**UNITED STATES PATENT AND TRADEMARK OFFICE**

---

**BEFORE THE PATENT TRIAL AND APPEAL BOARD**

---

INTEL CORPORATION,

Petitioner

---

IPR2025-00794
IPR2025-00795
U.S. Patent No. 10,333,768

---

**DECLARATION OF CHANDRAJIT L. BAJAJ, PH.D., UNDER 37 C.F.R. § 1.68 IN SUPPORT OF PETITION FOR *INTER PARTES* REVIEW**

**TABLE OF CONTENTS**

I. INTRODUCTION ..... 5
II. QUALIFICATIONS ..... 8
III. LEVEL OF ORDINARY SKILL IN THE ART ..... 14
IV. RELEVANT LEGAL STANDARDS ..... 15
V. BACKGROUND ..... 17
VI. THE '768 PATENT ..... 33
A. Summary of the '768 Patent ..... 33
B. Prosecution History ..... 37
VII. CLAIM CONSTRUCTION ..... 38
A. "*peer-to-peer architecture*" ..... 39
B. "*a mechanism for*" ..... 39
VIII. PRIOR ART ..... 46
A. Menon ..... 47
1. Summary of Menon ..... 47
2. Availability of Menon ..... 54
B. Trefethen ..... 55
1. Summary of Trefethen ..... 55
2. Availability of Trefethen ..... 58
C. RS6000 ..... 59
1. Summary of RS6000 ..... 59
2. Availability of RS6000 ..... 60
D. POEref ..... 61
1. Summary of POEref ..... 61
2. Availability of POEref ..... 62
E. MPIref ..... 64

1. Summary of MPIref ..... 64
2. Availability of MPIref ..... 66
IX. DETAILED UNPATENTABILITY ANALYSIS ..... 67
A. Ground 1: Claims 1-26, 29-30, 35, 36, and 39 are obvious over Menon in view of Trefethen, RS6000, and POEref ..... 68
1. Combining Trefethen with Menon ..... 68
2. Combining RS6000 with Menon ..... 73
3. Combining POEref with Menon ..... 76
4. Claim 1 ..... 79
5. Claim 2 ..... 141
6. Claim 3 ..... 143
7. Claim 4 ..... 144
8. Claim 5 ..... 149
9. Claim 6 ..... 152
10. Claim 7 ..... 157
11. Claim 8 ..... 165
12. Claim 9 ..... 168
13. Claim 10 ..... 170
14. Claim 11 ..... 174
15. Claim 12 ..... 176
16. Claim 13 ..... 178
17. Claim 14 ..... 180
18. Claim 15 ..... 182
19. Claim 16 ..... 183
20. Claim 17 ..... 186
21. Claim 18 ..... 189
22. Claim 19 ..... 189

23. Claim 20 ..... 190
24. Claim 21 ..... 192
25. Claim 22 ..... 193
26. Claim 23 ..... 194
27. Claim 24 ..... 198
28. Claim 25 ..... 198
29. Claim 26 ..... 201
30. Claim 29 ..... 216
31. Claim 30 ..... 220
32. Claim 35 ..... 226
33. Claim 36 ..... 282
34. Claim 39 ..... 286
B. Ground 2: Claims 27-28, 31-34, 37, and 38 are obvious over Menon in view of Trefethen, RS6000, and POEref, further in view of MPIref ..... 287
35. Combining MPIref with the MultiMATLAB system of Menon, Trefethen, RS6000 and POEref ..... 287
36. Claim 27 ..... 289
37. Claim 28 ..... 309
38. Claim 31 ..... 330
39. Claim 32 ..... 338
40. Claim 33 ..... 369
41. Claim 34 ..... 369
42. Claim 37 ..... 375
43. Claim 38 ..... 375
X. DECLARATION ..... 377

# I. INTRODUCTION

**1.** I, Chandrajit L. Bajaj, have been retained by counsel for Intel Corporation ("Petitioner") as a technical expert in connection with the proceeding identified above. I submit this declaration in support of Intel's Petition for *Inter Partes* Review ("IPR") of U.S. Patent No. 10,333,768 ("the '768 patent").

**2.** I am being compensated for my work in this matter at an hourly rate. I am also being reimbursed for reasonable and customary expenses associated with my work and testimony in this matter. My compensation is not contingent on the outcome of this matter or the specifics of my testimony. I have no personal or financial stake or interest in the outcome of this proceeding.

**3.** I have been asked to provide my opinions regarding whether the subject matter of claims 1 to 39 (the "Challenged Claims") of the '768 patent would have been obvious to a person having ordinary skill in the art ("POSITA") as of the earliest claimed priority date. After reviewing the patent claims, specification, and prosecution history, as well as the prior art references discussed below, it is my opinion that the Challenged Claims would have been obvious to a POSITA, as discussed below.

**4.** In the preparation of this declaration, I have considered:

(1) the '768 patent, Ex.1001;

(2) the prosecution history of the '768 patent, Ex.1002;

(3) "MultiMATLAB: Integrating MATLAB with High-Performance Parallel Computing," Menon et al., SC '97: Proceedings of the 1997 ACM/IEEE Conference on Supercomputing 1997 ("Menon"), Ex.1005;

(4) "MultiMATLAB: MATLAB on Multiple Processors," Trefethen et al., Cornell University 1996 ("Trefethen"), Ex.1006;

(5) "RS/6000 SP: Planning Vol. 1, Hardware and Physical Environment," IBM 2001, ("RS6000"), Ex.1007;

(6) "IBM Parallel Environment for AIX – Operation and Use, Volume 1, Using the Parallel Operating Environment," IBM 2001, ("POEref"), Ex.1008;

(7) "Single program multiple data" in Dictionary of Algorithms and Data Structures, P. E. Black, December 2004, Ex.1009;

(8) "The RS/6000 SP Inside Out" to Barrios et al., IBM 1999, Ex.1010;

(9) "High Performance Cluster Computing: Programming and Applications," Rajkumar Buyya, Vol. 2, 1999, Ex.1011;

(10) "Analysis of 100Mb/s Ethernet for the Whitney Commodity Computing Testbed," Fineberg et al., NAS Technical Report NAS-97-025 1997, Ex.1012;

(11) "A Parallel Linear Algebra Server for Matlab-like Environments," G. Morrow et al., SC '98: Proceedings of the 1998 ACM/IEEE Conference on Supercomputing 1998, Ex.1013;

(12) "Parallel MATLAB: Doing It Right," R. Choy et al., Proceedings of the IEEE, Vol. 93, No. 2, 2005, Ex.1014;

(13) "Mathematica Parallel Computing Toolkit - Unleash the Power of Parallel Computing," R. Maeder, 2005, Ex.1015;

(14) "Mastering MATLAB® 5 – A Comprehensive Tutorial and Reference," D. Hanselman et al., 1998, Ex.1016;

(15) "MPI: A Message-Passing Interface Standard," Message Passing Interface Forum, May 5, 1994 ("MPIref"), Ex.1017;

(16) "Modern Operating Systems," A.S. Tanenbaum, 2001, Ex.1018;

(17) Cornell Websites, Ex.1023;

(18) "pyMPI–An introduction to parallel Python using MPI," P. Miller 2002, Ex.1024;

(19) Amended Joint Claim Construction Chart for *Advanced Cluster Systems, Inc. v. NVIDIA Corporation*, 1:19-cv-02032 (D. DE.), Ex.1025;

(20) SC '97 Conference Materials, Ex.1026;

(21) Declaration of Gordon MacPherson, IEEE, Ex.1027;

(22) A copy of Menon with list of publications citing Menon, Ex.1028;

(23) Affidavit of Mina Ching, Internet Archive, Ex.1029;

(24) "Visual Supercomputing to be Demonstrated at SC97," HPC wire, 1997, Ex.1030;

(25) SC1997 flyer, 1997, Ex.1031;

(26) IEEE Xplore's "About IEEE Xplore" page, Ex.1032;

(27) "As Cornell Theory Center winds up Microsoft pact, it seeks faculty advice on future, direction," Cornell Chronicle, 2005; Ex.1033;

(28) ACM's description of "ACM Digital Library," Ex.1034;

(29) A copy of Trefethen with list of publications citing Trefethen, Ex.1035;

(30) "A Beginner's Guide to the IBM SP," Stephanie Wolf et al., University of Tennessee, 2000, Ex.1036;

(31) List of publications citing MPIref, Ex.1037;

(32) IBM website's documentation library for RS6000, Ex.1038

(33) IBM website's documentation library for POEref, Ex.1039; and

(34) ACS Preliminary Response, IPR2021-00019, Ex.1040.

**5.** In forming my opinions, I also have considered: the relevant legal standards, including the standard for obviousness, which Intel's counsel has explained to me; and my own knowledge and experience based upon my work in the field of cluster computing, which includes computer cluster systems and cluster computing techniques as described below; and relevant background additional materials cited in this declaration.

**6.** Unless otherwise noted, all emphasis in any quoted material has been added.

## II. QUALIFICATIONS

**7.** The details of my background and education, and a listing of my publications that I have authored, are provided in my *Curriculum Vitae*, a copy of which is submitted as Ex.1004. The following is a brief summary of my relevant qualifications and professional experience.

**8.** I received my Ph.D. and M.Sc. degrees in computer science from Cornell University in 1984 and 1983, respectively. I also received my Bachelor of Technology degree in electrical engineering from the Indian Institute of Technology Delhi in 1980.

**9.** During my college studies from 1975 to 1980, I focused on designing and using programmable micro-controllers based on commodity microprocessors, as well as very large scale integrated (VLSI) circuit design. I took specialized courses including graduate courses in computer algorithms and mathematical programming. As part of my doctoral research at Cornell from 1980 to 1984, I worked on developing and implementing methods for large combinatorial and geometric optimization problems.

**10.** From 1984 to 1997, I was a professor of computer sciences at Purdue University while also serving as the director of the Image Analysis and Visualization Center at the university. From 1990 to 1991, I was a visiting associate professor of computer sciences at Cornell University. As part of my work at Purdue and Cornell, I researched, taught, and used high-performance, distributed, and parallel computer systems. I developed new algorithms to model, simulate, and visualize natural and synthetic 3D and 4D virtual environments, combining computational scanning, and rapid scene generation algorithms deployed on distributed, collaborative, and parallel architectures. During this time, I authored or co-authored about 150 papers, book chapters, conference papers, and technical reports. (*see* my *Curriculum Vitae*). These include the following sample list of peer-reviewed papers directed to high-performance, distributed, and parallel computer systems, which demonstrated remote and shared audio-video and CAD design

applications developed over peer-to-peer networked computer graphics workstations: "Shastra: Multimedia Collaborative Design Environment," "SHASTRA - An Architecture for Development of Collaborative Applications," "Collaborative Multimedia in SHASTRA," "Distributed and Collaborative Synthetic Environments," etc.

**11.** From 1997 to the present, I have been a professor of computer sciences at the University of Texas at Austin. During this time, I have researched, taught, and used high-performance, distributed, and parallel computer systems including commodity of-the-shelf (COTS) clusters systems. I have also been an active member of a team developing hardware and software technology that allowed multiple computers with multiple programmable graphics cards (GPUs) to simultaneously and synchronously display to large multi-screen immersive displays. Our team called this multi-displays and many-graphics networked processors, cluster programming technology the UT Meta-Buffer solution. One of the peer-reviewed publications that resulted from this is titled "Active Visualization in a Multi-display Immersive Environment."

**12.** Much of my past and current work involves issues relating to 2D and 3D image capture, and reconstruction, compression, image and geometric data processing, modeling, and quantitative analysis. I have participated in the design and use of several computer systems spanning handhelds, laptops, and graphics

workstations to PC/Linux clusters, as well as very large memory supercomputers for capturing, modeling, and displaying virtual and scientific phenomena. My experience with computational modeling and displaying computer graphics imagery touches many fields, such as interactive games, molecular, biomedical and industrial diagnostics, oil and gas exploration, geology, cosmology, and military industries. During this time, I authored or co-authored about 300 papers, book chapters, conference papers, and technical reports, a select sample of which include the following papers directed at high-performance, distributed, and parallel computer systems or cluster (*see* my *Curriculum Vitae*):

- "Parallel Accelerated Isocontouring for Out-of-Core Visualization," *In Proceedings of the 1999 IEEE Symposium on Parallel Visualization and Graphics* (1999);
- "Web based Collaborative Visualization of Distributed and Parallel Simulation," *In Proceedings of the 1999 IEEE Symposium on Parallel Visualization and Graphics* (1999);
- "Parallel and Out-of-core View-dependent Isocontour Visualization Using Random Data Distribution," *Joint Eurographics-IEEE TCVG Symposium on Visualization* (2002);

- "Scalable Isosurface Visualization of Massive Datasets on Commodity off-the-Shelf Clusters," *Journal of Parallel and Distributed Computing* (2009);
- "Visualization-Specific Compression of Large Volume Data," *Proceedings of Pacific Graphics* (2001);
- "Visualization of Very Large Oceanography Time-Varying Volume Datasets," *Lecture Notes in Computer Science (LNCS)* (2004);
- "SIMD Optimization of Linear Expressions for Programmable Graphics Hardware," *Computer Graphics Forum* (2004);
- "Multi-level Grid Algorithms for Faster Molecular Energetics," *Proceedings of the ACM Symposium on Solid and Physical Modeling* (2010), and
- "Accelerated Molecular Mechanical and Solvation Energetics on Multicore CPUs and Many-Core GPUs," *Proc. Of the 6th ACM Conference on Bioinformatics Computational Biology and Health Informatics* (2015).

**13.** I am knowledgeable about and have experience in developing both parallel hardware and software in the field of high-performance, distributed, and parallel computer systems or clusters, including for many core GPUs and multi-core CPU cluster systems. Specifically, I led a team that built the first data intensive and

display intensive visualization commodity cluster with 128 workstations interconnected with InfiniBand and used in driving immersive displays and visualization walls.

**14.** During my tenure at the University of Texas at Austin, I also began to create spatially-realistic 3D graphical environments of nature's molecules and cells with a combination of different types of acquired and reconstructed imagery within which a user may explore, query, and learn. My publication titled "From Voxel Maps to Models," which appeared in an Oxford University Press book called *Imaging Life: Biological Systems from Atoms to Tissues*, Gary C. Howard, William E. Brown & Manfred Auer eds., Oxford University Press, 2015, is an example of my research in computational imaging.

**15.** As outlined in my *Curriculum Vitae*, I have authored approximately 169 peer-reviewed journal articles, 35 peer-reviewed book chapters, and 173 peer-reviewed journal and 162 peer-reviewed conference publications. I am also a co-inventor of U.S. Patent No. 6,438,266, titled "Encoding Images of 3-D Objects with Improved Rendering Time and Transmission Processes," issued on August 20, 2002.

**16.** I have written and edited four books, on topics ranging from image processing, geometric modeling, and visualization techniques to algebraic geometry and its applications. I have given 234 invited speaker, plenary and keynotes on

these topics. I am a Fellow of the American Association for the Advancement of Science, a Fellow of the Institute of Electrical and Electronics Engineers (IEEE), a Fellow of the Society of Industrial and Applied Mathematics (SIAM), and a Fellow of the Association of Computing Machinery (also known as ACM), which is the world's largest education and scientific computing society. ACM Fellow is ACM's most prestigious member grade and recognizes the top 1% of ACM members for their outstanding accomplishments in computing and information technology and/or outstanding service to ACM and the larger computing community.

**17.** Additional relevant experience in the field of distributed and parallel algorithms and numeric, symbolic and geometric computing, including machine learning are listed in my *Curriculum Vitae*.

## III. LEVEL OF ORDINARY SKILL IN THE ART

**18.** I understand from Intel's counsel that there are multiple factors relevant to determining the level of ordinary skill in the pertinent art, including (1) the levels of education and experience of persons working in the field at the time of the invention; (2) the sophistication of the technology; (3) the types of problems encountered in the field; and (4) the prior art solutions to those problems.

**19.** A POSITA in the field of the '768 patent, as of the earliest claimed priority date of June 13, 2006, would have been someone knowledgeable and familiar with computer cluster systems and cluster computing techniques available

at the time. Such a POSITA would have a bachelor's degree in computer science, computer engineering, electrical engineering, or an equivalent training, and approximately two years of experience working in the field of cluster computing or parallel processing and would be knowledgeable regarding high-level scientific computing languages. Additional work experience can substitute for specific educational background, and vice versa.

**20.** Such a POSITA would also have had a general knowledge of the internet and would have been familiar in the use of the internet to search for and retrieve documents. In particular, a POSITA would have known how to search for and download technical documents from the websites of universities, research institutions, companies, etc.

**21.** For purposes of this Declaration, in general, and unless otherwise noted, my statements and opinions, such as those regarding my own experience and what a POSITA would have understood or known generally (and specifically related to the references I consulted herein), reflect the knowledge that existed in the relevant field as of the priority date of the '768 patent.

## IV. RELEVANT LEGAL STANDARDS

**22.** I am not an attorney. In preparing and expressing my opinions and considering the subject matter of the '768 patent, I am relying on certain legal principles that counsel has explained to me.

**23.** I understand that prior art to the ’768 patent includes patents and printed publications in the relevant art that predate the priority date of the ’768 patent. For purposes of this Declaration, I am applying the earliest claimed priority date of June 13, 2006, as the priority date of the ’768 patent.

**24.** I have been informed by Intel’s counsel that a claimed invention is unpatentable under 35 U.S.C. § 103 if the differences between the claimed invention and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a POSITA. I have also been informed by Intel’s counsel that the obviousness analysis considers factual inquiries, including the level of ordinary skill in the art, the scope and content of the prior art, and the differences between the prior art and the claimed subject matter.

**25.** I understand from Intel’s counsel that several rationales can demonstrate how a POSITA would have combined or modified prior art teachings to arrive at the claimed invention. These rationales include: (a) combining prior art elements according to known methods to yield predictable results; (b) simple substitution of one known element for another to obtain predictable results; (c) use of a known technique to improve a similar device (method, or product) in the same way; (d) applying a known technique to a known device (method, or product) ready for improvement to yield predictable results; (e) choosing from a finite number of identified, predictable solutions, with a reasonable expectation of success; and (f)

some teaching, suggestion, or motivation in the prior art that would have led a POSITA to modify the prior art or to combine prior art teachings to arrive at the claimed invention.

**26.** Also, I have been informed and understand that obviousness does not require physical combination/bodily incorporation, but rather consideration of what the combined teachings would have suggested to a POSITA at the time of the alleged invention.

**27.** I have also been informed and understand that for purposes of *Inter Partes* Review, the claim terms must be construed according to their ordinary and customary meaning as would have been understood by a POSITA during the relevant timeframe.

## V. BACKGROUND

**28.** I discuss in this section general background information regarding computer clusters and cluster computing techniques. It is my opinion that the information I discuss in this "BACKGROUND" section would have been background knowledge to a POSITA.

**29.** **<u>Cluster Computing</u>**

**30.** The '768 patent claims computer clusters which, according to the patent, "include a group of two or more computers, microprocessors, and/or processor cores ('nodes') that intercommunicate so that the nodes can accomplish a

task as though they were a single computer," and where the communication mechanism is a message passing interface (MPI). Ex.1001, 1:19-22, claims 1 and 3. Computer clusters (and cluster computing techniques), however, were well known in the art prior to the earliest priority date of the '768 patent. For example, "High Performance Cluster Computing: Programming and Applications" by R. Buyya, Vol. 2 ("Buyya," Ex.1011), published in 1999, describes computer clusters comprising a plurality of nodes and mechanisms such as MPI for the nodes to communicate in a peer-to-peer fashion, where the nodes have single-node programs executing in a single-program multiple-data (SPMD) cluster computing paradigm, and parallel algorithms running on the computer clusters can employ data pipeline structures for data flows between adjacent nodes. Ex.1011, 4-7 (cluster computers), 19-22 (SPMD cluster computing paradigm and data pipelining), 53-56 (peer-to-peer communication), and 56-60 and 64-68 (mechanisms for cluster nodes to communicate with each other).

**31.** Advances in "commodity" computer technology and "off-the-shelf" networking technology in the early 1990s, more than a decade before the '768 patent's earliest priority date, enabled "[s]calable computing clusters, ranging from a cluster of (homogeneous or heterogeneous) PCs or workstations, to SMPs [symmetric multiprocessors]" to become "the standard platforms for high-performance and large-scale computing." Ex.1012, 2; Ex.1011, 4. "A cluster

generally refers to two or more computers (nodes) connected together" and "is a type of parallel or distributed processing system, which consists of a collection of interconnected stand-alone computers working together as a single integrated computing resource." Ex.1011, 5-6. FIG. 1.1 of Buyya below shows "a typical architecture of a cluster." Ex.1011, 6.

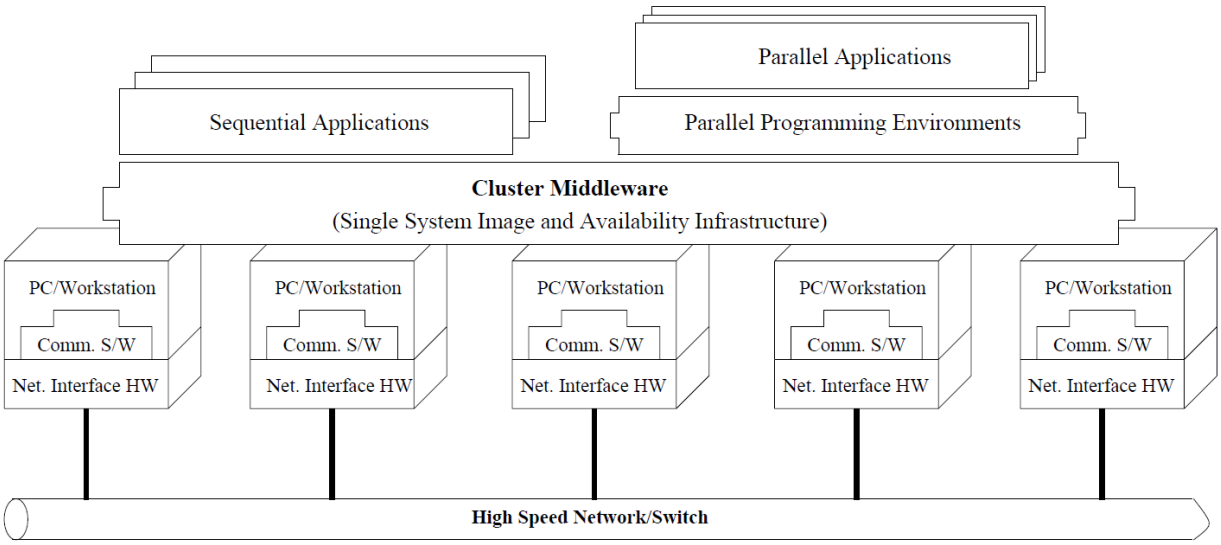



**Figure 1.1** Cluster computer architecture.

**Ex.1011, FIG. 1.1**

**32.** Each computer node of a cluster "can be a single or multiprocessor system (PCs, workstations, or SMPs) with memory, I/O facilities, and an operating system," and can work collectively with the other nodes of the cluster "as an integrated computing resource" or "can operate as [an] individual computer[]." Ex.1011, 5-6, 7. Communication between cluster nodes is enabled by a "network interface hardware [that] acts as a communication processor and is responsible for

transmitting and receiving packets of data between cluster nodes via a network/switch" and a "[c]ommunication software [that] offers a means of fast and reliable data communication among cluster nodes and to the outside world." Ex.1011, 7. Parallel programming environments such as message passing interface (MPI) provide efficient tools such as message passing libraries to facilitate the development of parallel applications on computer clusters. Ex.1011, 7.

**33. <u>Parallel Mathematical Processing</u>**

**34.** At the relevant time frame of the '768 patent, there were multiple well-known computer programs used to assist engineers with complex mathematical equations. Two such programs were Mathematica and MATLAB. Mathematica and MATLAB were and are very powerful tools used by engineers, supporting linear algebra as well as matrix and vector manipulation. They were well known to require extensive processing power to provide a timely result. Of course it was generally desirable to speed up the time it took to get results, so that the corresponding engineering problems being analyzed could be more quickly addressed.

**35.** The '768 patent identifies, and recites in its claims, a Mathematica kernel as a single-node kernel of a cluster node. Ex.1001, 1:38-41; claim 2. However, this was nothing new, as it was known prior to the '768 patent's earliest priority date that any number of Mathematica kernels can run individually on

single- or multi-processor machines communicating with each other via a network communication protocol. Ex.1015, Preface.

**36.** Specifically, prior to the '768 patent's earliest priority date, Mathematica was parallelized via a parallel computing toolkit designed to allow a user "to control several *Mathematica* kernel processes from within *Mathematica*," where the several Mathematica kernel processes run at "a number of remote computers capable of running *Mathematica*" or "a multiprocessor local machine" that has the required Mathematica licenses. Ex.1015, 1-2 (emphasis original).

**37.** Besides the parallel computing toolkit, there were several efforts in the 1990s "to exploit parallelism from within interactive mathematical software packages." Ex.1013, 1. For example, there were several efforts dedicated to parallelizing MATLAB, a product that was a competing product to Mathematica–by one count, there were 27 projects at the time for parallelizing MATLAB alone. Ex.1014, 332. One of these projects was MultiMATLAB. Ex.1014, 333; Ex.1005, Abstract; Ex.1006, Abstract. MultiMATLAB, "one of the most well known parallel MATLABs," "is a general extension of the MATLAB environment to any distributed memory multiprocessors." Ex.1014, 333; Ex.1005, 1. Specifically, it is an "extension[] of the Matlab interpreter that essentially run on each node of the parallel machine," i.e., the MultiMATLAB architecture is a computer cluster where MATLAB runs on each node of the computer cluster. Ex.1013, 1. MultiMATLAB's

applicability is not limited to MATLAB only, as the designers "present[ed] the MultiMATLAB architecture as a promising design for a parallel numerical computing environment." Ex.1005, 16. This follows from the designers' decision to "to build upon MATLAB itself" while avoiding "any changes in the MATLAB architecture; indeed, we have not had access to the MATLAB source code." Ex.1006, 12.

**38. <u>Peer-to-peer Architecture</u>**

**39.** Cluster nodes each having single-node programs were known in the art prior to the '768 patent's earliest priority date, as Buyya explains that each process executing on a node of a cluster "executes basically the same piece of code but on a different part of the data" in the most common parallel programming paradigms, the SPMD cluster computing paradigm. Ex.1011, 19. SPMD cluster computing paradigm "involves the splitting of application data among the available processors," where "[p]rocessors communicate with neighbouring processors," "some global synchronization [may be performed] periodically among all the processes," and "data … may be read from the disk during the initialization stage." Ex.1011, 19-20. FIG. 1.5 of Buyya below shows a schematic representation of the SPMD cluster computing paradigm.

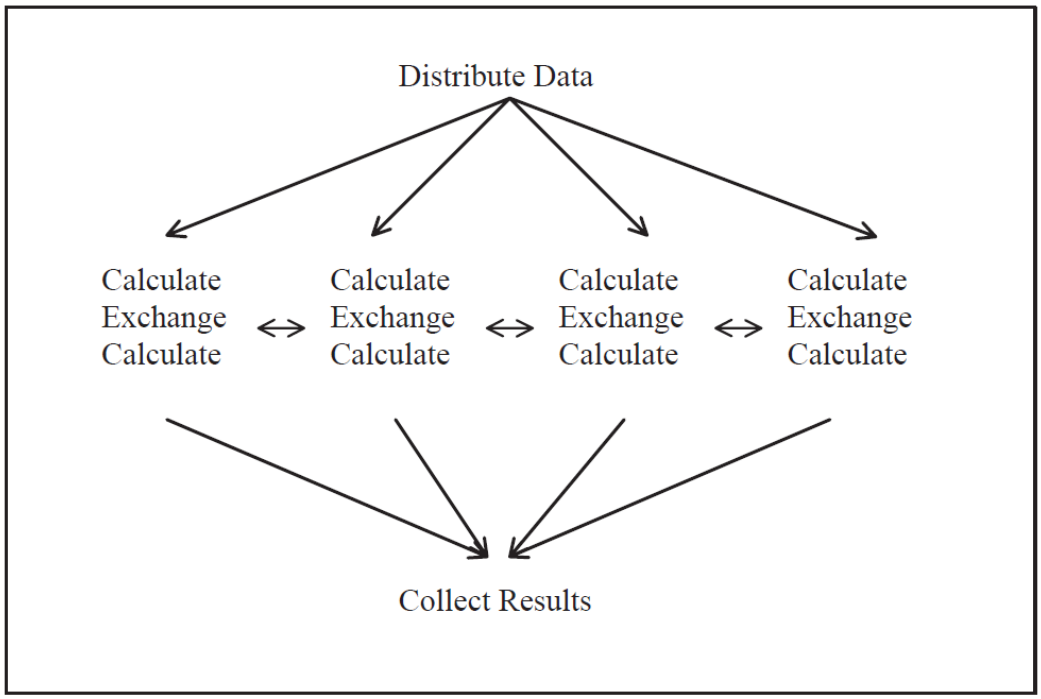



**Figure 1.5** Basic structure of a SPMD program.

**Ex.1011, FIG. 1.5**

**40.** As noted above, the '768 patent claims MPI as the "mechanism for the nodes [of the computer cluster] to communicate … with each other using a peer-to-peer architecture." Ex.1001, claims 1 and 3. The use of MPI to enable processes on cluster nodes in a peer-to-peer network to communicate with each other, however, was known before the earliest priority date of the '768 patent, as exemplified by Buyya's explanation that "MPI consists of a specification of a set of library functions that tasks of a parallel program can use to communicate among themselves" and "[t]he MPI environment supports [] the all-peers … programming model[] very well." Ex.1011, 56, 58. MPI enables "both point-to-point and

collective [communications] within a group of tasks," where point-to-point communication refers to "direct communication between a source and a destination task" using "point-to-point send and receive functions" and collective communication refers to "communication between a group of MPI tasks, where all tasks belonging to that group participate in the communication" using a "set of collective communication primitives." Ex.1011, 57-58. Example send and receive functions include MPI_Send() and MPI_Recv(), respectively. Ex.1011, 60.

**41.** Buyya provides an example implementation of MPI using the SPMD cluster computing paradigm in a peer-to-peer network. Ex.1011, 53, 55, 64-68. The implementation is with reference to the parallel Floyd algorithmic solution to "the all pairs shortest path problem":

> each task is a worker, with the *task identifier* (*taskid*) assigned by MPI serving as its instance identifier. One of the tasks is designated as the one responsible for reading in the input matrix and distributing the relevant portions to its peers. This task is the one with the taskid of 0 []. At the end, it collects the results from its peers, which together yield the solution to the all pairs problem.

Ex.1011, 53, 55-56 (emphasis original). During "the main computation (interspersed with communication) phase," "the peers work on their respective stripes of the input data matrix," communicating "among themselves in every iteration of updates to the matrix to get the most up-to-date value." Ex.1011, 64.

"At the end of this main phase of the algorithm, the peers send their results to the pseudo-master. Since all the tasks are peers, and the pseudo-master is arbitrarily chosen to be the task with rank MASTERTID, all the peers know a priori who to send the results back to." Ex.1011, 64. Excerpts of Buyya's Listing 3.3, which is "a code excerpt from the Parallel Floyd's MPI program, with the MPI calls [] **highlighted**," are shown below. Ex.1011, 64 (emphasis original). The two excerpts of Listing 3.3 show taskid = 0 (the pseudo-master or cluster computer coordinating task) distributing calls (i.e., matrix stripes) to the other tasks (using MPI command MPI_Send) and the other tasks receiving the calls sent by taskid = 0 (using MPI command MPI_Recv). Ex.1011, 65-66.

```
/* Get our task ID (our rank in the basic group). */
MPI_Comm_rank(MPI_COMM_WORLD, &myTid);

/* Get the number of tasks. */
MPI_Comm_size(MPI_COMM_WORLD, &nTasks);

/* The master reads the input, and distributes it to the workers. */
if (myTid == MASTERTID) {
    /*
     * Read in the number of vertices and the input matrix itself from a
     * specified file.
     */
    :
    /* Assign stripes to everyone, including ourselves. */
    sendWork();
}
else {
    collectWork();
}

/* Calculate shortest paths for the stripe assigned to us. */
calcShort();
```

**Ex.1011, Listing 3.3 (excerpted)**

```
void sendWork()
{
    for (i = 0; i < nTasks; i++) {
        if (i == MASTERTID) {
            /* Read in my stripeMatrix entries directly from inMatrix. */
            :
        }
        else {
            /* Calculate the stripe for each peer task and send it to them. */
            bufPos = 0;
            MPI_Pack(&nVert, 1, MPI_INT, buf, BUFSIZE, &bufPos,
                     MPI_COMM_WORLD);
            for (j = 0; j < nVert; j++) {
                MPI_Pack(&(inMatrix[j][begCol]), width, MPI_INT,
                         buf, BUFSIZE, &bufPos, MPI_COMM_WORLD);
            }
            MPI_Send(buf, bufPos, MPI_PACKED, i, WORK_MSG,
                     MPI_COMM_WORLD);
        }
    }
} /* end of "sendWork" */

void collectWork()
{
    bufPos = 0;

    /* Get stripe matrix from master. */
    MPI_Recv(buf, BUFSIZE, MPI_PACKED, MPI_ANY_SOURCE, WORK_MSG,
             MPI_COMM_WORLD, &status);

    /* Unpack data. */
    MPI_Unpack(buf, BUFSIZE, &bufPos, &nVert, 1, MPI_INT,
               MPI_COMM_WORLD);

    /* Calculate my stripe width and allocate stripeMatrix. */
    :
    for (j = 0; j < nVert; j++) {
        MPI_Unpack(buf, BUFSIZE, &bufPos, stripeMatrix[j],
                   stripeWidth, MPI_INT, MPI_COMM_WORLD);
    }
} /* end of "collectWork" */
```

**Ex.1011, Listing 3.3 (excerpted)**

**42.** The parallel MATLAB architecture discussed above, MultiMATLAB, also employs the MPI communication standard for communication between MATLAB processes running on the processor nodes of the MultiMATLAB architecture. Ex.1005, 3. The MultiMATLAB architecture comprises multiple processors where "each processor in [the] parallel platform individually runs a

MATLAB process," as shown below. Ex.1005, 3. "MATLAB processes [are run] on multiple processors, with full access to all the usual capabilities such as Toolboxes. These processes communicate via simple MATLAB-style commands built on MPI … Both master/slave and SPMD paradigms are implemented." Ex.1006, 12.

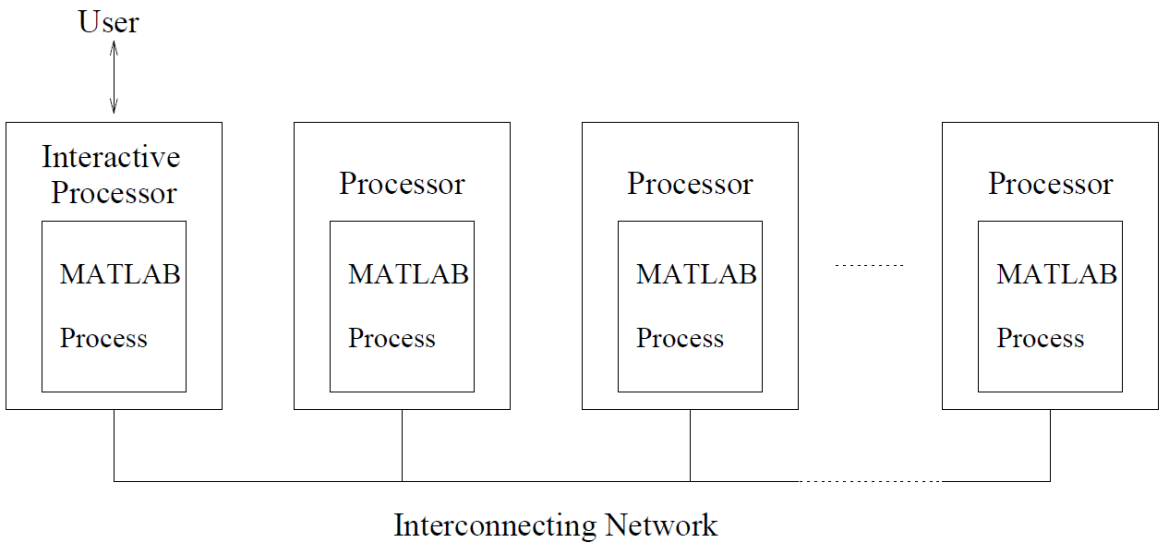



Figure 1: MultiMATLAB Architecture

**Ex.1005, FIG. 1**

**43.** As shown in the figure above, the MultiMATLAB architecture has an interconnection network that connects the processors of the MultiMATLAB architecture to each other. Ex.1005, 3, FIG. 1. The communication layer MPI "runs over the parallel platform's interconnection network," providing each MATLAB process "with the ability to communicate with other [MATLAB] processes," i.e., the interconnection network is a peer-to-peer network. Ex.1005, 3.

**44.** Other literature also illustrates a POSITA's understanding of MPI. For example, "MPI provides explicit point-to-point messaging operations." Ex.1024, 6. MPI implementations use the "send, recv" functions, as well as queues to send data among processes. Ex.1024, 6, 11.

**45.** **<u>Sequential Processing from One Node to the Next</u>**

**46.** Also known in the art prior to the '768 patent's earliest priority date was the fact that data communication between the cluster nodes can be in a sequential or pipelined fashion. Ex.1011, 21-22. In this approach, "the tasks of the algorithm, which are capable of concurrent operation, are identified and each processor executes a small part of the total algorithm." Ex.1011, 21. Data, such as the results of a process's computation of a task at a cluster node, is then sent to the process at the next cluster node in the pipeline, as explained by Buyya:

> Processes are organized in a pipeline - each process corresponds to a stage of the pipeline and is responsible for a particular task. The communication pattern can be very simple since the data flows between the adjacent stages of the pipeline. For this reason, this type of parallelism is also sometimes referred to as data flow parallelism. The communication may be completely asynchronous.

Ex.1011, 21. The "pipeline is one of the simplest and most popular functionality decomposition paradigms." Ex.1011, 21. FIG. 1.6 of Buyya below shows a schematic representation of the data pipeline structure.

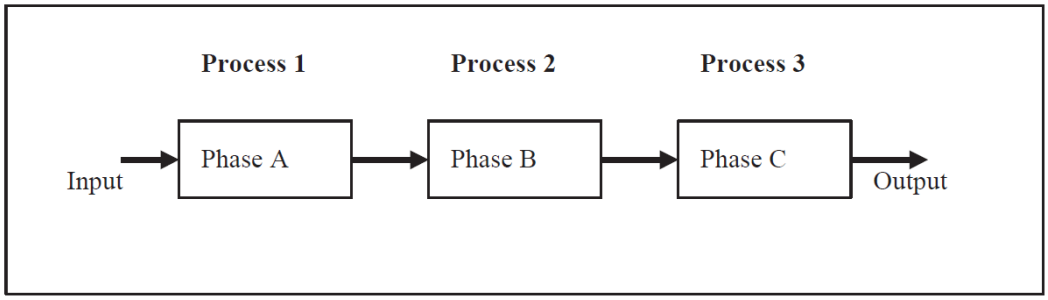



**Figure 1.6** Data pipeline structure.

**Ex.1011, FIG. 1.6**

**47.** As is notable from Figure 1.6 of Ex.1011, POSITAs knew and understand that a pipeline paradigm has communication flow from the first process to a second process to a third process and then provides the resulting output.

**48.** Buyya teaches combining paradigms (e.g., such as the SPMD cluster computing paradigm and the data pipeline structure for data flows) when discussing "the need, [in some applications,] to mix elements of different paradigms," which may involve "situations where it makes sense to mix data and task parallelism simultaneously." Ex.1011, 23. For example, the SPMD cluster computing paradigm and the data pipeline structure for data flows, depicted above in Buyya's FIG. 1.5 and FIG. 1.6, respectively, can be combined to implement an example situation where data is distributed, as shown in FIG. 1.5, to one process (e.g., process 1 in

FIG. 1.6 above), and then data propagates from this process to all the other processes on the cluster nodes, as shown in FIG. 1.6 above, while the processes perform calculations on their respective received data, as shown in FIG. 1.5.

**49.** Buyya explains that the SPMD cluster computing paradigm (and its combination with the data pipeline structure for data flows) can be implemented in computer clusters with the cluster nodes communicating with each other "in the all-peers paradigm," i.e., via a peer-to-peer network. Ex.1011, 53-55. In a peer-to-peer network setting, one process at a cluster node may function as a coordinating process that, for example, receives user instructions and distributes commands or calls to the other processes of the cluster computer:

> In the all-peers paradigm, all tasks participate in the computation and are referred to as worker tasks in this scenario. Each of these tasks decides what portion of the problem it will solve, based on some common algorithm. In general, all tasks control the computation as a whole, communicating with each other as necessary both for computation and control purposes. Only one of the peer tasks may still read in the input data and distribute it to the rest of its peers, and then collect the results in the end.

Ex.1011, 53. FIG. 3.3 of Buyya below shows a peer-to-peer network comprising *n* cluster nodes where "a task communicates with all other tasks." Ex.1011, 53.

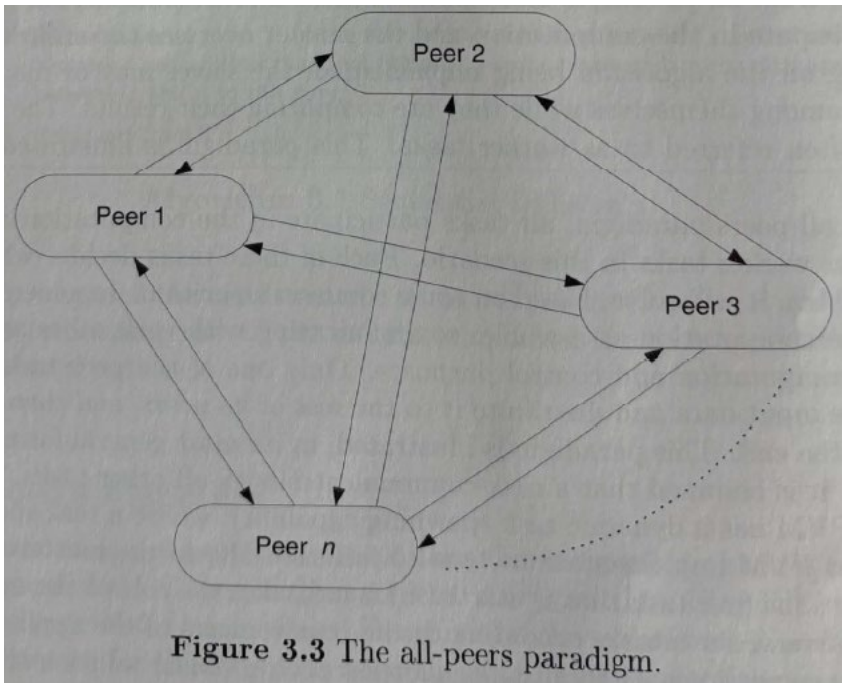



**Figure 3.3** The all-peers paradigm.

**Ex.1011, FIG. 3.3**

**50.** The aforementioned MultiMATLAB architecture employs a combination of a SPMD cluster computing paradigm with a data pipeline structure for data flows, as illustrated with the following example script (m-file), "cycle.m," executed on 6 processors that constitute a computer cluster implementing a MultiMATLAB architecture:

```
if ID==0                    % first p
   a = 1
   Send(ID+1,a)
elseif ID == Nproc-1        % last process: receive and double
   a = 2*Recv
else                        % middle
   a = 2*Recv
   Send(ID+1,a)
end;
```

MATLAB process ID=0 creates variable a=1 and sends it to MATLAB process ID=1

Last MATLAB process receives the variable "a" and doubles it

Process 0 creates the variable a with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of a, and sends it along to process 2; and so on. If there are six processors the command `Eval( 'cycle' )` produces the output

```
a = 1
a = 2
a = 4
a = 8
a = 16
a = 32
```

MATLAB processes ID receive the variable "a," double it, and send the doubled variable "a" to the next MATLAB process ID+1

Ex.1006, 6 (annotated).

**51.** I note that the m-file "cycle.m," when executed in a MultiMATLAB architecture comprising multiple processors communicating in a SPMD paradigm, results in the sequential evaluations of mathematical expressions by processors that make up the computer cluster, where the final result is communicated to the node that distributed the call for the execution of the mathematical expressions. Ex.1006, 6.

**52.** Accordingly, it was well known, and a POSITA would have understood, before the priority date of the '768 patent for a computer cluster to have a plurality of nodes in a peer-to-peer network each having a processor and a memory, where each node has a single-node program that executes using the processor and communicates with any other node using MPI, and where one of the

nodes functions as a coordinating node of the computer cluster during parallel computation in that it receives user instructions, distributes calls to all the other nodes based on the user instructions, and then collects results from the other nodes upon completion of each node's computation.

## VI. THE '768 PATENT

### A. Summary of the '768 Patent

**53.** The '768 patent describes and claims nothing more than well-known computer cluster systems and cluster computing techniques. After describing computer clusters as systems that "include a group of two or more computers, microprocessors, and/or processor cores ('nodes') that intercommunicate so that the nodes can accomplish a task as though they were a single computer," the '768 patent states that "[m]any computer application programs are not currently designed to benefit from advantages that computer clusters can offer, even though they may be running on a group of nodes that could act as a cluster." Ex.1001, 1:19-26. The purported novelty of the '768 patent is that it discloses "systems and methods for adding cluster computing functionality to a computer program." Ex.1001, 1:14-16. Besides the "Mathematica software," the '768 patent lists "Maple®, MATLAB®, … other applications employing an interpreter or a kernel," as candidate programs for which the systems and methods can be used to add cluster computing functionality. Ex.1001, 4:14-25.

**54.** An example implementation is shown in FIG. 2, reproduced below, where software modules such as "kernel modules 206 a-e are designed for single-threaded execution," and "each … is in communication with a single cluster node module 204 a-e, respectively," run on a computer cluster. Ex.1001, 5:19-20, 33-35. "MPI calls and advanced cluster commands are used to parallelize program code received from an optional user interface module 208 and distribute tasks among the kernel modules 206 a-e. The cluster node modules 204 a-e provide communications among kernel modules 206 a-e while the tasks are executing." Ex.1001, 6:16-21. Examples of the kernel modules 206 a-e are Mathematica kernels and those of the cluster node modules 204 a-e are Message-Passing Interfaces ("MPIs"). Ex.1001, 11:42-48. The tasks include "performing calculations, processing, or other work" from the "simple (for example, '1+1'), [to] entire subroutines and sequences of code (such as, for example, Mathematica code)." Ex.1001, 25:41-42, 24:58-61.

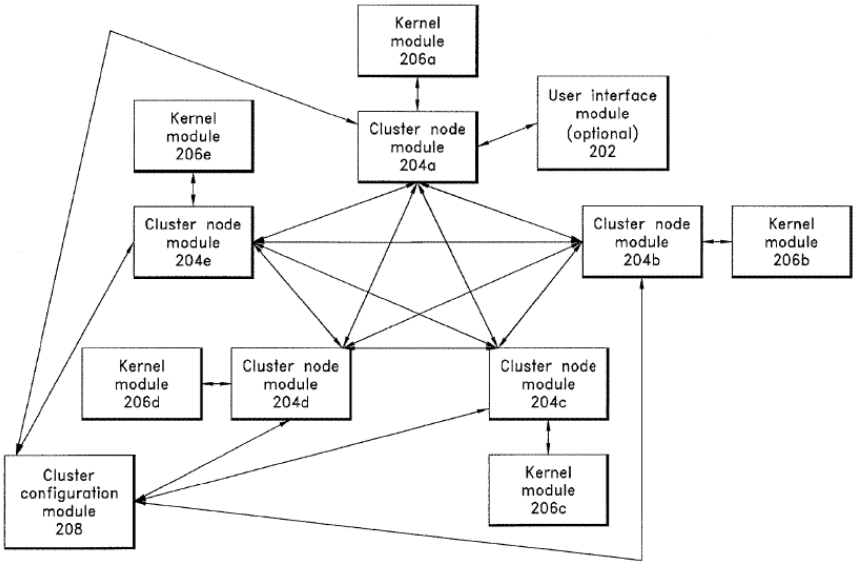



*FIG. 2*

**Ex.1001, FIG. 2**

**55.** Representative Claim 1 of the ’768 patent is reproduced below (Section VI.B has discussion related to “Amended limitation”):

**1**. A computer cluster comprising:

a plurality of nodes, wherein each of the plurality of nodes comprises a hardware processor, wherein one or more of the nodes are configured to receive a command to start a cluster initialization process for the computer cluster, and wherein each of the nodes is configured to access a non-transitory computer-readable medium comprising program code for a single-node kernel that, when executed, is capable of causing the hardware processor to evaluate mathematical expressions; and

a mechanism for the nodes to communicate results of mathematical expression evaluation with each other using a peer-to-peer architecture;

wherein the plurality of nodes comprises:

a first node comprising a first hardware processor configured to access a first memory comprising program code for a user interface and program code for a first single-node kernel, the first single-node kernel configured to interpret user instructions and distribute calls to at least one of a plurality of other nodes for execution; and

a second node comprising a second hardware processor with a plurality of processing cores, wherein the second node is configured to receive calls from the first node, execute at least a first mathematical expression evaluation, and communicate a result of the first mathematical expression evaluation to a third node;

wherein the third node comprises a third hardware processor with a plurality of processing cores, wherein the third node is configured to receive the result of the first mathematical expression evaluation from the second node, execute at least a second mathematical expression evaluation using the received result, and communicate the result of the second mathematical expression evaluation to the first node;

wherein the first node is configured to return the result of the second mathematical expression evaluation to the user interface;

wherein one or more of the nodes are configured to:

accept user instructions;

after accepting user instructions, communicate at least some of the user instructions using the mechanism for the nodes to communicate with each other; and

after communicating at least some of the user instructions using the mechanism, communicate at least some of the user instructions to one or more single-node kernels.

Amended limitation

**56.** As I explain in detail below, there is nothing novel about the alleged invention disclosed and claimed in the ’768 patent. For example, MultiMATLAB, “a general extension of the MATLAB environment to any distributed memory multiprocessors,” based on “based upon the MPI communication standard” and “useful for … fast and convenient execution of easily parallelizable numerical computations on multiple processors” was already known well before the earliest priority date of the ’768 patent. Ex.1005, Abstract; Ex.1006, Abstract.

**B. Prosecution History**

**57.** The ’768 patent was filed on February 14, 2014 and issued on June 25, 2019. In response to a rejection by the Examiner, Applicant amended the pending claims to recite “a mechanism for the nodes to communicate … with each other using a peer-to-peer architecture.” Ex.1002, 481. The Applicant explained that “peer-to-peer architecture … makes intermediate results of computation available to other nodes,” and supported this description of “peer-to-peer architecture” by pointing to a portion of the specification where it is discussed that “gridMathematica software package,” which “connect Mathematica kernels in a master-slave relationship rather than a peer-to-peer relationship,” “does not provide a means for instances of the same code running on different nodes to communicate, collaborate, or coordinate work among the instances.” Ex.1002, 488; Ex.1001, 12:26-32. The Applicant reinforced this understanding (of “peer-to-peer

architecture" as one that allows inter-node communication) by pointing to another portion of the specification where it is discussed that "[g]rid computers [that] include at least one node known as a master node that manages a plurality of slave nodes or computational nodes" "include a plurality of nodes that generally do not communicate with one another as peers." Ex.1002, 488; Ex.1001, 1:46-53.

**58.** The Examiner allowed the claims to issue after the Applicant authorized the Examiner's proposal to incorporate the subject matter identified as "Amended limitation" in Section VI.A into the independent claims:

> The following is an examiner's statement of reasons for allowance.
>
> Independent Claims 2 as amended distinguishes itself over the prior art due to the amended limitation in combination with the rest of the limitations. It is to be noted that it is the combination of all limitations that renders the claims allowable. Claims 3-25, and 27-41 are allowed based on the same reason(s).

Ex.1002, 79.

**59.** Notably, as I demonstrate below, there is nothing new about claim 1 (or any of the other claims) of the '768 patent since the amended limitation, as well as its combination with all the other limitations of the claim(s), which appear to be the reason for allowance, were known in the art and it would have been obvious to combine the prior art as claimed.

## VII. CLAIM CONSTRUCTION

**60.** It is my understanding that in order to properly consider whether the '768 patent would have been obvious to a POSITA, it is necessary to give meaning

to claim terms in view of the claim language itself, the specification, and prosecution history. In determining how a POSITA would understand claim terms, I have applied the legal principles discussed above in Section IV regarding construing claim terms according to their ordinary and customary meaning. I have also been informed that claim terms only need to be construed to the extent necessary to resolve the obviousness inquiry. It is my opinion that the following terms have the meanings discussed below.

**A. "*peer-to-peer architecture*"**

**61.** The limitation "peer-to-peer architecture" appears in each of independent claims 1 and 35.

**62.** Patent Owner has construed "peer-to-peer architecture" as requiring "an architecture in which each node can communicate tasks and data with other nodes without the tasks and data being required to go through a central server or master node." Ex.1040, 8-12. I will apply this construction for the term when applying the prior art presented herein to evaluate the patentability of the claims.

**B. "*a mechanism for*"**

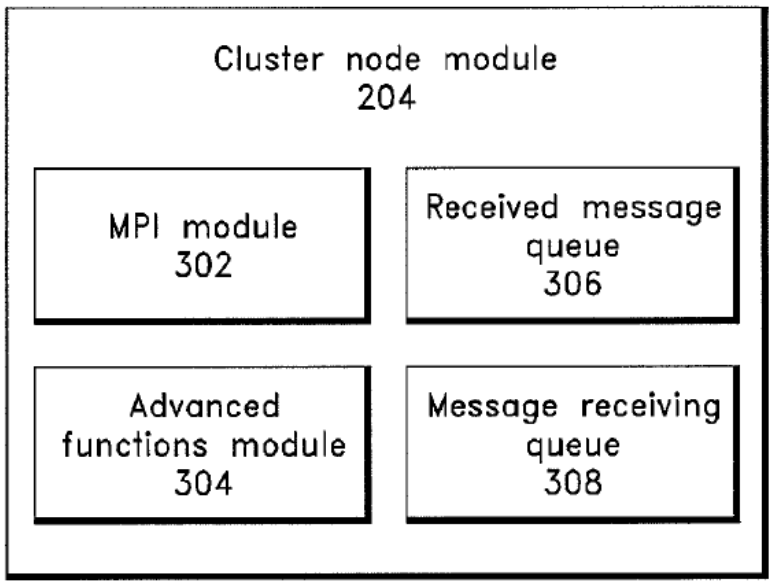
**63.** This limitation appears in each of independent claim 1 ("a mechanism for the nodes to communicate results of mathematical expression evaluation with each other using a peer-to-peer architecture"), claim 26 ("a mechanism for the nodes to communicate results of mathematical expression evaluation with each

other using asynchronous calls"), claim 29 ("a mechanism for the nodes to communicate results of mathematical expression evaluation with each other"), and claim 35 ("a mechanism to communicate results of evaluation with other computer cluster nodes using a peer-to-peer architecture").

**64.** The claims do not suggest anything about the specific structure of the term "mechanism," but recite that the "mechanism" performs a function (e.g., communicate results of mathematical expression evaluation or communicate results of evaluation). The specification twice mentions "mechanism" in the context of inter-node communications when discussing i) "MPI module 302" and ii) cluster node modules of nodes employing a mechanism for exchanging messages "on a pair-wise or collective basis" in a "process [that] provides the peer-to-peer behavior of the cluster node modules 204 a-e." Ex.1001, 14:41-43, 25:22-47, Claim 4 (indicating each node comprises one or more cluster node modules). These mentions of the term "mechanism" are with reference to MPI calls, as discussed below; however, the specification does not provide a definite or specific meaning for the mechanism's structure.

**65.** The '768 patent explains that "the cluster node modules 204 a-e provide a way for many kernel modules 206 … to communicate with one another. A cluster node module 204 can include at least a portion of an application programming interface ("API") known as the Message-Passing Interface ('MPI')."

Ex.1001, 11:41-48. Exchanging messages on a "pair-wise basis," i.e., "sending expressions from one node to another," is enabled by basic MPI calls. Ex.1001, 13:17-18. MPI module 302 includes "basic MPI calls such as, for example, relatively low-level routines that map MPI calls." "In some embodiments, basic MPI calls include calls that send data, equations, formulas, and other/or other expressions." Ex.1001, 13:9-16, Table B. "[M]essages [] being sent and received" (while "other work" is being done) is enabled by asynchronous MPI calls mpiISend[ ] and mpiIRecv[ ]. Ex.1001, 27:38-40, 13:41-14:16 (indicating mpiISend[ ] and mpiIRecv[ ] are asynchronous MPI calls). Figure 3 of the '768 patent shows "a cluster node module 204 implementing MPI calls and advanced MPI functions" and including "MPI module 302, advanced functions module 304, received message queue 306, and message receiving queue 308." Ex.1001, 12:41-46.




**Ex.1005, FIG. 3**

**66.** Neither the received message queue 306 nor the message receiving queue 308 includes anything that may be understood as "calls that send data, equations, formulas, and other/or other expressions." Ex.1001, 13:9-16, Table B. "The received message queue 306 includes a data structure for storing messages received from other cluster node modules," while "[t]he message receiving queue 308 includes a data structure for storing information about the location to which an expression is expected to be sent and the processor from which the expression is expected." Ex.1001, 21:64-22:13, 22:18-24. The data structures are "queue[s] and/or [other] type of data structure[s] such as, for example, a stack, a linked list, an array, a tree, etc." Ex.1001, 22:13-15, 22-24.

**67.** The advanced functions module 304 also does not include MPI or anything that may be understood as calls that send data, equations, formulas, and other/or other expressions." Ex.1001, 13:9-16, Table B. Instead, the advanced functions module 304 includes "a custom set of directives or functions" or "program code that provides a toolkit of functions inconvenient or impractical to do with MPI instructions and calls implemented by the MPI module 302." Ex.1001, 16:48-50, 43-45. "The advanced functions module 304 relies "on [MPI] calls and instructions implemented by the MPI module 302 in the implementation of advanced functions." Ex.1001, 16:46-48, 16:62 (explaining that the advanced functions module 304 functions or calls are "[b]uilt on the MPI calls"). Examples of the advanced functions module 304 calls include "functions providing for basic parallelization such as, for example, routines that would perform the same operations on many data elements or inputs, stored on many nodes," "functions providing for linear algebra operations such as … matrix and vector multiplication," "program code for implementing large-scale parallel fast Fourier transforms ('FFTs')," "parallel disk input and output calls," and "[a]utomatic [l]oad [b]alancing." Ex.1001, 17:2-6, 18:36-40, 20:3-4, 20:29, 21:12.

**68.** The "basic MPI calls include calls that send data, equations, formulas, and/or other expressions" and the '768 patent explains that "[s]imply sending expressions from one node to another is possible with these most basic MPI calls."

Ex.1001, 13:15-18. In other words, "the MPI module 302 can include basic MPI calls" that allow nodes to communicate mathematical expressions such as "equations, formulas, and/or other expressions" with each other, where "[o]ne node can call to send an expression while the other calls a corresponding routine to receive the sent expression." Ex.1001, 13:10-11, 16, 18-20. Examples of such calls include mpiSend[expr, target, comm, tag] call that "[s]ends an expression expr to a node with the ID target" and mpiRecv[expr, target, comm, tag] call that "[r]eceives an expression into expr from a node with the ID target." Ex.1001, 13:28-34 (Table B).

**69.** Asynchronous MPI calls, the other types of calls implemented by the MPI Module 302, allow kernels "to do work while communications are proceeding simultaneously." Ex.1001, 13:42-43. Examples of asynchronous MPI calls include mpiISend[expr, target, comm, tag, req] that "[s]ends an expression expr to a processor with the ID target" and mpiIRecv[expr, target, comm, tag, req] that "[r]eceives an expression expr from a proessor with the ID target." Ex.1001, 13:51-56 (Table C). "The mpiISend[ ] command … creates a packet containing the Mathematica expression to be sent as payload and where the expression should be sent" while the "mpiIRecv[ ] command … creates a packet specifying where it expects to receive an expression and from which processor this expression is expected." Ex.1001, 14:1-4, 9-12. The asynchronous MPI calls allow nodes to

communicate expressions such as Mathematica expressions asynchronously with each other so that the kernels of the nodes can continue performing their tasks while inter-node communications continue. Ex.1001, 13:41-14:16.

**70.** Communications between nodes using basic and asynchronous MPI calls are peer-to-peer communications, as evidenced by the '768 patent's description of both these calls as "[p]eer-to-[p]eer MPI" calls. Ex.1001, 26:65-27:49. The '768 patent provides an example illustration where two nodes use peer-to-peer MPI calls to communicate mathematical expressions with each other. Ex.1001, 27, 10-49.

**71.** For example, the basic MPI calls "mpiSend[N[Pi,22],targetProc, mpiCommWorld,d]" and "mpiRecv[a,targetProc,mpiCommWorld,d]" cause "the odd processor [to] send[] 22 digits of Pi, while the even [paired with the odd processor] receives that message." Ex.1001, 27, 11-25. The same applies to the asynchronous MPI calls mpiISend and mpiIRecv, except that the communications exhibit "asynchronous behavior, making it possible to do other work while messages are being sent and received, or if the other processor is busy." Ex.1001, 27:38-48.

**72.** Because the '768 patent describes the MPI module 302 as including or implementing MPI calls that enable nodes to communicate mathematical expressions with each other in a peer-to-peer and/or asynchronous fashion, as

discussed above, a POSITA would have understood the specification of the '768 patent as providing the following structure and functions for "mechanism" in claims 1, 26, 29, and 35:

**Structure**:

- Claims 1, 26, 29, and 35: MPI module 302.

**Function**:

- Claim 1: communicate results of mathematical expression evaluation with each other using a peer-to-peer architecture;
- Claim 26: communicate results of mathematical expression evaluation with each other using asynchronous calls;
- Claim 29: communicate results of mathematical expression evaluation with each other; and
- Claim 35: communicate results of evaluation with other computer cluster nodes using a peer-to-peer architecture.

If only the plain and ordinary meaning of "mechanism" is applied, my analysis below shows that the prior art meets that definition, as well.

## VIII. PRIOR ART

**73.** Below is a detailed discussion of the prior art references I considered as part of my analysis of '768 patent claims 1-39:

(1) "MultiMATLAB: Integrating MATLAB with High-Performance Parallel Computing" to Menon et al. 1997 ("Menon," Ex.1005);

(2) "MultiMATLAB: MATLAB on Multiple Processors" to Trefethen et al. 1996 ("Trefethen," Ex.1006);

(3) "RS/6000 SP: Planning Vol. 1, Hardware and Physical Environment," IBM 2001 ("RS6000," Ex.1007);

(4) "IBM Parallel Environment for AIX – Operation and Use, Volume 1, Using the Parallel Operating Environment," IBM 2001 ("POEref," Ex.1008); and

(5) "MPI: A Message-Passing Interface Standard," MPI Forum 1994 ("MPIref," Ex.1017).

## A. Menon

### 1. Summary of Menon

**74.** Like the '768 patent, Menon discloses a computing architecture of "distributed memory multiprocessors" that enables "high-performance parallel computing" by the "scientific computing environment" MATLAB. Ex.1005, Abstract, Title.

**75.** In more detail, Menon discloses MultiMATLAB, "a general extension of the MATLAB environment to any distributed memory multiprocessors." Ex.1005, Abstract. The MultiMATLAB architecture comprises a plurality of processors, each configured to run a MATLAB process and connected to the other processors via an interconnection network. Ex.1005, 3, FIG. 1. Each MATLAB process is "provided with the ability to communicate with other processes through a

communication layer that runs over the parallel platform's interconnection network." Ex.1005, 3. Examples of said communication layer include "MPI [Message Passing Interface], PVM [Parallel Virtual Machine], BLACS [Basic Linear Algebra Communication Subprograms], or any other package available on the platform," as well as versions of MPI such as MPICH and MPI-F. Ex.1005, 3, 5. FIG. 1 of Menon shown below depicts an illustration of an example MultiMATLAB architecture.

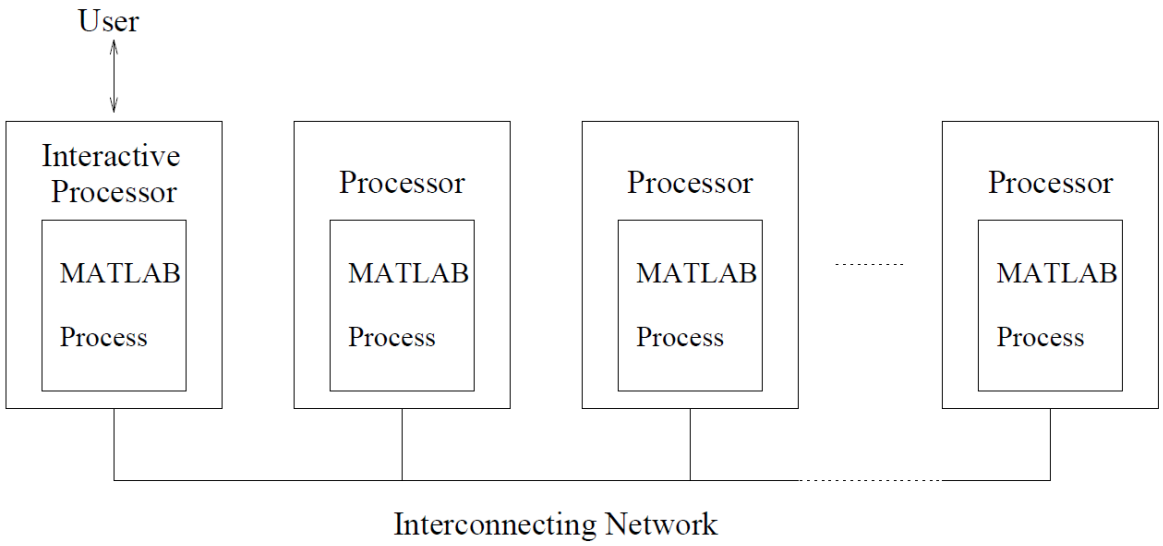



Figure 1: MultiMATLAB Architecture

**Ex.1005, FIG. 1.**

**76.** MultiMATLAB architecture is capable of operating on any parallel platform provided each processor of the platform is configured to run a MATLAB process. Ex.1005, 2. The parallel platform can be "any network of workstations supporting MATLAB," such as "a generic network of Unix workstations," or "the

IBM SP2, a modern high performance distributed memory multiprocessor." Ex.1005, 2, 5. Menon discloses example implementations of the MultiMATLAB architecture on these parallel platforms, where "MPICH, a popular public domain version of MPI" is used as the communication layer for the network of Unix workstations and MPI-F, "IBM's proprietary version of MPI specifically optimized for the SP2," is used as the communication layer for the IBM SP2 implementation. Ex.1005, 5.

**77.** Focusing on the IBM SP2 implementation, "all MATLAB processes are started via POE, IBM's Parallel Operating Environment. For each of *p* MATLAB processes, POE assigns an identification number between 0 and *p*-1. The process numbered 0 is considered the interactive MATLAB. All other processes wait for commands from the interactive MATLAB process." Ex.1005, 6. A user utilizing the MultiMATLAB architecture "interacts directly with one MATLAB process, called the interactive process, and operates within that process's MATLAB environment." Ex.1005, 3. The "other processes are used either by explicitly sending MATLAB commands to them or by executing routines that, in turn, use them." Ex.1005, 3.

**78.** FIG. 2 of Menon shown below depicts the address space of a MATLAB process of a MultiMATLAB architecture. Ex.1005, 4. An "address space" of a process is "a list of memory locations from some minimum (usually 0)

to some maximum, which the process can read and write. The address space contains the executable program, the program's data, and its stack." Ex.1018, 67-68. The MATLAB address space includes MATLAB, MEX Routines which "are executables originally written in C or Fortran that may be run directly from MATLAB," a MultiMATLAB interface module that "is responsible for initializing an underlying communication layer … and exposing it to the rest of the system," the communication layer, and an indirection table via which the MultiMATLAB interface module "provides MEX Routines access to the underlying communication layer." Ex.1005, 3-5.

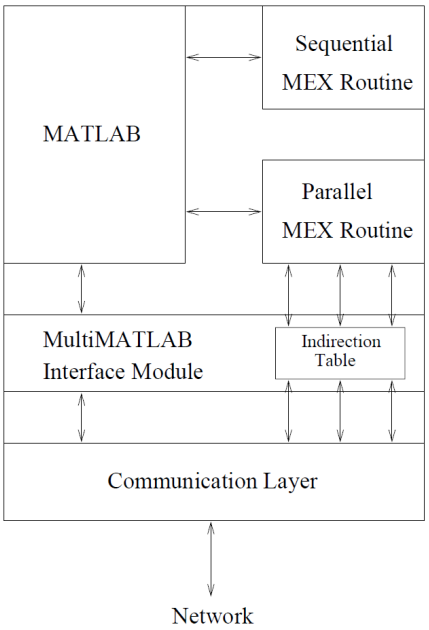



Figure 2: MATLAB Process Address Space

**Ex.1005, FIG. 2**

**79.** Upon initialization of the MultiMATLAB architecture, for example by using POE on the IBM SP2 as discussed above, the MultiMATLAB interface

module builds the indirection table, which is "a table of pointers to all functions and data in the underlying communication layer that need to be exposed to parallel MEX Routines." Ex.1005, 5. A loaded and executing parallel MEX Routine then accesses the MultiMATLAB interface module to obtain the indirection table, which enables it to have "the capability to access any function provided by the underlying communication layer through its corresponding offset in the table." Ex.1005, 5. This way, "[a]ll parallel MEX Routines access the underlying communication layer and, hence, the network, through the MultiMATLAB interface module." Ex.1005, 4.

**80.** The MEX Routines are responsible for all the parallel functionalities of the MultiMATLAB architecture. Ex.1005, 3. For example, Eval routine, which allows users to perform parallel evaluation on the MultiMATLAB architecture, is implemented on the interactive process "as a MEX Routine which accepts a vector of MATLAB process IDs and a MATLAB command" (shown in Table 1 below). Ex.1005, 4, Table 1. "On non-interactive processes, a separate top-level MEX Routine is immediately run upon initialization of the system. This MEX Routine runs in a loop in which it waits for MATLAB commands to arrive from the interactive process and executes them in its own MATLAB environment." Ex.1005, 4.

**81.** Menon explains parallel programming paradigms in MATLAB language used in cluster computing. Ex.1005, 7-11. The first paradigm is the "master/slave" environment, where the "interactive MATLAB process is the master and other MATLAB processes are slaves." Ex.1005, 8-9. The second paradigm is "SPMD/Message passing" that provides peer-to-peer MATLAB process communication, namely "point to point and collective communication MATLAB routines." Ex.1005, 9-10.

**82.** Menon explains that "[p]arallel MEX Routines may be run in a SPMD [single program multiple data] fashion on any subset of MATLAB processes via the Eval command." Ex.1005, 6. SPMD refers to a "style of parallel programming, where all processors use the same program, though each has its own data." Ex.1009, 1. However, "[i]n order to accommodate applications requiring a finer degree of communication," i.e., "finer grain communication of MATLAB data structures between any processes," Menon discloses "a set of MATLAB message passing routines" that "provide point to point and collective communication MATLAB routines operating directly on MATLAB data structures." Ex.1005, 9, 8. Examples of such MATLAB message passing routines are shown in Table 1 of Menon below depicting several MultiMATLAB commands.

| MultiMATLAB commands | |
|---|---|
| Starting and stopping MultiMATLAB (MPICH only) | |
| Start(n) | Initializes n remote MATLAB processes. |
| Quit | Closes remote MATLAB processes. |
| Running commands on multiple processors | |
| Nproc | Returns number of MATLAB processes. |
| ID | Returns ID of calling MATLAB process. |
| Eval(pid,'command') | Evaluates a command on one or more MATLAB process. |
| Master/Slave Communication | |
| Put(pid,dataname) | Put data from the master process onto one or more remote processes. |
| Get(pid,dataname) | Get data from a remote process onto the master process. |
| SPMD Communication | |
| Send(pid,data) | Send data from one process to another. |
| Recv(pid) | Receive data sent from another process. |
| Barrier | Synchronize processes. |
| Bcast(pid,data) | Broadcasts data from processor pid to all processes. |
| Sum(data) | Adds data across all processors to form a global sum. |
| Collect(data) | Collects local data segments into a single global one. |

Table 1: Selected commands in the MultiMATLAB system

Commands for point-to-point communication (including Send and Recv)

**Ex.1005, Table 1 (annotated)**

**83.** The MATLAB data structures can be data structures such as "matrices, matrix sections," "sparse matrices, cell arrays, and structures." Ex.1005, 9. Menon discloses an example use of these structures when discussing "MultiMATLAB implementations of the conjugate gradients algorithm on the IBM SP2." Ex.1005, 11. In this implementation the MATLAB processes of the MultiMATLAB architecture evaluate mathematical expressions such as matrix equations, since "[t]he conjugate gradients algorithm is iterative method to solve a linear system of the form $Ax = b$, where $A$ is a symmetric positive definite matrix and $x$ and $b$ are

vectors. The computational core of this method is a single matrix-vector multiplication at each iteration." Ex.1005, 11.

### 2. Availability of Menon

**84.** Menon was originally presented on November 19, 1997 at "SC '97: International Conference for High Performance Computing, Networking, Storage and Analysis," held on November 15-21, 1997, in San Jose, CA. Ex.1026, 1, 2. It was published in "SC '97: Proceedings of the 1997 ACM/IEEE conference on Supercomputing," which was made available no later than the last day of the conference, November 21, 1997. Ex.1027, 2. It was also available on the conference website no later than July 7, 1998. Ex.1026, 6-26. It is currently available for public download on the IEEE digital library, IEEE Xplore, which the IEEE uses to publish and make available technical articles and standards as part of its ordinary course of business. Ex.1027, 1-2. It was published and made available on a Cornell University website (associated with one of the authors, Vijay Menon) no later than November 18, 2003. Ex.1023, 3-23. Menon was also listed (with a link) on a webpage of Vijay Menon no later than January 28, 2001. Ex.1023, 40-41. It has been cited 17 times in papers with online publication dates between the original publication of Menon and the earliest priority date of the '768 patent. Ex.1028, 4-9.

**85.** SC '97 conference, the 1997 edition of the annual SC [supercomputing] conference, was recognized for being "the annual conference for leaders in high-

performance networking and computing" and included topics from the field of computer cluster systems and cluster computing techniques, such as "Beowulf clusters," "[h]eterogeneous distributed parallel computing," "[c]heckpointing and recovery on distributed parallel systems," etc. Ex.1030, 3, Ex.1031, 1. "IEEE *Xplore* is the flagship digital platform for discovery and access to scientific and technical content published by the IEEE (Institute of Electrical and Electronics Engineers)." Ex.1032, 1. Cornell, at the relevant timeframe, was recognized as a pioneer in "the development of high-performance 'cluster' computers made by linking off-the-shelf shelf computers in parallel." Ex.1033, 2.

**86.** Accordingly, a POSITA interested in computer cluster systems and cluster computing techniques and exercising reasonable diligence would have located Menon from the SC '97 conference proceedings, the SC '97 conference website, IEEE Xplore, and/or the Cornell websites before the earliest priority date of the '768 patent, as evidenced by the afore-mentioned 17 citations to Menon in papers with online publication dates in between the original publication of Menon and the earliest priority date of the '768 patent. Ex.1028, 4-9.

### B. Trefethen

#### 1. Summary of Trefethen

**87.** Trefethen details an implementation of the MultiMATLAB architecture, discussed above in Section VIII.A.1. Trefethen discloses that a user

"connected to a node of the IBM SP2, running MATLAB," in a MultiMATLAB architecture that includes additional MATLAB processes, can execute on all the MATLAB processes using Eval, "[t]he standard MultiMATLAB command for executing commands on one or more processors." Ex.1006, 3. That is, "[e]ach command passed to Eval was executed on all MATLAB processes." Ex.1006, 4. The outputs from the MATLAB processes are "sent to the master process as soon as it is ready," i.e., without order or synchronicity between the MATLAB processes. Ex.1006, 4. In the context of SPMD point-to-point communication, a "master process" is a "pseudo-master [that] is arbitrarily chosen to be the task [that] all the peers know a priori who to send the results back to." Ex.1011, 64.

**88.** Trefethen also discloses peer-to-peer communication, in that "[m]ore general point-to-point communication is accomplished by send and receive commands, which can be executed on any of the MATLAB processes." Ex.1006, 5. For example, "SPMD programs can be built upon Send and Recv commands." Ex.1006, 6. Trefethen illustrates "point-to-point communication" in the MultiMATLAB architecture via an example script (m-file), "cycle.m," executed on a MultiMATLAB architecture of 6 processors each running a MATLAB process (to execute the example script (an m-file), a user can use the command Eval( 'filename' )). Ex.1006, 6:

*user instructions*

Instruction for MATLAB process ID=0 to create a=1 and send "a" to MATLAB process ID=1

```
if ID==0                    % first process: send
   a = 1
   Send(ID+1,a)
elseif ID == Nproc-1        % last process: receive and double
   a = 2*Recv
else                        % middle processes: receive, double, and send
   a = 2*Recv
   Send(ID+1,a)
end;
```

Instruction for the last MATLAB process to receive the variable a and double it

Process 0 creates the variable a with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of a, and sends it along to process 2; and so on. If there are six processors the command `Eval( 'cycle' )` produces the output

Instruction for a middle MATLAB process ID to receive the variable "a," double it, and send the doubled variable to the next MATLAB process ID=ID+1

```
a = 1
a = 2
a = 4
a = 8
a = 16
a = 32
```

Output of final result (a=32)

Ex.1006, 6 (annotated).

The sample code shows:

- node 0 sending data to node 1,
- node 1 performing a mathematical calculation and then sending that result to node 2,
- node 2 performing a mathematical calculation and then sending that result to the next node ("and so on"), and
- the result is output to the user. Ex.1006, 6.

**89.** I note that the m-file "cycle.m," when executed in a MultiMATLAB architecture comprising multiple networked processors communicating in a SPMD paradigm, results in the sequential evaluations of mathematical expressions by the

networked processors, where the final result is communicated to the node (i.e., the pseudo-master node) that distributed the call for the execution of the mathematical expressions. Ex.1006, 6. Although, the mathematical evaluation in cycle.m is an arithmetic operation (doubling a variable), a POSITA would understand that the MATLAB processes in MultiMATLAB architecture execute other types of mathematical operations such as matrix operations. Ex.1006, 4-5 (discussing the computation of "the condition numbers of the Hilbert matrices" or eigenvalues of "matrices of dimension 400.").

**2. Availability of Trefethen**

**90.** Trefethen was published and made available on a Cornell University website no later than May 10, 1996. Ex.1023, 24-39. It was listed (with a link) along with Menon (*See* Section VIII.A.2 above) on a webpage of one of the authors, Vijay Menon, no later than January 28, 2001. Ex.1023, 40. It is currently available for public download on the ACM digital library, which the ACM uses to publish and make available technical articles. Ex.1034, 1. Trefethen has been cited 10 times in papers with online publication dates between the original publication of Trefethen and the earliest priority date of the '768 patent, including by Menon. Ex.1035, 2-3.

**91.** As discussed in Section VIII.A.2 above, Cornell was recognized at the relevant timeframe as a pioneer in "the development of high-performance 'cluster'

computers made by linking off-the-shelf shelf computers in parallel." Ex.1033, 2. Further, "[t]he ACM Digital Library (DL) is the world's most comprehensive database of full-text articles and bibliographic literature covering computing and information technology." Ex.1034, 1. Accordingly, a POSITA interested in computer cluster systems and cluster computing techniques and exercising reasonable diligence would have located Trefethen from either the Cornell website and/or the ACM Digital Library, as evidenced by the afore-mentioned 10 citations to Trefethen in papers with online publication dates in between the original publication of Trefethen and the earliest priority date of the '768 patent. Ex.1035, 2-3. Furthermore, an interested POSITA that locates Menon as discussed above in Section VIII.A.2 would also have located Trefethen as well because both Menon and Trefethen were listed with links on a webpage of one of the authors no later than January 28, 2001. Ex.1023, 40-41.

### C. RS6000

#### 1. Summary of RS6000

**92.** RS6000 is part of an IBM installation guide for IBM RS/6000 SP, which is the same as the IBM SP2 on which the MultiMATLAB architecture discussed above in Sections VIII.A.1 and Sections VIII.B.1 is implemented. Ex.1010, 6 (indicating that the IBM RS/6000 SP is the same as IBM SP2 because "the SP2 was renamed to simply the SP"). The IBM RS/6000 SP System, which

"represents the high-end of the RS/6000 family," is a "scalable, parallel computing" system that includes processor nodes and "is scalable from one to 128 processor nodes." Ex.1010, 6; Ex.1007, 1-2. The processor nodes of IBM RS/6000 SP have internal as well as external hard disk drives. Ex.1007, 6, 10.

**2. Availability of RS6000**

**93.** RS6000 was published and made available via IBM's website no later than July 26, 2001. Ex.1029, 47-56. RS6000 was catalogued and indexed according to IBM's documentation library structure on its website no later than August 14, 2001, where the link chain "Products & Services" → "Servers" → "UNIX Servers" → "Learn More" (under "Large scale servers") → "RS/6000 SP product documentation" → "RS/6000 SP Planning" → "Browse HTML" under volume 1" → "Table of Contents" starting on http://www.ibm.com/us/, lands on a page that has links to the contents of a document named "RS/6000 SP: Planning Volume 1, Hardware and Physical Environment" with a document number of GA22-7280-12 available no later than June 17, 2001. Ex.1038, 1-11. RS6000 is a combination of the contents available under the links "Introducing the RS/6000 SP" and "375 MHz POWER3 SMP High Node (F/C 2058)" at the afore-mentioned landing page. Ex.1038, 12, 13. Also, a document with the same document number and a date of December 2002 was available on IBM's website no later than October 4, 2003 (at "http://www.ibm.com/us/" → "Products" → "Unix" → "Library" → "RS/6000 SP

hardware and software documentation" → "SP Planning"). Ex.1029, 63-72, 75-76. A document with the same control number and the same title is currently available on IBM's website (can be found by searching IBM's website using the document title or control number), with the control number included both in a link to the document and the footer of a copy of the document itself.

**94.** Accordingly, an interested POSITA exercising reasonable diligence would have located RS6000 from IBM's website.

### D. POEref

#### 1. Summary of POEref

**95.** POEref is an IBM product documentation for POE that "describes the IBM Parallel Environment (PE)" and "its Parallel Operating Environment (POE)" that is used to launch all the MATLAB processes on the MultiMATLAB architecture discussed above in Sections VIII.A.1 and Sections VIII.B.1. Ex.1008, ix; Ex.1005, 6.

**96.** Among other things, POEref discloses that one of "the steps involved in … executing [] parallel C, C++, or Fortran programs using either an IBM RS/6000 SP" is "load[ing] and execut[ing] an SPMD program onto all [remote] nodes of your partition" using the command **poe**. Ex.1008, 7, 28. When the command **poe** is invoked, "the Partition Manager allocates processor nodes for each task and initializes the local environment. It then loads your program, and

reproduces your local environment, on each processor node … If you are using the dynamic message passing interface, the appropriate communication subsystem library implementation (IP or US) is automatically loaded at this time." Ex.1008, 28. The IP and US communication subsystems are "two separate implementations of the communication subsystem library – the Internet Protocol (IP) Communication Subsystem and the User Space (US) Communication Subsystem." Ex.1008, 2.

### 2. Availability of POEref

**97.** POEref was published and made available for download via IBM's website no later than August 3, 2003. Ex.1039, 1-8. It was catalogued and indexed according to IBM's documentation library structure on its website no later than December 30, 2004, where the link chain "Products" → "Unix" → "Library" → "RS/6000 SP hardware and software documentation" → "Parallel Environment (PE)," starting on http://www.ibm.com/us/, lands on a page that has a link to a document named "Operations and Use, Volume 1" (under a column labelled "PE for AIX 5L Version 3 Release 2") with a document number of SA22-7425-01 and a date of December 2001. Ex.1039, 1-8. A document with the same control number and the same title is currently available on IBM's website (can be found by searching IBM's website using the document title or control number), with the

control number included both in a link to the document and the footer of a copy of the document itself.

**98.** It is also my understanding that the IBM SP2 parallel computing platform was distributed with manuals, including the "IBM AIX Parallel Environment – Operations and Use" manuals. Ex.1036, 14; Ex.1008, 157-159 (stating "PE documentation is shipped with the PE licensed program in a variety of formats" and indicating "IBM Parallel Environment for AIX: Operation and Use, Volume 1, SA22-7425" as one of the PE publications). The "IBM AIX Parallel Environment – Operations and Use" provided users web addresses where users "can find most of the IBM product information for RS/6000 SP products on the World Wide Web" with "[f]ormats for both viewing and downloading." Ex.1008, 157. Specifically, it provided the link http://www.ibm.com/servers/eserver/pseries, which takes one to the "Library" link in the link chain discussed above. Ex.1008, 157; Ex.1039, 3.

**99.** Accordingly, an interested POSITA exercising reasonable diligence would have located POEref from IBM's website and/or would have obtained it from IBM.

## E. MPIref

### 1. Summary of MPIref

**100.** MPIref is a documentation from the Message Passing Interface Forum "contain[ing] all the technical features proposed for the [message passing] interface." Ex.1017, ii. MPI is used as a communication layer allowing all the MATLAB processes on the MultiMATLAB architecture to communicate with each other as discussed above in Section VIII.A.1. Ex.1005, 3.

**101.** MPIref discloses that MPI includes "[p]oint-to-point communication," and that the "basic point-to-point communication operations are **send** and **receive**." Ex.1017, 3, 15. The **send** operation MPI_SEND "specifies a **send buffer** in the sender memory from which the message data is taken." Ex.1017, 15. The send buffer "consists of **count** successive entries of the type indicated by **datatype**, starting with the entry at address **buf**." Ex.1017, 16 (emphasis original). MPIref also discloses that:

> In addition to the data part, messages carry information that can be used to distinguish messages and selectively receive them. This information consists of a fixed number of fields, which we collectively call the **message envelope**. These fields are
>
> source

> destination
>
> tag
>
> communicator
>
> …
>
> The message destination is specified by the **dest** argument.

Ex.1017, 18 (emphasis original). The syntax of the **send** operation MPI_SEND is shown below:

Send buffer information  Message envelope

The syntax of the blocking send operation is given below.

MPI_SEND(buf, count, datatype, dest, tag, comm)

| | | |
|---|---|---|
| IN | buf | initial address of send buffer (choice) |
| IN | count | number of elements in send buffer (nonnegative integer) |
| IN | datatype | datatype of each send buffer element (handle) |
| IN | dest | rank of destination (integer) |
| IN | tag | message tag (integer) |
| IN | comm | communicator (handle) |

Ex.1017, 16 (annotated).

**102.** MPIref discloses that processes receive a message "with the **receive** operation MPI_RECV," where "[t]he message to be received is selected according to the value of its [message envelope], and the message data is stored into the **receive buffer**." Ex.1017, 16 (emphasis original). "The receive buffer consists of

the storage containing **count** consecutive elements of the type specified by **datatype**, starting at address **buf**." Ex.1017, 16 (emphasis original). The syntax of the **receive** operation MPI_RECV is shown below:

Recv buffer information  Message envelope

The syntax of the blocking receive operation is given below.

MPI_RECV (buf, count, datatype, source, tag, comm, status)

| | | |
|---|---|---|
| OUT | buf | initial address of receive buffer (choice) |
| IN | count | number of elements in receive buffer (integer) |
| IN | datatype | datatype of each receive buffer element (handle) |
| IN | source | rank of source (integer) |
| IN | tag | message tag (integer) |
| IN | comm | communicator (handle) |
| OUT | status | status object (Status) |

Ex.1017, 19 (annotated).

### 2. Availability of MPIref

**103.** MPIref was published by MPI Forum, a group of organizations that defined MPI, on May 5, 1994. Ex.1017, cover page. At the time of its publication, it was made available and distributed, including via anonymous FTP mail servers, by the University of Tennessee and Oak Ridge National Laboratory. Ex.1017, vi. It was common practice at the relevant timeframe for MPI documentations to be distributed to interested parties as they became available and my research team at the University of Texas at Austin would have received a copy as well. MPIref was

available on the MPI Forum's website no later than July 3, 1998. Ex.1029, 1-3, 80-316. MPIref has been cited 74 times in papers with publication dates between the original publication of MPIref and the earliest priority date of the '768 patent. Ex.1037, 2-16.

**104.** Accordingly, an interested POSITA exercising reasonable diligence would have located MPIref at the MPI Forum website and/or would have obtained it from the University of Tennessee and/or Oak Ridge National Laboratory, as evidenced by the afore-mentioned 74 citations to MPIref in papers with online publication dates in between the original publication of Menon and the earliest priority date of the '768 patent. Ex.1037, 2-16.

## IX. DETAILED UNPATENTABILITY ANALYSIS

**105.** I have been asked to provide my opinion as to whether the Challenged Claims of the '768 patent would have been obvious to a POSITA as of the relevant timeframe. The discussion below provides a detailed analysis of how the prior art references I reviewed teach the elements of the Challenged Claims of the '768 patent.

**106.** Throughout my analysis, I applied the legal principles as set out in Section IV and evaluated the prior art teachings from the perspective of a POSITA as of the relevant timeframe. Specifically, I have considered the scope and content of the prior art and any potential differences between the claimed subject matter and

the prior art as of the relevant timeframe. For the ’768 patent, the relevant timeframe is the earliest claimed priority date of June 13, 2006. It is my opinion, following this analysis, that the claimed invention, as a whole, would have been obvious. I note that my analysis and proposed prior art combinations rely on the teachings of the references and not on physical incorporation of the elements. Additionally, as part of my analysis, I have reviewed and cite to other prior art references as demonstrating knowledge of a POSITA prior to the earliest claimed priority date.

### A. Ground 1: Claims 1-26, 29-30, 35, 36, and 39 are obvious over Menon in view of Trefethen, RS6000, and POEref

**107.** The combination of Menon, Trefethen, RS6000, and POEref renders obvious claims 1-26, 29-30, 35, 36, and 39 as discussed below.

#### 1. Combining Trefethen with Menon

##### a. ***Menon and Trefethen Are Analogous Art***

**108.** Menon and Trefethen are each analogous art at least because Menon and Trefethen are each directed to the same field of endeavor as the ’768 patent. The ’768 patent is directed to “the field of cluster computing.” Ex.1001, 1:14-15. Likewise, Menon is directed to the MultiMATLAB system, “a step towards an effective parallel computing environment.” Ex.1005, 15. Similarly, Trefethen is directed to a system “that enables one to run MATLAB conveniently on multiple processors.” Ex.1006, Abstract.

**109.** Furthermore, Menon and Trefethen are each analogous art to the ’768 patent because each is reasonably pertinent to a problem the inventors of the ’768 patent attempted to solve. The ’768 patent addresses the problem of “adding cluster computing functionality to a computer program” such as Mathematica, MATLAB, etc. Ex.1001, 1:15-17, 4:15-19. As an example, the ’768 patent describes an embodiment that “adapts a software module designed to run on a single node, such as, for example, the Mathematica kernel, to support cluster computing.” Ex.1001, 2:18-21. The ’768 patent describes using “Message Passing Interface (‘MPI’) calls directly from within a user interface, such as, for example, the Mathematica programming environment.” Ex.1001, 2:24-27.

**110.** Likewise, Menon’s “MultiMATLAB[] is a general extension of the MATLAB environment to any distributed memory multiprocessors.” Ex.1005, Abstract. Each MATLAB process of the MultiMATLAB architecture is “provided with the ability to communicate with other processes through a communication layer that runs over the parallel platform’s interconnection network.” Ex.1005, 3. Examples of said communication layer include “MPI [Message Passing Interface].” Ex.1005, 3, 5.

**111.** Similarly, Trefethen attempts to enable users “to run MATLAB conveniently on multiple processors.” Ex.1006, Abstract. “Using short, MATLAB-style commands like Eval, Send, Recv, Bcast, Min, and Sum, the user operating

within one MATLAB session can start MATLAB processes on other machines and then pass commands and data between [] these various processes in a fashion that maintains MATLAB's traditional user-friendliness." Ex.1006, Abstract.

**112.** Accordingly, in my opinion, a POSITA would understand that both Menon and Trefethen are prior art that are analogous to the claimed subject matter of the '768 patent.

**b. *Motivation to Combine Menon with Trefethen***

**113.** A POSITA would have considered and combined Trefethen with Menon because it is a combination of prior art elements (Menon describing example implementations of the MultiMATLAB architecture on a "high performance distributed memory multiprocessor" and Trefethen describing an implementation of the MultiMATLAB architecture and an "illustrat[ion of] how the [MultiMATLAB] system is used" with concrete examples of codes executed on a MultiMATLAB system to perform mathematical calculations) according to known methods (executing MultiMATLAB codes on a MultiMATLAB system, as taught by Trefethen) to yield the predictable result discussed in both Menon and Trefethen of parallel execution of MATLAB codes on a high performance distributed memory multiprocessor to perform mathematical calculations. Ex.1005, Abstract; Ex.1006, 3-9.

**114.** Additionally, Menon suggests the combination by expressly citing to Trefethen and explaining that Menon builds on the teaching of Trefethen. Ex.1005, 1-2. Trefethen discloses a MultiMATLAB architecture where "MATLAB processes [are run] on multiple processors, with full access to all the usual capabilities such as Toolboxes. These processes communicate via simple MATLAB-style commands built on MPI" implemented using MPICH, and based on "SPMD paradigm[]." Ex.1006, 12, 9. Menon points to Trefethen's MultiMATLAB architecture and further discloses a "MultiMATLAB architecture [that] enables MATLAB applications to take advantage of high performance distributed memory multiprocessors" "based upon the MPI communication standard" implemented using MPICH and MPI-F. Ex.1005, 2, 16, Abstract, 5.

**115.** Furthermore, a POSITA seeking to understand and develop MultiMATLAB's capabilities would have considered the teachings of Menon and Trefethen. During the relevant timeframe, researchers commonly made their findings available on websites, particularly the websites associated with their research institution, here Cornell University. A POSITA working with the MultiMATLAB system would have consulted the Cornell website dedicated to MultiMATLAB, which contained links to both papers. Ex.1023, 1-39. Co-author Vijay Menon likewise included links to both papers within his web page on the Cornell website. Ex.1023, 40-41. Furthermore, the authors "Vijay Menon" and

"Anne E. Trefethen" are common across both papers. *See* Ex.1005, 1 and Ex.1006, 1. The authors Menon and Trefethen were both at the same university, Cornell University, working on the same research team to develop the MultiMATLAB system. Ex.1005, 1 and Ex.1006, 1. Accordingly, any POSITA seeking to understand MultiMATLAB would have been presented with both papers to consider.

**116.** The results of the combination of Menon and Trefethen would have been predictable (using the techniques of executing MultiMATLAB codes, as taught by Trefethen, on the implementations of the MultiMATLAB architecture, as taught by Menon) and there would have been a reasonable expectation of success at least because Menon expressly builds on the teaching of Trefethen with respect to the MultiMATLAB architecture. Ex.1005, 2. In addition, during the relevant timeframe, researchers had a strong desire to parallelize MATLAB, as exemplified by Trefethen's statement that "[m]any people must have thought about parallelizing MATLAB over the years." Ex.1006, 11. This desire had given rise to multiple efforts that were expected to succeed at the stated goal of parallelizing MATLAB. Ex.1006, 12. Trefethen reflected this sentiment when stating that "[w]e are aware of seven projects than have been undertaken elsewhere that share some of the goals and capabilities of MultiMATLAB" and that "we believe that the authors of all of these systems join us in expecting that it is inevitable that the MATLAB world will

soon take the step from single to multiple processors." Ex.1006, 11, 12. The combination of Menon and Trefethen succeeded in achieving a MultiMATLAB system that provides parallel execution of MATLAB on a high-performance distributed memory multiprocessor to perform mathematical calculations. Ex.1005, 2 ("In this paper, we … achieve … MATLAB code with high-performance parallel routines"), because MultiMATLAB is "a general extension of the MATLAB environment to distributed memory multiprocessors" that is "easily portable onto any parallel platform that allows MATLAB to run on each processor" and has "high-performance parallel routines." Ex.1005, 2.

### 2. Combining RS6000 with Menon

#### a. ***RS6000 is Analogous Art***

**117.** RS6000 is analogous art at least because it is directed to the same field of endeavor as the '768 patent. The '768 patent is directed to "the field of cluster computing." Ex.1001, 1:14-15. Similarly, RS6000 is directed to "provid[ing] a state-of-the-art parallel computing system." Ex.1007, 1.

**118.** Furthermore, RS6000 is analogous art to the '768 patent because RS6000 is reasonably pertinent to a problem the inventors of the '768 patent attempted to solve. The '768 patent purports to provide a solution "for adding cluster computing functionality to a computer program." Ex.1001, 1:14-16. The '768 patent describes "[c]omputer clusters includ[ing] … processor cores ('nodes')

that intercommunicate so that the nodes can accomplish a task as though they were a single computer." The '768 patent describes an embodiment that "allows a user to create applications, using a high-level language such as Mathematica, that are able to run on a computer cluster having supercomputer-like performance." Ex.1001, 2:10-13.

**119.** Likewise, RS6000 discloses the IBM RS/6000 SP, a "family of scalable, parallel computing solutions" that "provides … industry-leading application enablers and applications." Ex.1007, 1. RS6000 describes a system that "execute[s] both serial and parallel applications simultaneously, while managing your system from a single workstation." Ex.1007, 1.

**120.** Accordingly, in my opinion, a POSITA would understand that RS6000 is a prior art that is analogous to the '768 patent.

**a. *Motivation to Combine RS6000 with Menon***

**121.** A POSITA would have considered and combined the disclosures of RS6000 with Menon because Menon expressly suggests doing so. Specifically, Menon teaches a MultiMATLAB architecture implemented in "IBM SP2, a modern high performance distributed memory multiprocessor," and RS6000, which is part of an installation guide for the IBM RS/6000 SP, provides "an overview of the IBM RS/6000 SP."[1] Ex.1005, 5; Ex.1007, 1. For example, RS6000 discloses that "[t]he

[1] As discussed in ¶92, IBM SP2 is the same as IBM RS/6000 SP.

basic components of the RS/6000 SP system are: [p]rocessor nodes [and] … [e]xternal disk drives," where the nodes are "scalable from one to 128 processor nodes" and the "[h]ard disk drives for the SP system can be either of two types as follows: [i]nternal … [or] [e]xternal." Ex.1007, 2, 6.

**122.** A POSITA seeking to understand MultiMATLAB would be motivated to seek the documentation corresponding to the systems expressly discussed in Menon, namely, the IBM SP2 hardware for which "[t]he MPI-F implementation of MultiMATLAB was design[ed] specifically." Ex.1005, 6. IBM provided extensive documentation of its products on its web site and a POSITA would have been motivated to seek out the documentation in order to more fully understand MultiMATLAB's operation. *See* Section VIII.C.2.

**123.** A POSITA would also have considered and combined RS6000 with Menon because it is a combination of prior art elements (Menon describing an implementation of the MultiMATLAB architecture in "IBM SP2, a modern high performance distributed memory multiprocessor" and RS6000 providing "an overview of the IBM RS/6000 SP … introduc[ing] the processor nodes" and memory requirements) according to known methods (as taught by Menon of implementing the MultiMATLAB architecture in IBM SP2) to yield the predictable result of an implementation of the MultiMATLAB architecture in an IBM SP2

having the processor nodes and memory devices disclosed by RS6000. Ex.1005, Abstract; Ex.1007, 1, 12-13.

**124.** Combining RS6000 with Menon (and Trefethen) would have been predictable and would have had a reasonable expectation of success at least because Menon expressly discloses implementing its MultiMATLAB architecture in IBM SP2 (i.e., IBM RS/6000 SP), which is the subject of RS6000, explaining that "[t]he MPI-F implementation of MultiMATLAB was design[ed] specifically for the IBM SP2." Ex.1005, 5. Specifically, Menon describes "a MultiMATLAB system" "in which parallel libraries or applications written in MPI can easily be integrated into MATLAB on the [IBM] SP2." Ex.1005, 6. In other words, Menon expressly describes performing the proposed combination, with "results [that] demonstrate that the MultiMATLAB system can be an effective platform for parallel computing." Ex.1005, 14.

### 3. Combining POEref with Menon

#### a. ***POEref is Analogous Art***

**125.** POEref is analogous art at least because it is directed to the same field of endeavor as the '768 patent. The '768 patent is directed to "the field of cluster computing." Ex.1001, 1:14-15. Similarly, POEref is directed to "compil[ing], execut[ing], and analyz[ing] parallel programs." Ex.1008, ix.

**126.** Furthermore, POEref is analogous art to the ’768 patent because it is reasonably pertinent to a problem the inventors of the ’768 patent attempted to solve. The ’768 patent purports to provide a solution “for adding cluster computing functionality to a computer program.” Ex.1001, 1:14-16. The ’768 patent describes an embodiment that “allows a user to create applications, using a high-level language such as Mathematica, that are able to run on a computer cluster having supercomputer-like performance.” Ex.1001, 2:10-13. The ’768 patent describes enabling “computer programs [that] can run on only a single node because, for example, they are coded to perform tasks serially” to “benefit from a plurality of nodes in a cluster.” Ex.1001, 1:14-17, 26-28, 2:9-10.

**127.** Likewise, POEref discloses the parallel operating environment (POE), which is “a simple and friendly environment designed to ease the transition from serial to parallel application development and execution,” and “is designed to run on an … RS/6000 network cluster.” Ex.1008, 7, ix.

**128.** Accordingly, in my opinion, a POSITA would understand that POEref is a prior art that is analogous to the ’768 patent.

**b. *Motivation to Combine POEref with Menon***

**129.** A POSITA would have considered and combined the teachings of POEref with Menon because Menon suggests the combination via its teaching that “POE, IBM’s Parallel Operating Environment” is used to start and assign

identification numbers to all the MATLAB processes when implementing the MultiMATLAB architecture in IBM SP2, and POEref is directed to "describ[ing] … [the] Parallel Operating Environment (POE)" Ex.1005, 6; Ex.1008, ix. For example, POEref discloses the use of the command **poe** "to load and execute programs on remote nodes," which includes "the Partition Manager allocat[ing] processor nodes for each task and initializ[ing] the local environment. It then loads your program, and reproduces your local environment, on each processor node." Ex.1008, 28.

**130.** A POSITA seeking to understand MultiMATLAB would be motivated to seek the documentation corresponding to the systems expressly discussed in Menon, namely, the POE system expressly discussed in Menon. Ex.1005, 6. IBM provided extensive documentation of its products on its web site and a POSITA would have been motivated to seek out the documentation to more fully understand MultiMATLAB's operation. *See* Section VIII.D.2.

**131.** A POSITA would also have considered and combined POEref with Menon because it is a combination of prior art elements (Menon describing an implementation of the MultiMATLAB architecture where POE is used to start and assign identification numbers to all the MATLAB processes and POEref "describ[ing] … [the] Parallel Operating Environment (POE)" that uses the command **poe** "to load and execute programs on remote nodes") according to

known methods (as taught by Menon of using POE in implementing the MultiMATLAB architecture) to yield the predictable result of an implementation of the MultiMATLAB architecture utilizing a POE with the command **poe** as disclosed by POEref. Ex.1005, 6; Ex.1008, ix, 28.

**132.** The results of the combination would have been predictable and there would have been a reasonable expectation of success at least because Menon expressly discusses using POE, which is the subject of POEref, to implement its MultiMATLAB architecture in IBM SP2, with "results [that] demonstrate that the MultiMATLAB system can be an effective platform for parallel computing." Ex.1005, 6; Ex.1008, ix; Ex.1005, 14.

**4. Claim 1**

**a. [1.0]** ***A computer cluster comprising:***

**133.** Menon in combination with RS6000 renders obvious the preamble. The '768 patent states that "computer clusters include a group of two or more computers, microprocessors, and/or processor cores ('nodes') that intercommunicate so that the nodes can accomplish a task as though they were a single computer." Ex.1001, 1:19-22. Likewise, Menon discloses a MultiMATLAB cluster with a series of MATLAB processor nodes, implemented on an IBM SP2 that uses multiple processor nodes, switches, network connectivity, and disk drives, for executing software, to work in parallel to calculate mathematical formulas, and

RS6000 describes the IBM SP2 as a networked cluster computing system for executing software in parallel, rendering obvious *a computer cluster*.

**134.** Menon discloses a "MultiMATLAB architecture" designed to provide "high-performance parallel routines" and "performance gains." Ex.1005, 2. The MultiMATLAB architecture comprises a series of processors that each run a MATLAB process and communicate with each other:

> In the MultiMATLAB architecture, illustrated in Figure 1, each processor in a parallel platform individually runs a MATLAB process. Each process is then provided with the ability to communicate with other processes through a communication layer that runs over the parallel platform's interconnection network. The user interacts directly with one MATLAB process, called the interactive process, and operates within that process's MATLAB environment. Other MATLAB processes await commands from the interactive process. These other processes are used either by explicitly sending MATLAB commands to them or by executing routines that, in turn, use them.

Ex.1005, 3. FIG. 1 is illustrated below:

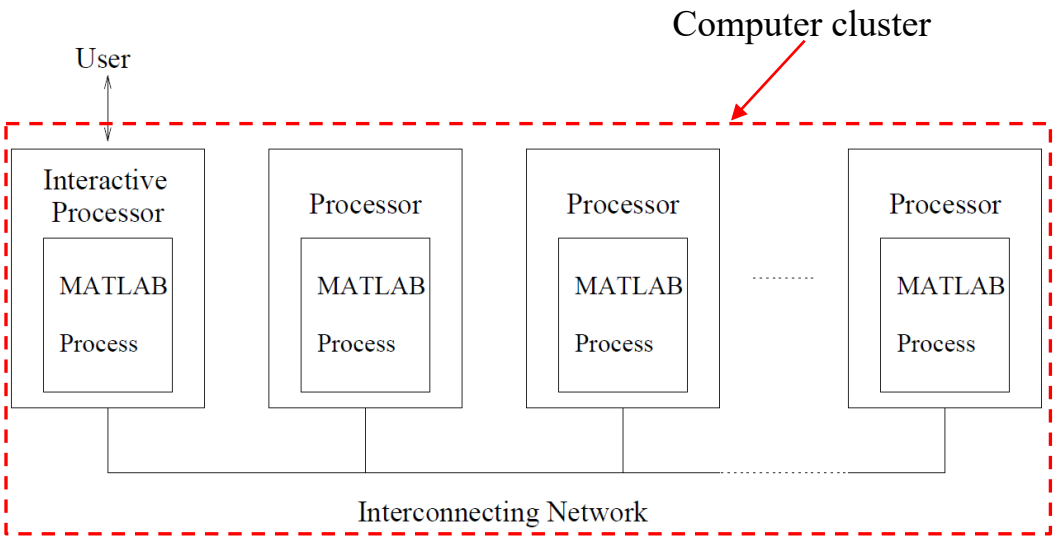



Figure 1: MultiMATLAB Architecture

**Ex.1005, FIG. 1 (annotated)**

**135.** Menon discloses "two implementations of the MultiMATLAB architecture," one "designed to run on a generic network of Unix workstations" and another that "was designed specifically for the IBM SP2," "a modern high performance distributed memory multiprocessor." Ex.1005, 5, 6. Menon describes implementing MultiMATLAB on a plurality of processors, such as 8 processors or "32 processors[,] of the IBM SP2," as shown in FIGs. 7 and 8 below. Ex.1005, 13.

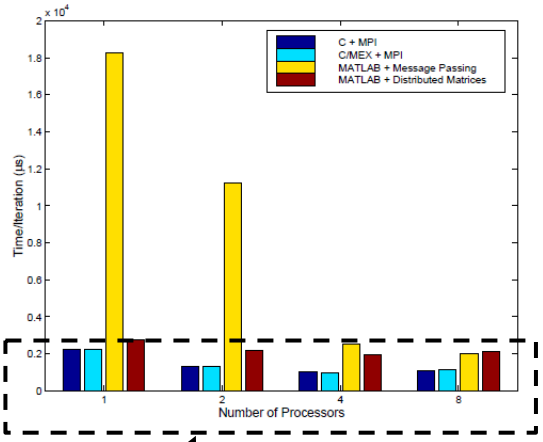



Figure 7: Parallel Conjugate Gradients on the IBM SP-2: 256 × 256 matrix

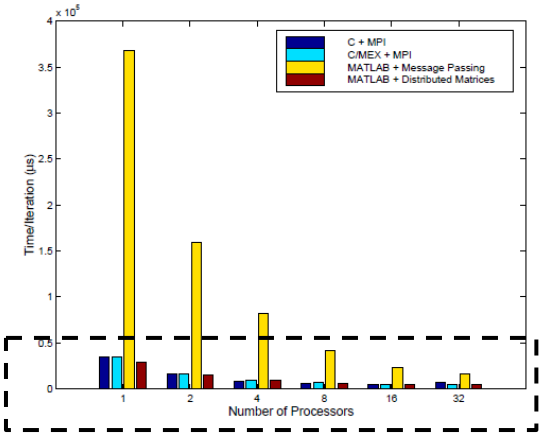



Figure 8: Parallel Conjugate Gradients on the IBM SP-2: 1024 × 1024 matrix

8 processors

32 processors

**Ex.1005, FIGs. 7 and 8**

**136.** The IBM SP2 is "a state-of-the-art parallel computing system" designed to "handle data-intensive, computer-intensive jobs" and "execute serial and parallel applications simultaneously." Ex.1007, 1. The IBM SP2 system includes:

- Processor nodes
- Frames with integral power subsystems
- Switches
- Extension nodes
- Control workstations
- Network connectivity adapters
- External disk drives

Ex.1007, 2.

**137.** The IBM SP2 system is used to "execute parallel programs on … a networked cluster of [IBM SP2] RS/6000 processors." Ex.1008, 1. For example, Menon describes the IBM SP2 system being used to perform mathematical computations, where a user provides a mathematical formula into a user interface on a first node of the IBM SP2 and then the other nodes perform mathematical computations in parallel to provide the user an answer. Ex.1005, 6-8, 11-14 (example "MultiMATLAB implementations of the conjugate gradients algorithm on the IBM SP2"). As another example, Trefethen discusses the IBM SP2 system performing mathematical evaluations such as arithmetic operations (e.g., doubling a value, calculating square root of numbers, etc.) and matrix operations (e.g., calculations of eigenvalues and condition number of Hilbert matrices, etc.). Ex.1006, 4-6.

**138.** In short, Menon's MATLAB processes are designed to run in parallel on individual processors, as implemented on an IBM SP2 with a plurality of processor nodes, and the IBM SP2's parallel computing system (as described in RS6000) has a series of processor nodes to execute parallel applications. A POSITA would then understand that Menon's teaching of a series of MATLAB processes in view of RS6000's teaching of the parallel computing system with multiple

processor nodes would result in Menon's MATLAB processes each running on a processor node of the IBM SP2, as exemplified below:

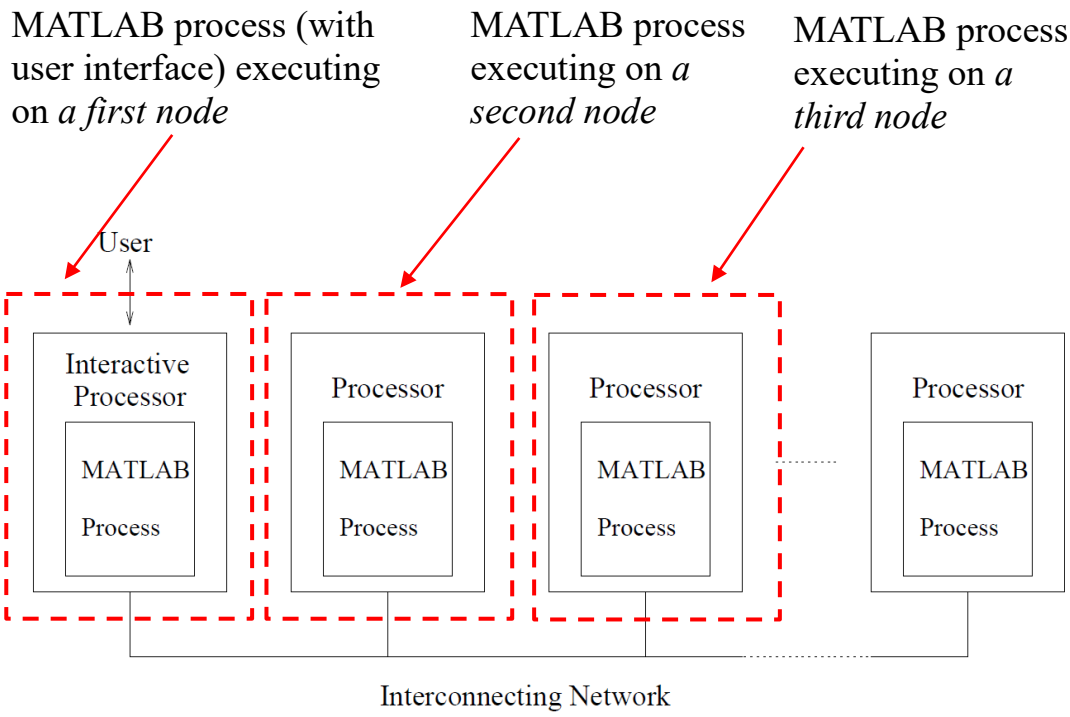



Figure 1: MultiMATLAB Architecture

**Ex.1005, FIG. 1 (annotated).**

**139.** Accordingly, Menon's MultiMATLAB architecture with a series of MATLAB processes, implemented on an IBM SP2 using 32 processors, to work in parallel to calculate mathematical formulas, in view of RS6000's description that the IBM SP2 is a computing system having processor nodes, switches, network connectivity, and disk drives, for executing software, renders obvious *a computer cluster*.

### b. [1.1.1] *a plurality of nodes, wherein each of the plurality of nodes comprises a hardware processor*

**140.** Menon in view of RS6000 renders obvious this element. Specifically, Menon discloses a parallel processing system executing a series of MATLAB processes on 32 processor nodes, implemented on the IBM SP2, and RS6000 discloses that the IBM SP2 is a parallel processing system having up to 128 or more processor nodes, where each processor node has a symmetric multiprocessor, which can have up to sixteen processors, the combination of Menon and RS6000 thus rendering obvious *a plurality of nodes, wherein each of the plurality of nodes comprises a hardware processor*.

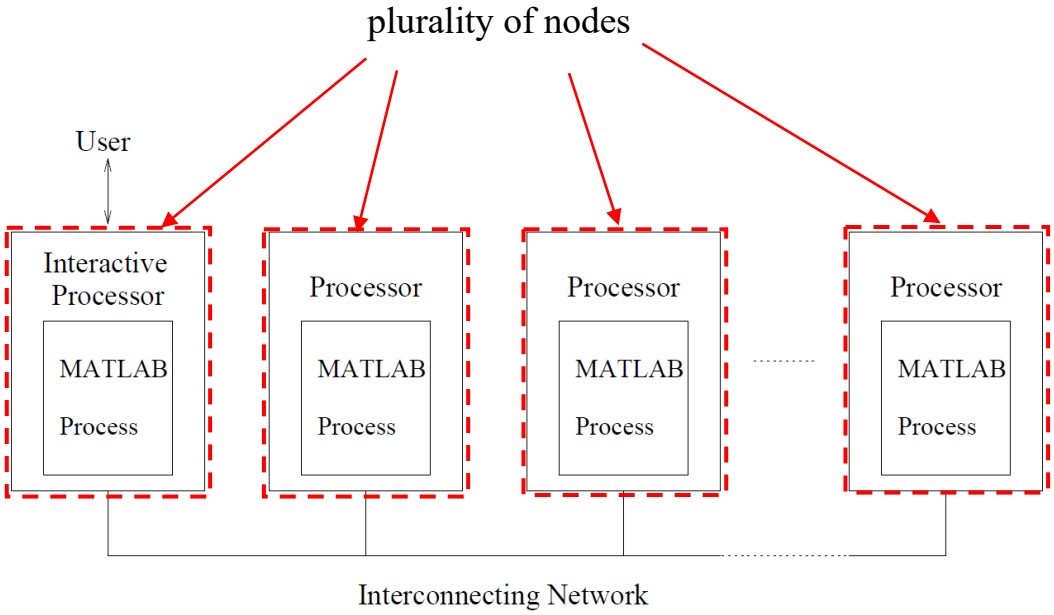



Figure 1: MultiMATLAB Architecture

**Ex.1005, FIG. 1 (annotated)**

**141.** Menon describes an "implementation of MultiMATLAB" architecture "for the IBM SP2" that comprises "32 processors of the IBM SP2." Ex.1005, 5, 13. The number of processors is not limited to 32 processors as the IBM SP2 is "scalable from one to 128 processor nodes" and "systems with more than 128 processor nodes are [also] available." Ex.1007, 2. Each processor node of the IBM SP2 can "have either a Symmetric MultiProcessor (SMP) configuration or a uniprocessor configuration." Ex.1007, 2. The IBM SP2 processor nodes enable the execution of "both serial and parallel applications." Ex.1007, 1. RS6000 provides each "375 MHz POWER3 SMP High Node (F/C 2058)" with a symmetric multiprocessor having "four, eight, twelve, or sixteen 375 MHz 630FP 64-bit processors per node." Ex.1007, 9.

**142.** Thus, Menon discloses an IBM SP2 system having 32 processor nodes for executing a series of parallel processes and RS6000 teaches that the IBM SP2 consists of up to 128 or more processor nodes, where each processor node has a symmetric multiprocessor having up to sixteen processors, the combination of Menon and RS6000 thus rendering obvious *a plurality of nodes, wherein each of the plurality of nodes comprises a hardware processor*.

### c. [1.1.2] *wherein one or more of the nodes are configured to receive a command to start a cluster initialization process for the computer cluster, and*

**143.** Menon in view of POEref renders obvious this element. Specifically, Menon discloses an IBM SP2 processor from which a user, by utilizing IBM's POE, launches an interactive MATLAB process on that IBM SP2 processor and a non-interactive MATLAB process on each of the other IBM SP2 processors, where the non-interactive MATLAB processes wait for further commands from the interactive MATLAB process. Further, POEref discloses that POE initializes a local environment, i.e., a MATLAB process, on a processor node and reproduces that local environment on each of the remote processor nodes. The combination of Menon and POEref thus renders obvious *wherein one or more of the nodes are configured to receive a command to start a cluster initialization process for the computer cluster*.

**144.** Menon discloses that MATLAB processes of a MultiMATLAB architecture implemented on the IBM SP2 are started by a user utilizing IBM's POE:

> In this implementation, all MATLAB processes are started via POE, IBM's Parallel Operating Environment. For each of *p* MATLAB processes, POE assigns an identification number between 0 and *p* - 1. The process numbered 0 is considered the interactive MATLAB. All

> other processes wait for commands from the interactive MATLAB process.

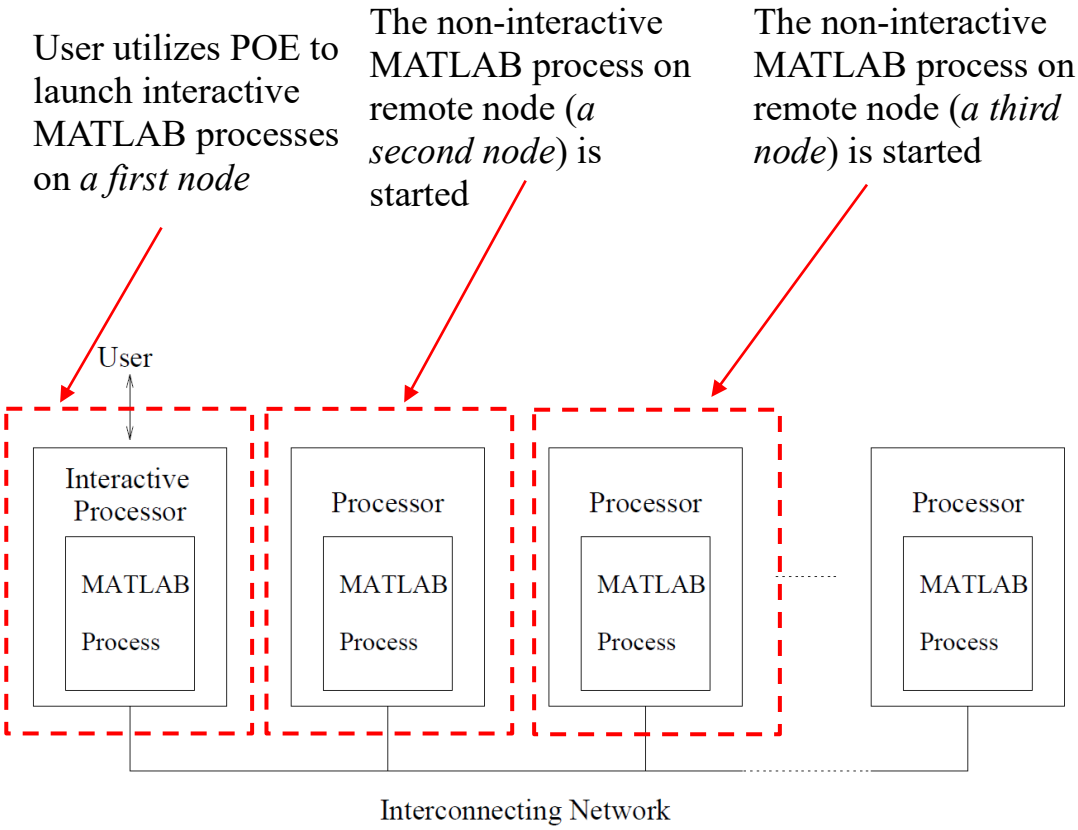
Ex.1005, 6. The interactive MATLAB process is the MATLAB process with which the user interacts. Ex.1005, 3. The user invokes the initialization process using the "poe" command:

> When you invoke **poe**, the Partition Manager allocates processor nodes for each task and initializes the local environment. It then loads your program, and reproduces your local environment, on each processor node. The Partition Manager also passes the option list to each remote node. … If you are using the dynamic message passing interface, the appropriate communication subsystem library implementation (IP or US) is automatically loaded at this time.

Ex.1008, 28.

**145.** In other words, when a user on an IBM SP2 invokes **poe**, a local environment, i.e., a MATLAB process, is initiated on the IBM SP2 processor with which the user is interacting and reproduced on each of the other IBM SP2 processors (called "remote nodes" by POEref), and the MATLAB processes (called "non-interactive processes" by Menon) on these remote nodes await for commands from the interactive MATLAB process. Ex.1005, 6, 4 (calling the MATLAB processes on the remote nodes "non-interactive processes"); Ex.1008, 28, 3 (discussing "remote nodes"). Specifically, "[o]n non-interactive processes, a

separate top-level MEX Routine is immediately run upon initialization of the system. This MEX Routine runs in a loop in which it waits for MATLAB commands to arrive from the interactive process and executes them in its own MATLAB environment." Ex.1005, 4. The initialization of MATLAB processes on the IBM SP2 are illustrated in FIG. 1 of Menon below.




**Ex.1005, FIG. 1 (annotated)**

**146.** Accordingly, Menon discloses an IBM SP2 processor from which a user, utilizing IBM's POE, launches an interactive MATLAB process on that IBM SP2 processor and a non-interactive MATLAB process on each of the other IBM

SP2 processors (where the non-interactive MATLAB processes wait for further commands from the interactive MATLAB process), and POEref discloses that POE initializes a local environment, i.e., a MATLAB process, on the processor node the user is using and reproduces that local environment on each of the remote processor nodes, the combination of Menon and POEref thus rendering obvious *wherein one or more of the nodes are configured to receive a command to start a cluster initialization process for the computer cluster*.

**d. [1.1.3] *wherein each of the nodes is configured to access a non-transitory computer-readable medium comprising program code for a single-node kernel that, when executed, is capable of causing the hardware processor to evaluate mathematical expressions; and***

**147.** Menon in view of Trefethen and RS6000 renders obvious this element. Specifically, Menon discloses each processor accesses the MATLAB process address space (*computer-readable medium*) having MATLAB (*computer program code for a single-node kernel*) for executing single matrix-vector multiplication (*evaluate mathematical expressions*), Trefethen teaches each processor performing mathematical evaluations such as matrix operations, doubling a variable, singular value decomposition, fast Fourier transform, and polynomial zerofinding (*evaluate mathematical expressions*), and RS6000 discloses that each processor node of the IBM SP2 has hard drives and up to 64 GB memory for storing program code accessible by the processor nodes, the combination of Menon and RS600 this

rendering obvious *wherein each of the nodes is configured to access a non-transitory computer-readable medium comprising program code for a single-node kernel that, when executed, is capable of causing the hardware processor to evaluate mathematical expressions*.

**148. First**, Menon discloses that each one of the IBM SP2 processors has a MATLAB process, explaining that "[i]n the MultiMATLAB architecture, illustrated in Figure 1, each processor in a parallel platform individually runs a MATLAB process." Ex.1005, 3. FIG. 1 of Menon below shows a MATLAB process executing on each IBM SP2 processor of the MultiMATLAB architecture.

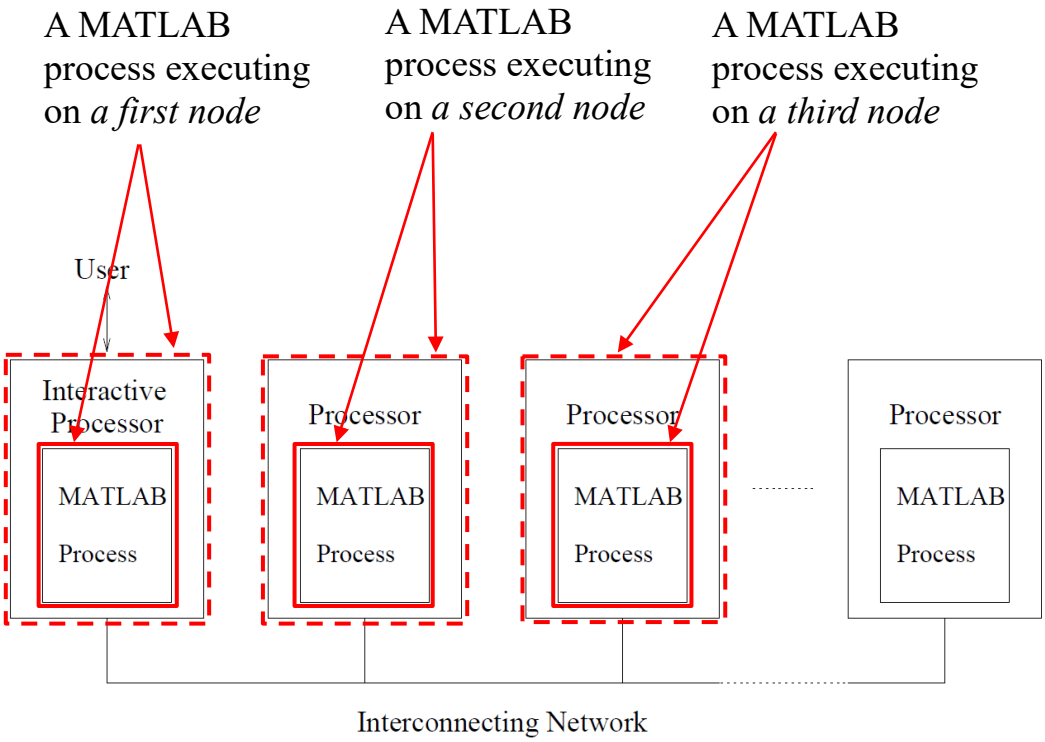



Figure 1: MultiMATLAB Architecture

**Ex.1005, FIG. 1 (annotated).**

**149.** Menon explains that each MATLAB process contains MATLAB, "the numerical computing environment of choice on uniprocessors for hundreds of thousands of engineers and scientists." Ex.1005, 1, FIG. 2 (showing a MATLAB process containing MATLAB). MATLAB is a "scientific computing environment" that provides users with "an extensive library of high quality numerical routines." Ex.1005, 1. For example, MATLAB "provides a high-level matrix based language that permits users to express computations in an exceptionally concise manner via matrix or vector formulations." Ex.1005, 1. FIG. 2 of Menon below shows a MATLAB process containing MATLAB.

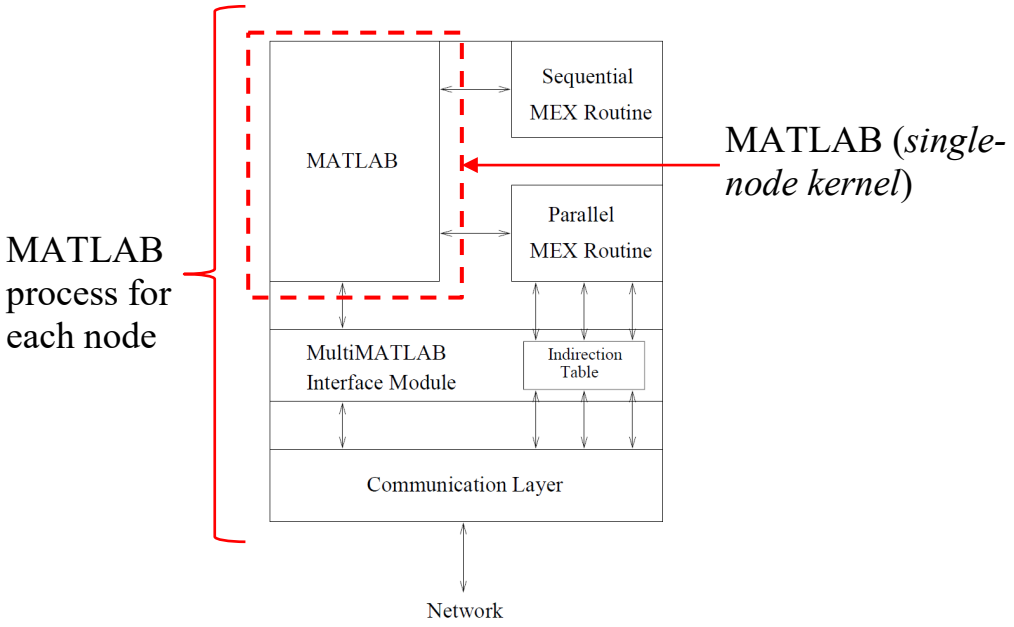



Figure 2: MATLAB Process Address Space

**Ex.1005, FIG. 2 (annotated)**

**150.** Because MATLAB is "MATLAB software," a POSITA would recognize MATLAB is a *program code* that is executed by a processor. Ex.1006, 2; Ex.1005, 3. Accordingly, Menon's disclosure of MATLAB that is executed by a processor teaches *a program code for a single-node kernel* that is *executed*.

**151.** **Second**, Menon and Trefethen teach MATLAB is capable of causing a processor to evaluate mathematical expressions, as illustrated by the "MultiMATLAB implementations of the conjugate gradients algorithm on the IBM SP2" discussed in Menon and mathematical evaluations such as arithmetic operations (e.g., doubling a value, calculating square root of numbers, etc.), singular value decomposition, fast Fourier transform, polynomial zerofinding, and matrix operations (e.g., calculations of eigenvalues and condition number of Hilbert matrices, etc.) discussed in Trefethen. Ex.1005, 11-13; Ex.1006, 2-6.

**152.** Menon explains that the "parallel MATLAB implementation of conjugate gradients" includes the step of "matrix A and the different vectors [] each [being] distributed by row over all processes." Ex.1005, 11. FIG. 6 of Menon shown below illustrates this step where "[m]atrix A and vector x [are] distributed by row over *P* processors" when the MultiMATLAB implementation includes "*p* MATLAB processes." Ex.1005, 12, 6 (emphasis original).

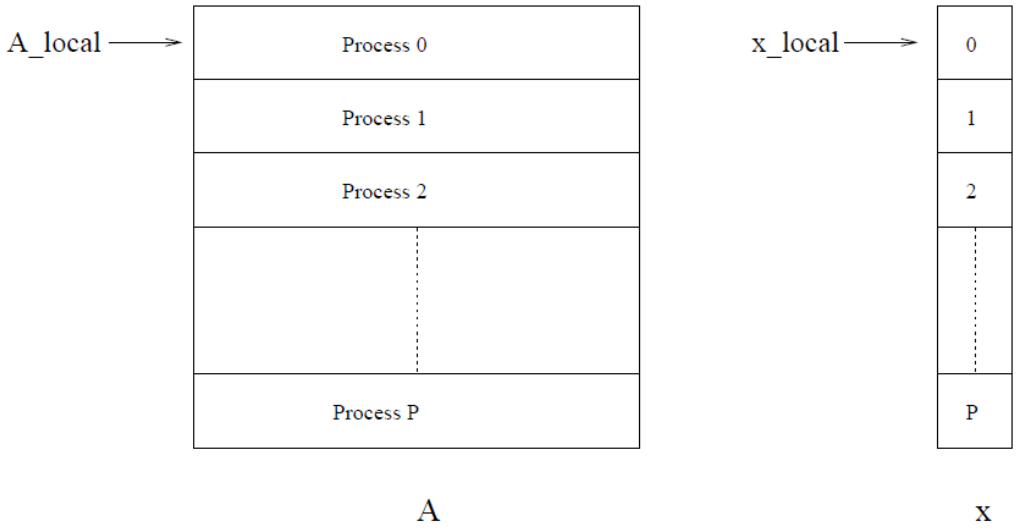



Figure 6: Matrix $A$ and vector $x$ distributed by row over $P$ processors

**Ex.1005, FIG. 6**

**153.** Each MATLAB process performs its own computation locally but exchanges data with other MATLAB processes:

> [i]n each iteration, a process only does the computation required for its local data. However, communication is required for two different operations: the dot-products needed to compute *rho* and *alpha* and the matrix-vector multiply needed to compute *w_local*. The dot products only require communication of a single value over all processes. On the other hand, the matrix-vector multiplication needed for *w_local* requires the global vector *p*, and, thus, more expensive communication."

Ex.1005, 11-13 (emphasis original). The conjugate gradients algorithm computations are *evaluat[ions of] mathematical expressions*, such as performing single matrix-vector multiplication: "[t]he conjugate gradients algorithm is iterative method to solve a linear system of the form $Ax = b$, where $A$ is a symmetric positive definite matrix and $x$ and $b$ are vectors. The computational core of this method is a single matrix-vector multiplication at each iteration." Ex.1005, 11 (emphasis original). FIG. 4 of Menon shown below illustrates the conjugate gradients algorithm computations, i.e., the *evaluat[ions of] mathematical expressions*, by MATLAB (*a single-node kernel*).

code to *evaluate mathematical expressions*

```
r = b;
rho = r'*r;
k = 0;
while((k < kmax))
  k = k+1;
  if (k == 1)
    p = r;
  else
    beta = rho/oldrho;
    p = r + beta*p;
  end

  w = A * p;

  alpha = rho/(p'*w);
  x = x + alpha * p;
  r = r - alpha * w;
  oldrho = rho;
  rho = r'*r;
end
```

a. Sequential MATLAB

```
r_local = b_local;
rho = Sum(r_local'*r_local);
k = 0;
while((k < kmax))
  k = k+1;
  if (k == 1)
    p_local = r_local;
  else
    beta = rho/oldrho;
    p_local = r_local + beta*p_local;
  end

  p = Gather(p_local);
  w_local = A_local * p;

  alpha = rho/(Sum(p_local'*w_local));
  x_local = x_local + alpha * p_local;
  r_local = r_local - alpha * w_local;
  oldrho = rho;
  rho = Sum(r_local'*r_local);
end
```

b. Message Passing MATLAB

Figure 4: Conjugate Gradients

**Ex.1005, FIG. 4 (annotated)**

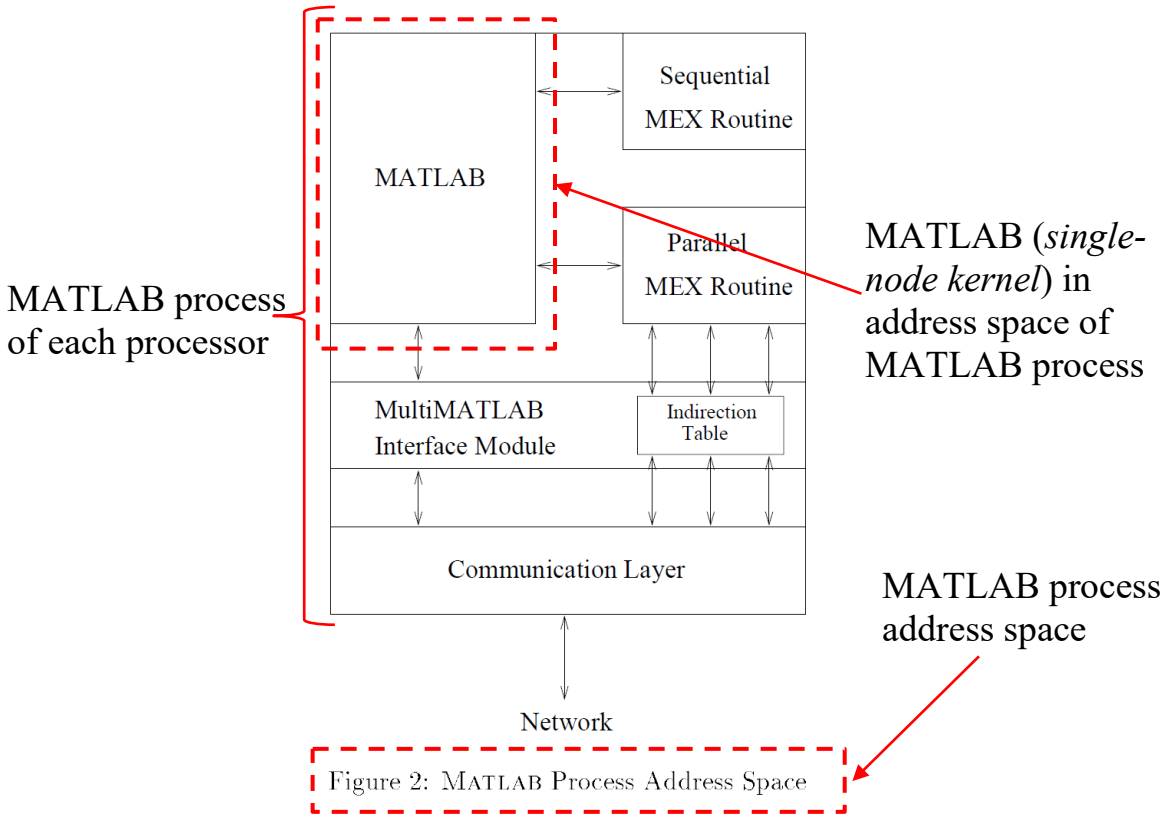
**154.** Further, Trefethen discloses "[a] system … that enables one to run … MATLAB conveniently on multiple processors" to perform *evaluat[ions of] mathematical expressions* such as doubling a value, calculating square root of numbers, singular value decomposition, fast Fourier transform, polynomial zerofinding, eigenvalue and Hilbert matrix condition number, etc. Ex.1006, Abstract, 2-6.

**155.** Accordingly, Menon and Trefethen teach that MATLAB (*a single-node kernel*) of a MATLAB process cause *a hardware processor* that includes the MATLAB process to *evaluate mathematical expressions*.

**156.** **Third**, Menon teaches that a processor accesses MATLAB from a *computer-readable medium*, because Menon describes the MATLAB process, which includes MATLAB, as being in an "address space" and further describes the building of "a table of pointers to all functions and data" that the parallel MEX Routines use "to access any function provided by the underlying communication layer" during the initialization of the MATLAB process. Ex.1005, FIG. 2.

**157.** As discussed previously, a POSITA would understand an "address space" (as described in Menon) to be a "a list of memory locations from some minimum (usually 0) to some maximum, which the process can read and write. The address space contains the executable program, the program's data, and its stack." Ex.1018, 67-68. Because Menon's address space refers to a computer's memory, a

POSITA would recognize that the MATLAB process and MATLAB are in a *computer-readable medium*, and that MATLAB is accessed from a *computer-readable medium*, i.e., *a hardware processor* accesses MATLAB from a *computer-readable medium*. FIG. 2 of Menon below shows MATLAB as part of the MATLAB process address space. The *computer-readable medium* of the processors of the IBM SP2, on which the conjugate gradients algorithm is implemented, can be in the form of internal and external hard disk drives, or memory cards, as taught by RS6000. Ex.1007, 6, 10 (disclosing IBM SP2 processors have internal and external hard disk drives), 10 (disclosing IBM SP2 processors have "one to four memory cards" that support a "maximum of 64 GB").




**Ex.1005, FIG. 2 (annotated)**

**158.** Accordingly, Menon's disclosure of each processor accessing the MATLAB process address space (*computer-readable medium*) having MATLAB (*computer program code for a single-node kernel*) for executing single matrix-vector multiplication (*evaluate mathematical expressions*) in view of Trefethen teaching each processor performing mathematical evaluations such as matrix operations, doubling a variable, singular value decomposition, fast Fourier transform, and polynomial zerofinding (*evaluate mathematical expressions*) and RS6000 teaching that each processor node of the IBM SP2 has hard drives and up to 64 GB memory for storing program code accessible by that processor node,

renders obvious *wherein each of the nodes is configured to access a non-transitory computer-readable medium comprising program code for a single-node kernel that, when executed, is capable of causing the hardware processor to evaluate mathematical expressions*.

**e. [1.2] *a mechanism for the nodes to communicate results of mathematical expression evaluation with each other using a peer-to-peer architecture;***

**159.** Menon in view of Trefethen render obvious this element. Specifically, Menon describes MEX routines, having code to implement MPI calls, that enable SPMD point-to-point communication so that each processor can communicate directly with each other over an interconnecting network, and Trefethen discloses that the SPMD point-to-point communication enables a peer-to-peer architecture where mathematical expression evaluation results from a first MATLAB process on a first processor that is communicated to a second MATLAB process on a second processor, and the mathematical expression results obtained from the second MATLAB process on the second processor are communicated to a third MATLAB process on a third processor, and so on, rendering obvious *a mechanism for the nodes to communicate results of mathematical expression evaluation with each other using a peer-to-peer architecture*.

**160.** **First**, Menon discloses that MultiMATLAB uses "MEX routines" and that "[a]ll parallel functionality in the MultiMATLAB system is provided through

MEX routines." Ex.1005, 3. For example, "the MultiMATLAB system provides users with an Eval routine for parallel evaluation. This routine allows users to execute commands on the other processes." Ex.1005, 3-4. "[T]he Eval routine is implemented [on the interactive process] as a MEX routine which accepts a vector of MATLAB process IDs and a MATLAB command" that the MEX Routine sends to the MATLAB processes identified by the MATLAB process IDs. Ex.1005, 4. "On non-interactive processes, a separate top-level MEX routine is immediately run upon initialization of the system. This MEX routine runs in a loop in which it waits for MATLAB commands to arrive from the interactive process and executes them in its own MATLAB environment." Ex.1005, 4. The MEX Routines send the "commands over the underlying communication layer to the processes corresponding to the specified IDs." Ex.1005, 4.

**161.** The underlying communication layer of the MultiMATLAB architecture "runs over the parallel platform's interconnection network" and "provide[s] [a MATLAB process on a processor] with the ability to communicate with other processes." Ex.1005, 3. An example of the underlying communication layer is MPI. Ex.1005, 3. Table 1 of Menon below shows MultiMATLAB commands implemented as MEX Routines. Further, FIG. 2 and FIG. 1 of Menon below show the MEX Routines of a MATLAB process and the interconnection

network of the MultiMATLAB architecture linking the processors thereof, respectively.

| MultiMATLAB commands | |
|---|---|
| Starting and stopping MultiMATLAB (MPICH only) | |
| Start(n) | Initializes n remote MATLAB processes. |
| Quit | Closes remote MATLAB processes. |
| Running commands on multiple processors | |
| Nproc | Returns number of MATLAB processes. |
| ID | Returns ID of calling MATLAB process. |
| Eval(pid,'command') | Evaluates a command on one or more MATLAB process. |
| Master/Slave Communication | |
| Put(pid,dataname) | Put data from the master process onto one or more remote processes. |
| Get(pid,dataname) | Get data from a remote process onto the master process. |
| SPMD Communication | |
| Send(pid,data) | Send data from one process to another. |
| Recv(pid) | Receive data sent from another process. |
| Barrier | Synchronize processes. |
| Bcast(pid,data) | Broadcasts data from processor pid to all processes. |
| Sum(data) | Adds data across all processors to form a global sum. |
| Collect(data) | Collects local data segments into a single global one. |

Table 1: Selected commands in the MultiMATLAB system

Commands implemented in MEX Routines

**Ex.1005, Table 1 (annotated)**

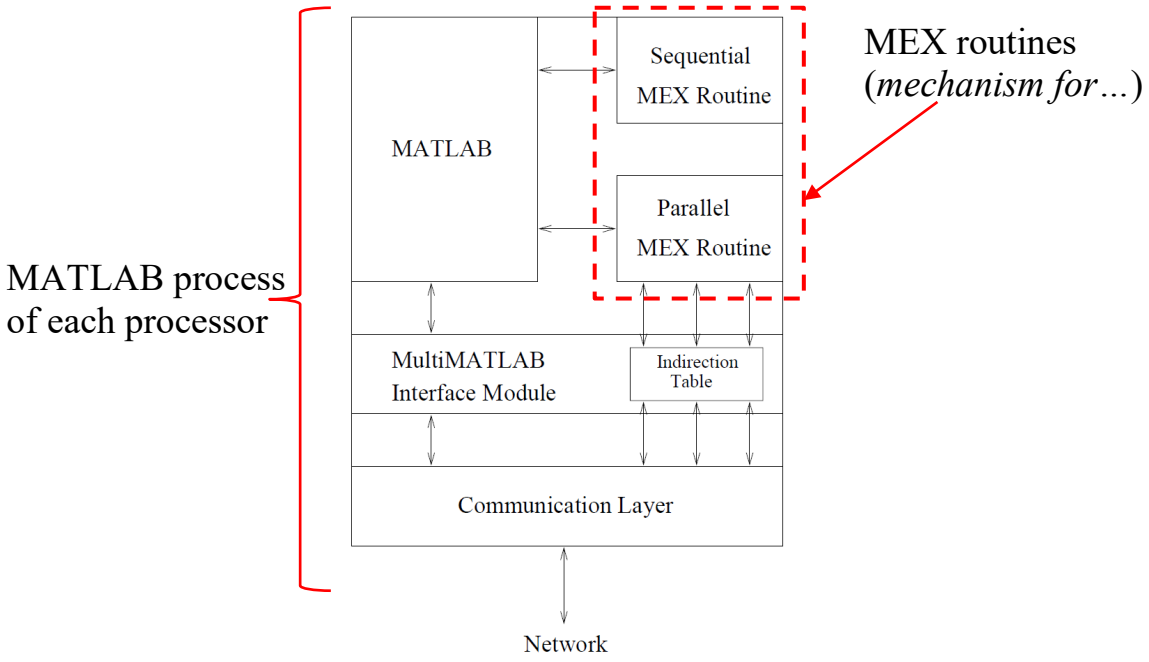



Figure 2: MATLAB Process Address Space

**Ex.1005, FIG. 2 (annotated)**

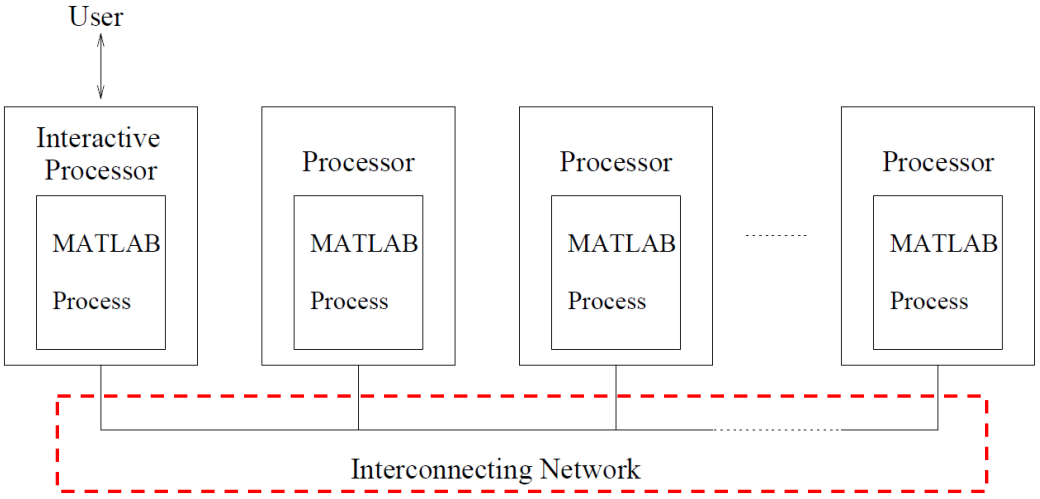



Figure 1: MultiMATLAB Architecture

**Ex.1005, FIG. 1 (annotated)**

Interconnecting network connecting each MATLAB process on a processor node to any other in MultiMATLAB

**162. Second**, Trefethen discloses that the commands used in MultiMATLAB are "written as a C file" and "interfaced to MATLAB via the standard MATLAB Fortran/C/C++ interface known as MEX." Ex.1006, 10. The "[h]igher-level MultiMATLAB commands are usually built on higher-level MPI commands." Ex.1006, 10. Trefethen teaches that "MultiMATLAB is currently built upon the standard send and receive variants" of MPI. Ex.1006, 10. That is, "[m]ore general point-to-point communication is accomplished by send and receive commands, which can be executed on any of the MATLAB processes." Ex.1006, 5.

**163.** Trefethen also discloses sending tasks and data to other processors using these MultiMATLAB commands. Ex.1006, 3-7. "The standard MultiMATLAB command for executing commands on one or more processors is Eval." Ex.1006, 3. For example,

> The command
>
> Eval( 'ID^ID' )
>
> might produce
>
> ans = 1
>
> ans = 1
>
> ans = 256
>
> ans = 3125
>
> ans = 27

ans = 4.

Ex.1006, 4. That is, Eval( 'ID^ID' ) sends the task "ID^ID" to each processor ID to perform the task "ID^ID" using the data ID. Ex.1006, 4.

**164.** Trefethen discloses that a user can specify which MATLAB processes should perform the tasks, explaining that "one can select a subset of the processes by passing two arguments to the Eval command, the first being a vector of process IDs." Ex.1006, 4. For example,

Eval( [4 5] , 'cond(hilb(ID))' )

might return

ans = 1.5514e+04

ans = 4.7661e+05

the condition numbers of the Hilbert matrices of

dimensions 4 and 5.

Ex.1006, 4. That is, the MATLAB processes on processor ID=4 and ID=5 each perform the task "cond(hilb(ID))" using the data ID, i.e., MATLAB process on processor ID=4 performs the task "cond(hilb(ID = 4))" using the data ID=4 and generates the result "ans = 1.5514e+04," "the condition number[] of the Hilbert [matrix] of dimension[] 4," and MATLAB process on processor ID=5 performs the task "cond(hilb(ID = 5))" using the data ID=5 and generates the result "ans =

4.7661e+05," "the condition number[] of the Hilbert [matrix] of dimension[] 5." Ex.1006, 4.

**165.** The MEX Routines that contain program code to implement the standard MPI send and receive calls to send commands (e.g., tasks) and data directly to other processes renders obvious an MPI module.

**166.** **Third**, Menon's MEX Routines are implemented using "point to point … communication" paradigm used to enable communication between the MATLAB processes operating in parallel in the MultiMATLAB architecture. Ex.1005, 9. Menon explains that, in contrast to a "master/slave paradigm" that "provid[es] a very simple mechanism for coarse-grain distributed computing" and "is often adequate in situations where little or no communication is required," Menon's SPMD/Message Passing paradigm "accommodate[s] applications requiring a finer degree of communication" and provides "point to point … communication." Ex.1005, 8-9. SPMD [single program multiple data] refers to a "style of parallel programming, where all processors use the same program, though each has its own data." Ex.1009, 1. Table 1 of Menon below shows example point-to-point commands for SPMD communication paradigm, such as the "Send(pid,data)" command that "send[s] data from one process to another," the "Recv(pid)" command that "receive[s] data sent from another process," and the "Bcast(pid,data)" command that "broadcasts data from processor pid to all

processes." Ex.1005, Table 1. The SPMD point-to-point communication paradigm where each node can communicate tasks and data directly with each other node renders obvious *a peer-to-peer architecture*.

| MultiMATLAB commands | |
|---|---|
| Starting and stopping MultiMATLAB (MPICH only) | |
| Start(n) | Initializes n remote MATLAB processes. |
| Quit | Closes remote MATLAB processes. |
| Running commands on multiple processors | |
| Nproc | Returns number of MATLAB processes. |
| ID | Returns ID of calling MATLAB process. |
| Eval(pid,'command') | Evaluates a command on one or more MATLAB process. |
| Master/Slave Communication | |
| Put(pid,dataname) | Put data from the master process onto one or more remote processes. |
| Get(pid,dataname) | Get data from a remote process onto the master process. |
| SPMD Communication | |
| Send(pid,data) | Send data from one process to another. |
| Recv(pid) | Receive data sent from another process. |
| Barrier | Synchronize processes. |
| Bcast(pid,data) | Broadcasts data from processor pid to all processes. |
| Sum(data) | Adds data across all processors to form a global sum. |
| Collect(data) | Collects local data segments into a single global one. |

Table 1: Selected commands in the MultiMATLAB system

Commands for point-to-point communication

**Ex.1005, Table 1 (annotated)**

**167. Fourth**, Trefethen describes a MATLAB process on a processor of the MultiMATLAB architecture using the SPMD point-to-point commands to communicate *results of mathematical expression evaluation* to another MATLAB process on another processor. Ex.1006, 6. Specifically, Trefethen provides the following example code in which a MATLAB process communicates a result of a

mathematical evaluation to a next MATLAB process, which in turn communicates the next mathematical result to the next MATLAB process, and so on:

```
if ID==0                    % first process: send
  a = 1
  Send(ID+1,a)
elseif ID == Nproc-1        % last process: receive and double
  a = 2*Recv
else                        % middle processes: receive, double
  a = 2*Recv
  Send(ID+1,a)
end;
```

Recv command receives result from preceding node

Send() command sends result to the next node

Process 0 creates the variable a with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of a, and sends it along to process 2; and so on. If there are six processors the command `Eval( 'cycle' )` produces the output

```
a = 1
a = 2
a = 4
a = 8
a = 16
a = 32
```

Point-to-point communications of mathematical evaluation results between MATLAB processes on different processors

Ex.1006, 6 (annotated).

While Trefethen's example uses six processors, it shows output from the third processor (a=4), and it would have been obvious to a POSITA that Trefethen contemplates any number of processors ("and so on"), including three processors.

**168.** Trefethen also discloses that the MultiMATLAB commands can deliver a mathematical result to a designated process. Ex.1006, 7. For example, Trefethen explains that:

> The command
>
> Eval( 'Sum(1,[1 ID Nproc])' )

> executed on six processes will return the vector
>
> [6 15 36]
>
> to process 1. If the first argument is omitted, the result is
>
> returned (broadcast) to all processes.

Ex.1006, 7.

**169.** Accordingly, Menon's MEX routines, having code to implement MPI calls, that enable SPMD point-to-point communication so that each processor can communicate directly with each other over an interconnecting network, further in view of Trefethen teaching that the SPMD communication enables mathematical expression evaluation results from a first MATLAB process on a first processor to be communicated to a second MATLAB process on a second processor, and the mathematical expression results obtained there to be communicated to a third MATLAB process on a third processor, and so on, renders obvious *a mechanism for the nodes to communicate results of mathematical expression evaluation with each other using a peer-to-peer architecture*.

**f. [1.3.0] *wherein the plurality of nodes comprises:***

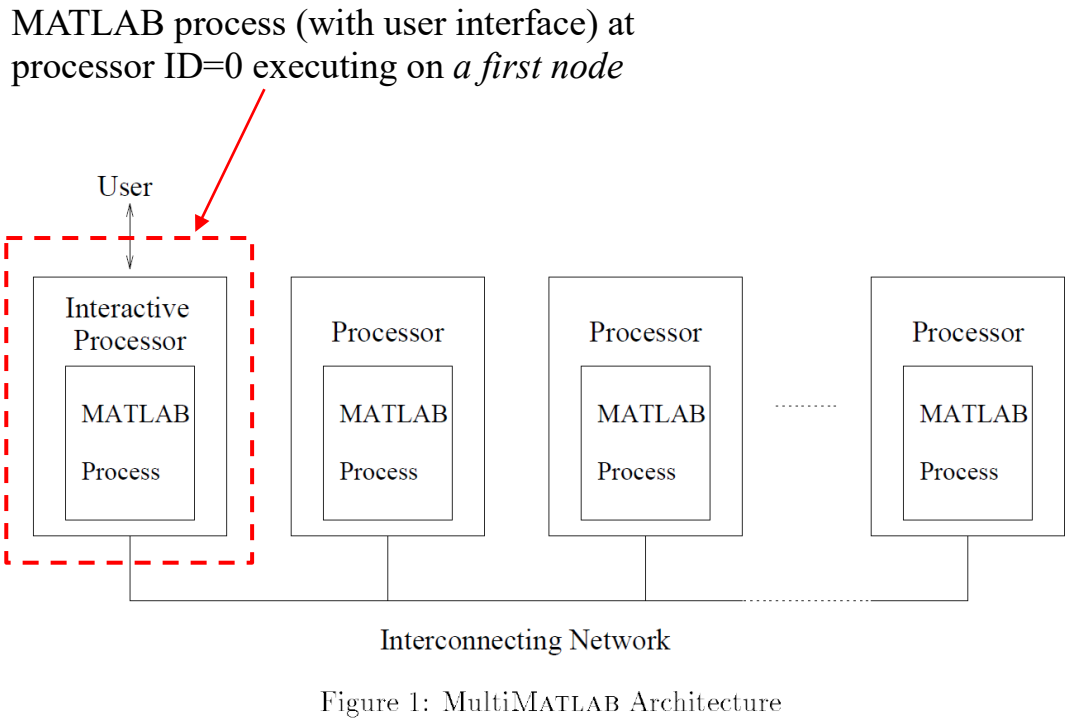
**170.** Limitation [1.3.0] is rendered obvious by Menon in view of RS6000 for the same reasons presented above for limitation [1.1.1].

**g. [1.3.1] *a first node comprising a first hardware processor configured to access a first memory comprising program code for a user interface and program code for a first single-node kernel,***

**171.** Menon in view of RS6000 render obvious this element. Specifically, Menon and RS6000 describe a node having (i) a symmetric multiprocessor that accesses *program code* stored in *memory* to be executed by a *processor*, and (ii) the MATLAB code stored in the address space of, and accessed by, the processor, includes a toolkit with a graphical interface to allow users to provide equations and directly interact with the interactive processor node (*user interface*), thereby rendering obvious *a first node comprising a first hardware processor configured to access a first memory comprising program code for a user interface and program code for a first single-node kernel.*

**172.** **First**, it was shown in limitation [1.1.1] that the combination of Menon and RS6000 renders obvious *a node comprising a hardware processor*. Further, it was shown in limitation [1.1.3] that the combination of Menon and RS6000 renders obvious *a node configured to access* Menon's address space, which refers to a computer's memory, comprising *program code for a single-node kernel*, where the memory can be in the form of memory cards having up to 64 GB memory for storing program code accessible by that processor. Thus, Menon and RS6000 render obvious that *a first node comprising a first hardware processor configured to access a first memory comprising … program code for a single-node kernel*.

**173. Second**, Menon describes that when MATLAB processes of a MultiMATLAB architecture are initialized, one of the MATLAB processes is an interactive MATLAB process with which a user can directly interact. Ex.1005, 3, 6. Specifically, Menon discloses that when the user utilizes IBM's POE to activate the MATLAB processes, "all MATLAB processes are started via POE, IBM's Parallel Operating Environment. For each of $p$ MATLAB processes, POE assigns an identification number between 0 and $p - 1$. The process numbered 0 is considered the interactive MATLAB." Ex.1005, 6 (emphasis original). "The user interacts directly with one MATLAB process, called the interactive process, and operates within that process's MATLAB environment." Ex.1005, 3.




**Ex.1005, FIG. 1 (annotated).**

**174.** Menon discloses a "toolkit" that provides "an intuitive graphical interface" to allow users to specify equations. Ex.1005, 6. This toolkit is "MATLAB's Partial Differential Equation Toolkit," in which "users may interactively create and solve rather general PDE systems using the finite element method. The PDE toolkit provides an intuitive graphical interface allowing users to specify geometries, differential equations, and boundary conditions." Ex.1005, 6. In other words, MATLAB's partial differential equation (PDE) toolkit has a graphical interface that allow users to directly interact with MATLAB.

**175.** Accordingly, because MATLAB is "MATLAB software," a POSITA would recognize the PDE toolkit with a graphical interface for users to provide equations is *program code for a user interface*. Ex.1006, 2.

**176.** Accordingly, Menon and RS6000 describe a node having (i) a symmetric multiprocessor that accesses *program code* stored in *memory* to be executed by a *processor*, and (ii) the MATLAB code stored in the address space of, and accessed by, the processor, and a toolkit with a graphical interface to allow users to provide equations and directly interact with the interactive processor node (*user interface*), rendering obvious *a first node comprising a first hardware processor configured to access a first memory comprising program code for a user interface and program code for a first single-node kernel.*

**h. [1.3.2] *the first single-node kernel configured to interpret user instructions and distribute calls to at least one of a plurality of other nodes for execution; and***

**177.** Menon in view of Trefethen render obvious this element. Specifically, Menon's teaching that MATLAB (*the first single-node kernel*) receives the Eval command including MATLAB commands and data including processor ID and causes calls to be sent to (*distribute calls to*) the MATLAB processes on other processors (*at least one of a plurality of other nodes*) in accordance with the IDs specified by the user instructions, in view of Trefethen's teaching that MATLAB interprets Eval commands (*interpret user instructions*) in order to assign different actions to different processes for those actions to be performed by those processes (*for execution*), renders obvious *the first single-node kernel configured to interpret user instructions and distribute calls to at least one of a plurality of other nodes for execution*.

**178.** Menon discloses that the interactive MATLAB process allows the user to directly interact with the MultiMATLAB architecture, explaining that "[t]he user interacts directly with one MATLAB process, called the interactive process, and operates within that process's MATLAB environment." Ex.1005, 3. The user can use "an Eval routine for parallel evaluation" to provide user instructions to the interactive MATLAB process so that the user instructions are evaluated in parallel fashion by the MultiMATLAB architecture. Ex.1005, 4. "This routine allows users

to execute commands on the other processes. On the interactive process, the Eval routine is implemented as a MEX Routine which accepts a vector of MATLAB process IDs and a MATLAB command. This routine sends the commands over the underlying communication layer to the processes corresponding to the specified IDs." Ex.1005, 4. In limitation [1.3.1], it was shown that MATLAB of the interactive MATLAB process teaches *a first single-node kernel*. Further, the user interface is configured so that the MEX routines "may be run directly from [the] MATLAB" environment. Ex.1005, Abstract, 3. Thus, on the interactive MATLAB process, MATLAB (*the first single-node kernel*) receives *user instructions* including MATLAB commands from a user and sends the commands to the other MATLAB processes for evaluation.

**179.** Trefethen provides example illustrations where *user instructions* are interpreted in order to send different actions for different MATLAB processes. Ex.1006, 3-7. For example, Trefethen discloses MATLAB (*the first single-node kernel*) receiving Eval( [4 5] , 'cond(hilb(ID))' ), which includes the data "ID," from the user. Ex.1006, 4. The Eval command (*user instruction*) instructs MATLAB (*the first single-node kernel*) to "select a subset of the processes by passing two arguments to the Eval command, the first being a vector of process IDs." Ex.1006, 4. In accordance with that instruction, the calls "cond(hilb(ID))" are distributed to the MATLAB processes on processor ID = 4 and ID = 5 for each to

perform that call, i.e., processor ID=4 calculates cond(hilb(ID=4)) and processor ID=5 calculates cond(hilb(ID=5)), where the results are:

"ans = 1.5514e+04,

ans = 4.7661e+05

the condition numbers of the Hilbert matrices of dimensions 4 and 5." Ex.1006, 4.

**180.** Trefethen discloses that a user can input instructions via an "m-file." Ex.1006, 5. The "m-file" contains instructions, such as "svd," "fft," and "roots," which are interpreted by the interactive MATLAB process to calculate the mathematical expressions "singular value decomposition," "fast Fourier transform," and "polynomial zerofinding," respectively. Ex.1006, 2. Trefethen describes that, rather than entering explicit MATLAB commands as an argument string, the user can use an "m-file" to provide instructions. Ex.1006, 5. That is, a user may "want to execute a program (an m-file) rather than a single line of text," and "[a] command such as Eval( 'filename' ) achieves this effect." Ex.1006, 5.

**181.** As an example, Trefethen discloses the *user instructions* Eval ('cycle') where the m-file "cycle.m" instructs: (i) the interactive MATLAB process to create a variable "a" with a value and send the variable to the next MATLAB process, (ii) the next middle MATLAB processes to receive the variable "a" from the immediately preceding MATLAB process, double the value of the received variable "a," and send the result to the next MATLAB process, and (iii) the last MATLAB

process to receive the variable "a" and double its value. Ex.1006, 6. The m-file cycle.m and the final output from a MultiMATLAB architecture that has six MATLAB processes are shown below:

*user instructions*

Instruction for MATLAB process ID=0 to create a=1 and send "a" to MATLAB process ID=1

```
if ID==0                    % first process: send
   a = 1
   Send(ID+1,a)
elseif ID == Nproc-1        % last process: receive and double
   a = 2*Recv
else                        % middle processes: receive, double, and send
   a = 2*Recv
   Send(ID+1,a)
end;
```

Instruction for the last MATLAB process to receive the variable a and double it

Process 0 creates the variable a with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of a, and sends it along to process 2; and so on. If there are six processors the command `Eval( 'cycle' )` produces the output

Instruction for a middle MATLAB process ID to receive the variable "a," double it, and send the doubled variable to the next MATLAB process ID=ID+1

```
a = 1
a = 2
a = 4
a = 8
a = 16
a = 32
```

Output of final result (a=32)

Ex.1006, 6 (annotated).

**182.** Thus, Trefethen discloses that MATLAB (*the first single-node kernel*) receives and interprets the command Eval ('cycle') (*configured to interpret user instructions and*) to send respective MATLAB commands to the MATLAB processes on the processors (*distribute calls to at least one of a plurality of other nodes*) so that those MATLAB processes perform the MATLAB commands (*for execution*). Accordingly, Menon's teaching that MATLAB is configured to interpret command and then causes calls to be sent to the MATLAB processes on other

nodes in accordance with the IDs specified by the user instructions, in view of Trefethen's teaching that MATLAB interprets Eval commands in order to assign different actions to different processes for those actions to be performed by those processes, renders obvious *the first single-node kernel configured to interpret user instructions and distribute calls to at least one of a plurality of other nodes for execution*.

#### i. [1.4.1] *a second node comprising a second hardware processor with a plurality of processing cores,*

**183.** Menon in view of RS6000 render obvious this element. Specifically, Menon discloses executing a series of parallel processes on 32 processors of the IBM SP2 system and RS6000 teaches that the IBM SP2 consists of up to 128 nodes, where each node has a symmetric multiprocessor having up to sixteen processors (each having a core), rendering obvious *a second node comprising a second hardware processor with a plurality of processing cores*.

**184.** In limitation [1.1.1], it was shown that Menon discloses a parallel processing system, the MultiMATLAB architecture, executing a series of MATLAB processes on 32 processors, implemented on the IBM SP2, and RS6000 discloses that the IBM SP2 is a parallel processing system having up to 128 or more processor nodes, where each processor node has a symmetric processor having up to sixteen processors. Further, in limitation [1.1.3], it was shown that each processor node of the IBM SP2 has hard drives and up to 64 GB memory for storing program

code accessible by the processor nodes. A processor node of the MultiMATLAB architecture other than the interactive processor (*first node*) of FIG. 1 discussed in limitation [1.3.1], such as processor ID = 1, teaches *a second node comprising a second hardware processor*. FIG. 1 below shows an example *second node comprising a second hardware processor*:

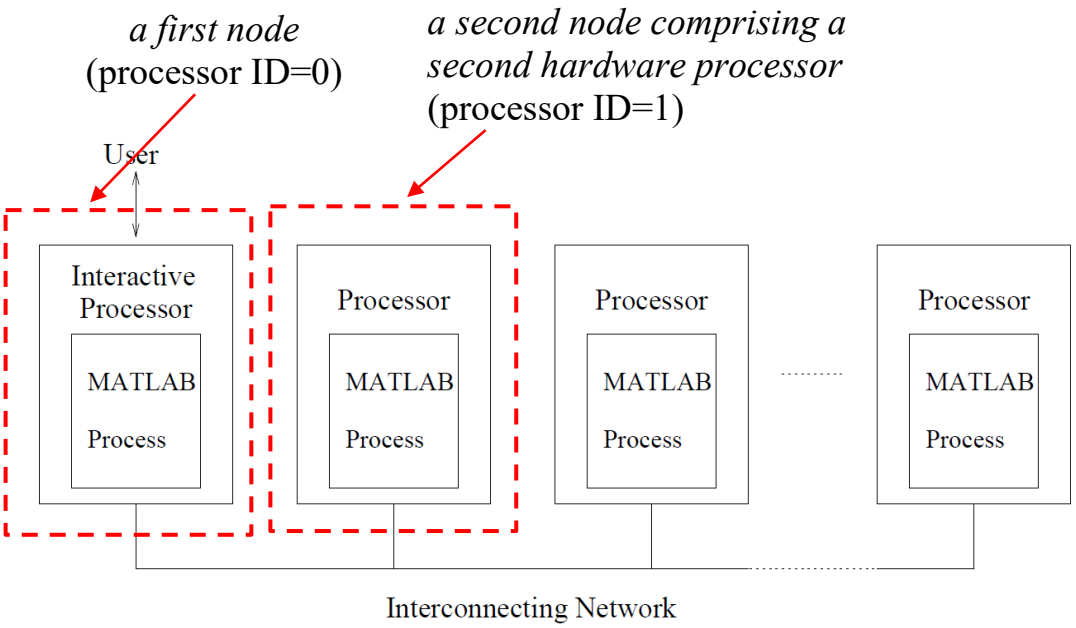



Figure 1: MultiMATLAB Architecture

**Ex.1005, FIG. 1 (annotated).**

**185.** RS6000 also teaches that each of the nodes in the IBM SP2 has "a Symmetric MultiProcessor (SMP)" ("*hardware processor*"). Ex.1007, 2. The SMP includes "four, eight, twelve, or sixteen 375 MHz 630FP 64-bit processors" (each having a core). Ex.1007, 2, 9. Thus, each node has *hardware processor with a plurality of processing cores*.

**186.** Accordingly, Menon discloses executing a series of parallel processes on 32 processors of the IBM SP2 system and RS6000 teaches that the IBM SP2 consists of up to 128 nodes, where each node has a symmetric multiprocessor having up to sixteen processors (each having a core), rendering obvious *a second node comprising a second hardware processor with a plurality of processing cores*.

**j. [1.4.2] *wherein the second node is configured to receive calls from the first node, execute at least a first mathematical expression evaluation, and communicate a result of the first mathematical expression evaluation to a third node;***

**187.** Menon in view of Trefethen renders obvious *wherein the second node is configured to receive calls from the first node, execute at least a first mathematical expression evaluation, and communicate a result of the first mathematical expression evaluation to a third node* because Menon teaches a MultiMATLAB architecture using point to point communication among the nodes to communicate messages and Trefethen teaches an example instruction that uses point-to-point message passing to send a call from a first node to a second node, and then send a mathematical result from the second node to a third node.

**188.** In limitation [1.2], it was shown that Menon teaches MEX routines having code that implement standard variants of MPI calls to enable SPMD point-to-point communication so that the MATLAB processes can communicate tasks and data directly with each other. Ex.1005, 9. The point-to-point communication is

established by IBM's POE, where "[w]hen you invoke **poe**, the Partition Manager allocates processor nodes for each task and initializes the local environment. It then loads your program, and reproduces your local environment, on each processor node. … If you are using the dynamic message passing interface, the appropriate communication subsystem library implementation (IP or US) is automatically loaded at this time." Ex.1008, 28 (emphasis original). "For each of *p* MATLAB processes, POE assigns an identification number between 0 and *p – 1*. The process numbered 0 is considered the interactive MATLAB. All other processes wait for commands from the interactive MATLAB process." Ex.1005, 6 (emphasis original).

**189.** When a user then "interacts directly with" the interactive MATLAB process and enters a command identifying one or more IDs of the other processors, the command is executed on the identified MATLAB processes, indicating that MATLAB processes are initialized and running on all the processors of the MultiMATLAB architecture:

> the command Eval( 'ID' ) [when entered by the user] calls the MultiMATLAB command ID on each of the processors running. This command returns the number of the current process, an integer from 0 to Nproc-1. Running it on all nodes might give the result
>
> ans = 0

ans = 1

ans = 5

ans = 2

ans = 3

ans = 4

Ex.1005, 3; Ex.1006, 3-4. Thus, Menon in view of Trefethen and POEref disclose a point-to-point communication between all the processors of the MultiMATLAB architecture, in particular between the processor (*the first node*) of the interactive MATLAB process, processor ID = 0, and the processors of any of the other MATLAB processes including *the second node* shown in FIG. 1 in limitation [1.4.1].

**190.** In limitation [1.3.2], it was shown that Trefethen discloses an example user instruction, Eval ('cycle'), where "the m-file cycle.m," instructs: (i) the interactive MATLAB process to create a variable "a" with a value and send the variable to the next MATLAB process, and (ii) the next MATLAB process to receive the variable "a" from the interactive MATLAB process, double the value of the received variable "a," and send the result to the next MATLAB process. Ex.1006, 6. That is, Trefethen teaches that processor ID =1 (*the second node*) that is next to the interactive processor (i.e., processor ID = 0) receives a command (*is configured to receive calls*) from the processor ID = 0 (*from the first node*) of the

interactive MATLAB that the user is interacting with to receive and double the variable "a" (*execute at least a first mathematical expression evaluation*), and send the result of the mathematical operation doubling the value of the received variable (*and communicate a result of the first mathematical expression evaluation*) to the next MATLAB process at the next processor, processor ID = 2 (*to a third node*), as shown below:

```
if ID==0                    % first p
    a = 1
    Send(ID+1,a)
elseif ID == Nproc-1        % last pr
    a = 2*Recv
else                        % middle
    a = 2*Recv
    Send(ID+1,a)
end;
```

processor ID=0 (*first node*) sends a=1 (*calls*) to processor ID=1 (*second node*)

processor ID=Nproc-1 (last *node*) receives a (*receives calls*) and doubles it (*execute at least a first mathematical expression evaluation*)

Process 0 creates the variable a with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of a, and sends it along to process 2; and so on. If there are six processors the command `Eval( 'cycle' )` produces the output

```
a = 1
a = 2
a = 4
a = 8
a = 16
a = 32
```

processor ID=ID+1 (*second node*) sends *(calls)* a=2 (*a result of the first mathematical expression evaluation)* to processor ID=2 (*third node*)

Output of third node (a=4)

Ex.1006, 6 (annotated).

**191.** Accordingly, Menon teaching MEX routines having code that implement standard variants of MPI calls to enable SPMD point-to-point communication so that the MATLAB processes can communicate tasks and data directly with each other in view of Trefethen teaching example instructions that uses point to message passing to send calls from a first node that are received by the

second node, where the second node sends a mathematical result to a third node, renders obvious *wherein the second node is configured to receive calls from the first node, execute at least a first mathematical expression evaluation, and communicate a result of the first mathematical expression evaluation to a third node.*

#### k. [1.5.1] *wherein the third node comprises a third hardware processor with a plurality of processing cores,*

**192.** Menon in view of RS6000 render obvious this element. Specifically, Menon discloses executing a series of parallel processes on 32 processors of the IBM SP2 system and RS6000 teaches that the IBM SP2 consists of up to 128 nodes, where each node has a symmetric multiprocessor having up to sixteen processors (each having a core), rendering obvious *a third node comprising a third hardware processor with a plurality of processing cores*.

**193.** In limitation [1.4.1], it was shown that each node of the IBM SP2 has hard drives, up to 64 GB memory for storing program code accessible by the nodes, and a symmetric multiprocessor having "four, eight, twelve, or sixteen 375 MHz 630FP 64-bit processors," each having a core. Ex.1007, 9. A non-interactive processor of the MultiMATLAB architecture (i.e., a processor other than the interactive processor (*first node*)) of FIG. 1 discussed in limitation [1.3.1] then teaches *the third node comprising a third hardware processor having a plurality of processing cores*.

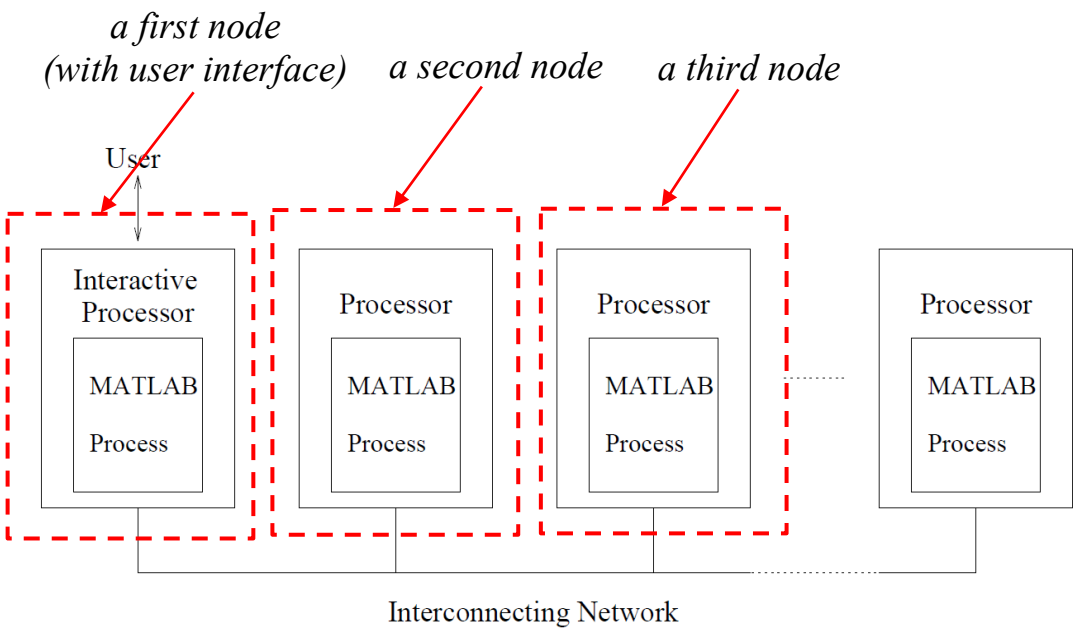



Figure 1: MultiMATLAB Architecture

**Ex.1005, FIG. 1 (annotated).**

**l. [1.5.2] *wherein the third node is configured to receive the result of the first mathematical expression evaluation from the second node, execute at least a second mathematical expression evaluation using the received result, and communicate the result of the second mathematical expression evaluation to the first node;***

**194.** Menon in view of Trefethen render obvious this element. Specifically, Menon teaches a MultiMATLAB architecture using point to point communication among the nodes to communicate messages and Trefethen teaches example instructions that use point-to-point message passing to send a mathematical result from the second node to a third node, which performs a mathematical calculation, and then sends the result back to the first node, thus rendering obvious *wherein the*

*third node is configured to receive the result of the first mathematical expression evaluation from the second node, execute at least a second mathematical expression evaluation using the received result, and communicate the result of the second mathematical expression evaluation to the first node*.

**195.** In limitation [1.4.2], it was shown that Trefethen teaches the MATLAB process at processor ID = 1 (*the second node*) using its processor to send the result of a mathematical operation doubling the value of a received variable (*result of the first mathematical expression evaluation*) to the next processor, processor ID = 2 (*a third node*). That is, processor ID = 2 (*a third node*) receives (*is configured to receive*) a variable that is doubled by the MATLAB process at processor ID = 1 (*the result of the first mathematical expression evaluation*) from processor ID = 1 (*from the second node*).

**196.** Trefethen also teaches that the MATLAB process at processor ID = 2, after receiving the doubled variable from processor ID = 1, uses its processor to double the received variable, as shown below:

```
if ID==0                    % fir
   a = 1
   Send(ID+1,a)
elseif ID == Nproc-1        % las
   a = 2*Recv
else                        % mid
   a = 2*Recv
   Send(ID+1,a)
end;
```

processor ID=2 (*third node*) receives the variable a doubled by processor ID=1 (*a result of the first mathematical expression evaluation*) from processor ID=1 (*second node*) and doubles it (*execute at least a second mathematical expression evaluation using the received result*)

Process 0 creates the variable a with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of a, and sends it along to process 2; and so on. If there are six processors the command `Eval( 'cycle' )` produces the output

```
a = 1
a = 2
a = 4
a = 8
a = 16
a = 32
```

Output of third node (a=4)

Output of final result (a=32)

Ex.1006, 6 (annotated). That is, processor ID=2 (*the third node*) receives the variable a that was doubled by processor ID=1 (*a result of the first mathematical expression evaluation*) from processor ID=1 (*the first node*), and the MATLAB process at processor ID=2 doubles the received variable (*execute at least a second mathematical expression evaluation using the received result*).

**197.** Trefethen further discloses that each MATLAB process transmits its output to the interactive processor (processor ID=0) as soon as a computation is completed, indicating that the MATLAB process uses its processor ID=2 to send the result of its variable doubling operation to the interactive processor "as soon as [the result] is ready." Ex.1006, 4. Specifically, Trefethen discloses that running the command Eval( 'ID' ) on all six nodes "give[s] the result

ans = 0

ans = 1

ans = 5

ans = 2

ans = 3

ans = 4

The ordering of these numbers is arbitrary, since the processors are not synchronized and **output is sent to the master process as soon as it is ready**." Ex.1006, 4. In other words, as soon as a MATLAB process at a processor completes its computation, it uses its processor to send the result of the computation to the interactive processor. A POSITA would recognize that the "master process" described by Trefethen would be the interactive process of Menon, because the interactive process is the MATLAB process in the MultiMATLAB architecture with which the user interacts.[2] Ex.1005, 3. As a specific example, as soon as the MATLAB process at processor ID=2 completes the operation of doubling the

[2] In the context of SPMD point-to-point communication, a "master process" is a "pseudo-master [that] is arbitrarily chosen to be the task [that] all the peers know a priori who to send the results back to." Ex.1011, 64. That is, here, the "master" process is the process with which the user is directly interacting in a peer-to-peer paradigm, and not a "master" in a master/slave paradigm.

variable that it has received from processor ID=1, the MATLAB process uses that processor ID=2 (*third node*) to send (*communicate*) the variable that it has received and then doubled (*the result of the second mathematical expression evaluation*) to the interactive processor (*to the first node*).

**198.** Further, Menon "describe[s] different MultiMATLAB implementations of the conjugate gradients algorithm on the IBM SP2," where the algorithm is an "iterative method to solve a linear system of the form $Ax = b$, where $A$ is a symmetric positive definite matrix and $x$ and $b$ are vectors. The computational core of this method is a single matrix-vector multiplication at each iteration." Ex.1005, 11. The MultiMATLAB architecture implementation of the conjugate gradients algorithm is shown below in FIG. 4 of Menon below:

```
r = b;
rho = r'*r;
k = 0;
while((k < kmax))
  k = k+1;
  if (k == 1)
    p = r;
  else
    beta = rho/oldrho;
    p = r + beta*p;
  end

  w = A * p;

  alpha = rho/(p'*w);
  x = x + alpha * p;
  r = r - alpha * w;
  oldrho = rho;
  rho = r'*r;
end
```

a. Sequential MATLAB

MultiMATLAB implementation of the conjugate gradients algorithm

```
r_local = b_local;
rho = Sum(r_local'*r_local);
k = 0;
while((k < kmax))
  k = k+1;
  if (k == 1)
    p_local = r_local;
  else
    beta = rho/oldrho;
    p_local = r_local + beta*p_local;
  end

  p = Gather(p_local);
  w_local = A_local * p;

  alpha = rho/(Sum(p_local'*w_local));
  x_local = x_local + alpha * p_local;
  r_local = r_local - alpha * w_local;
  oldrho = rho;
  rho = Sum(r_local'*r_local);
end
```

b. Message Passing MATLAB

Figure 4: Conjugate Gradients

**Ex.1005, FIG. 4 (annotated)**

**199.** Menon teaches that, during the execution of the conjugate gradient algorithm, "the matrix *A* and the different vectors are each distributed by row over all processes as in Figure 6," shown below:

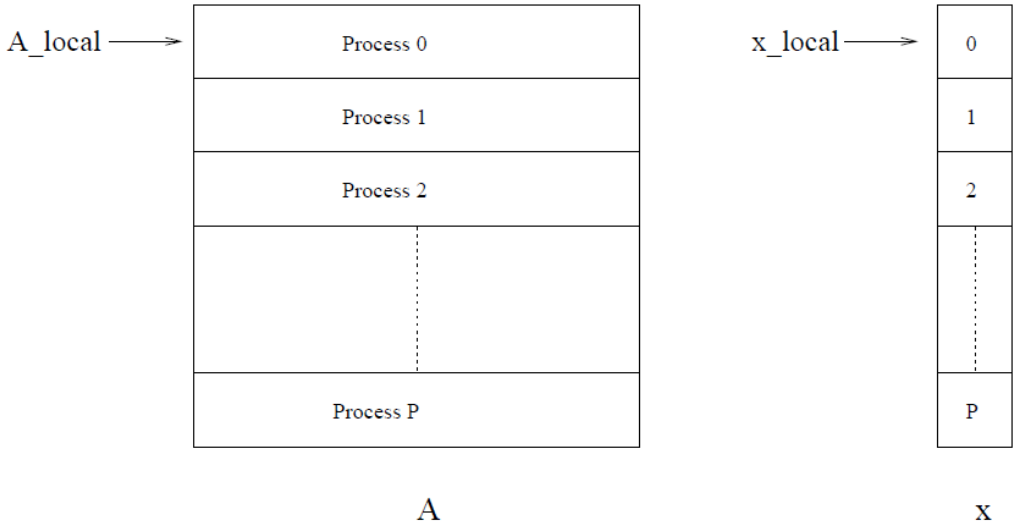



Figure 6: Matrix $A$ and vector $x$ distributed by row over $P$ processors

**Ex.1005, FIG. 6**

Ex.1005, 11.

**200.** The communication between the processors of the MultiMATLAB architecture then includes "communication of a single value over all processes" (for the processors to perform "the dot-products needed to compute *rho* and *alpha*," which "require communication of a single value over all processes") and "expensive communication" involving a "global vector *p*" (for the processors to perform the "the matrix-vector multiply needed to compute *w_local*," which "requires the global vector *p*"). Ex.1005, 11-13 (emphasis original).

**201.** Accordingly, Menon teaching a point to point computing system for passing messages among MATLAB nodes with identification numbers in view of Trefethen teaching passing mathematical results from a second MATLAB node that are received by a third MATLAB node, which perform a seconds calculation, and that, when the output is ready, sends the output to the interactive MATLAB process (*first node*), renders obvious *wherein the third node is configured to receive the result of the first mathematical expression evaluation from the second node, execute at least a second mathematical expression evaluation using the received result, and communicate the result of the second mathematical expression evaluation to the first node.*

**m. [1.6] *wherein the first node is configured to return the result of the second mathematical expression evaluation to the user interface;***

**202.** Menon in view of Trefethen render obvious this element. Specifically, Menon teaches an interactive MATLAB process at an interactive processor that allows a user to directly interact with, and operate within, the MATLAB process environment, and Trefethen teaches that the progress of computations on several processors of the MultiMATLAB architecture are displayed on the user's screen as soon as the results are ready, thus rendering obvious *wherein the first node is configured to return the result of the second mathematical expression evaluation to the user interface*.

**203.** Menon discloses that "[t]he user interacts directly with one MATLAB process, called the interactive process, and operates within that process's MATLAB environment." Ex.1005, 3. An example of said interaction is taught by Trefethen, which teaches "produc[ing] plots in a distributed fashion that are then sent to the user's screen. This can be particularly useful when one wishes to monitor the progress of computations on several processors graphically." Ex.1006, 7. In particular, Trefethen teaches "set[ting] up a MATLAB figure window in each process and arrang[ing] them in a grid on the screen. This is easily done using standard MATLAB handle graphics commands." Ex.1006, 7. In other words, "the progress of computations on several processors," such as the results of computations performed by the processors of the MultiMATLAB architecture and returned to the interactive processor, are presented to the user at the user's screen. Ex.1006, 7.

**204.** In limitation [1.5.2], it was shown that as soon as the MATLAB process at processor ID=2 completes the operation of doubling the variable that it has received from processor ID=1, it uses its processor ID=2 (*third node*) to send the variable that it has received and then doubled (*the result of the second mathematical expression evaluation*) to the interactive processor (*to the first node*). Then, Menon and Trefethen render it obvious that the variable (*the result of the second mathematical expression evaluation*) that the interactive processor received

from processor ID=2 is displayed on the user's screen. Therefore, Menon and Trefethen render it obvious that the interactive processor (*the first node*) displays (*is configured to return*) the received variable (*the result of the second mathematical expression evaluation*) on the user's screen (*to the user interface*).

**n. [1.7.0] *wherein one or more of the nodes are configured to:***

**205.** Limitation [1.7.0] is disclosed or rendered obvious by Menon in view of RS6000 for the same reasons presented above for limitation [1.1.1].

**o. [1.7.1] *accept user instructions;***

**206.** Menon in view of Trefethen render obvious this element. Specifically, Menon teaches that the interactive processor (*one or more of the nodes*) executes the interactive MATLAB process, which includes a user interface for receiving *user instructions*, and Trefethen teaches that the interactive MATLAB process receives instructions input via an m-file into MATLAB, rendering obvious *wherein one or more of the nodes are configured to … accept user instructions*.

**207.** In limitation [1.3.1], it was shown that Menon discloses the interactive processor of the MultiMATLAB architecture has a MATLAB process with MATLAB that provides a toolkit with a graphical interface to allow users to provide equations. FIG. 1 of Menon below shows the MATLAB process at the interactive processor of the MultiMATLAB architecture.

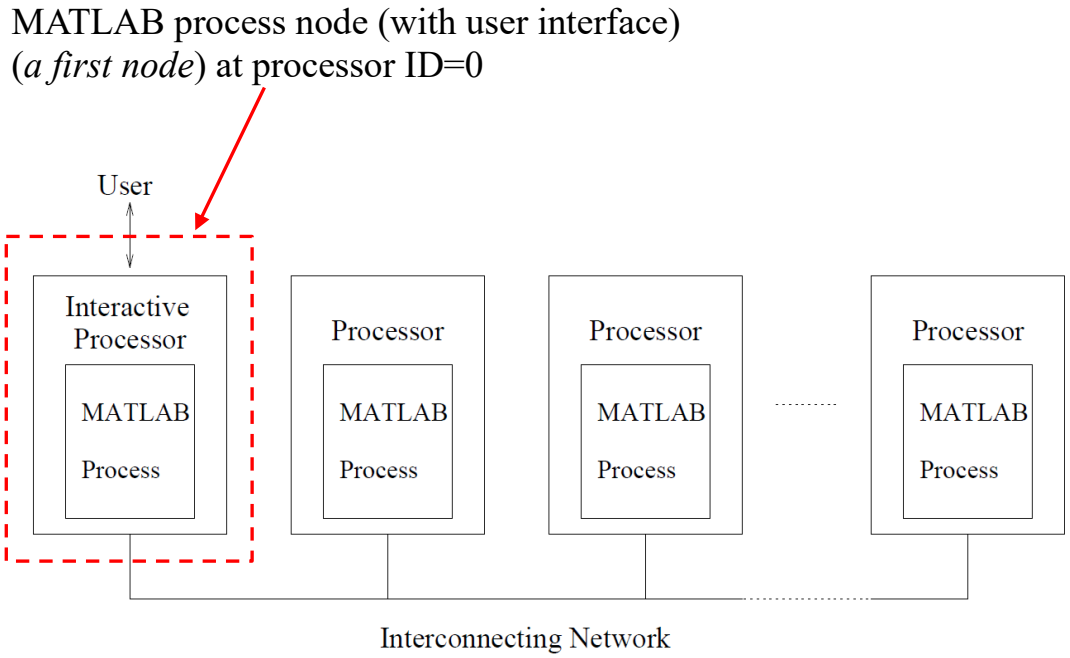



Figure 1: MultiMATLAB Architecture

**Ex.1005, FIG. 1 (annotated)**

**208.** Menon explains that users can provide the graphical interface "geometries, differential equations, and boundary conditions." Ex.1005, 6. For example, users can provide mathematical formulas to calculate a conjugate gradient, as shown below in FIG. 4 of Menon:

```
r = b;
rho = r'*r;
k = 0;
while((k < kmax))
  k = k+1;
  if (k == 1)
    p = r;
  else
    beta = rho/oldrho;
    p = r + beta*p;
  end

  w = A * p;

  alpha = rho/(p'*w);
  x = x + alpha * p;
  r = r - alpha * w;
  oldrho = rho;
  rho = r'*r;
end
```

a. Sequential MATLAB

User instructions to perform mathematical expression evaluation

```
r_local = b_local;
rho = Sum(r_local'*r_local);
k = 0;
while((k < kmax))
  k = k+1;
  if (k == 1)
    p_local = r_local;
  else
    beta = rho/oldrho;
    p_local = r_local + beta*p_local;
  end

  p = Gather(p_local);
  w_local = A_local * p;

  alpha = rho/(Sum(p_local'*w_local));
  x_local = x_local + alpha * p_local;
  r_local = r_local - alpha * w_local;
  oldrho = rho;
  rho = Sum(r_local'*r_local);
end
```

b. Message Passing MATLAB

Figure 4: Conjugate Gradients

**Ex.1005, FIG. 4 (annotated)**

**209.** Further, in limitation [1.3.2], it was shown that a user can provide *user instructions* via an "m-file." Trefethen teaches that m-files are programs of the MATLAB programming language that has its origins in the attempt "to provide interactive access to the Fortran linear algebra software packages EISPACK and LINPACK," but soon became "a programming language … containing dozens of high-level commands such as svd (singular value decomposition), fft (fast Fourier transform), and roots (polynomial zerofinding)," where "programs conventionally have the extension .m and are called "m-files." Ex.1006, 2. Trefethen explains that "for most applications, … a user will want to execute a program (an m-file) rather than a single line of text. A command such as Eval ( 'filename' ) achieves this effect." Ex.1006, 5.

**210.** Accordingly, Menon's teaching of the interactive processor (*one or more of the nodes*) executing the interactive MATLAB process, which includes a user interface for receiving *user instructions*, in view of Trefethen's teaching of the interactive MATLAB process receiving instructions input via an m-file into MATLAB, renders obvious *where wherein one or more of the nodes are configured to … accept user instructions*.

**p. [1.7.2] *after accepting user instructions, communicate at least some of the user instructions using the mechanism for the nodes to communicate with each other; and***

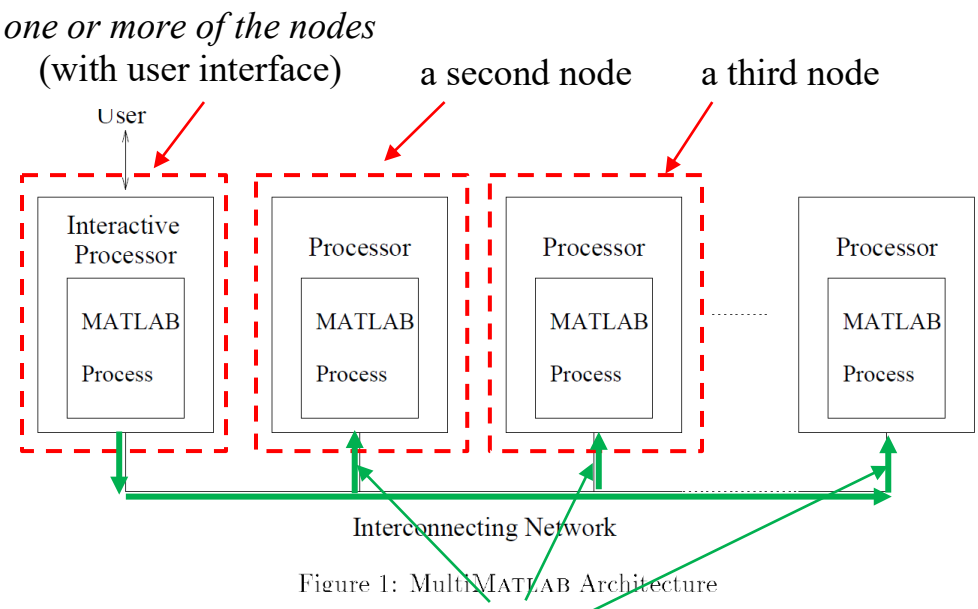
**211.** Menon in view of Trefethen render obvious this element. Specifically, Menon and Trefethen describe that (i) the MATLAB process at the interactive processor receives the user instructions including IDs identifying processors in the MultiMATLAB architecture and (ii) those instructions are communicated to the other MATLAB processes at the processors corresponding to the IDs specified in the user instructions using MEX routines having code that implement standard variants of MPI calls so that the MATLAB processes can communicate tasks and data directly with each other, rendering obvious *after accepting user instructions, communicate at least some of the user instructions using the mechanism for the nodes to communicate with each other*.

**212.** In limitations [1.3.2] and [1.7.1], it was shown that *user instructions* can be provided via m-files that assign different actions for different MATLAB

processes. Further, in limitation [1.7.1], it was shown that the interactive MATLAB process at the interactive processor receiving instructions input via an m-file into MATLAB renders obvious *wherein one or more of the nodes are configured to accept[] user instructions*.

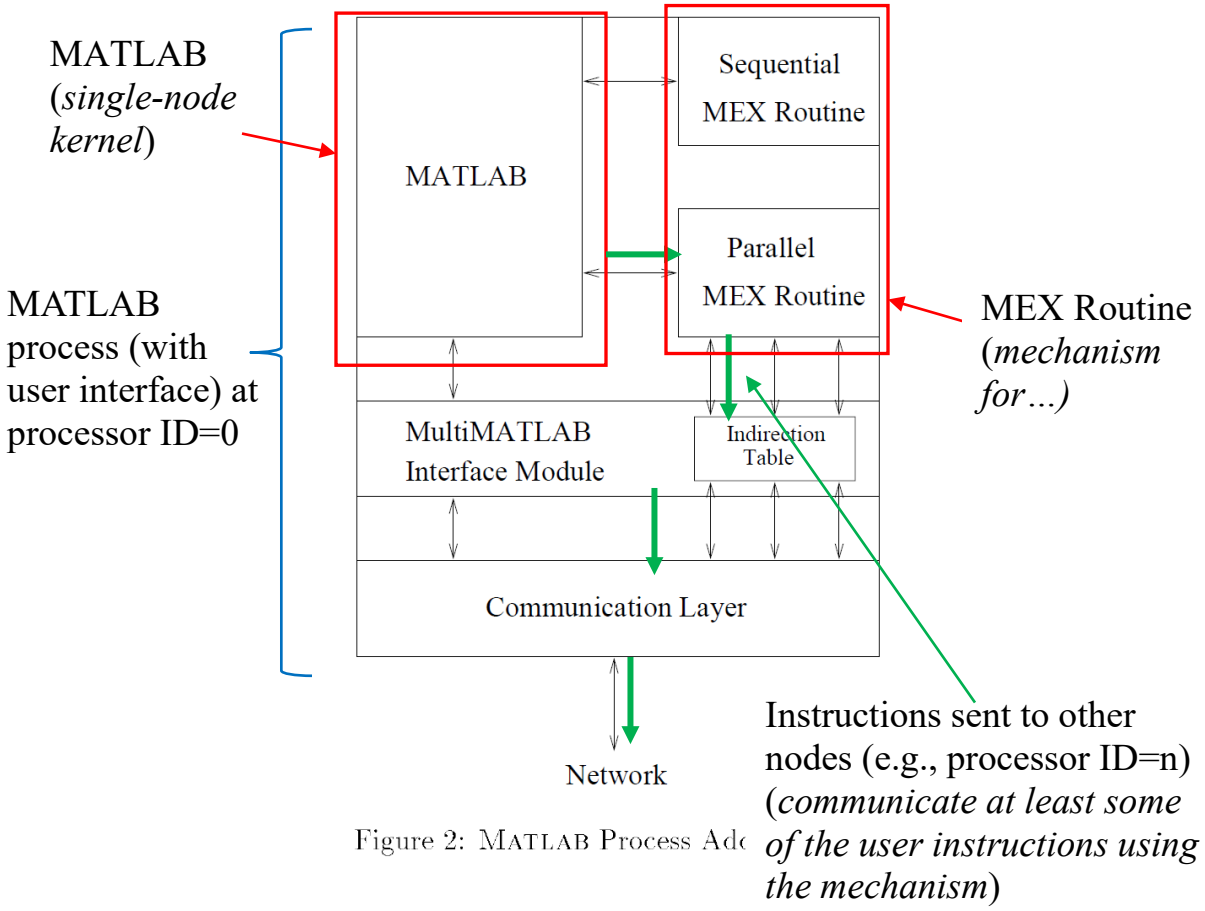
**213.** In limitation [1.2], it was shown that Menon teaches MEX routines having code that implement standard variants of MPI calls so that the MATLAB processes can communicate tasks and data directly with each other, rendering obvious *a mechanism for the nodes to communicate with each other*.

**214.** Further, Menon describes how the interactive processor (e.g., processor ID=0) sends commands to the other MATLAB processes, stating that the Eval routine "is implemented as a MEX routine which accepts a vector of MATLAB process IDs and a MATLAB command. This routine sends the commands over the underlying communication layer to the processes corresponding to the specified IDs." Ex.1005, 4. The "communication layer [] runs over the parallel platform's interconnection network." Ex.1005, 3. FIG. 1 and FIG. 2 of Menon shown below illustrate the interactive processor sending commands to the MATLAB processes on the other processors of the MultiMATLAB architecture over the communication layer.




The interactive processor (*one or more of the nodes are configured to*) transmits instructions to the other processors via the communication layer running over the interconnection network (*communicate at least some of the user instructions using the mechanism*)

**Ex.1005, FIG. 1 (annotated)**




**Ex.1005, FIG. 2 (annotated)**

**215.** Accordingly, Menon in view of Trefethen describes that (i) the interactive processor receives the user instructions including IDs identifying MATLAB processes in the MultiMATLAB architecture and (ii) those instructions are evaluated in order to send commands to the other MATLAB processes corresponding to the IDs specified in the user instructions using MEX routines having code that implement standard variants of MPI calls so that the MATLAB processes can communicate tasks and data directly with each other, thereby rendering obvious *after accepting user instructions, communicate at least some of*

*the user instructions using the mechanism for the nodes to communicate with each other*.

**q. [1.7.3] *after communicating at least some of the user instructions using the mechanism, communicate at least some of the user instructions to one or more single-node kernels.***

**216.** Menon in view of Trefethen render obvious this element. Specifically, Menon and Trefethen describe that the commands are communicated using the MEX routines from the interactive processor (ID=0), and once received by each non-interactive node (processor ID=n), are then communicated to MATLAB (*single-node kernel*) for processing, thereby rendering obvious *after communicating at least some of the user instructions using the mechanism, communicate at least some of the user instructions to one or more single-node kernels*.

**217.** In limitation [1.7.2] above, it was shown that Menon and Trefethen describe that (i) the MATLAB process at the interactive processor receives *user instructions* including IDs identifying MATLAB processes in the MultiMATLAB architecture and (ii) those instructions are evaluated in order to send commands to the other MATLAB processes corresponding to the IDs specified in the user instructions using MEX routines having code that implement standard variants of MPI calls so that the MATLAB processes can communicate tasks and data directly with each other, rendering obvious *communicating at least some of the user instructions using the mechanism*. Menon explains that the commands received at

the other MATLAB processes (referred to as "non-interactive MATLAB processes") are processed by MATLAB (*one or more single-node kernels*), stating that "[o]n non-interactive processes … [a] MEX routine runs in a loop in which it waits for MATLAB commands to arrive from the interactive process and executes them in its own MATLAB environment." Ex.1005, 4.

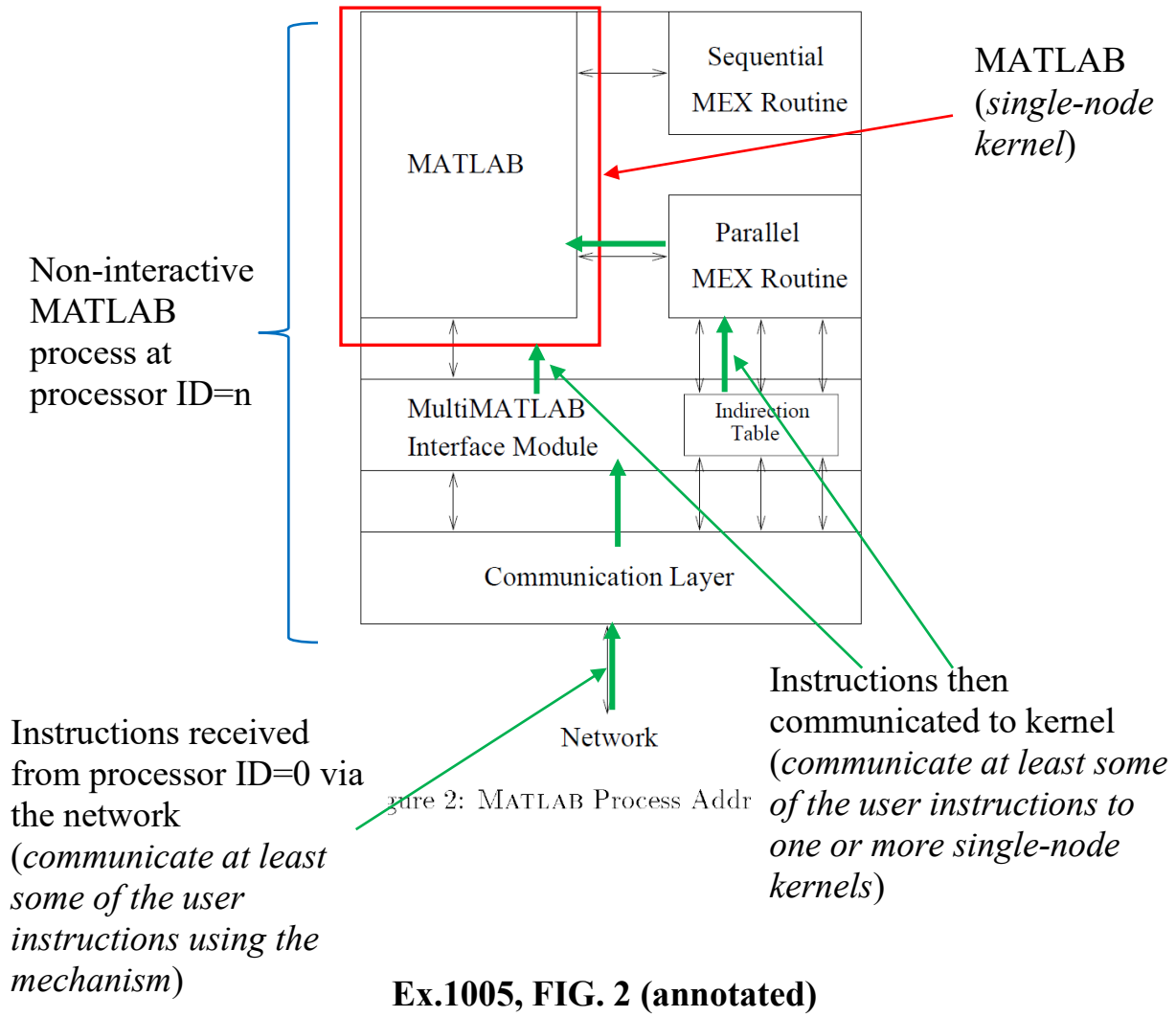
**218.** Trefethen provides example illustrations showing that commands sent to the other or non-interactive MATLAB processes corresponding to the IDs specified in the *user instructions* are processed by MATLAB at the respective MATLAB processes identified by the IDs. Ex.1006, 3-7. For example, as discussed in limitation [1.3.2], when the user provides the *user instructions* Eval( [4 5] , 'cond(hilb(ID))' ) to the interactive MATLAB process. Then, in accordance with that instruction, the calls "cond(hilb(ID))" are then distributed to the MATLAB processes on processor ID = 4 and ID = 5 for each to perform that call, i.e., MATLAB of the MATLAB process at processor ID=4 calculates cond(hilb(ID=4)) and MATLAB of the MATLAB process at processor ID=5 calculates cond(hilb(ID=5)), generating the results:

"ans = 1.5514e+04

ans = 4.7661e+05,

the condition numbers of the Hilbert matrices of dimensions 4 and 5." Ex.1006, 4.

FIG. 2 of Menon below shows MATLAB (*one or more single-node kernels*) of a non-interactive MATLAB process that executes the received instructions.




**Ex.1005, FIG. 2 (annotated)**

**219.** Accordingly, Menon and Trefethen describe that the commands are communicated using the MEX routines from the interactive processor (ID=0), and once received by each non-interactive node (processor ID=n), are communicated to MATLAB (*single-node kernel*) for processing, thereby rendering obvious *after*

*communicating at least some of the user instructions using the mechanism, communicate at least some of the user instructions to one or more single-node kernels*.

**5. Claim 2**

**a. [2.0] *The computer cluster of claim 1, wherein the single-node kernel comprises a Mathematica kernel.***

**220.** Menon discloses *wherein the single-node kernel comprises a Mathematica kernel* because it teaches a general architecture that is a "promising design" for a parallel numerical computing environment.

**221.** Menon discloses that "the MultiMATLAB architecture [is] a promising design for a parallel numerical computing environment." Ex.1005, 16. POSITAs knew and recognized that Mathematica, just like MATLAB, was a "mathematical software package" that "allow[ed] scientists and engineers to perform complex analysis and modeling in their familiar workstation environment." Ex.1013, 1. It would have been obvious to apply the architecture proposed by Menon to other numerical computing environments, such as Mathematica, since Mathematica shared the same issue as MATLAB – it operated in a single-node environment and did not provide parallel computing enhancements. Applying Menon's solution of a multiprocessor layer that worked with MATLAB's core functionality, without modifying MATLAB, to Mathematica would have predictable result of a parallel

numerical computing environment envisioned in Menon, but using the mathematical features of Mathematica.

**222.** Further, POSITAs would have been motivated to apply Menon's solution to other numerical computing environments, such as Mathematica, because there was a recognition and interest at the time for "combining the user-friendliness and interactivity of the commercially-available mathematical packages [such as Mathematica, Matlab, HiQ and others] with the computing power of the parallel machines." Ex.1013, 1. Further, Trefethen explains that "[a]ll of this [applying the MultiMATLAB architecture to MATLAB] happens without any changes in the MATLAB architecture; indeed, we have not had access to the MATLAB source code," indicating that the architecture can be applied to other numerical computing environments, such as Mathematica. Ex.1006, 12. A POSITA would have had a reasonable expectation of success because Menon describes "the MultiMATLAB architecture as a promising design for a parallel numerical computing environment," which is what Mathematica is. Ex.1014, 332 (describing Mathematica as one of "very popular technical computing environments").

### 6. Claim 3

#### a. [3.0] *The computer cluster of claim 1, wherein the mechanism comprises a message passing interface.*

**223.** Menon and Trefethen render obvious *wherein the mechanism comprises a message passing interface* because they describe using MEX routines built upon the message passing interface (MPI).

**224.** As discussed in [1.2], Menon renders obvious a *mechanism for…* because it discloses MEX routines having code that implement standard variants of MPI calls to enable SPMD point-to-point communication so that the MATLAB processes can communicate tasks and data directly with each other.

**225.** The '768 patent describes using "Message Passing Interface ('MPI') calls directly from within a user interface." Ex.1001, 2:24-26. Similarly, Menon describes using "MPI as [the] underlying communication substrate" for its implementation of the MultiMATLAB architecture "due [to] its broad popularity, its suitability to high performance parallel computing, and the virtually universal availability of vendor-optimized versions on modern parallel computers." Ex.1005, 5. One implementation is "designed for the IBM SP2, a modern high performance distributed memory multiprocessor. This implementation is based upon MPI-F [], IBM's proprietary version of MPI specifically optimized for the SP2." Ex.1005, 5. The other implementation "is based upon MPICH [], a popular public domain version of MPI." Ex.1005, 5.

**226.** Accordingly, Menon and Trefethen describe using MEX routines built upon the message passing interface (MPI), rendering obvious *wherein the mechanism comprises a message passing interface*.

**7. Claim 4**

**a. [4.0] *The computer cluster of claim 1, wherein each of the nodes comprises one or more cluster node modules.***

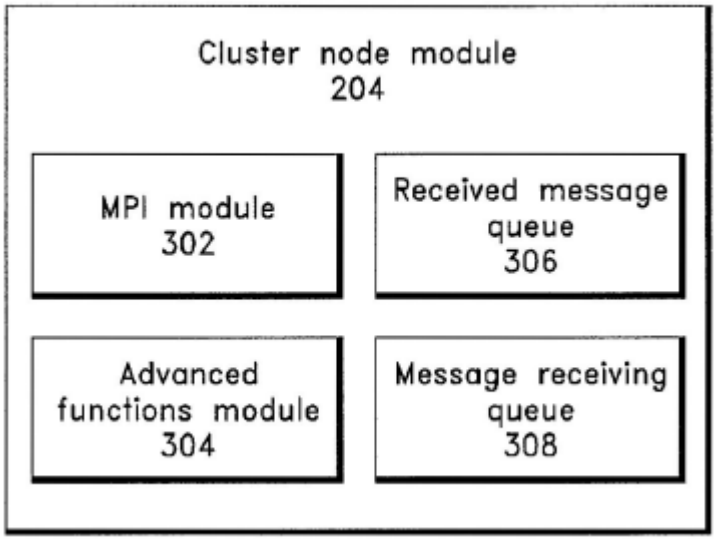
**227.** Menon discloses *wherein each of the nodes comprises one or more cluster node modules* because it teaches each processor of the MultiMATLAB architecture having MEX Routines, a MultiMATLAB interface module, an indirection table, and a communication layer.

**228.** The ’768 patent describes a cluster node module as follows:

> the cluster node modules 204a-e provide a way for many kernel modules 206a-e such as, for example, Mathematica kernels, running on a computer cluster 100 to communicate with one another. A cluster node module 204 can include at least a portion of an application programming interface (“API”) known as the Message-Passing Interface (“MPI”), which is used in several supercomputer and cluster installations. A network of connections (for example, the arrows shown in FIG. 2) between the cluster node modules 204a-e can be implemented using a communications network 102, such as, for example, TCP/IP over Ethernet, but the

> connections could also occur over any other type of network or local computer bus.

Ex.1001, 11:40-54. Fig. 3 depicts a single cluster node module comprising multiple sub-components:




**Ex.1001, FIG. 3**

**229.** FIG. 2 of the ’768 patent depicting the cluster node modules 204a-e is shown below.

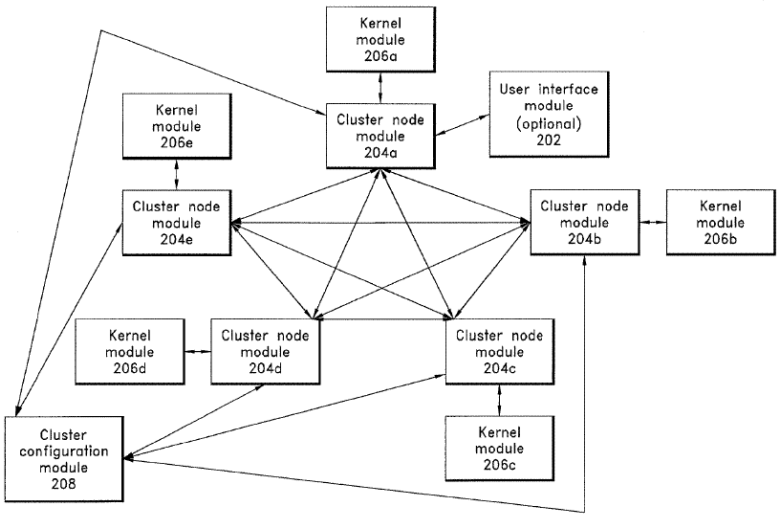



*FIG. 2*

**Ex.1001, FIG. 2**

**230.** Similarly, Menon discloses that each processor of "the MultiMATLAB architecture … individually runs a MATLAB process" that includes MEX Routines, a set of components to provide interconnection with other modules, a "MultiMATLAB interface module," a "key component in [the MultiMATLAB] architecture," along with an "indirection table" and "communication layer." Ex.1005, 3, 5. The "MEX routines" provide "[a]ll parallel functionality in the MultiMATLAB system." Ex.1005, 3. "The interface module, shown in Figure 2, is responsible for initializing an underlying communication layer, such as MPI …, or any other package available on the platform, and exposing it to the rest of the system." Ex.1005, 3. A POSITA would recognize that the MEX Routines, interface module, indirection table, and communication layer correspond to a *cluster node*

*module* of the '768 patent because the '768 patent describes its cluster node module as a grouping of subcomponents, as shown in its Fig. 3 depicted above.

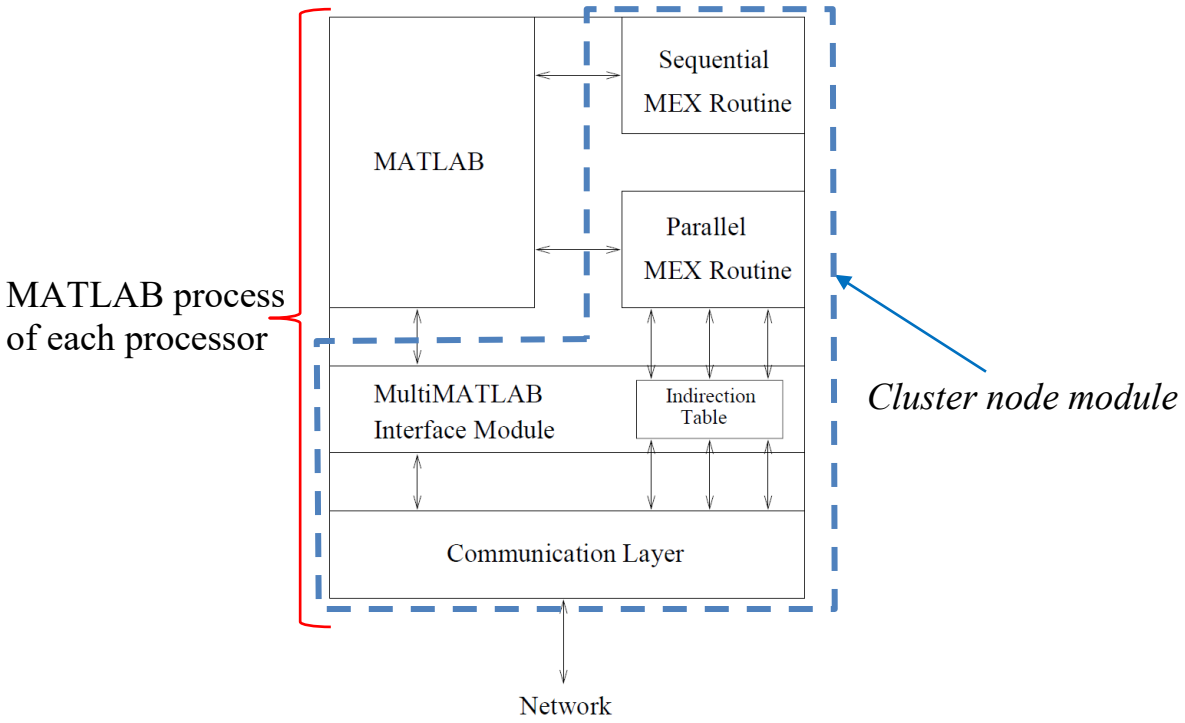
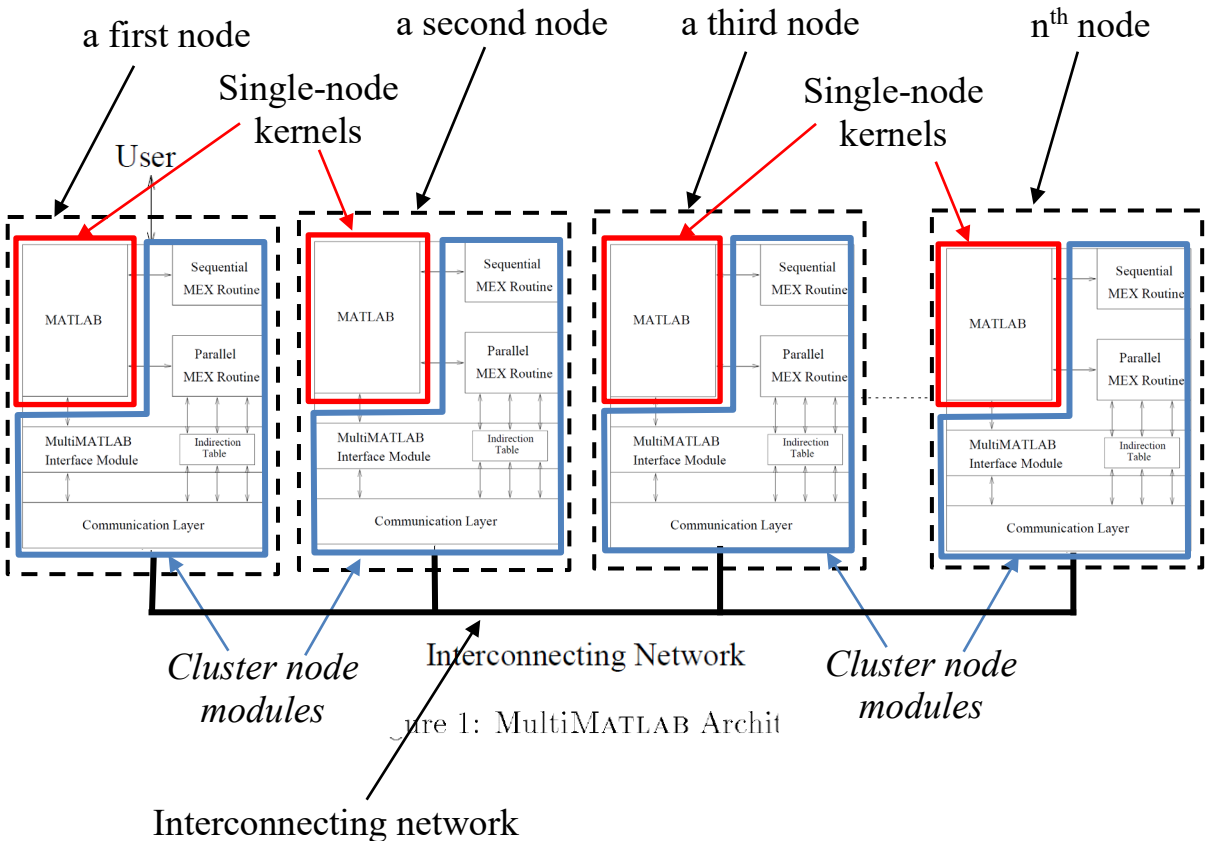



Figure 2: MATLAB Process Address Space

**Ex.1005, FIG. 2 (annotated)**

**231.** A combination of FIG. 1 and FIG. 2 of Menon below shows the relationship between the communication components (i.e., MEX routines, MultiMATLAB interface module, the communication layer), and the single-node kernel for each node of the MultiMATLAB architecture. In operation, the MEX routines (such as Send() and Recv) are run directly from the user interface. Ex.1005, 3; *see also* Ex.1006, 3, 6. The MEX routines (such as Send() and Recv() are high-level calls that are mapped (using the indirection table) to the analogous

MPI calls (such as MPI_SEND() and MPI_RECV()) found in the communication layer. Ex.1005, 3, 6. The communication layer, using the MPI calls, communicates over the interconnection network with the communication layer in the other nodes. Ex.1005, 3.




**Ex.1005, FIGs. 1 and 2 (combined and annotated)**

**232.** Accordingly, Menon describes each processor of the MultiMATLAB architecture having a MATLAB process, which in turn has MEX Routines, a MultiMATLAB interface module, indirection table, and a communication layer

(*cluster node module*), rendering obvious *wherein each of the nodes comprises one or more cluster node modules*.

**8. Claim 5**

**a. [5.0] *The computer cluster of claim 1, wherein each single-node kernel is stored in a non-transitory computer-readable medium and configured to accept and execute a request.***

**233.** Menon discloses *wherein each single-node kernel is stored in a non-transitory computer-readable medium and configured to accept and execute a request* because Menon teaches that each node has a single-node kernel (MATLAB) that (i) is stored in hard disk or RAM and executed by a processor and (ii) upon initialization, awaits MATLAB commands to arrive and then executes those commands in its own MATLAB environment.

**234. First**, in limitation [1.1.3], it was shown that each processor of the MultiMATLAB architecture stores MATLAB (*single-node kernel*) in memory address space (*computer-readable medium*) on either internal or external hard disk drives, or memory cards (e.g., a RAM).

**235. Second**, Menon discloses that MATLAB of a MATLAB process on each processor awaits for MATLAB commands and executes routines based on those MATLAB commands when received. Ex.1005, 3-4. Specifically, Menon teaches that to have user instructions executed on all the MATLAB processes of the MultiMATLAB architecture, a "user interacts directly with one MATLAB process,

called the interactive process, and operates within that process's MATLAB environment." Ex.1005, 3. For example, the user may use the "Eval routine for parallel evaluation," which is a "routine [that] allows users to execute commands on the other processes." Ex.1005, 4. "Other MATLAB processes await commands from the interactive process. These other processes are used either by explicitly sending MATLAB commands to them or by executing routines that, in turn, use them." Ex.1005, 3.

**236.** The parallel execution of the user's instructions is facilitated by the MEX Routines, which provide the MultiMATLAB architecture "all [its] parallel functionality." Ex.1005, 3. On the interactive MATLAB process, the user's "Eval routine is implemented as a MEX Routine which accepts a vector of MATLAB process IDs and a MATLAB command. This routine sends the commands over the underlying communication layer to the processes corresponding to the specified IDs." Ex.1005, 4. On the other non-interactive MATLAB processes, "a separate top-level MEX Routine is immediately run upon initialization of the system. This MEX Routine runs in a loop in which it waits for MATLAB commands to arrive from the interactive process and executes them in its own MATLAB environment." Ex.1005, 4. The interactive as well as the non-interactive MATLAB processes' MEX Routines use their respective MATLAB process's MultiMATLAB interface module to access the interconnection network and communicate with each other, as

"[a]ll parallel MEX Routines access the underlying communication layer and, hence, the network, through the MultiMATLAB interface module." Ex.1005, 4.

**237.** Trefethen provides an example illustration where a user "connected to a node of the IBM SP2, running MATLAB" is operating in a MultiMATLAB architecture where "6 MultiMATLAB processes [are] running." Ex.1006, 3. If the user types the command Eval( 'ID' ), the result is

ans = 0

ans = 1

ans = 5

ans = 2

ans = 3

ans = 4,

indicating that "the MultiMATLAB command ID" is executed on each MATLAB process. Ex.1006, 3. That is the case because "Eval routine is implemented as a MEX Routine which accepts a vector of MATLAB process IDs and a MATLAB command. This routine sends the commands over the underlying communication layer to the processes corresponding to the specified IDs." Ex.1005, 4. "[E]ach command passed to Eval was executed on all MATLAB processes" unless a "subset of the processes [is selected] by passing two arguments to the Eval command, the first being a vector of process IDs." Ex.1006, 4. In other words, the interactive as

well as all the non-interactive MATLAB processes of "the 6 MultiMATLAB processes" receive and execute commands thereon, generating the above result. Ex.1006, 3.

**238.** Accordingly, Menon teaches that each node has MATLAB (*a single-node kernel*) that (i) is stored in hard disk or RAM and executed by a processor and (ii) upon initialization, awaits MATLAB commands to arrive and then executes those commands in its own MATLAB environment, rendering obvious *wherein each single-node kernel is stored in a non-transitory computer-readable medium and configured to accept and execute a request*.

**9. Claim 6**

**a. [6.0] *The computer cluster of claim 5, wherein each of the nodes comprises one or more cluster node modules, wherein each of the cluster node modules comprises instructions stored in a non-transitory computer-readable medium, and wherein the instructions, when executed by the hardware processor, cause the cluster node module to communicate with the single-node kernel and with one or more other cluster node modules.***

**239.** Menon discloses *wherein each of the nodes comprises one or more cluster node modules, wherein each of the cluster node modules comprises instructions stored in a non-transitory computer-readable medium, and wherein the instructions, when executed by the hardware processor, cause the cluster node module to communicate with the single-node kernel and with one or more other cluster node modules* because it teaches a set of components (MEX routines, a

MultiMATLAB interface module, an indirection table, and a communication layer of a MATLAB process) are computer software that, when executed by the processor of the MATLAB process, communicates with MATLAB and the communications components of the other nodes.

**240. First**, in limitation [4.0], it was shown that each processor of the MultiMATLAB architecture has a MATLAB process, which in turn has a set of components for handling communications with other processors (MEX routines, a MultiMATLAB interface module, an indirection table, and a communication layer) (*cluster node module*).

**241. Second**, in limitation [1.1.3], it was shown that the MATLAB process address space on each processor is in a computer-readable medium in the form of internal or external hard disk drives, or memory cards (e.g., RAMs) that can be accessed by the processor of the MATLAB process. A POSITA would recognize that the MATLAB process, which includes the MEX routines, the MultiMATLAB interface module, the indirection table, and the communication layer (*cluster node module*), is software (*computer instructions*) stored in memory address space on either a hard drive or in RAM for execution by the processor. Ex.1005, 4 (Menon disclosing that "[i]t is important to note that the interface module exists in the corresponding MATLAB process's address space."); FIG. 2 (showing the communication layer for handling communications with other processors in the

MATLAB process address space). FIG. 2 of Menon below shows the MEX routines, the MultiMATLAB interface module, the indirection table, and the communication layer (*cluster node module*) in the MATLAB process address space.

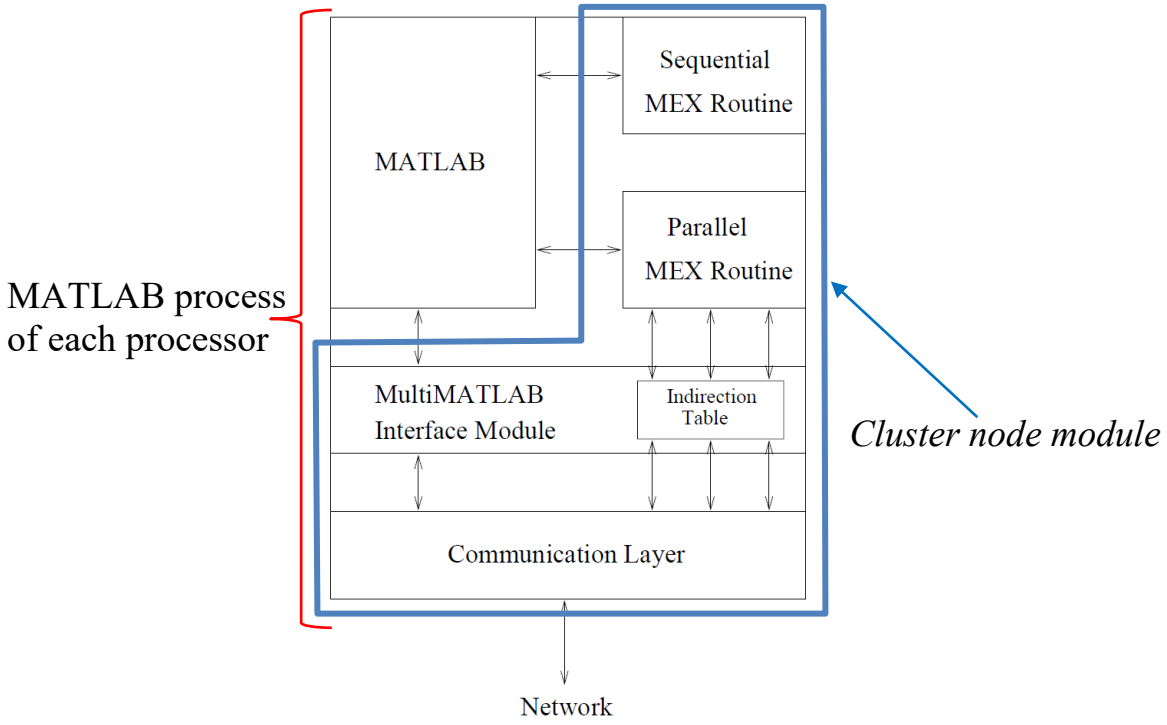
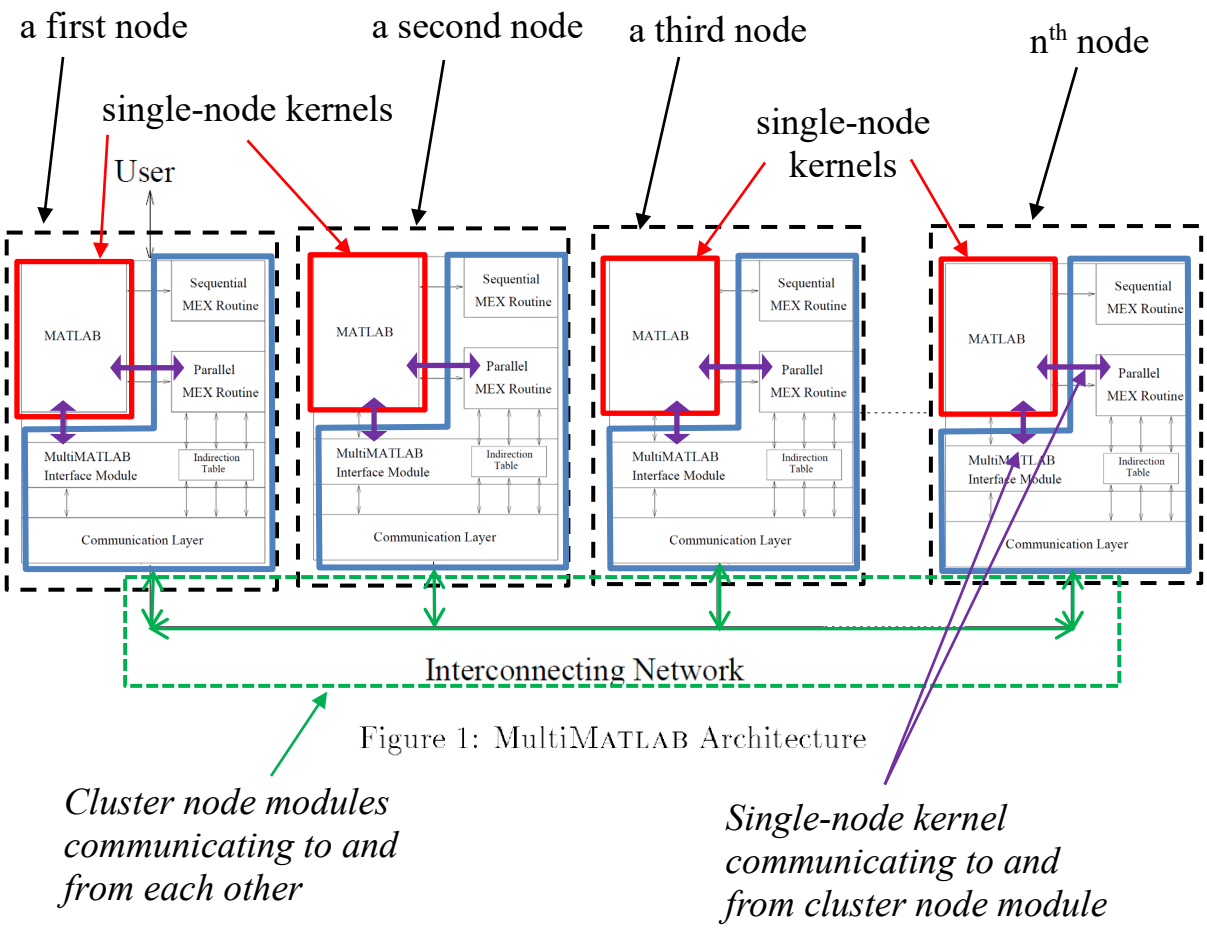



Figure 2: MATLAB Process Address Space

**Ex.1005, FIG. 2 (annotated)**

**242. Third**, Menon describes that the set of components for handling communications with other processors (MEX routines, a MultiMATLAB interface module, indirection table, and communication layer) (*cluster node module*) facilitate communication to and from the MATLAB (*single-node kernel*) and also across the network to the other nodes. Ex.1005, 3-4. Specifically, Menon discloses that "[a]ll parallel MEX routines access the underlying communication layer and,

hence, the network, through the MultiMATLAB interface module." Ex.1005, 4. Further, the combination of FIG. 1 and FIG. 2 of Menon below shows back and forth communication between MATLAB (*single-node kernel*) and MEX routines, a MultiMATLAB interface module, indirection table, and communication layer (*cluster node module*). Menon also explains that each MATLAB process on a processor of the MultiMATLAB is "provided with the ability to communicate with other processes through a communication layer that runs over the parallel platform's interconnection network," as shown in FIG. 1 of Menon below. Ex.1005, 3.




**Ex.1005, FIGs. 1 and 2 (combined and annotated)**

**243.** Accordingly, Menon describes a set of components (MEX routines, a MultiMATLAB interface module, an indirection table, and a communication layer) that (i) are stored in hard disk or RAM and executed by the processor, (ii) communicates with MATLAB (*single-node kernel*), and (iii) communicates with the communications components of the other nodes (*one or more other cluster node modules*), thereby rendering obvious *wherein each of the nodes comprises one or more cluster node modules, wherein each of the cluster node modules comprises instructions stored in a non-transitory computer-readable medium, and wherein the*

*instructions, when executed by the hardware processor, cause the cluster node module to communicate with the single-node kernel and with one or more other cluster node modules*.

**10. Claim 7**

**a. [7.0] *The computer cluster of claim 6, wherein the plurality of cluster node modules act as a cluster.***

**244.** Menon, Trefethen, and POEref renders obvious *wherein the plurality of cluster node modules act as a cluster* because Menon describes that the MEX routines, the MultiMATLAB interface module, the indirection table, and the communication layer of each MATLAB process work together to enable a user to solve a mathematical problem as if they were a single computer, POEref describes how POE executes parallel programs on a networked cluster, and Trefethen describes how the user can issue a single command to cause all the processors to run at the same time.

**245.** Menon discloses that "[i]n the MultiMATLAB architecture … each processor in a parallel platform individually runs a MATLAB process." Ex.1005, 3. An "important goal of the MultiMATLAB system is to allow users to integrate high performance parallel routines." Ex.1005, 6. A user can run commands on all the MATLAB processes by "interact[ing] directly with one MATLAB process, called the interactive process, and operat[ing] within that process's MATLAB environment," and then by "explicitly sending MATLAB commands to [the other

MATLAB processes] or by executing routines that, in turn, use them" as the "[o]ther MATLAB processes await commands from the interactive process." Ex.1005, 3. For example, the user can utilize:

> an Eval routine for parallel evaluation. This routine allows users to execute commands on the other processes. On the interactive process, the Eval routine is implemented as a MEX routine which accepts a vector of MATLAB process IDs and a MATLAB command. This routine sends the commands … to the processes corresponding to the specified IDs. On non-interactive processes, a separate top-level MEX routine … runs in a loop in which it waits for MATLAB commands to arrive from the interactive process and executes them in its own MATLAB environment.

Ex.1005, 4. "[E]ach command passed to Eval was executed on all MATLAB processes." Ex.1006, 4. FIG. 1 of Menon below shows the interactive and non-interactive MATLAB processes on the processors of the MultiMATLAB architecture that execute commands in parallel as if the processors were a single computer (*act as a cluster*) when a user provides the instructions from the interactive MATLAB process.

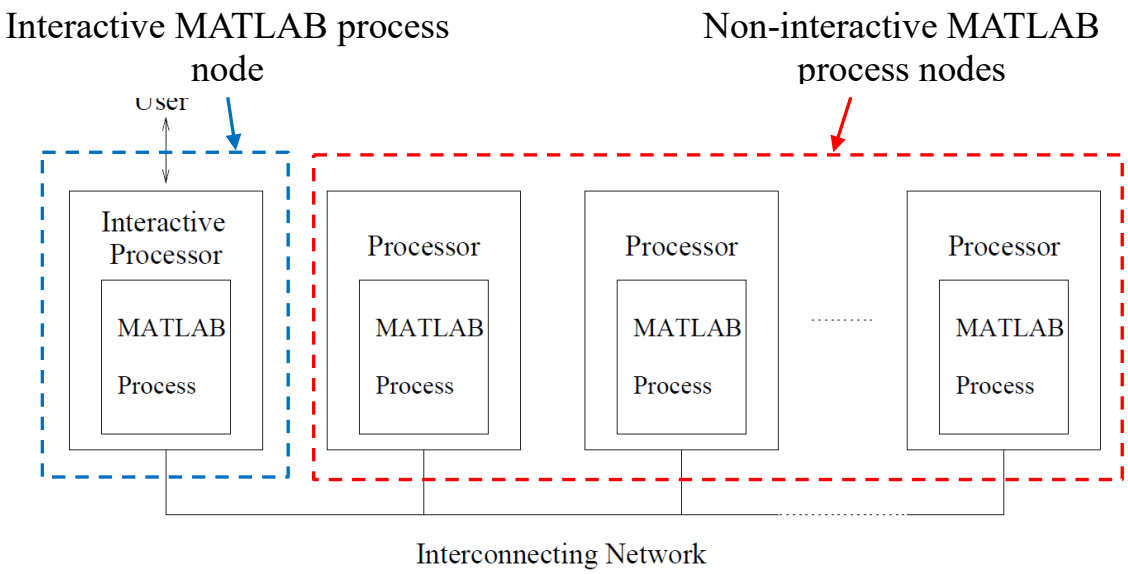



Figure 1: MultiMATLAB Architecture

**Ex.1005, FIG. 1 (annotated)**

**246.** Menon explains that the MultiMATLAB architecture can be implemented in the IBM SP2, describing a:

> MPI-F implementation of MultiMATLAB [that] was design specifically for the IBM SP2. … In this implementation, all MATLAB processes are started via POE, IBM's Parallel Operating Environment. For each of *p* MATLAB processes, POE assigns an identification number between 0 and *p-1*. The process numbered 0 is considered the interactive MATLAB. All other processes wait for commands from the interactive MATLAB process.

Ex.1005, 6 (emphasis original). IBM's Parallel Environment is used "to execute parallel programs on … a networked cluster of [IBM SP2] RS/6000 processors."

POEref, 1. That is, the interactive and non-interactive MATLAB processes of the MultiMATLAB architecture executing commands in parallel are a group of processors communicating with each other to accomplish a mathematical computation as if they were a single computer (*act as a cluster*). Ex.1005, 6; POEref, 1.

**247.** Trefethen provides several illustrative examples of a user interacting directly with an interactive MATLAB process to have MATLAB commands executed in parallel in the MATLAB environments of the interactive and the non-interactive MATLAB processes by the processors of the MultiMATLAB architecture as if the processors were a single computer (*act as a cluster*). Ex.1006, 3-7. For example, Trefethen discusses a user that is "connected to a node of the IBM SP2, running MATLAB" and operating in a MultiMATLAB architecture where "6 MultiMATLAB processes [are] running." Ex.1006, 3. "If the user now types:

> Eval( 'sqrt(2)' )
>
> then the MATLAB command sqrt(2) is executed on all six processors. The result is six repetitions of 1.4142. … On the other hand the command
>
> Eval( 'ID' )
>
> calls the MultiMATLAB command ID on each of the processors running. This command returns the number of

the current process, an integer from 0 to Nproc-1.

Running it on all nodes might give the result

ans = 0

ans = 1

ans = 5

ans = 2

ans = 3

ans = 4."

Ex.1006, 3-4. That is, when the user enters its commands at the interactive MATLAB process of the IBM SP2 node, of the 6 processors or IBM SP2 nodes that make up the MultiMATLAB architecture, "the MATLAB command sqrt(2) is executed on all six processors" and "the MultiMATLAB command ID" is executed "on each of the processors running." Ex.1006, 3-4.

**248.** Further, as discussed in limitation [6.0], Menon discloses each MATLAB process on a node has a set of components (MEX routines, a MultiMATLAB interface modules, an indirection table, and a communication layer) (*the plurality of cluster node modules*) to communicate with other nodes in the MultiMATLAB architecture. The communications can be or include mathematical results. Ex.1005, 10-13. For example, Menon describes "MultiMATLAB implementations of the conjugate gradients algorithm on the IBM SP2" so that all

the MATLAB processes are working in parallel together (*act as a cluster*). Ex.1005, 11.

**249.** Menon describes "a parallel MATLAB implementation of conjugate gradients that uses the SPMD message passing … In this case, the matrix A and the different vectors are each **distributed by row over all processes** as in Figure 6" of Menon which is also shown below. Ex.1005, 11. Menon explains that the MATLAB processes perform their computations in parallel together:

> In each iteration, a process only does the computation required for its local data. However, communication is required for two different operations: the dot-products needed to compute *rho* and *alpha* and the matrix-vector multiply needed to compute *w_local*. The dot products only require communication of a single value over all processes. On the other hand, the matrix-vector multiplication needed for *w_local* requires the global vector *p*, and, thus, more expensive communication.

Ex.1005, 11-13 (emphasis original).

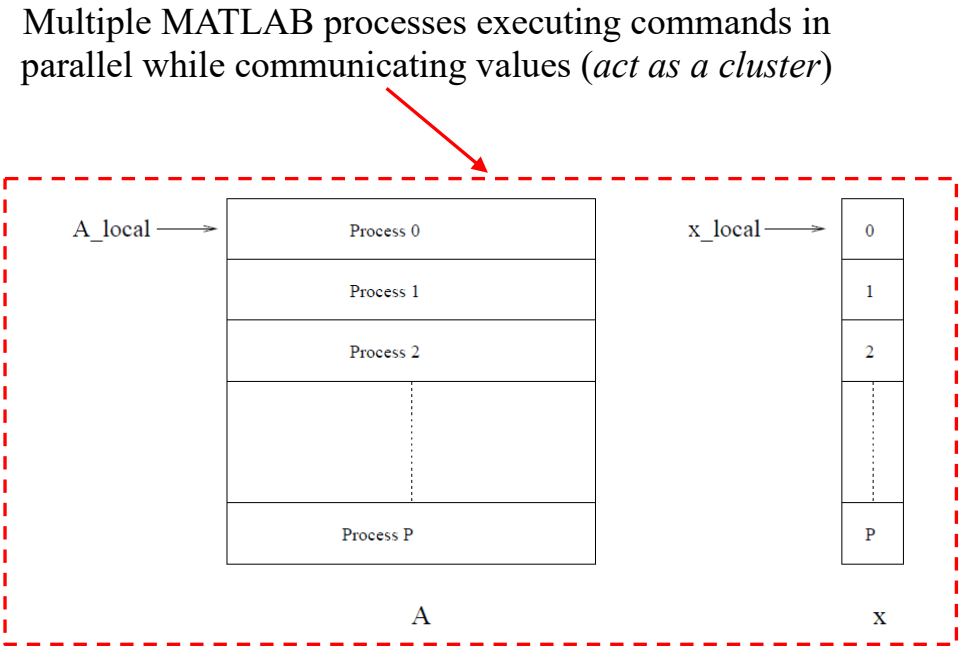



Figure 6: Matrix $A$ and vector $x$ distributed by row over $P$ processors

**Ex.1005, FIG. 6**

**250.** Menon describes another parallel MATLAB implementation of conjugate gradients where "the matrix A and all vectors are, in fact, distributed objects, and **the code must be run on all processors** in a SPMD fashion. … the communication and computation required by the parallel matrix-vector multiplication is folded into a single, more efficient routine." Ex.1005, 14. That is, the MATLAB processes on the processors perform their computations in parallel as if the processors were a single computer (*act as a cluster*). Ex.1005, 14.

**251.** Accordingly, Menon describes a MultiMATLAB architecture, shown below in FIG. 1 of Menon, in which all the MATLAB processes on the processors

of the MultiMATLAB architecture are running in parallel to solve a mathematical problem as if the processors were a single computer (*act as a cluster*):

MATLAB processes on processors working in parallel as if the processors were a single computer (*act as a cluster*) to solve mathematical problems

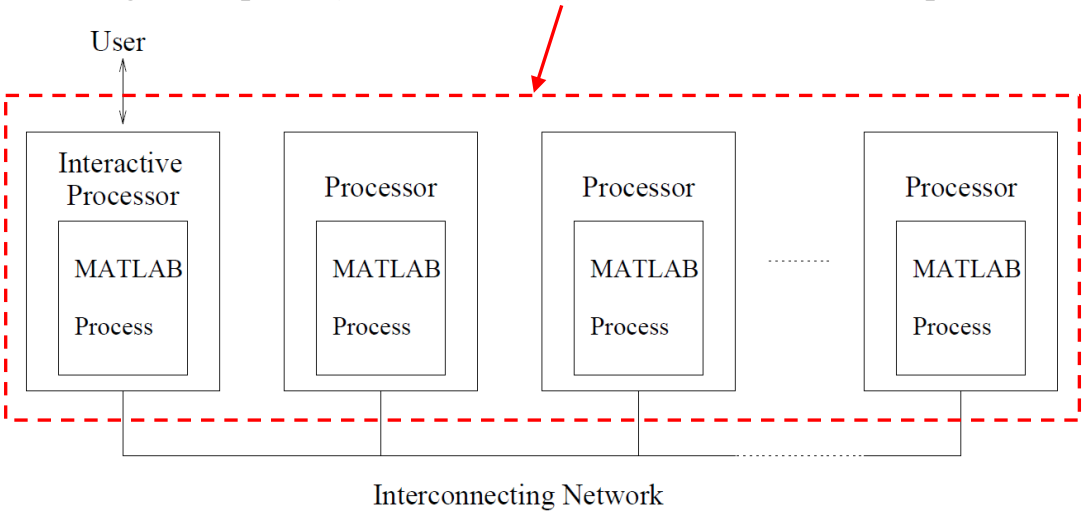



Figure 1: MultiMATLAB Architecture

**Ex.1005, FIG. 1 (annotated)**

**252.** Accordingly, Menon describes that the MEX routines, the MultiMATLAB interface module, the indirection table, and the communication layer of each MATLAB process (*the plurality of cluster node modules*) work together to enable a user to solve a mathematical problem as if they were a single computer, POEref describes how POE executes parallel programs on a networked cluster, and Trefethen describes how the user can issue a single command to cause all the processors to run at the same time, thereby rendering obvious *wherein the plurality of cluster node modules act as a cluster*.

**11. Claim 8**

**a. [8.0] *The computer cluster of claim 6, wherein the plurality of cluster node modules communicate with one another to act as a cluster.***

**253.** Menon and Trefethen render obvious *wherein the plurality of cluster node modules communicate with one another to act as a cluster* because Menon describes that the MEX routines, the MultiMATLAB interface module, the indirection table, and the communication layer of each MATLAB process in the MultiMATLAB architecture work together to enable a user to solve a mathematical problem by using all the MATLAB instances in parallel, where each MATLAB process performs its computation and communicates with other MATLAB processes for all the MATLAB processes on the processors of the MultiMATLAB architecture to perform computations in parallel as if the processors were a single computer and Trefethen describes how the user can issue a single command to cause all the processors to run at the same time.

**254.** As discussed in limitation [7.0], Menon describes that the MEX routines, the MultiMATLAB interface module, the indirection table, and the communication layer of each MATLAB process (*the plurality of cluster node modules*) work together to enable a user to solve a mathematical problem as if they were a single computer and Trefethen describes how the user can issue a single command to cause all the processors of the MultiMATLAB architecture to execute

at the same time. It was also shown in limitation [7.0] that the MultiMATLAB architecture can be implemented in IBM SP2 as "a networked cluster of [IBM SP2] RS/6000 processors," in which case all the MATLAB processes execute commands in parallel are a group of processors communicating with each other to accomplish a mathematical computation as if they were a single computer. Ex.1005, 6; POEref, 1.

**255.** In particular, Menon describes "**a parallel MATLAB implementation** of conjugate gradients that uses the SPMD message passing … In this case, the matrix A and the different vectors are each **distributed by row over all processes** as in Figure 6" of Menon which is shown below. Ex.1005, 11. Menon explains that the MATLAB processes on the processors of the MultiMATLAB architecture communicate with each other to perform their computations in parallel as if the processors were a single computer (*communicate with one another to act as a cluster*):

> In each iteration, a process only does the computation required for its local data. However, **communication is required for two different operations**: the dot-products needed to compute rho and alpha and the matrix-vector multiply needed to compute w local. The dot products only **require communication of a single value over all processes**. On the other hand, the matrix-vector multiplication needed for w local **requires the global vector** p, and, thus, **more expensive communication**.

Ex.1005, 11-13 (emphasis original).

Multiple MATLAB processes on processors communicating with each other to execute commands in parallel as if the processors were a single computer (*communicate with one another to act as a cluster*)

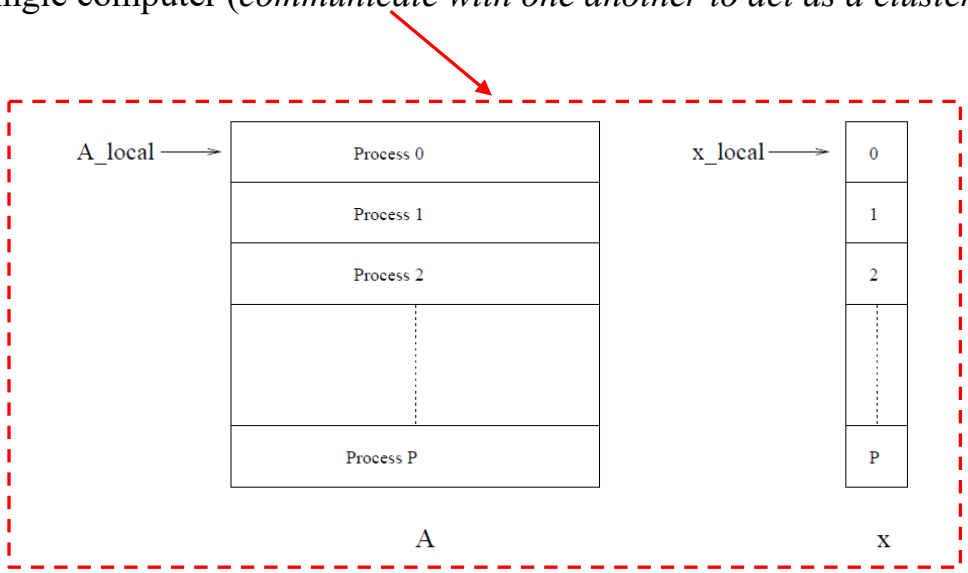



Figure 6: Matrix $A$ and vector $x$ distributed by row over $P$ processors

**Ex.1005, FIG. 6**

**256.** Accordingly, Menon describes that the MEX routines, the MultiMATLAB interface module, the indirection table, and the communication layer of each MATLAB process (*the plurality of cluster node modules*) in the MultiMATLAB architecture work together to enable a user to solve a mathematical problem by using all the MATLAB instances in parallel, where each MATLAB process performs its computation and communicates with other MATLAB processes for all the MATLAB processes on the processors of the MultiMATLAB architecture to perform computations in parallel as if the processors were a single computer (*communicate with one another to act as a cluster*) and Trefethen

describes how the user can issue a single command to cause all the processors to run at the same time, thereby rendering obvious *wherein the plurality of cluster node modules communicate with one another to act as a cluster*.

**12. Claim 9**

**a. [9.0] *The computer cluster of claim 8, wherein the computer cluster includes the user interface.***

**257.** Menon describes *wherein the computer cluster includes the user interface* because it teaches a MultiMATLAB architecture that includes a toolkit with a graphical interface (*user interface*) to allow users to specify equations, and a user interface for the user to directly interact with the interactive processor node (additionally, *user interface*).

**258.** As discussed in limitation [7.0], Menon describes a MultiMATLAB architecture (*computer cluster*), shown below in FIG. 1 of Menon, in which all the MATLAB processes are running in parallel to solve a mathematical problem.

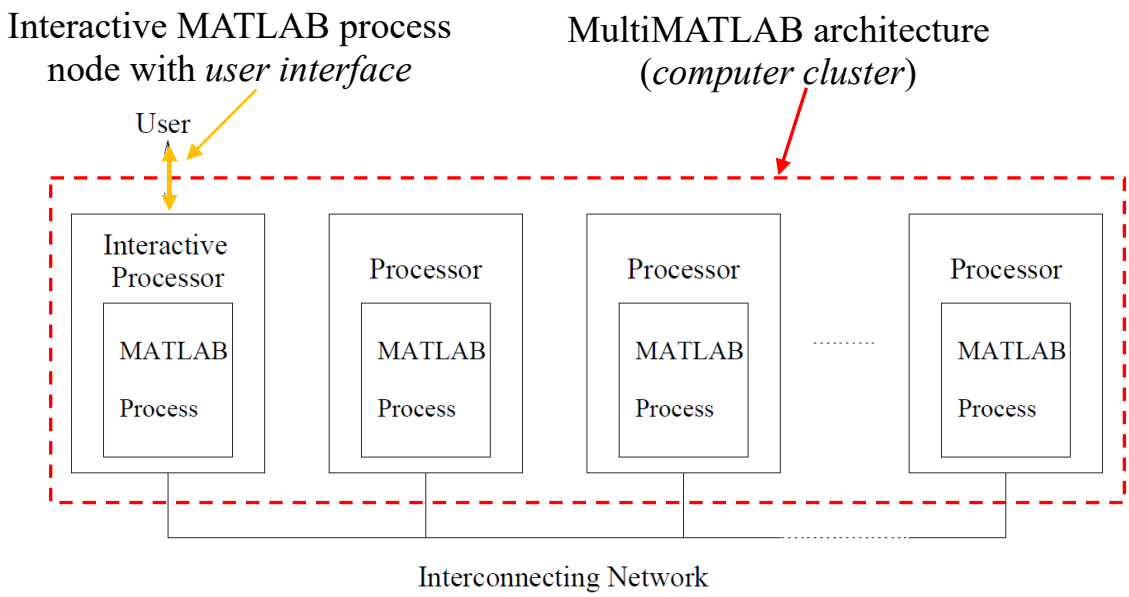



Figure 1: MultiMATLAB Architecture

**Ex.1005, FIG. 1 (annotated)**

**259.** Further, as discussed in limitation [1.3.1], Menon discloses a toolkit with a graphical interface (*user interface*) to allow users to specify equations, and a user interface (additionally, *user interface*) for the user to directly interact with the interactive processor node. Ex.1005, 6.

**260.** Accordingly, Menon's user interface by which a user specifies an equation that is part of the MultiMATLAB architecture (*computer cluster*) for solving mathematical equations in parallel renders obvious *wherein the computer cluster includes the user interface*.

**13.Claim 10**

**a. [10.0] *The computer cluster of claim 9, wherein each cluster node module accepts instructions from the user interface and interprets one or more of the instructions.***

**261.** Menon discloses *wherein each cluster node module accepts instructions from the user interface and interprets one or more of the instructions* because it teaches that a user enters instructions (*user instructions*) into a user interface (*user interface*) in the MultiMATLAB architecture in order to directly run the MEX routines of a *cluster node module*, those instructions are then communicated to the MATLAB processes of the MultiMATLAB architecture via their respective MultiMATLAB interface module (*each cluster node module accepts instructions from the user interface*), and the MultiMATLAB interface module of each MATLAB process maps MPI calls for MEX routines to the table offset (*interprets one or more of the instructions*).

**262. First**, as described in limitation [1.3.1], the MultiMATLAB architecture includes an interactive processor having a user interface to directly interact with and specify equations. From the user interface, the user can "directly" run the MEX routines of the *cluster node module*. Ex.1005, 3; s*ee also* Ex.1006, 3, 6. FIG. 1 of Menon below shows a schematic of the interactive MATLAB process with the user interface.

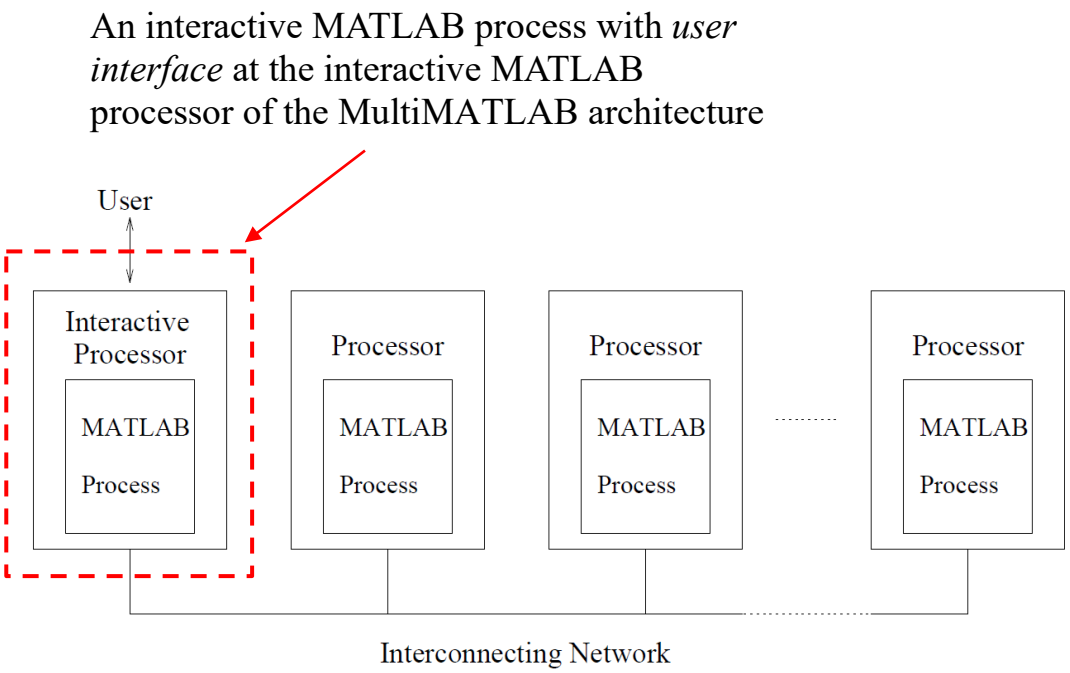
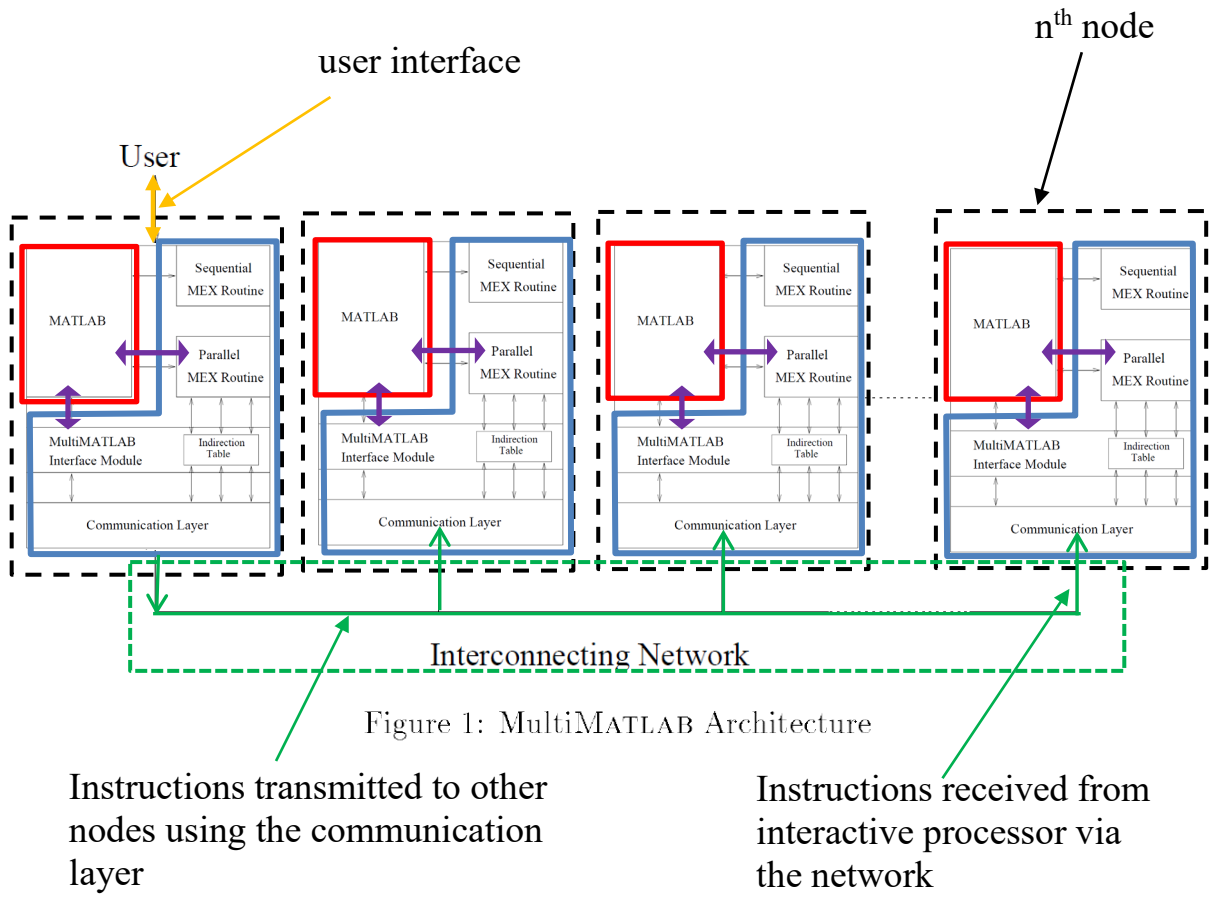



Figure 1: MultiMATLAB Architecture

**Ex.1005, FIG. 1 (annotated).**

**263.** **Second**, as discussed in [6.0], Menon discloses that each MATLAB process of the MultiMATLAB architecture has MEX routines, a MultiMATLAB interface module, an indirection table, and a communication layer (*a cluster node module*) for communicating with other nodes. **Third**, as discussed in limitations [1.7.2] and [1.7.3], Menon describes that (i) the interactive MATLAB process receives the user instructions and (ii) those instructions are evaluated in order to send commands to the other MATLAB processes. A combination figure of FIG. 1 and FIG. 2 of Menon below schematically shows the *cluster node modules* of each MATLAB process and communication of user instructions from the interactive MATLAB process to all the other (non-interactive) MATLAB processes.




**Ex.1005, FIGs. 1 and 2 (combined and annotated)**

**264. Fourth**, Menon describes that the MultiMATLAB interface module (*cluster node module*) of each MATLAB process interprets MPI (message passing interface) calls for MEX routines using an "indirection table":

> The MultiMATLAB interface module provides parallel MEX Routines with access to the underlying communication layer. In our implementations, MEX Routines are granted access to MPI routines. Moreover, the indirection imposed by the interface module is transparent to the programmer. There are macros,

> provided to all parallel MEX Routines, which map MPI calls directly to the appropriate table offset while maintaining the syntax of MPI.

Ex.1005, 6.

**265.** Accordingly, Menon teaches that the user can give instructions according to the known syntax for MPI calls (*one or more instructions*) and that the MultiMATLAB interface module then maps those MPI calls to the appropriate table offset (*interprets one or more instructions*).

**266.** In combination, Menon teaches that (i) the user enters instructions in order to directly run the MEX routines of *the cluster node module*, (ii) the MultiMATLAB *cluster node module* accepts the instructions and interprets the instructions to determine the destination of the instructions, (iii) each of the non-interactive MATLAB processes accepts the instructions via its *cluster node module*, and (iv) the MultiMATLAB interface module (*cluster node module*) of each MATLAB process maps the MPI calls to the table offset, rendering obvious *wherein each cluster node module accepts instructions from the user interface and interprets one or more of the instructions*.

## 14. Claim 11

### a. [11.0] *The computer cluster of claim 1, wherein the cluster initialization process comprises establishing communication among two or more of the nodes.*

**267.** Menon and POEref render obvious *wherein the cluster initialization process comprises establishing communication among two or more of the nodes* because they render obvious an initialization process for a MATLAB process on each processor node of a MultiMATLAB architecture that includes communicating with each of the processor nodes using MPI.

**268.** In limitation [1.1.2], it was shown that Menon discloses a user utilizing IBM's POE on an IBM SP2 system, in which a MultiMATLAB architecture is implemented, to launch an interactive MATLAB process on that IBM SP2 processor node and the non-interactive MATLAB processes of each of the other IBM SP2 processor nodes, and that POEref discloses the use of the "poe" command of POE to initialize a MATLAB process on each of the IBM SP2 processor nodes of the MultiMATLAB architecture. FIG. 1 of Menon below illustrates the initialization of MATLAB processes on the IBM SP2 MultiMATLAB architecture.

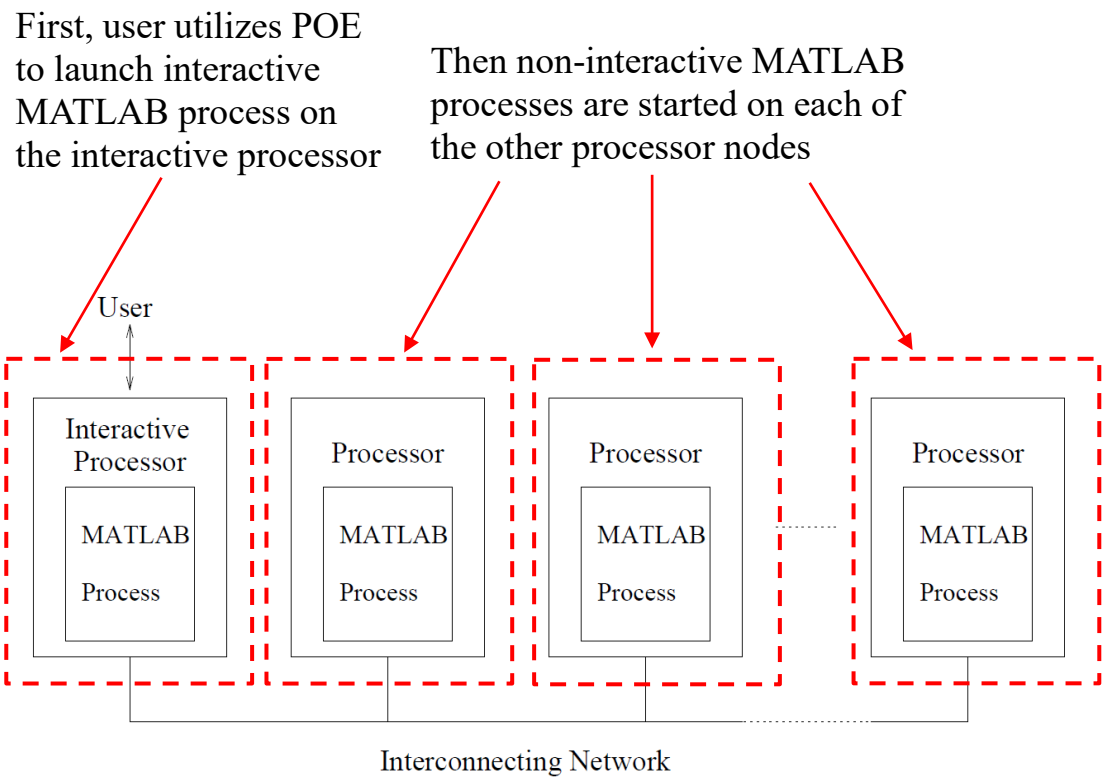



Figure 1: MultiMATLAB Architecture

**Ex.1005, FIG. 1 (annotated)**

**269.** POEref further discloses that during the initialization of MATLAB processes, POE ensures that the communication API (such as MPI) is initialized properly, explaining that "POE now provides an option to enable you to specify whether your program will use MPI, LAPI [low-level application programming interface], or both. Using this option, POE ensures that each API initializes properly and informs LoadLeveler which APIs are used so each node is set up completely." Ex.1008, 3. LoadLeveler is used "to allocate nodes" and LAPI is used to "run [a program] as a number of parallel tasks." Ex.1008, 3 (LoadLeveler), 1-2 (LAPI). Further, POEref describes that the Partition Manager, which the command "poe" of

POE starts to "allocate[] the nodes of your partition and initialize[] the local environment," "connects standard I/O to each remote node so the parallel tasks can communicate with the home node." Ex.1008, 3.

**270.** Accordingly, Menon teaching that the use of POE on the interactive MATLAB process to start MATLAB processes on each processor node of the MultiMATLAB architecture (*the cluster initialization process*) in view of POEref teaching the "poe" command for POE that ensures that the communication API for MPI is initialized properly and that each processor node can communicate (*establishing communication among two or more of the nodes*) renders obvious *wherein the cluster initialization process comprises establishing communication among two or more of the nodes*.

**15. Claim 12**

**a. [12.0] *The computer cluster of claim 1, wherein the cluster initialization process comprises configuring access by one or more of the nodes to a computer-readable medium comprising program code for the single-node kernel.***

**271.** Menon in view of POEref renders obvious *wherein the cluster initialization process comprises configuring access by one or more of the nodes to a computer-readable medium comprising program code for the single-node kernel* because Menon teaches using POE to initialize an IBM SP2 system in which a MultiMATLAB architecture is implemented (*the cluster initialization process*) by

launching a MATLAB process having a MATLAB process address space including MATLAB (*a computer-readable medium comprising program code for the single-node kernel*) on each IBM SP2 processor node and POEref teaching using POE to determine whether a user is authorized to access a node and that the node is permitted to access (*configuring access by one or more of the nodes to*) the user's directory containing the program, i.e., MATLAB (*a computer-readable medium comprising program code for the single-node kernel*).

**272. First**, in limitation [1.1.3], it was shown that the MATLAB process address space having MATLAB teaches *a computer-readable medium comprising computer program code for a single-node kernel*.

**273. Second**, in limitation [1.1.2], it was shown that Menon discloses a user utilizing IBM's POE on an IBM SP2 system, in which a MultiMATLAB architecture is implemented, to launch a MATLAB process (which includes MATLAB (*single-node kernel*)) on each IBM SP2 processor node of the MultiMATLAB architecture, teaching *the cluster initialization process*.

**274. Third**, when a user starts a job with POE, the LoadLeveler of POE "allocates [] nodes to the job." Ex.1008, 54. Accordingly, a POSITA would understand that, when a job (i.e., the *single-node kernel*) is allocated to a node when the job is being started (i.e., during the initialization process), the node is configured to access the computer-readable medium having the program code for the job.

**275.** Accordingly, Menon teaching using POE to initialize (*the cluster initialization process*) the IBM SP2 processor nodes that each contain a MATLAB process address space having MATLAB (*a computer-readable medium comprising computer program code for a single-node kernel*), in view of POEref teaching using POE so that each processor node checks for access to a directory that contains the parallel job to be performed (*configuring access by one or more of the nodes to … the single-node kernel*), renders obvious *wherein the cluster initialization process comprises configuring access by one or more of the nodes to a computer-readable medium comprising program code for the single-node kernel*.

**16. Claim 13**

**a. [13.0] *The computer cluster of claim 12, wherein the cluster initialization process comprises establishing message-passing support among the nodes in the cluster.***

**276.** Menon in view of POEref render obvious *wherein the cluster initialization process comprises establishing message-passing support among the nodes in the cluster* because Menon describes using POE to launch a MATLAB process on each IBM SP2 processor node of the MultiMATLAB architecture and POEref describes that POE ensures that the MPI communication API is initialized properly so that each node can communicate.

**277.** In limitation [1.1.2], it was shown that Menon discloses a user utilizing IBM's POE on an IBM SP2 system, "a modern high performance distributed

memory multiprocessor" in which a MultiMATLAB architecture is implemented, to launch a MATLAB process on each IBM SP2 processor node, teaching *the cluster initialization process*. Ex.1005, 5. This implementation of MultiMATLAB architecture "is based upon MPI-F [6], IBM's proprietary version of MPI specifically optimized for the SP2." Ex.1005, 5.

**278.** Also, POEref discloses that during the initialization of MATLAB processes, POE ensures that the communication API (such as MPI) is initialized properly and that the Partition Manager "connects standard I/O to each remote node so the parallel tasks can communicate with the home node." Ex.1008, 3. Therefore, POEref discloses that during the initialization of MATLAB processes, communication API such as MPI is initialized properly so that each node communicates, teaching *establishing message-passing support among the nodes in the cluster*.

**279.** Accordingly, Menon describes using POE to launch a MATLAB process on each IBM SP2 processor node of the MultiMATLAB architecture, in view of POEref describing that POE ensures that the MPI communication API is initialized properly so that each node can communicate, renders obvious *wherein the cluster initialization process comprises establishing message-passing support among the nodes in the cluster*.

**17. Claim 14**

**a. [14.0] *The computer cluster of claim 12, wherein the cluster initialization process comprises launching cluster node modules by the cluster.***

**280.** Menon discloses *wherein the cluster initialization process comprises launching cluster node modules by the cluster* because Menon teaches using POE to launch on each IBM SP2 processor node of the MultiMATLAB architecture a MATLAB process (*the cluster initialization process*), and each of the MATLAB processes launched by the POE includes MEX Routines, a MultiMATLAB interface module, indirection table, and a communication layer (*launching cluster node modules by the cluster*).

**281.** In limitation [1.1.2], it was shown that Menon discloses a user utilizing IBM's POE on an IBM SP2 system, in which a MultiMATLAB architecture is implemented, to launch a MATLAB process on each IBM SP2 processor node of the MultiMATLAB architecture, teaching *the cluster initialization process*. In limitation [4.0], it was shown that each MATLAB process includes MEX Routines, a MultiMATLAB interface module, indirection table, and a communication layer (*a cluster node module*), as shown in FIG. 2 of Menon below.

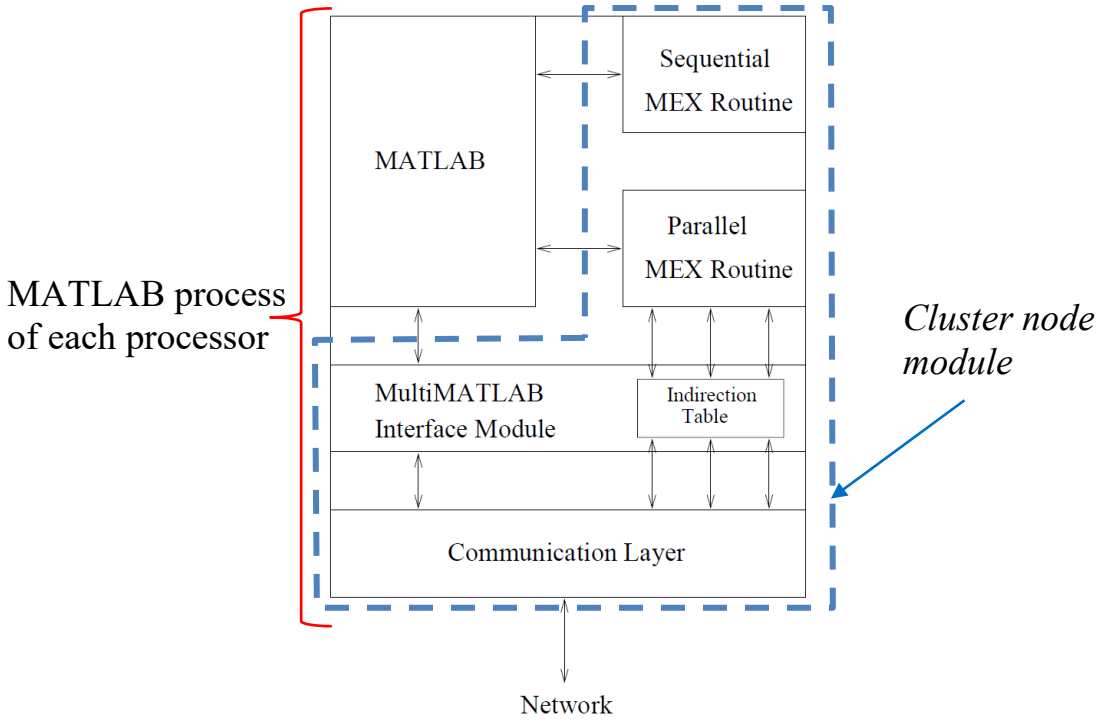



Figure 2: MATLAB Process Address Space

**Ex.1005, FIG. 2 (annotated)**

**282.** Accordingly, Menon teaches that the user starts a MATLAB process on each processor of the MultiMATLAB architecture by using IBM's POE (*the cluster initialization process*), which results in the interactive MATLAB process communicating with all the other processors using its communication layer to start (*comprises launching*) on each of those processors a MATLAB process that includes MEX Routines, a MultiMATLAB interface module, indirection table, and a communication layer (*a cluster node module*), rendering obvious *wherein the cluster initialization process comprises launching cluster node modules by the cluster*.

**18. Claim 15**

**a. [15.0] *The computer cluster of claim 14, wherein the cluster initialization process comprises assigning a processor identification number to each of the cluster node modules.***

**283.** Menon describes *wherein the cluster initialization process comprises assigning a processor identification number to each of the cluster node modules* because it teaches assigning an identification number to each of the MATLAB processes when the processes are started.

**284.** In limitation [1.1.2], it was shown that Menon discloses a user utilizing IBM's POE on an IBM SP2 system, in which a MultiMATLAB architecture is implemented, to launch a MATLAB process on each IBM SP2 processor node, teaching *the cluster initialization process*. Specifically, Menon explains that "[f]or each of *p* MATLAB processes, POE assigns an identification number between 0 and *p-1*. The process numbered 0 is considered the interactive MATLAB." Ex.1005, 6.

**285.** In limitation [4.0], it was shown that each MATLAB process includes MEX Routines, a MultiMATLAB interface module, indirection table, and a communication layer (*a cluster node module*), as shown in FIG. 2 of Menon below.

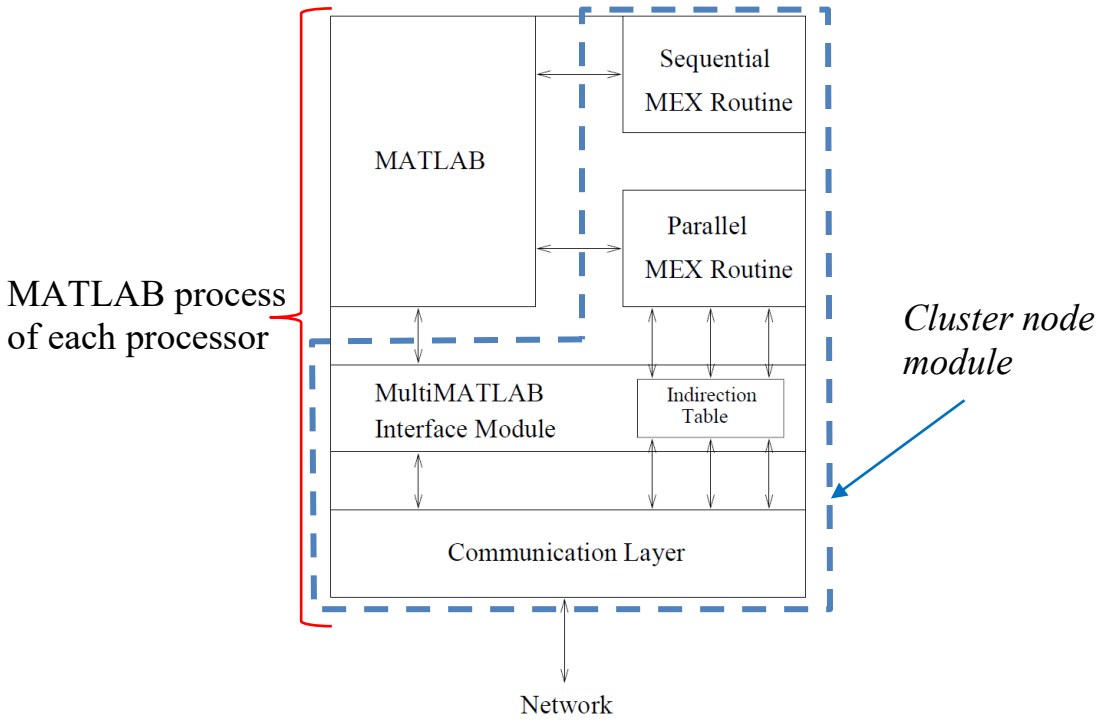



Figure 2: MATLAB Process Address Space

**Ex.1005, FIG. 2 (annotated)**

**286.** Accordingly, Menon assigning an identification number to each of the MATLAB processes when the processes are started renders obvious *wherein the cluster initialization process comprises assigning a processor identification number to each of the cluster node modules*.

**19. Claim 16**

**a. [16.0] *The computer cluster of claim 1, wherein the cluster initialization process comprises:***

**287.** Limitation [16.0] is disclosed or rendered obvious for the same reasons presented above for limitation [1.1.2].

**b. [16.1] *launching cluster node modules by the cluster; and***

**288.** Limitation [16.1] is disclosed or rendered obvious for the same reasons presented above for limitation [14.0].

**c. [16.2] *after launching the cluster node modules, configuring access by one or more of the nodes to a non-transitory computer-readable medium comprising program code for the single-node kernel.***

**289.** As discussed in limitation [14.0], Menon teaches *launching cluster node modules*.

**290.** Further, as discussed in limitation [12.0], Menon teaches using POE to initialize an IBM SP2 system in which a MultiMATLAB architecture is implemented (*the cluster initialization process*) by launching a MATLAB process having a MATLAB process address space (that includes MATLAB (*single-node kernel*)) on each IBM SP2 processor node and POEref discloses that when a user starts a job with POE, the LoadLeveler of POE "allocates [] nodes to the job." Ex.1008, 54. Accordingly, a POSITA would understand that, when a job (i.e., the *single-node kernel*) is allocated to a node when the job is being started (i.e., during the initialization process), the node is configured to access the computer-readable medium having the program code for the job, rendering obvious *configuring access by one or more of the nodes to a non-transitory computer-readable medium comprising program code for the single-node kernel*.

**291.** Because logic dictates that a MATLAB process has to be launched on a node before the node can access MATLAB contained therein, *the cluster initialization process*, which launches the MATLAB processes on the nodes, necessarily occurs before *configuring access by one or more of the nodes to a non-transitory computer-readable medium comprising program code for the single-node kernel*. In other words, Menon in view of POEref renders obvious *after launching the cluster node modules, configuring access by one or more of the nodes to a non-transitory computer-readable medium comprising program code for the single-node kernel*.

**292.** Further, a POSITA would have found it obvious to *configur[e] access by one or more of the nodes to a non-transitory computer-readable medium comprising program code for the single-node kernel* ***after*** *launching the cluster node modules* because there are only two predictable options available for timing the *configuring* and the *launching* steps with respect to each other, i.e., one **before** or **after** the other, rendering both options obvious. It is my understanding that, under the law of obviousness, when a POSITA has two predictable choices for when a step can occur, such as **before** or **after**, it is a simple and obvious design choice for a POSITA to make.

**20.Claim 17**

**a. [17.0] *The computer cluster of claim 1, wherein the cluster initialization process comprises:***

**293.** Limitation [17.0] is disclosed or rendered obvious for the same reasons presented above for limitation [1.1.2].

**b. [17.1] *launching cluster node modules by the cluster;***

**294.** Limitation [17.1] is disclosed or rendered obvious for the same reasons presented above for limitation [14.0].

**c. [17.2] *after launching the cluster node modules, establishing communication among two or more of the nodes; and***

**295.** As discussed in limitation [14.0], Menon teaches *the cluster initialization process comprises launching cluster node modules*.

**296.** Further, as discussed in limitation [11.0], Menon teaches the use of POE on the interactive MATLAB process to start MATLAB processes on each processor node of the MultiMATLAB architecture (*the cluster initialization process*) and POEref teaches the "poe" command for POE that ensures that the communication API for MPI is initialized properly and that each processor node can communicate (*establishing communication among two or more of the nodes*), renders obvious *establishing communication among two or more of the nodes*.

**297.** In limitation [11.0], it was shown the "poe" command of POE can be used to initialize a MATLAB process on each of the IBM SP2 processor nodes, and

during the initialization of MATLAB processes, POE ensures that the communication API (such as MPI) is initialized properly and each node can communicate. Thus, the initialization of the MATLAB processes on each processor node (*the cluster initialization process*, which *comprises launching cluster node modules*) occurs before the use of the "poe" command to ensure that the communication API for MPI is initialized properly and that each processor node can communicate (*establishing communication among two or more of the nodes*). Accordingly, Menon discloses *after launching the cluster node modules, establishing communication among two or more of the nodes.*

**298.** Further, a POSITA would have found it obvious to *establish[] communication among two or more of the nodes* ***after*** *launching the cluster node modules* because there are only two predictable options available for timing the *establishing* and the *launching* steps with respect to each other, i.e., one **before** or **after** the other, rendering both options obvious. It is my understanding that, under the law of obviousness, when a POSITA has two predictable choices for when a step can occur, such as **before** or **after**, it is a simple and obvious design choice for a POSITA to make.

**d. [17.3] *after establishing communication among two or more of the nodes, assigning a processor identification number to each of the cluster node modules.***

**299.** As discussed in limitation [11.0], Menon in view of POEref teaches that *the cluster initialization process comprises establishing communication among two or more of the nodes establishing communication among two or more of the nodes*. Specifically, POEref teaches the "poe" command for POE that ensures that the communication API for MPI is initialized properly and that each processor node can communicate (*establishing communication among two or more of the nodes*)

**300.** Further, as discussed in limitation [15.0], Menon teaches that *the cluster initialization process comprises assigning a processor identification number to each of the cluster node modules*. Specifically, Menon explains that "[f]or each of *p* MATLAB processes, POE assigns an identification number between 0 and *p-1*. The process numbered 0 is considered the interactive MATLAB." Ex.1005, 6.

**301.** A POSITA would have found it obvious to *assign[] a processor identification number to each of the cluster node modules* ***after*** *establishing communication among two or more of the nodes* because POEref teaches that the user enters the "poe" command to establish communication among and with the nodes. Ex. 1008, 30. After the poe command, then the nodes are identified by the node id, as shown below:

For additional illustration, the following shows the command prompts that would appear, as well as the program names you would enter, to load the example *master* and *workers* programs. This example assumes that the **MP_PROCS** environment variable is set to 5.

```
% poe

0:host1_name> master [options]

1:host2_name> workers [options]

2:host3_name> workers [options]

3:host4_name> workers [options]

4:host5_name> workers [options]

Partition loaded...
```

Ex. 1008, 31.

**21. Claim 18**

**a. [18.0] *The computer cluster of claim 1, wherein one or more of the nodes are configured to communicate at least some of the user instructions.***

**302.** Limitation [18.0] is disclosed or rendered obvious for the same reasons presented above for limitations [1.7.2].

**22. Claim 19**

**a. [19.0] *The computer cluster of claim 18, wherein one or more of the nodes are configured to communicate at least some of the user instructions to one or more single-node kernels.***

**303.** Limitation [19.0] is disclosed or rendered obvious for the same reasons presented above for limitations [1.7.3].

**23. Claim 20**

**a. [20.0] *The computer cluster of claim 1, wherein one or more of the nodes are configured to accept user instructions via one or more of the nodes.***

**304.** Menon discloses *wherein one or more of the nodes are configured to accept user instructions via one or more of the nodes* because it teaches that the non-interactive nodes await commands from the interactive processor, and upon receiving those commands, executing the commands.

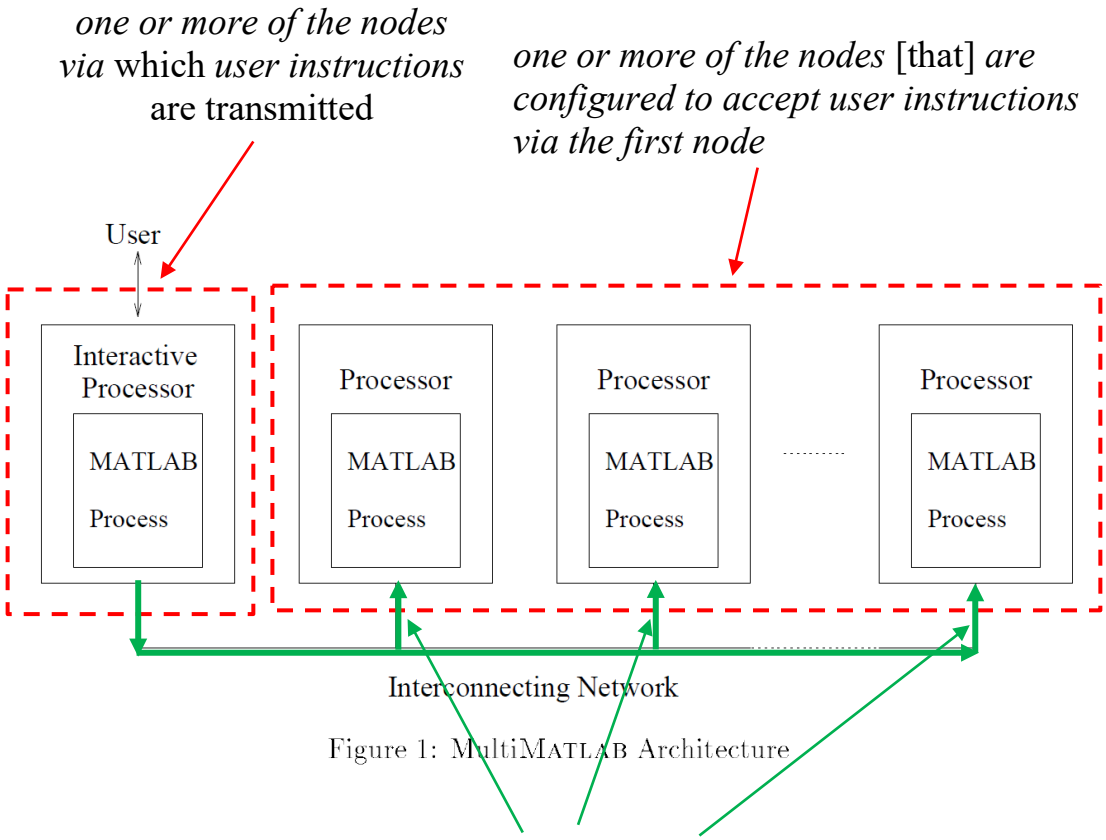
**305.** As discussed in limitation [1.7.3], Menon teaches an interactive processor that communicates the *user instructions* to the other processors, which are received at the communication layer and communicated to the respective MATLAB of the processors.

**306.** Menon further discloses that each of its processors has a MATLAB process that awaits commands from the interactive MATLAB process and then executes those commands, explaining:

> In the MultiMATLAB architecture, illustrated in Figure 1, each processor in a parallel platform individually runs a MATLAB process. … The user interacts directly with one MATLAB process, called the interactive process … Other MATLAB processes await commands from the interactive process. These other processes are used either

> by explicitly sending MATLAB commands to them or by executing routines that, in turn, use them. Ex.1005, 3.

FIG. 1 of Menon below illustrates the interactive MATLAB process at the interactive processor transmitting commands that are based on *user instructions* to the non-interactive MATLAB processes on the other processors of the MultiMATLAB architecture, which then execute the commands. Limitation [1.7.3] shows example illustrations from Trefethen showing that commands sent to the other or non-interactive MATLAB processes corresponding to IDs specified in the *user instructions* are processed by MATLAB at the respective MATLAB processes identified by the IDs (and as such, are accepted by the processors of the non-interactive MATLAB processes).




**Ex.1005, FIG. 1 (annotated)**

**24. Claim 21**

**a. [21.0]** ***The computer cluster of claim 20, wherein one or more of the nodes are configured to communicate at least some of the user instructions using the mechanism for the nodes to communicate with each other.***

**307.** Limitation [21.0] is rendered obvious for the same reasons presented above for [1.7.2].

**25. Claim 22**

**a. [22.0] *The computer cluster of claim 1, wherein one or more of the nodes are configured to transmit at least some of the user instructions that originate from a user interface.***

**308.** Menon discloses *wherein one or more of the nodes are configured to transmit at least some of the user instructions that originate from a user interface* because it teaches an interactive processor of the MultiMATLAB architecture that receives user instructions from a user interface and transmits those user instructions to the other processors.

**309.** As discussed in limitation [1.3.1], the MATLAB code stored in the address space of, and accessed by, the interactive processor, includes a toolkit with a graphical interface to allow users to provide equations and directly interact with the interactive processor node (*user interface*). Further, as discussed in limitation [1.7.2], the interactive MATLAB processor communicates the *user instructions* to the other (i.e., non-interactive) processors of the MultiMATLAB architecture.

**310.** Accordingly, Menon's interactive processor that (i) provides *a user interface* by which a user can submit *user instructions* and (ii) transmits those *user instructions* to the other processors of the MultiMATLAB architecture, rendering obvious *wherein one or more of the nodes are configured to transmit at least some of the user instructions that originate from a user interface*.

**26. Claim 23**

**a. [23.0] *The computer cluster of claim 1, wherein one or more of the nodes are configured to parallelize at least some of the user instructions before communicating at least some of the user instructions to one or more single-node kernels.***

**311.** Menon discloses *wherein one or more of the nodes are configured to parallelize at least some of the user instructions before communicating at least some of the user instructions to one or more single-node kernels* because it teaches that the interactive MATLAB processor receives the *user instructions* specifying an equation and then the parallel MEX Routines cause each MATLAB process to hold a different portion of the data so that parallel processing can be performed.

**312.** Limitation [1.7.3] shows that Menon's interactive MATLAB processor communicates *user instructions* to the other processors of the MultiMATLAB architecture, which are received at the communication layer and communicated to MATLAB (*single-node kernel*). Limitation [1.2] shows that Menon teaches MEX routines having code that implement standard variants of MPI calls to enable SPMD point-to-point communication so that the MATLAB processes can communicate task and data directly with each other.

**313.** Menon further describes that the interactive MATLAB processor uses "parallel MEX Routines" to provide the "necessary message passing and data distribution." Ex.1005, 10. An example of such "parallel MEX Routines" is "an

Eval routine for parallel evaluation." Ex.1005, 4. Because "low-level details of the SPMD paradigm may make it undesirable to certain users and for certain applications," Menon proposes:

> A possible alternative is to use parallel MEX Routines that provide similar functionality to high-level MATLAB operations. These parallel MEX Routines would perform matrix multiplications, solve for eigenvalues, compute Fourier transforms, and so on. In fact, these MEX Routines could provide an interface identical to corresponding sequential MATLAB functions. In particular, they would distribute data from the interactive MATLAB process among all processes, perform the parallel computation, and collect results back to the interactive process.

Ex.1005, 10. The parallel MEX Routines can "distribute data from the interactive MATLAB process among all processes" because "[a]ll parallel MEX Routines access the underlying communication layer and, hence, the network." Ex.1005, 10, 4. The "parallel MEX Routines" in each MATLAB process of the MultiMATLAB architecture is shown in FIG. 2 of Menon below.

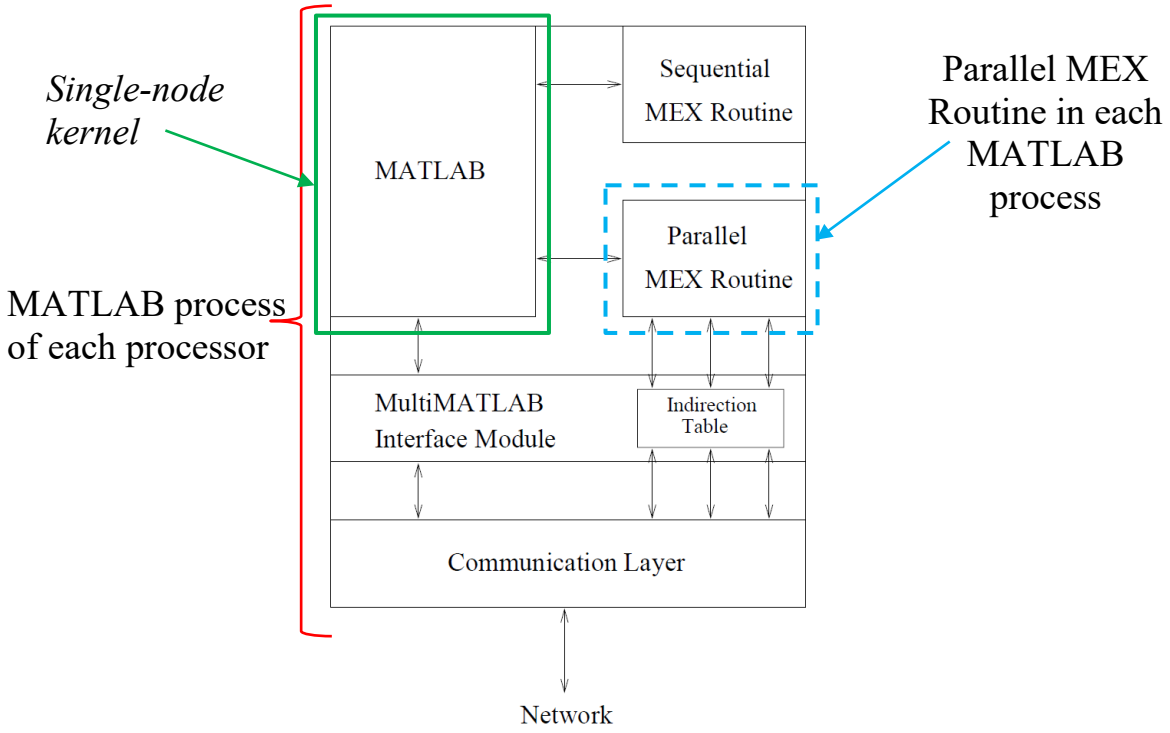



Figure 2: MATLAB Process Address Space

**Ex.1005, FIG. 2 (annotated)**

**314.** Menon provides an example of the use of parallel MEX Routines to "distribute data from the interactive MATLAB process among all processes," "an iterative algorithm such as conjugate gradients." Ex.1005, 10. With reference to the conjugate gradients algorithm, Menon discusses how to distribute calculating conjugate gradients by distributing a matrix across all the MATLAB processes, so that each MATLAB process holds a different portion of the matrix:

> The primary computation in this algorithm consists of a matrix-vector multiplication at every iteration. However, the matrix is unmodified over all iterations. In an

> efficient parallel implementation, the matrix would remain distributed over the course of the algorithm. We are currently investigating a paradigm based on distributed matrices. A distributed matrix is stored across all MATLAB processes. Each process holds a different portion of the matrix within a structure containing the local data and a descriptor denoting the data distribution of the entire matrix. Users may align MATLAB processes in a multi-dimensional grid and then distribute data across this grid.

Ex.1005, 10.

**315.** An application that uses MPI calls is called "parallelizing an application," as explained by POEref:

> The application developer begins by creating a parallel program's source code. … **the developer places calls to Message Passing Interface (MPI)** or Low-level Application Programming Interface (LAPI) routines **so that it can run as a number of parallel tasks. This is known as parallelizing the application**. … MPI

> provides message passing capabilities for the current version of PE [IBM parallel environment].

Ex.1008, 1-2. A POSITA would recognize that Menon's use of MPI calls with its parallel MEX routines is parallelizing the user instructions.

**316.** Accordingly, Menon's interactive MATLAB processor that receives an equation (such as the conjugate gradients algorithm) (*user instructions*), uses the parallel MEX Routines in connection with MPI calls to distribute a different portion of the matrix used in the matrix-vector calculation at each of the MATLAB processes (and corresponding MEX Routines) renders obvious *wherein one or more of the nodes are configured to parallelize at least some of the user instructions before communicating at least some of the user instructions to one or more single-node kernels*.

### 27. Claim 24

**a. [24.0]** ***The computer cluster of claim 1, wherein one or more of the nodes are configured to accept user instructions before communicating at least some of the user instructions to one or more single-node kernels.***

**317.** Limitation [24.0] is rendered obvious for the same reasons presented above for limitations [1.7.0]-[1.7.3].

### 28. Claim 25

**a. [25.0]** ***The computer cluster of claim 1, wherein the plurality of nodes are configured to communicate with***

***one another to interpret and translate commands for execution by a plurality of single-node kernels.***

**318.** Menon discloses *wherein the plurality of nodes are configured to communicate with one another to interpret and translate commands for execution by a plurality of single-node kernels* because it teaches that each node has a cluster node module to enable the processors to communicate with one another and a MultiMATLAB interface module that maps those MPI calls to the appropriate table offset (*to interpret and translate commands*) to expose all functions and data to MATLAB.

**319.** **First**, as discussed in limitation [4.0], Menon describes each processor of the MultiMATLAB architecture having a MATLAB process, which in turn has a MEX Routines, a MultiMATLAB interface module, indirection table, and a communication layer (*cluster node module*).

**320.** **Second**, limitation [1.2] shows that Menon teaches MEX routines having code to implement standard variants of MPI calls to enable SPMD point-to-point communication so that the MATLAB processes can communicate tasks and data directly with each other.

**321.** **Third**, as discussed in limitation [10.0], Menon discloses that the MultiMATLAB interface module of each MATLAB process acts as an interpretation layer that maps MPI calls to the indirection table so tasks and data can be provided to MATLAB for computation. Menon describes the indirection

table as containing "pointers to all functions and data in the underlying communication layer that needed to be exposed to parallel MEX Routines," explaining:

> The MultiMatlab interface module provides MEX Routines access to the underlying communication layer via an indirection table. Upon initialization, the [MultiMATLAB] interface module builds a table of pointers to all functions and data in the underlying communication layer that need to be exposed to parallel MEX Routines. When a parallel MEX Routine is first loaded and executed, it accesses the MultiMATLAB interface module through a function call to MATLAB and is given the location of this table. … Once the table is obtained, the MEX Routine has the capability to access any function provided by the underlying communication layer through its corresponding offset in the table.

Ex.1005, 5.

**322.** Accordingly, Menon teaches that (i) each processor node uses a communication layer that uses MPI calls to enable the processors to communicate with one another and (ii) the MultiMATLAB interface module on each processor

maps those MPI calls to the appropriate table offset (*interprets and translate commands*) to expose all functions and data to MATLAB (*single-node kernel*) for execution, thereby rendering obvious *wherein the plurality of nodes are configured to communicate with one another to interpret and translate commands for execution by a plurality of single-node kernels*.

**29. Claim 26**

**a. [26.0] *A computer cluster comprising:***

**323.** Limitation [26.0] is disclosed or rendered obvious for the same reasons presented above for limitation [1.0].

**b. [26.1.0] *a plurality of nodes:***

**324.** Limitation [26.1.0] is disclosed or rendered obvious for the same reasons presented above for limitation [1.1.0].

**c. [26.1.1] *wherein one or more of the nodes are configured to receive: a command to start a cluster initialization process for the computer cluster,***

**325.** Limitation [26.1.1] is disclosed or rendered obvious for the same reasons presented above for limitation [1.1.2].

**d. [26.1.2] *wherein the cluster initialization process comprises establishing communication among two or more of the nodes; and***

**326.** Limitation [26.1.2] is disclosed or rendered obvious for the same reasons presented above for limitation [11.0].

**e. [26.1.3] [*wherein one or more of the nodes are configured to receive:*] *an instruction from a user interface or a script; and***

**327.** Menon in view of Trefethen discloses *wherein one or more of the nodes are configured to receive … an instruction from a user interface or a script* because Menon teaches that the user provides the mathematical equations to the interactive processor of the MultiMATLAB architecture, which then sends commands to the other processors, and Trefethen provides additional details of how MATLAB interprets user instructions input from an m-file, which is a script, and then sends the commands to the other processors.

**328.** Menon discloses that the interactive MATLAB process allows the user to directly interact with the MultiMATLAB architecture, explaining that "[t]he user interacts directly with one MATLAB process, called the interactive process, and operates within that process's MATLAB environment." Ex.1005, 3. The user can use "an Eval routine for parallel evaluation" to provide user instructions to the interactive MATLAB process so that the user instructions are evaluated in parallel fashion by the MultiMATLAB architecture. Ex.1005, 4. "This routine allows users to execute commands on the other processes. On the interactive process, the Eval routine is implemented as a MEX Routine which accepts a vector of MATLAB process IDs and a MATLAB command. This routine sends the commands over the underlying communication layer to the processes corresponding to the specified

IDs." Ex.1005, 4. In limitation [1.3.1], it was shown that MATLAB of the interactive MATLAB process teaches *a first single-node kernel*. Thus, Menon teaches that the user interacts directly with the interactive MATLAB process, and a MEX Routine receives *user instructions* including MATLAB commands and sends the commands to the other MATLAB processes for evaluation.

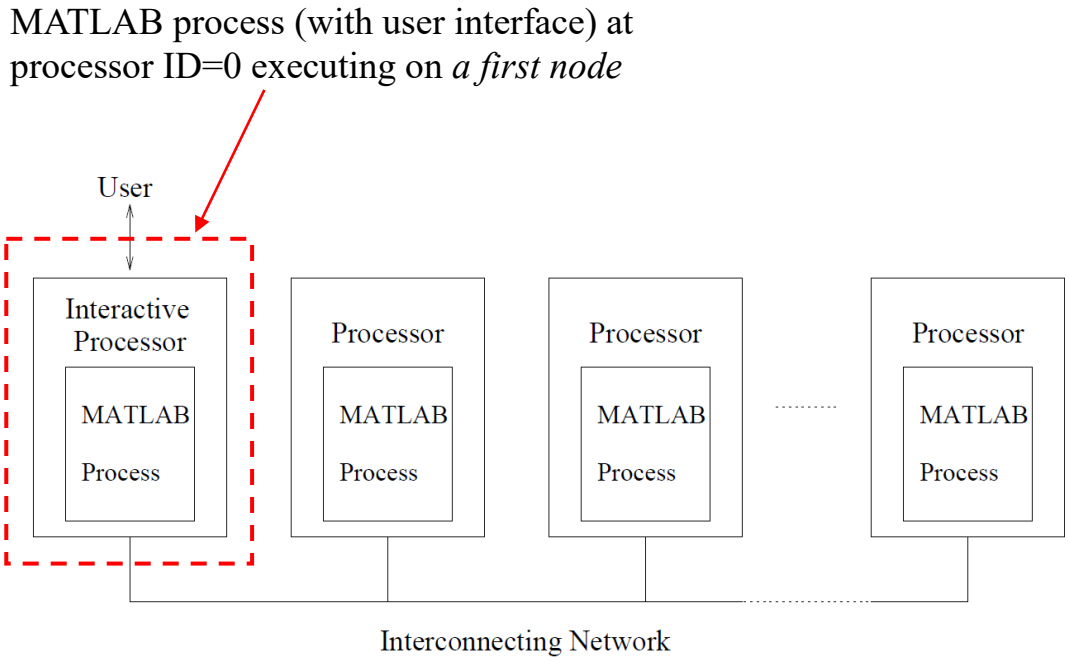



Figure 1: MultiMATLAB Architecture

**Ex.1005, FIG. 1 (annotated)**

**329.** Trefethen discloses that a user can input instructions via an "m-file." Ex.1006, 5. The "m-file" contains instructions, such as "svd," "fft," and "roots," which are interpreted by the interactive MATLAB process to calculate the mathematical expressions "singular value decomposition," "fast Fourier transform," and "polynomial zerofinding," respectively. Ex.1006, 2. Trefethen describes that,

rather than entering explicit MATLAB commands as an argument string, the user can use an "m-file" to provide instructions. Ex.1006, 5. That is, a user may "want to execute a program (an m-file) rather than a single line of text," and "[a] command such as Eval( 'filename' ) achieves this effect." Ex.1006, 5.

**330.** Trefethen provides example *user instructions* that specify different actions for different MATLAB processes. Ex.1006, 3-7. For example, Trefethen discloses the *user instructions* Eval( [4 5] , 'cond(hilb(ID))' ) that allows a user to "select a subset of the processes by passing two arguments to the Eval command, the first being a vector of process IDs." Ex.1006, 4. The MATLAB processes on processor ID = 4 and ID = 5 then perform different actions, i.e., processor ID=4 calculates cond(hilb(ID=4)) and processor ID=5 calculates cond(hilb(ID=5)), where the results are:

"ans = 1.5514e+04,

ans = 4.7661e+05

the condition numbers of the Hilbert matrices of dimensions 4 and 5." Ex.1006, 4.

**331.** As another example, Trefethen discloses the m-file "cycle.m," via which a user provides the user instructions to instruct: (i) the interactive MATLAB process to create a variable a with a value and send the variable to the next MATLAB process, (ii) the next middle MATLAB processes to receive the variable a from the immediately preceding MATLAB process, double the value of the

received variable a, and send the result to the next MATLAB process, and (iii) the last MATLAB process to receive the variable a and double its value. Ex.1006, 6. The *user instruction* cycle.m and the final output from a MultiMATLAB architecture that has six MATLAB processes are shown below:

*user instructions*

Instruction for MATLAB process ID=0 to create a=1 and send "a" to MATLAB process ID=1

```
if ID==0                    % first process: send
   a = 1
   Send(ID+1,a)
elseif ID == Nproc-1        % last process: receive and double
   a = 2*Recv
else                        % middle processes: receive, double, and send
   a = 2*Recv
   Send(ID+1,a)
end;
```

Instruction for the last MATLAB process to receive the variable a and double it

Process 0 creates the variable a with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of a, and sends it along to process 2; and so on. If there are six processors the command `Eval( 'cycle' )` produces the output

Instruction for a middle MATLAB process ID to receive the variable "a," double it, and send the doubled variable to the next MATLAB process ID=ID+1

```
a = 1
a = 2
a = 4
a = 8
a = 16
a = 32
```

Output of final result (a=32)

Ex.1006, 6 (annotated).

**332.** A POSITA would recognize and know that the "m-file" described in Trefethen is a "script." *See, e.g.*, Ex.1016, 25 ("These files are called ***script files*** or simply ***M-files***" (emphasis original)).

**333.** Accordingly, Menon's teaching that the MEX Routine receives the *user instructions* and then sends commands to the MATLAB processes on other processors of the MultiMATLAB architecture in accordance with the IDs specified

by the user, in view of Trefethen's teaching of a user inputting an m-file into MATLAB that is used by the main MATLAB process to assign different actions to different processes, renders obvious *wherein one or more of the nodes are configured to receive … an instruction from a user interface or a script*.

**f. [26.2] *a mechanism for the nodes to communicate results of mathematical expression evaluation with each other using asynchronous calls;***

**334.** Menon discloses *a mechanism for the nodes to communicate results of mathematical expression evaluation with each other using asynchronous calls* because Menon teaches MEX routines that have code to implement MPI calls and enable SPMD point-to-point communication so that each processor can communicate directly with the others over an interconnecting network, and Trefethen teaches that the SPMD communication enables mathematical expression evaluation results from a first MATLAB process on a first processor to be communicated asynchronously to a second MATLAB process on a second processor, and the mathematical expression results obtained there to be communicated asynchronously to a third MATLAB process on a third processor, and so on.

**335.** **First**, Menon discloses that MultiMATLAB uses "MEX routines" and that "[a]ll parallel functionality in the MultiMATLAB system is provided through MEX routines." Ex.1005, 3. For example, "the MultiMATLAB system provides

users with an Eval routine for parallel evaluation. This routine allows users to execute commands on the other processes." Ex.1005, 3-4. "[T]he Eval routine is implemented [on the interactive process] as a MEX routine which accepts a vector of MATLAB process IDs and a MATLAB command" that the MEX Routine sends to the MATLAB processes identified by the MATLAB process IDs. Ex.1005, 4. "On non-interactive processes, a separate top-level MEX routine is immediately run upon initialization of the system. This MEX routine runs in a loop in which it waits for MATLAB commands to arrive from the interactive process and executes them in its own MATLAB environment." Ex.1005, 4. The MEX Routines send the "commands over the underlying communication layer to the processes corresponding to the specified IDs." Ex.1005, 4.

**336.** The underlying communication layer of the MultiMATLAB architecture "runs over the parallel platform's interconnection network" and "provide[s] [a MATLAB process on a processor] with the ability to communicate with other processes." Ex.1005, 3. An example of the underlying communication layer is MPI. Ex.1005, 3. Table 1 of Menon below shows MultiMATLAB commands implemented as MEX Routines. Further, FIG. 2 and FIG. 1 of Menon below show the MEX Routines of a MATLAB process and the interconnection network of the MultiMATLAB architecture linking the processors thereof, respectively.

| MultiMATLAB commands | |
|---|---|
| Starting and stopping MultiMATLAB (MPICH only) | |
| Start(n) | Initializes n remote MATLAB processes. |
| Quit | Closes remote MATLAB processes. |
| Running commands on multiple processors | |
| Nproc | Returns number of MATLAB processes. |
| ID | Returns ID of calling MATLAB process. |
| Eval(pid,'command') | Evaluates a command on one or more MATLAB process. |
| Master/Slave Communication | |
| Put(pid,dataname) | Put data from the master process onto one or more remote processes. |
| Get(pid,dataname) | Get data from a remote process onto the master process. |
| SPMD Communication | |
| Send(pid,data) | Send data from one process to another. |
| Recv(pid) | Receive data sent from another process. |
| Barrier | Synchronize processes. |
| Bcast(pid,data) | Broadcasts data from processor pid to all processes. |
| Sum(data) | Adds data across all processors to form a global sum. |
| Collect(data) | Collects local data segments into a single global one. |

Table 1: Selected commands in the MultiMATLAB system

Commands implemented in MEX Routines

**Ex.1005, Table 1 (annotated)**

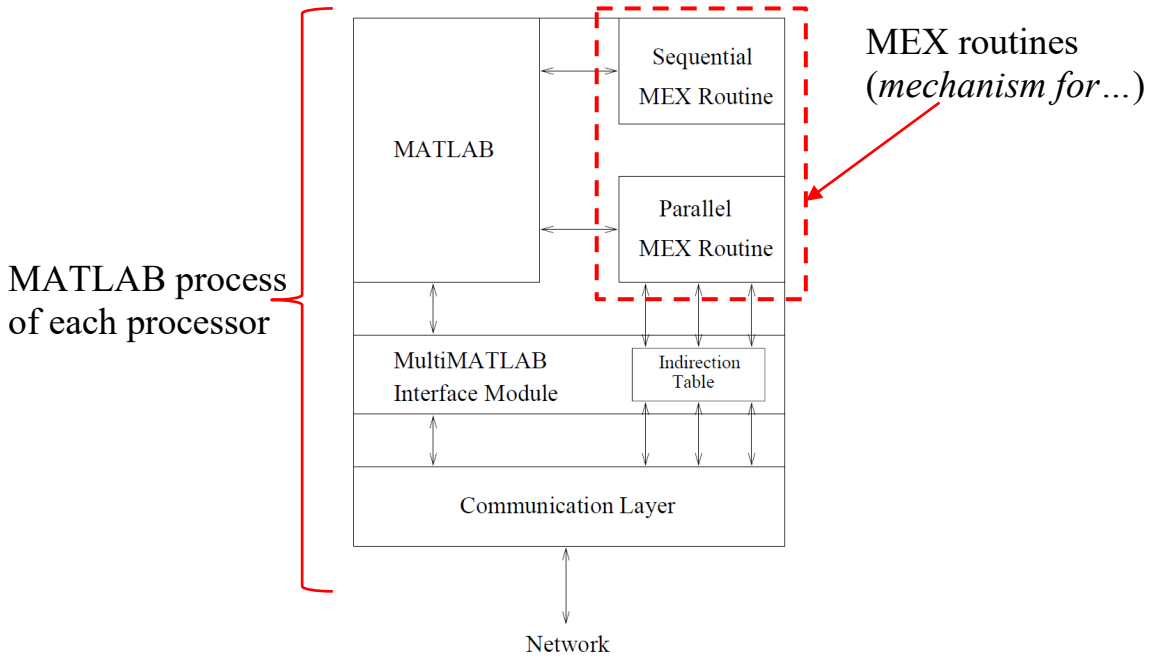
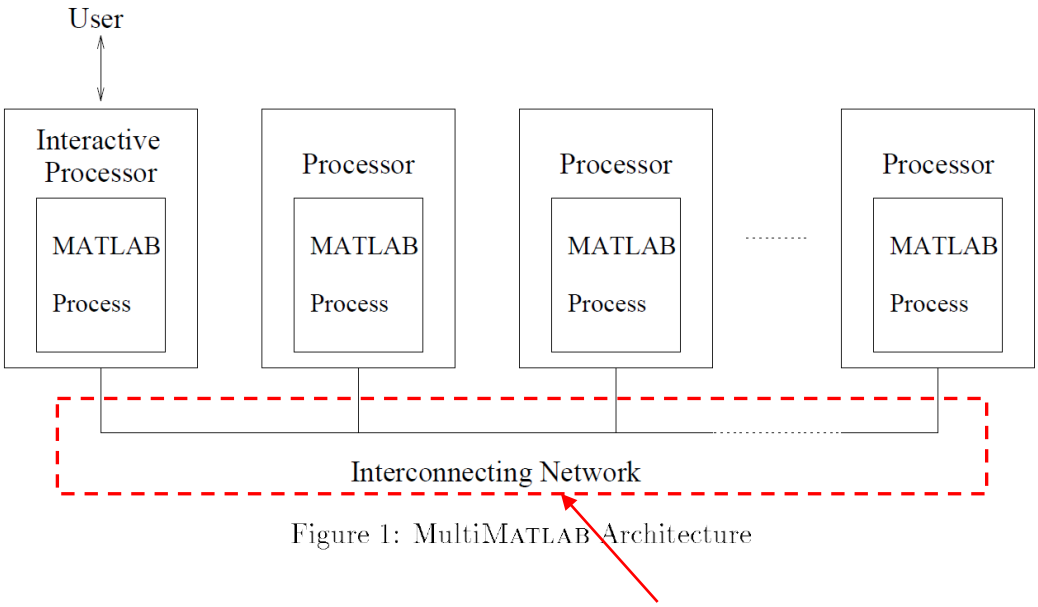



Figure 2: MATLAB Process Address Space

**Ex.1005, FIG. 2 (annotated)**




Interconnecting network connecting each MATLAB process on a processor to any other in the MultiMATLAB architecture

**Ex.1005, FIG. 1 (annotated)**

**337.** **Second**, Trefethen discloses that the commands used in MultiMATLAB are "written as a C file" and "interfaced to MATLAB via the standard MATLAB Fortran/C/C++ interface known as MEX." Ex.1006, 10. The "[h]igher-level MultiMATLAB commands are usually built on higher-level MPI commands." Ex.1006, 10. Trefethen teaches that "MultiMATLAB is currently built upon the standard send and receive variants" of MPI. Ex.1006, 10. That is, "[m]ore general point-to-point communication is accomplished by send and receive commands, which can be executed on any of the MATLAB processes." Ex.1006, 5.

**338.** Trefethen also discloses sending tasks and data directly to other processors using these MultiMATLAB commands. Ex.1006, 3-7. "The standard

MultiMATLAB command for executing commands on one or more processors is Eval." Ex.1006, 3. For example,

> The command
>
> Eval( 'ID^ID' )
>
> might produce
>
> ans = 1
>
> ans = 1
>
> ans = 256
>
> ans = 3125
>
> ans = 27
>
> ans = 4.

Ex.1006, 4. That is, Eval( 'ID^ID' ) sends the task "ID^ID" to each processor ID to perform the task "ID^ID" using the data ID. Ex.1006, 4.

**339.** Trefethen discloses that a user can specify which MATLAB processes should perform the tasks, explaining that "one can select a subset of the processes by passing two arguments to the Eval command, the first being a vector of process IDs." Ex.1006, 4. For example,

> Eval( [4 5] , 'cond(hilb(ID))' )
>
> might return
>
> ans = 1.5514e+04

ans = 4.7661e+05

the condition numbers of the Hilbert matrices of

dimensions 4 and 5.

Ex.1006, 4. That is, the MATLAB processes on processor ID=4 and ID=5 each perform the task "cond(hilb(ID))" using the data ID, i.e., MATLAB process on processor ID=4 performs the task "cond(hilb(ID = 4))" using the data ID=4 and generates the result "ans = 1.5514e+04," "the condition number[] of the Hilbert [matrix] of dimension[] 4," and MATLAB process on processor ID=5 performs the task "cond(hilb(ID = 5))" using the data ID=5 and generates the result "ans = 4.7661e+05," "the condition number[] of the Hilbert [matrix] of dimension[] 5." Ex.1006, 4.

**340.** Thus, the MEX Routines that contain program code to implement the standard MPI send and receive calls to send commands (e.g., tasks) and data directly to other processes renders obvious an MPI module.

**341.** **Third**, Trefethen also provides details of the SPMD communication commands. Ex.1006, 5-7. Trefethen describes a MATLAB process on a processor of the MultiMATLAB architecture using the SPMD point-to-point commands to communicate *results of mathematical expression evaluation* to another MATLAB process on another processor asynchronously. Ex.1006, 6. Specifically, Trefethen provides the following example code in which a MATLAB process communicates a

result of a mathematical evaluation to a next MATLAB process, which in turn communicates the next mathematical result to the next MATLAB process, and so on, where the communications of the mathematical results from one MATLAB process to another *use asynchronous calls* such as Send and Recv:

```
if ID==0                % first process: send
   a = 1
   Send(ID+1,a)
elseif ID == Nproc-1    % last process: receive and double
   a = 2*Recv
else                    % middle processes: receive, double, and send
   a = 2*Recv
   Send(ID+1,a)
end;
```

Process 0 creates the variable a with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of a, and sends it along to process 2; and so on. If there are six processors the command `Eval( 'cycle' )` produces the output

```
a = 1
a = 2
a = 4
a = 8
a = 16
a = 32
```

*Asynchronous calls*

The processes run asynchronously, but since each Send command is only executed after the corresponding Recv has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning.

Ex.1006, 6 (annotated).

**342.** Trefethen also discloses that the MultiMATLAB commands can deliver a mathematical result to a designated process. For example, Trefethen explains that:

> The command
>
> Eval( 'Sum(1,[1 ID Nproc])' )
>
> executed on six processes will return the vector
>
> [6 15 36]

> to process 1. If the first argument is omitted, the result is returned (broadcast) to all processes

Ex.1006, 7.

**343.** Accordingly, Menon's MEX routines, having code to implement MPI calls, that enable SPMD point-to-point communication so that each processor can communicate directly with the others over an interconnecting network, further in view of Trefethen's teaching that the SPMD communication enables mathematical expression evaluation results from a first MATLAB process on a first processor to be communicated to a second MATLAB process on a second processor asynchronously, and the mathematical expression results obtained there to be communicated to a third MATLAB process on a third processor asynchronously, and so on, renders obvious *a mechanism for the nodes to communicate results of mathematical expression evaluation with each other using asynchronous calls*.

**g. [26.3] *wherein each of the nodes is configured to access a non-transitory computer-readable medium comprising program code for a single-node kernel that, when executed, is capable of causing a hardware processor to evaluate mathematical expressions;***

**344.** Limitation [26.3] is disclosed or rendered obvious for the same reasons presented above for limitation [1.1.3].

**h. [26.4.0]** ***wherein the plurality of nodes comprises:***

**345.** Limitation [26.5.0] is disclosed or rendered obvious for the same reasons presented above for limitation [26.1.0].

**i. [26.4.1]** ***a first node comprising a first hardware processor configured to access a first memory comprising program code for a user interface and program code for a first single-node kernel,***

**346.** Limitation [26.4.1] is disclosed or rendered obvious for the same reasons presented above for limitation [1.3.1].

**j. [26.4.2]** ***the first single-node kernel configured to interpret user instructions and distribute calls to at least one of a plurality of other nodes for execution; and***

**347.** Limitation [26.4.2] is disclosed or rendered obvious for the same reasons presented above for limitation [1.3.2].

**k. [26.5.1]** ***a second node comprising a second hardware processor with a plurality of processing cores,***

**348.** Limitation [26.5.1] is disclosed or rendered obvious for the same reasons presented above for limitation [1.4.1].

**l. [26.5.2]** ***wherein the second node is configured to receive calls from the first node, execute at least a first mathematical expression evaluation, and communicate a result of mathematical expression evaluation to a third node;***

**349.** Limitation [26.5.2] is disclosed or rendered obvious for the same reasons presented above for limitation [1.4.2].

**m. [26.6.1]** ***wherein the third node comprises a third hardware processor with a plurality of processing cores,***

**350.** Limitation [26.6.1] is disclosed or rendered obvious for the same reasons presented above for limitation [1.5.1].

**n. [26.6.2]** ***wherein the third node is configured to receive the result of mathematical expression evaluation from the second node, execute at least a second mathematical expression evaluation using the received result, and communicate the result of the second mathematical expression evaluation to the first node;***

**351.** Limitation [26.6.2] is disclosed or rendered obvious for the same reasons presented above for limitation [1.5.2].

**o. [26.7]** ***wherein the first node is configured to return the result of the second mathematical expression evaluation to the user interface or the script;***

**352.** Limitation [26.7] is disclosed or rendered obvious for the same reasons presented above for limitation [1.6].

**p. [26.8.0]** ***wherein one or more of the nodes are configured to:***

**353.** Limitation [26.8.0] is disclosed or rendered obvious for the same reasons presented above for limitation [26.1.0].

**q. [26.8.1]** ***accept user instructions;***

**354.** Limitation [26.8.1] is disclosed or rendered obvious for the same reasons presented above for limitation [1.7.1].

**r. [26.8.2]** ***after accepting user instructions, communicate at least some of the user instructions using the***

***mechanism for the nodes to communicate with each other; and***

**355.** Limitation [26.8.2] is disclosed or rendered obvious for the same reasons presented above for limitation [1.7.2].

**s. [26.8.3]** ***after communicating at least some of the user instructions using the mechanism, communicate at least some of the user instructions to one or more single-node kernels.***

**356.** Limitation [26.8.3] is disclosed or rendered obvious for the same reasons presented above for limitation [1.7.3].

**30. Claim 29**

**a. [29.0]** ***A computer cluster comprising:***

**357.** Limitation [29.0] is disclosed or rendered obvious for the same reasons presented above for limitation [1.0] and [26.0].

**a. [29.1.1]** ***a plurality of nodes,***

**358.** Limitation [29.1.1] is disclosed or rendered obvious for the same reasons presented above for limitation [1.1.1] and [26.1.1].

**b. [29.1.2]** ***wherein one or more of the nodes are configured to receive: a command to start a cluster initialization process for the computer cluster,***

**359.** Limitation [29.1.2] is disclosed or rendered obvious for the same reasons presented above for limitation [1.1.2] and [26.1.2].

**c. [29.1.3] *wherein the cluster initialization process comprises establishing communication among two or more of the nodes; and***

**360.** Limitation [29.1.3] is disclosed or rendered obvious for the same reasons presented above for limitation [11.0].

**d. [29.1.4] [*wherein one or more of the nodes are configured to receive:*] *an instruction from a user interface or a script; and***

**361.** Limitation [29.1.4] is disclosed or rendered obvious for the same reasons presented above for limitation [26.1.3].

**e. [29.2] *a mechanism for the nodes to communicate results of mathematical expression evaluation with each other;***

**362.** Limitation [29.2] is disclosed or rendered obvious for the same reasons presented above for limitation [26.2].

**f. [29.3] *wherein each of the nodes is configured to access a non-transitory computer-readable medium comprising program code for a single-node kernel that, when executed, is capable of causing a hardware processor to evaluate mathematical expressions;***

**363.** Limitation [29.3] is disclosed or rendered obvious for the same reasons presented above for limitation [1.1.3].

**g. [29.4.0] *wherein the plurality of nodes comprises:***

**364.** Limitation [29.4.0] is disclosed or rendered obvious for the same reasons presented above for limitation [1.1.1].

**h. [29.4.1]** ***a first node comprising a first hardware processor configured to access a first memory comprising program code for a user interface and program code for a first single-node kernel,***

**365.** Limitation [29.4.1] is disclosed or rendered obvious for the same reasons presented above for limitation [1.3.1].

**i. [29.4.2]** ***the first single-node kernel configured to interpret user instructions and distribute calls to at least one of a plurality of other nodes for execution; and***

**366.** Limitation [29.4.2] is disclosed or rendered obvious for the same reasons presented above for limitation [1.3.2].

**j. [29.5.1]** ***a second node comprising a second hardware processor with a plurality of processing cores,***

**367.** Limitation [29.5.1] is disclosed or rendered obvious for the same reasons presented above for limitation [1.4.1].

**k. [29.5.2]** ***wherein the second node is configured to receive calls from the first node, execute at least a first mathematical expression evaluation, and communicate a result of mathematical expression evaluation to a third node;***

**368.** Limitation [29.5.2] is disclosed or rendered obvious for the same reasons presented above for limitation [1.4.2] and [26.5.2].

**l. [29.6.1]** ***wherein the third node comprises a third hardware processor with a plurality of processing cores,***

**369.** Limitation [29.6.1] is disclosed or rendered obvious for the same reasons presented above for limitation [1.5.1] and [26.6.1].

**m. [29.6.2]** ***wherein the third node is configured to receive the result of mathematical expression evaluation from the second node, execute at least a second mathematical expression evaluation using the received result, and communicate the result of the second mathematical expression evaluation to the first node; and***

**370.** Limitation [29.6.2] is disclosed or rendered obvious for the same reasons presented above for limitation [1.5.2] and [26.6.2].

**n. [29.7]** ***wherein the first node is configured to return the result of the second mathematical expression evaluation to the user interface or the script;***

**371.** Limitation [29.7] is disclosed or rendered obvious for the same reasons presented above for limitation [26.7].

**o. [29.8.0]** ***wherein one or more of the nodes are configured to:***

**372.** Limitation [29.8.0] is disclosed or rendered obvious for the same reasons presented above for limitation [29.1.0].

**p. [29.8.1]** ***accept user instructions;***

**373.** Limitation [29.8.1] is disclosed or rendered obvious for the same reasons presented above for limitation [1.7.1] and [26.8.1].

**q. [29.6.2]** ***after accepting user instructions, communicate at least some of the user instructions using the mechanism for the nodes to communicate with each other; and***

**374.** Limitation [29.8.2] is disclosed or rendered obvious for the same reasons presented above for limitation [1.7.2] and [26.8.2].

**r. [29.6.3] *after communicating at least some of the user instructions using the mechanism, communicate at least some of the user instructions to one or more single-node kernels.***

**375.** Limitation [29.8.3] is disclosed or rendered obvious for the same reasons presented above for limitation [1.7.3] and [26.8.3].

**31. Claim 30**

**a. [30.0] *The computer cluster of claim 4, wherein each of the plurality of nodes implements asynchronous calls that enable the single-node kernel to perform computation tasks while the cluster node modules are simultaneously communicating with one another.***

**376.** Menon and Trefethen render obvious *wherein each of the plurality of nodes implements asynchronous calls that enable the single-node kernel to perform computation tasks while the cluster node modules are simultaneously communicating with one another* because Menon teaches that the processors of MultiMATLAB (i) communicate mathematical results via SPMD point-to-point communication in the MEX routines of each processor and (ii) operate in parallel, and Trefethen teaches that the MEX routines implement asynchronous send() and recv() calls that enable MATLAB (*single-node kernel*) to evaluate mathematical calculations while simultaneously communicating results with other processors.

**377.** **First**, as discussed in limitation [4.0], Menon discloses that each node has communication components, i.e., MEX Routines, interface module, indirection table, and communication layer, that teach a *cluster node module* of that node.

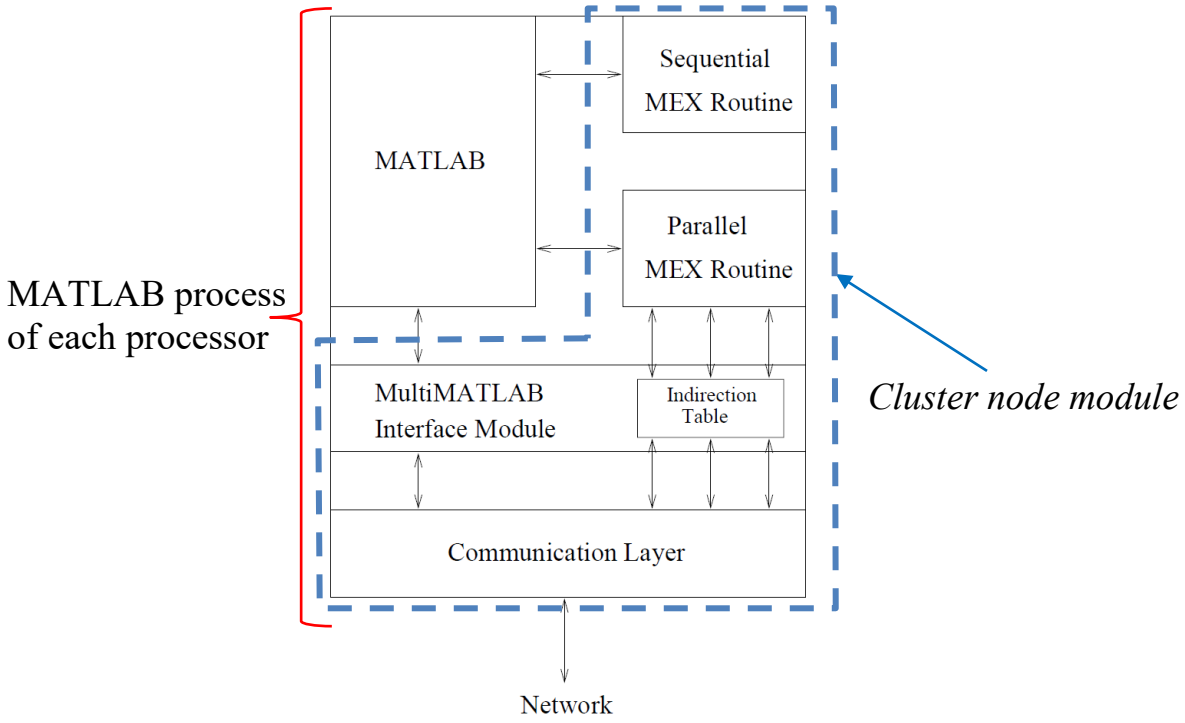



Figure 2: MATLAB Process Address Space

**Ex.1005, FIG. 2 (annotated)**

**378. Second**, as discussed in limitation [1.2], Menon teaches MEX routines having code that implement standard variants of MPI calls to enable SPMD point-to-point communication so that the MATLAB processes can communicate tasks and data directly with each other. The "MEX routines" provide "[a]ll parallel functionality in the MultiMATLAB system." Ex.1005, 3. For example, the MEX routines send the "commands over the underlying communication layer to the processes corresponding to the specified IDs." Ex.1005, 4. Table 1 of Menon below shows MultiMATLAB commands implemented as MEX Routines, including SPMD calls such as "Send(pid,data)" command that "send[s] data from one process

to another," the "Recv(pid)" command that "receive[s] data sent from another process," and the "Bcast(pid,data)" command that "broadcasts data from processor pid to all processes":

| MultiMATLAB commands | |
|---|---|
| Starting and stopping MultiMATLAB (MPICH only) | |
| Start(n) | Initializes n remote MATLAB processes. |
| Quit | Closes remote MATLAB processes. |
| Running commands on multiple processors | |
| Nproc | Returns number of MATLAB processes. |
| ID | Returns ID of calling MATLAB process. |
| Eval(pid,'command') | Evaluates a command on one or more MATLAB process. |
| Master/Slave Communication | |
| Put(pid,dataname) | Put data from the master process onto one or more remote processes. |
| Get(pid,dataname) | Get data from a remote process onto the master process. |
| SPMD Communication | |
| Send(pid,data) | Send data from one process to another. |
| Recv(pid) | Receive data sent from another process. |
| Barrier | Synchronize processes. |
| Bcast(pid,data) | Broadcasts data from processor pid to all processes. |
| Sum(data) | Adds data across all processors to form a global sum. |
| Collect(data) | Collects local data segments into a single global one. |

Table 1: Selected commands in the MultiMATLAB system

Commands implemented in MEX Routines

**Ex.1005, Table 1 (annotated)**

**379. Third**, also as discussed in [26.2], Trefethen provides further details of the SPMD communication commands. Ex.1006, 5-7. Specifically, Trefethen teaches that the Send() and Recv() calls are "asynchronous." Ex.1006, 6. Trefethen describes a MATLAB process on a processor of the MultiMATLAB architecture using the SPMD point-to-point commands to communicate mathematical results to another MATLAB process on another processor asynchronously. Ex.1006, 6. Trefethen provides the following example code in which a MATLAB process

communicates a mathematical result to a next MATLAB process, which in turn communicates the next mathematical result to the next MATLAB process, and so on, where the communications of the mathematical results from one MATLAB process to another use *asynchronous calls*:

```
if ID==0                 % first process: send
   a = 1
   Send(ID+1,a)
elseif ID == Nproc-1     % last process: receive and double
   a = 2*Recv
else                     % middle processes: receive, double, and send
   a = 2*Recv
   Send(ID+1,a)
end;
```

Process 0 creates the variable a with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of a, and sends it along to process 2; and so on. If there are six processors the command `Eval( 'cycle' )` produces the output

```
a = 1
a = 2
a = 4
a = 8
a = 16
a = 32
```

*Asynchronous calls*

The processes run asynchronously, but since each `Send` command is only executed after the corresponding `Recv` has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning.

Ex.1006, 6 (annotated).

**380. Fourth**, Menon discloses that a MATLAB process of each processor of the MultiMATLAB architecture evaluates mathematical expressions while simultaneously communicating to distribute the results of the evaluations to the other MATLAB processes. Ex.1005, 11-12. Discussing "a parallel MATLAB implementation of conjugate gradients that uses the SPMD message passing," Menon explains that:

> the matrix A and the different vectors are each distributed by row over all processes as in Figure 6. In each iteration, a process only does the computation required for its local data. However, communication is required for two different operations: the dot-products needed to compute *rho* and *alpha* and the matrix-vector multiply needed to compute *w_local*. The dot products only require communication of a single value over all processes. On the other hand, the matrix-vector multiplication needed for *w_local* requires the global vector *p*, and, thus, more expensive communication.

Ex.1005, 11-13 (emphasis original). That is, each MATLAB process performs its own computations while communicating values that may be needed for the computations with other MATLAB processes, as shown in FIG. 4b of Menon shown below. Ex.1005, 11. A POSITA would recognize that the computations or mathematical evaluations are performed by MATLAB (*single-node kernel*) in a MATLAB process.

Multiple processors evaluating or computing mathematical expressions

```
r = b;
rho = r'*r;
k = 0;
while((k < kmax))
  k = k+1;
  if (k == 1)
    p = r;
  else
    beta = rho/oldrho;
    p = r + beta*p;
  end

  w = A * p;

  alpha = rho/(p'*w);
  x = x + alpha * p;
  r = r - alpha * w;
  oldrho = rho;
  rho = r'*r;
end
```

a. Sequential MATLAB

```
r_local = b_local;
rho = Sum(r_local'*r_local);
k = 0;
while((k < kmax))
  k = k+1;
  if (k == 1)
    p_local = r_local;
  else
    beta = rho/oldrho;
    p_local = r_local + beta*p_local;
  end

  p = Gather(p_local);
  w_local = A_local * p;

  alpha = rho/(Sum(p_local'*w_local));
  x_local = x_local + alpha * p_local;
  r_local = r_local - alpha * w_local;
  oldrho = rho;
  rho = Sum(r_local'*r_local);
end
```

b. Message Passing MATLAB

Figure 4: Conjugate Gradients

Inter-processor communications take place for these computations

**Ex.1005, FIG. 4 (annotated)**

**381.** Accordingly, Menon teaching that the processors of MultiMATLAB (i) communicate mathematical results via SPMD point-to-point communication in the MEX routines of each processor and (ii) operate in parallel, further in view of Trefethen teaching that the MEX routines implement asynchronous send() and recv() calls that enable MATLAB (*single-node kernel*) to evaluate mathematical calculations while simultaneously communicating results with other processors, renders obvious *wherein each of the plurality of nodes implements asynchronous*

*calls that enable the single-node kernel to perform computation tasks while the cluster node modules are simultaneously communicating with one another*.

**32. Claim 35**

**a. [35.0] *A computer cluster node for evaluating expressions in parallel with other computer cluster nodes, the computer cluster node comprising:***

**382.** Menon and RS6000 render obvious the preamble. Specifically, Menon and RS600 render obvious *a computer cluster node for evaluating expressions in parallel with other computer cluster nodes* because Menon discloses individual processor nodes of the MultiMATLAB architecture that evaluate mathematic expressions in parallel as implemented on an IBM SP2 multiprocessor system, and RS6000 describes 128 nodes for parallel processing.

**383.** Menon discloses a "MultiMATLAB architecture" designed to provide "high-performance parallel routines" and "performance gains." Ex.1005, 2. "[T]he MultiMATLAB system provides users with an Eval routine for parallel evaluation." Ex.1005, 3-4. Menon discloses "two implementations of the MultiMATLAB architecture," one "designed to run on a generic network of Unix workstations" and another that "was designed specifically for the IBM SP2," "a modern high performance distributed memory multiprocessor." Ex.1005, 5, 6. Menon describes implementing MultiMATLAB on a plurality of processors, such as 8 processors or "32 processors[,] of the IBM SP2," as shown in FIGs. 7 and 8 below. Ex.1005, 13.

The number of processors is not limited to 32 processors as RS6000 discloses that the IBM SP2 is "scalable from one to 128 processor nodes" and "systems with more than 128 processor nodes are [also] available." Ex.1007, 2.

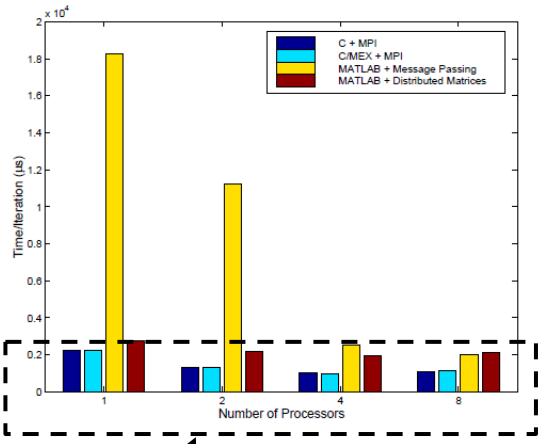



Figure 7: Parallel Conjugate Gradients on the IBM SP-2: 256 × 256 matrix

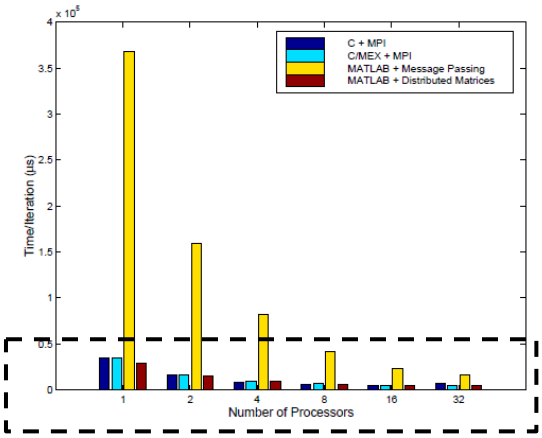



Figure 8: Parallel Conjugate Gradients on the IBM SP-2: 1024 × 1024 matrix

8 processors

32 processors

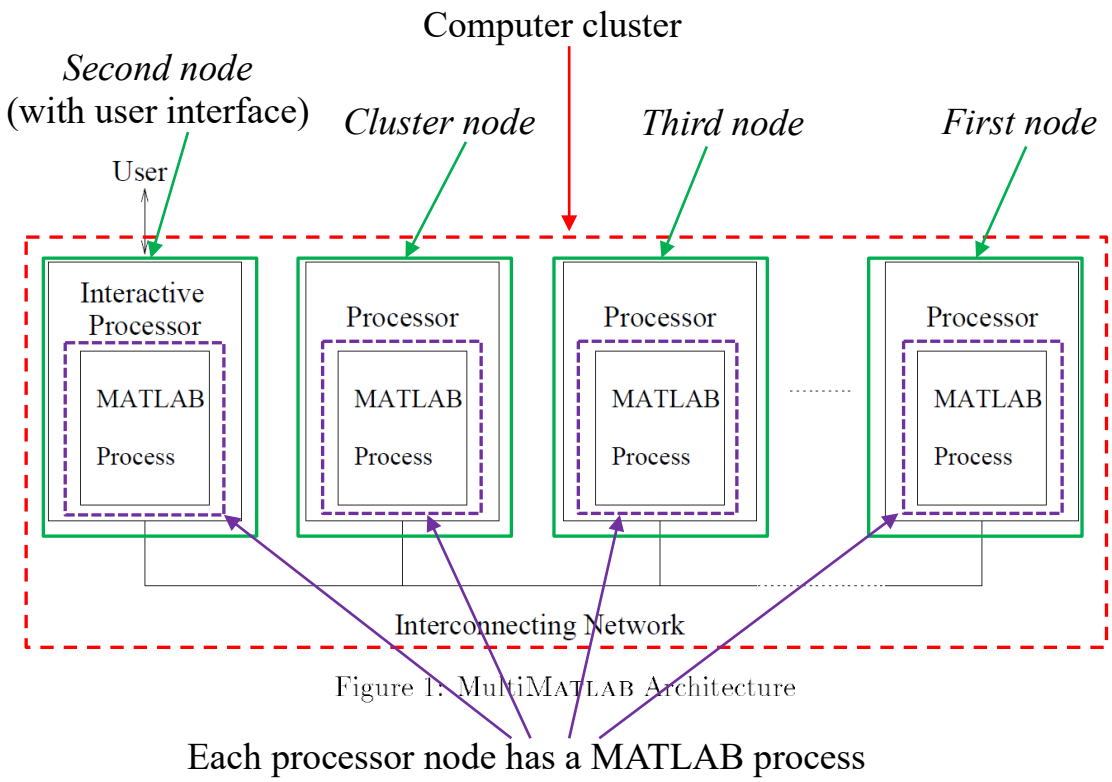
**Ex.1005, FIGs. 7 and 8 (annotated)**

**384.** The IBM SP2 system is used to "execute parallel programs on … a networked cluster of [IBM SP2] RS/6000 processors." Ex.1008, 1. Menon describes the IBM SP2 system being used to perform mathematical computations in parallel, where a user provides a mathematical formula into a user interface on a first node of the IBM SP2 and then the other nodes perform mathematical computations in parallel to provide the user an answer. Ex.1005, 6-8, 11-14. For example, Menon describes the MultiMATLAB architecture as comprising a series of processors that each run a MATLAB process and communicate with each other, explaining that:

> In the MultiMATLAB architecture, illustrated in Figure 1, each processor in a parallel platform individually runs a MATLAB process. Each process is then provided with the ability to communicate with other processes through a communication layer that runs over the parallel platform's interconnection network. The user interacts directly with one MATLAB process, called the interactive process, and operates within that process's MATLAB environment. Other MATLAB processes await commands from the interactive process. These other processes are used either by explicitly sending MATLAB commands to them or by executing routines that, in turn, use them.

Ex.1005, 3. FIG. 1 of Menon depicting the MultiMATLAB architecture with a plurality of processor nodes each executing a MATLAB process is shown below.[3]

[3] The functionality recited for the "first," "second," and "third" nodes of claim 35 is different than the functionality recited for the similarly named "first," "second," and "third" nodes of claim 1.




**Ex.1005, FIG. 2 (annotated)**

**385.** Menon provides another example where the IBM SP2 system implementation of the MultiMATLAB architecture is used to perform mathematical computations in parallel. Ex.1005, 11-13. Specifically, Menon teaches computing the conjugate gradients algorithm using the IBM SP2 system implementation of the MultiMATLAB architecture where the computations are evaluations of mathematical expressions, such as performing single matrix-vector multiplication, performed by MATLAB processes executing on multiple processors in parallel:

> The conjugate gradients algorithm is iterative method to solve a linear system of the form $Ax = b$, where $A$ is a symmetric positive definite matrix and $x$ and $b$ are

> vectors. The computational core of this method is a single matrix-vector multiplication at each iteration
>
> …
>
> the matrix A and the different vectors are each distributed by row over all processes as in Figure 6. In each iteration, a process only does the computation required for its local data. However, communication is required for two different operations: the dot-products needed to compute *rho* and *alpha* and the matrix-vector multiply needed to compute *w_local*. The dot products only require communication of a single value over all processes. On the other hand, the matrix-vector multiplication needed for *w_local* requires the global vector *p*, and, thus, more expensive communication.

Ex.1005, 11-13 (emphasis original). FIG. 4 of Menon shown below illustrates the parallel evaluation of mathematical expressions by the nodes of the MultiMATLAB architecture.

code to evaluate mathematical expressions in parallel

```
r = b;
rho = r'*r;
k = 0;
while((k < kmax))
  k = k+1;
  if (k == 1)
    p = r;
  else
    beta = rho/oldrho;
    p = r + beta*p;
  end

  w = A * p;

  alpha = rho/(p'*w);
  x = x + alpha * p;
  r = r - alpha * w;
  oldrho = rho;
  rho = r'*r;
end
```

a. Sequential MATLAB

```
r_local = b_local;
rho = Sum(r_local'*r_local);
k = 0;
while((k < kmax))
  k = k+1;
  if (k == 1)
    p_local = r_local;
  else
    beta = rho/oldrho;
    p_local = r_local + beta*p_local;
  end

  p = Gather(p_local);
  w_local = A_local * p;

  alpha = rho/(Sum(p_local'*w_local));
  x_local = x_local + alpha * p_local;
  r_local = r_local - alpha * w_local;
  oldrho = rho;
  rho = Sum(r_local'*r_local);
end
```

b. Message Passing MATLAB

Figure 4: Conjugate Gradients

**Ex.1005, FIG. 4 (annotated)**

**386.** Accordingly, Menon's individual processor nodes of the MultiMATLAB architecture that evaluate mathematic expressions in parallel as implemented on an IBM SP2 multiprocessor system, in view of RS6000 describing 128 nodes for parallel processing, renders obvious *a computer cluster node for evaluating expressions in parallel with other computer cluster nodes*.

**b.** **[35.1.1]** ***a hardware processor configured to access one or more non-transitory memory devices comprising program code for a single-node kernel that, when executed, causes the hardware processor to interpret user instructions, to evaluate mathematical expressions,***

***and to produce results of mathematical expression evaluation,***

**387.** Menon, Trefethen, and RS600 renders obvious *a hardware processor configured to access one or more non-transitory memory devices comprising program code for a single-node kernel that, when executed, causes the hardware processor to interpret user instructions, to evaluate mathematical expressions, and to produce results of mathematical expression evaluation* because Menon discloses IBM SP2's multiprocessor implementation having a node with a processor that accesses hard drives and RAM, as taught by RS6000, in view of Menon teaching that each node of the IBM SP2 executes a MATLAB process with a symmetric multiprocessor for evaluating mathematical expressions input by a user via the interface processor, further in view of Trefethen's teaching that MultiMATLAB displays the results of the mathematical computations.

**388.** **First**, Menon teaches each MATLAB process runs on a processor, explaining that "[i]n the MultiMATLAB architecture, … each processor in a parallel platform individually runs a MATLAB process." Ex.1005, 3. The MultiMATLAB architecture is implemented on the IBM SP2, "a modern high performance distributed memory multiprocessor." Ex.1005, 5. RS6000 teaches that each processor node of the IBM SP2 can "have either a Symmetric MultiProcessor (SMP) configuration or a uniprocessor configuration." Ex.1007, 2. The SMP ("*hardware processor*") has "four, eight, twelve, or sixteen 375 MHz 630FP 64-bit

processors." Ex.1007, 9. Accordingly, Menon and RS6000 disclose each processor of the MultiMATLAB architecture having *a hardware processor*.

**389. Second**, Menon discloses that each one of the IBM SP2 processors has a MATLAB process, explaining that "[i]n the MultiMATLAB architecture, illustrated in Figure 1, each processor in a parallel platform individually runs a MATLAB process." Ex.1005, 3. FIG. 1 of Menon below shows a MATLAB process executing on each IBM SP2 processor of the MultiMATLAB architecture. Ex.1005, FIG. 1.

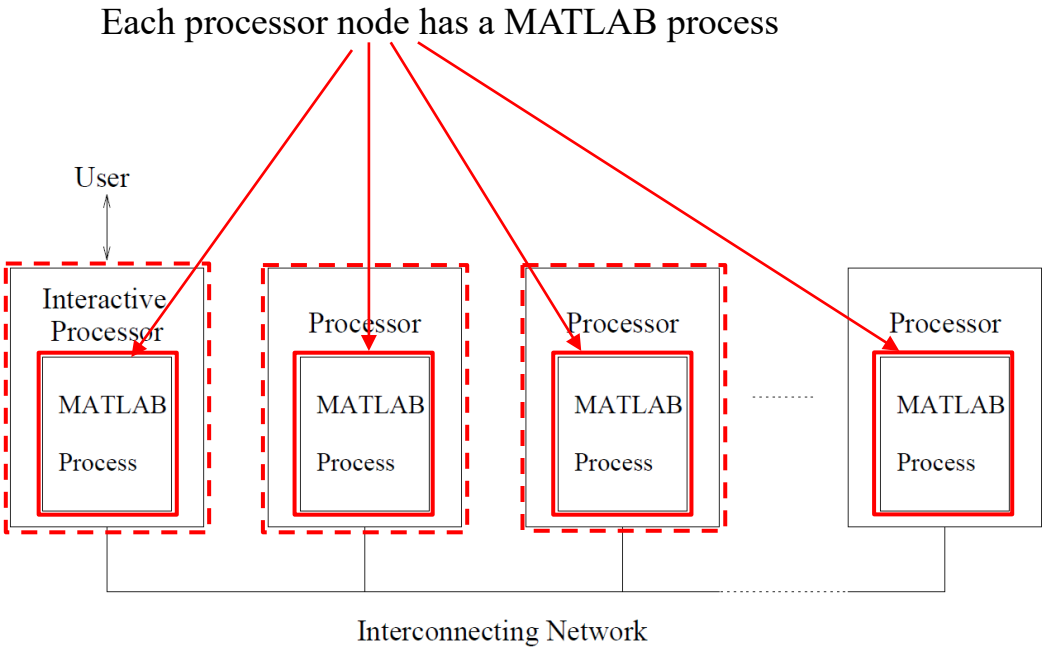
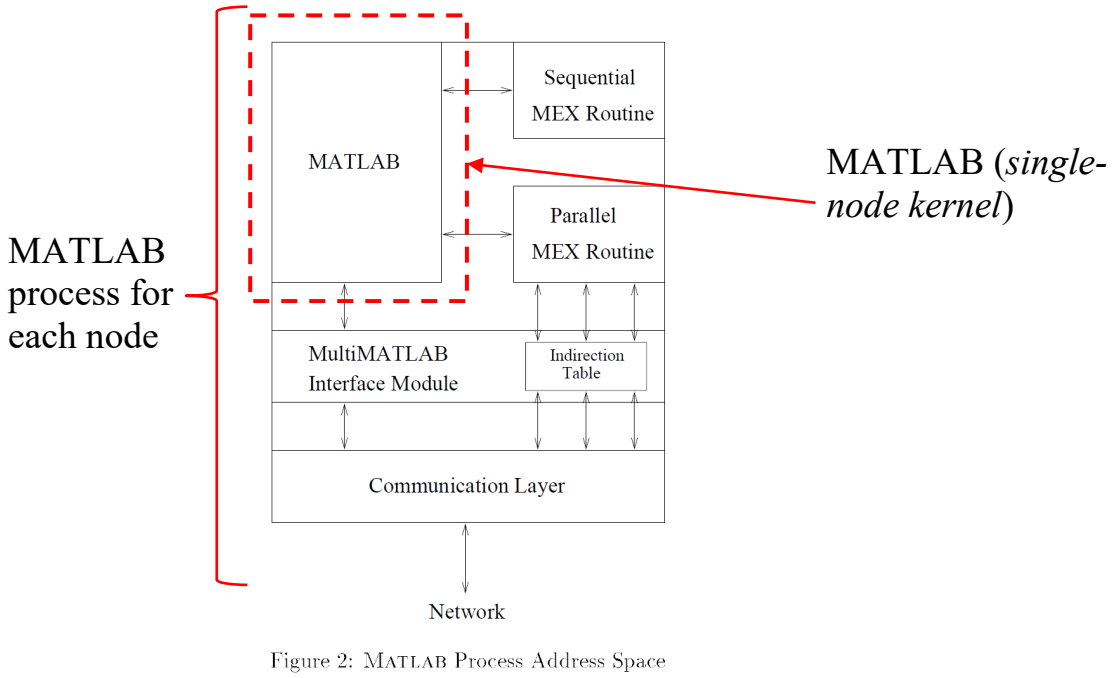



Figure 1: MultiMATLAB Architecture

**Ex.1005, FIG. 1 (annotated)**

**390.** Menon explains that each MATLAB process contains MATLAB, "the numerical computing environment of choice on uniprocessors for hundreds of

thousands of engineers and scientists." Ex.1005, 1, FIG. 2 (showing a MATLAB process containing MATLAB). MATLAB is a "scientific computing environment" that provides users with "an extensive library of high quality numerical routines." Ex.1005, 1. FIG. 2 of Menon showing a MATLAB process containing MATLAB is depicted below. MATLAB of each MATLAB process on a processor of the MultiMATLAB architecture teaches *a single-node kernel*.




**Ex.1005, FIG. 2 (annotated)**

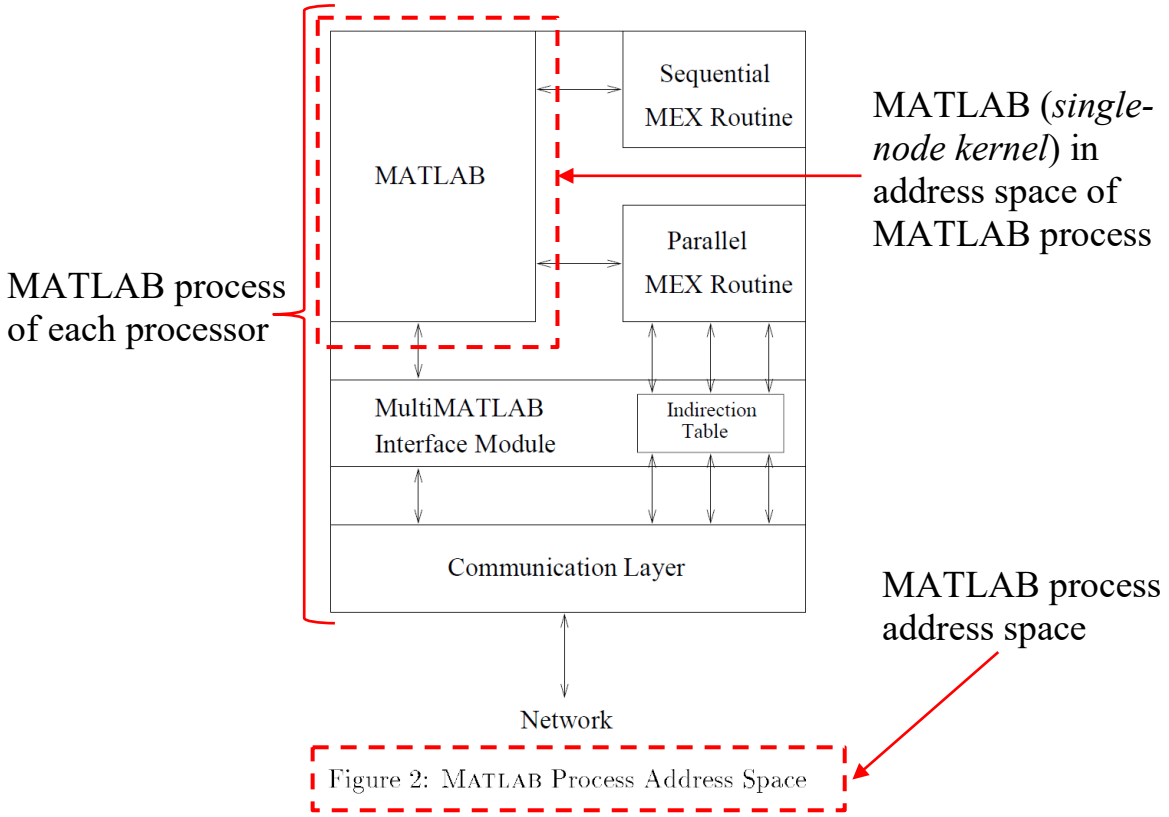
**391. Third**, Menon teaches that a processor accesses MATLAB from *one or more non-transitory memory devices*, because Menon describes the MATLAB process, which includes MATLAB, as being in an "address space" and further describes the building of "a table of pointers to all functions and data" that the

parallel MEX Routines use "to access any function provided by the underlying communication layer" during the initialization of the MATLAB process. Ex.1005, FIG. 2.

**392.** Further, RS6000 discloses that IBM SP2, on which the conjugate gradients algorithm is implemented, have computer readable media in the form of internal and external hard disk drives, or memory cards. Ex.1007, 6, 10 (disclosing IBM SP2 processors have internal and external hard disk drives), 10 (disclosing IBM SP2 processors have "one to four memory cards" that support a "maximum of 64 GB"). A POSITA would then understand Menon's "address space" to refer to a "a list of memory locations from some minimum (usually 0) to some maximum, which the process can read and write. The address space contains the executable program, the program's data, and its stack." *See* Ex.1018, 67-68. Because Menon's address space refers to a computer's memory, a POSITA would recognize that the MATLAB process and MATLAB are in *a non-transitory memory device*, and that MATLAB (*single-node kernel*) is accessed from *a non-transitory memory device*, i.e., *a hardware processor* accesses MATLAB (*single-node kernel*) from *one or more non-transitory memory devices*.

**393.** Further, because MATLAB is a "MATLAB software", a POSITA would recognize MATLAB on a processor is *program code for a single-node kernel* that is executed by *a hardware processor*. Ex.1006, 2; Ex.1005, 3.

Accordingly, a POSITA would recognize that *a hardware processor* accesses MATLAB (*a program code for a single-node kernel*) from *one or more non-transitory memory devices*. FIG. 2 of Menon below shows MATLAB and the MEX Routines as part of the MATLAB process address space.




**Ex.1005, FIG. 2 (annotated).**

**394. Fourth**, Menon teaches MATLAB (*single-node kernel*) is capable of causing the *hardware processor* to evaluate mathematical expressions, as illustrated by the "MultiMATLAB implementations of the conjugate gradients algorithm on the IBM SP2" discussed in Menon. Ex.1005, 11-13. The "parallel MATLAB implementation of conjugate gradients" includes the step of "matrix A and the

different vectors [] each [being] distributed by row over all processes." Ex.1005, 11. FIG. 6 of Menon shown below illustrates this step where "[m]atrix A and vector x [are] distributed by row over *P* processors" when the MultiMATLAB implementation includes "*p* MATLAB processes." Ex.1005, FIG. 6, 6 (emphasis original).

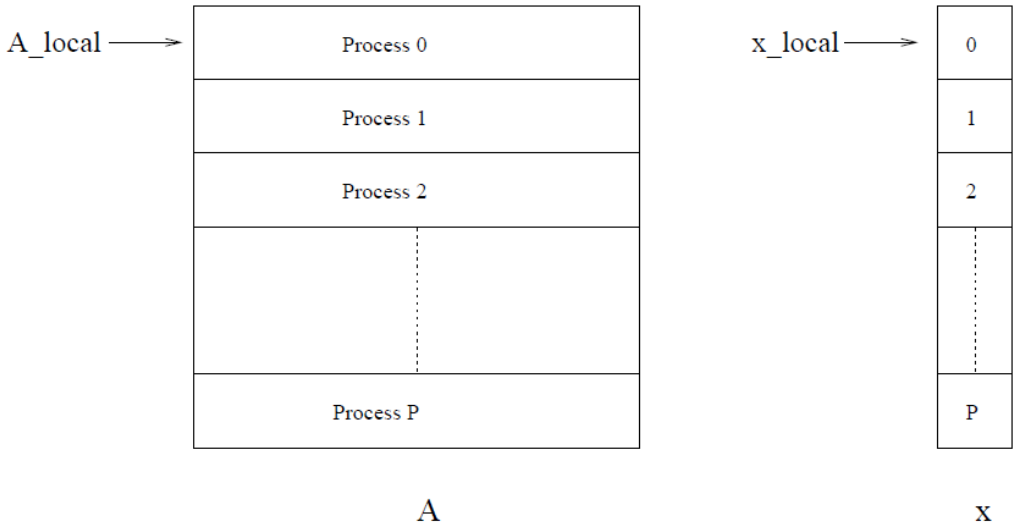



Figure 6: Matrix $A$ and vector $x$ distributed by row over $P$ processors

**Ex.1005, FIG. 6**

**395.** Each MATLAB process, where "the MEX Routine … [is] run from MATLAB," performs its own computation locally but exchanges data with other MATLAB processes, as explained by Menon:

> In each iteration, a process only does the computation required for its local data. However, communication is required for two different operations: the dot-products

> needed to compute *rho* and *alpha* and the matrix-vector multiply needed to compute *w_local*. The dot products only require communication of a single value over all processes. On the other hand, the matrix-vector multiplication needed for *w_local* requires the global vector *p*, and, thus, more expensive communication.

Ex.1005, 11-13. Accordingly, MATLAB (*a single-node kernel*) of a MATLAB process cause *a hardware processor* that includes the MATLAB process to perform conjugate gradients algorithm computations.

**396.** The conjugate gradients algorithm computations are *evaluat[ions of] mathematical expressions*, such as performing single matrix-vector multiplication: "[t]he conjugate gradients algorithm is iterative method to solve a linear system of the form $Ax = b$, where $A$ is a symmetric positive definite matrix and $x$ and $b$ are vectors. The computational core of this method is a single matrix-vector multiplication at each iteration." Ex.1005, 11. FIG. 4 of Menon shown below illustrates the *evaluat[ions of] mathematical expressions* by MATLAB and the MEX Routines. Ex.1005, FIG. 4. Accordingly, Menon teaches that MATLAB (*a single-node kernel*) of a MATLAB process cause *a hardware processor* that includes the MATLAB process to *evaluate mathematical expressions*.

code to *evaluate mathematical expressions*

```
r = b;
rho = r'*r;
k = 0;
while((k < kmax))
  k = k+1;
  if (k == 1)
    p = r;
  else
    beta = rho/oldrho;
    p = r + beta*p;
  end

  w = A * p;

  alpha = rho/(p'*w);
  x = x + alpha * p;
  r = r - alpha * w;
  oldrho = rho;
  rho = r'*r;
end
```

a. Sequential MATLAB

```
r_local = b_local;
rho = Sum(r_local'*r_local);
k = 0;
while((k < kmax))
  k = k+1;
  if (k == 1)
    p_local = r_local;
  else
    beta = rho/oldrho;
    p_local = r_local + beta*p_local;
  end

  p = Gather(p_local);
  w_local = A_local * p;

  alpha = rho/(Sum(p_local'*w_local));
  x_local = x_local + alpha * p_local;
  r_local = r_local - alpha * w_local;
  oldrho = rho;
  rho = Sum(r_local'*r_local);
end
```

b. Message Passing MATLAB

Figure 4: Conjugate Gradients

**Ex.1005, FIG. 4 (annotated)**

**397. Fifth**, Menon discloses that MATLAB (*single-node kernel*) of a MATLAB process on each processor of the MultiMATLAB architecture await for MATLAB commands from a user and execute routines based on those MATLAB commands when received. Ex.1005, 3-4. Specifically, Menon teaches that to have user instructions executed on all the MATLAB processes of the MultiMATLAB architecture, a "user interacts directly with one MATLAB process, called the interactive process, and operates within that process's MATLAB environment." Ex.1005, 3. For example, the user may use the "Eval routine for parallel evaluation," which is a "routine [that] allows users to execute commands on the other processes." Ex.1005, 4. "Other MATLAB processes await commands from

the interactive process. These other processes are used either by explicitly sending MATLAB commands to them or by executing routines that, in turn, use them." Ex.1005, 3.

**398.** The parallel execution of the user's instructions is facilitated by the MEX Routines, which provide the MultiMATLAB architecture "all [its] parallel functionality." Ex.1005, 3. On the interactive MATLAB process, the user's "Eval routine is implemented as a MEX Routine which accepts a vector of MATLAB process IDs and a MATLAB command. This routine sends the commands over the underlying communication layer to the processes corresponding to the specified IDs." Ex.1005, 4. On the other non-interactive MATLAB processes, "a separate top-level MEX Routine is immediately run upon initialization of the system. This MEX Routine runs in a loop in which it waits for MATLAB commands to arrive from the interactive process and executes them in its own MATLAB environment." Ex.1005, 4. The interactive as well as the non-interactive MATLAB processes' MEX Routines use their respective MATLAB process's MultiMATLAB interface module to access the interconnection network and communicate with each other, as "[a]ll parallel MEX Routines access the underlying communication layer and, hence, the network, through the MultiMATLAB interface module." Ex.1005, 4.

**399.** Trefethen provides an example illustration where a user "connected to a node of the IBM SP2, running MATLAB" is operating in a MultiMATLAB

architecture where "6 MultiMATLAB processes [are] running." Ex.1006, 3. If the user types the command Eval( 'ID' ), the result is

ans = 0

ans = 1

ans = 5

ans = 2

ans = 3

ans = 4,

indicating that "the MultiMATLAB command ID" is executed on each MATLAB process. Ex.1006, 3-4. That is the case because "Eval routine is implemented as a MEX Routine which accepts a vector of MATLAB process IDs and a MATLAB command. This routine sends the commands over the underlying communication layer to the processes corresponding to the specified IDs." Ex.1005, 4. "[E]ach command passed to Eval was executed on all MATLAB processes" unless a "subset of the processes [is selected] by passing two arguments to the Eval command, the first being a vector of process IDs." Ex.1006, 4. In other words, the interactive as well as all the non-interactive MATLAB processes of "the 6 MultiMATLAB processes" receive and execute thereon commands received from user instructions, generating the above result. Ex.1006, 3.

**400.** Accordingly, Menon teaches that each processor of the MultiMATLAB architecture has MATLAB (*single-node kernel*) that await for and receive commands from instructions provided by a user to execute the received commands (*interpret user instructions*) in its own MATLAB environment.

**401.** **Sixth**, Trefethen discloses that the MultiMATLAB architecture is capable of "produc[ing] plots in a distributed fashion that are then sent to the user's screen. This can be particularly useful when one wishes to monitor the progress of computations on several processors graphically." Ex.1006, 7. Trefethen explains that "calculations with a geometric flavor" can be plotted in the MATLAB processes of each processor node of the MultiMATLAB architecture, explaining that one can set up "a MATLAB figure window in each process" and then "arrange them [the calculations] in a grid on the screen." Ex.1006, 7. An example of the computations can be "the pseudospectra of a 64 by 64 matrix known as the 'Grcar matrix'." Ex.1006, 7. Accordingly, the MultiMATLAB architecture producing plots of calculations such as the pseudospectra of a matrix graphically teaches *to produce results of mathematical expression evaluation*.

**402.** Accordingly, IBM SP2's multiprocessor implementation having a node with a processor that accesses hard drives and RAM in view of Menon teaching that each node of the IBM SP2 executes a MATLAB process for evaluating mathematical expressions input by a user via the interface processor, further in view

of Trefethen's teaching that MultiMATLAB displays the results of the mathematical computations, renders obvious that *a hardware processor configured to access one or more non-transitory memory devices comprising program code for a single-node kernel that, when executed, causes the hardware processor to interpret user instructions, to evaluate mathematical expressions, and to produce results of mathematical expression evaluation*.

**c. [35.1.2] *wherein the hardware processor comprises multiple processor cores;***

**403.** As discussed in limitation [35.1.1], RS6000 teaches that each processor node of the IBM SP2 can be the "375 MHz POWER3 SMP High Node (F/C 2058)," where the SMP ("*hardware processor*") has "four, eight, twelve, or sixteen 375 MHz 630FP 64-bit processors." Ex.1007, 2, 9. Accordingly, RS6000 discloses or at least renders obvious *wherein the hardware processor comprises multiple processor cores*.

**d. [35.2] *a user connection interface configured to receive a command to start a cluster initialization process for a computer cluster;***

**404.** Menon discloses or at least renders obvious *a user connection interface configured to receive a command to start a cluster initialization process for a computer cluster* because it teaches that a processor of the MultiMATLAB architecture has a set of components to provide interconnection with other modules, including the "MEX Routines," "MultiMATLAB interface module," an "indirection

table" and "communication layer" (collectively, *a user connection interface*) to receive a user command to initialize the MATLAB processes (*a command to start a cluster initialization process*) on the processors of the MultiMATLAB architecture (*for a computer cluster*).

**405.** Menon discloses that MATLAB processes of a MultiMATLAB architecture implemented on the IBM SP2 are started by a user using IBM's POE:

> In this implementation, all MATLAB processes are started via POE, IBM's Parallel Operating Environment. For each of *p* MATLAB processes, POE assigns an identification number between 0 and *p* - 1. The process numbered 0 is considered the interactive MATLAB. All other processes wait for commands from the interactive MATLAB process.

Ex.1005, 6 (emphasis original). The interactive MATLAB is the MATLAB process with which the user interacts. Ex.1005, 3. POEref teaches that the user invokes the initialization process using the "poe" command:

> When you invoke **poe**, the Partition Manager allocates processor nodes for each task and initializes the local environment. It then loads your program, and reproduces your local environment, on each processor node. The Partition Manager also passes the option list to each remote node. … If you are using the dynamic message passing interface, the appropriate communication

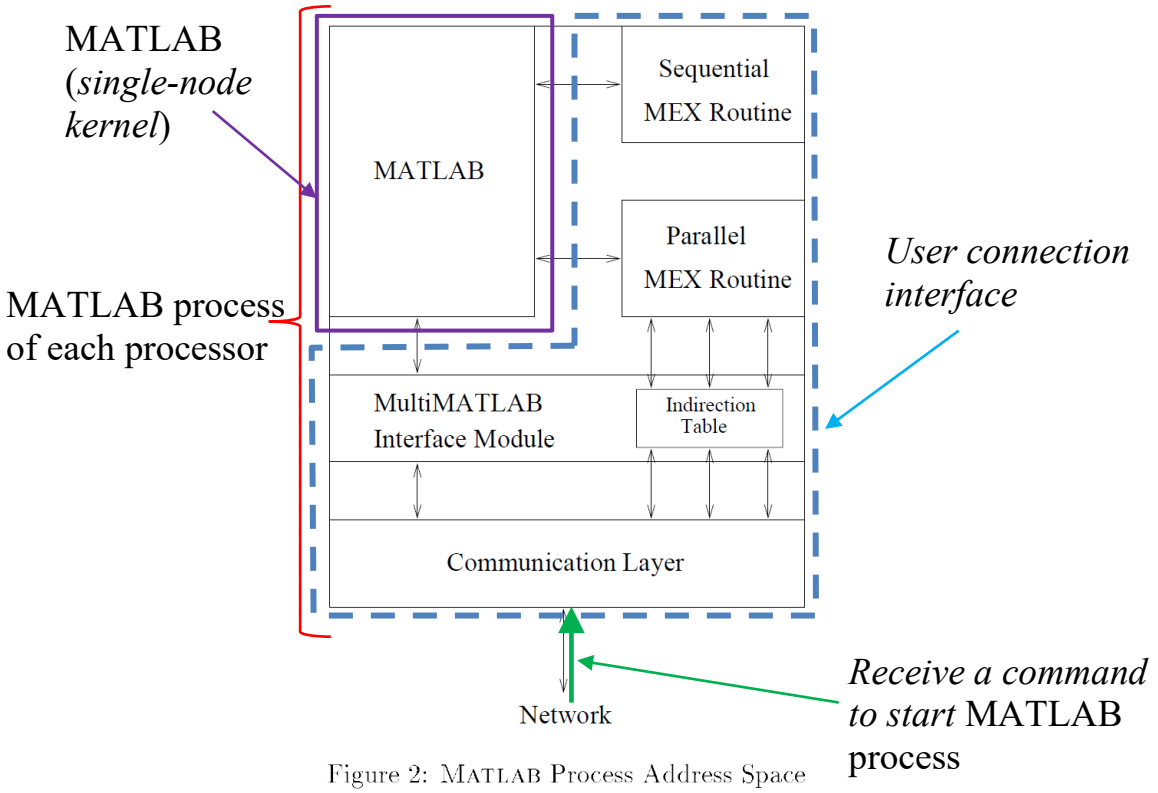
> subsystem library implementation (IP or US) is automatically loaded at this time.

Ex.1008, 28 (emphasis original).

**406.** In other words, when a user on an IBM SP2 invokes **poe**, a local environment, i.e., a MATLAB process, is initiated on the IBM SP2 processor with which the user is interacting and reproduced on each of the other IBM SP2 processors (called "remote nodes" by POEref), and the MATLAB processes (called "non-interactive processes" by Menon) on these remote nodes await for commands from the interactive MATLAB process. Ex.1005, 6, 4 (calling the MATLAB processes on the remote nodes "non-interactive processes"); Ex.1008, 28, 3 (discussing "remote nodes"). Specifically, "[o]n non-interactive processes, a separate top-level MEX Routine is immediately run upon initialization of the system. This MEX Routine runs in a loop in which it waits for MATLAB commands to arrive from the interactive process and executes them in its own MATLAB environment." Ex.1005, 4.

**407.** Menon discloses that each processor of "the MultiMATLAB architecture … individually runs a MATLAB process" that includes a set of components to provide interconnection with other modules, including the "MEX Routines," "MultiMATLAB interface module," an "indirection table" and "communication layer." Ex.1005, 3, 5. Menon describes that the MultiMATLAB

interface module on each processor is responsible for the underlying communication layer with other processors, explaining that "[t]he interface module, shown in Figure 2, is responsible for initializing an underlying communication layer, such as MPI …, or any other package available on the platform, and exposing it to the rest of the system." Ex.1005, 3. FIG. 2 of Menon shows the relationship between the MultiMATLAB interface module, the communication layer, and the single-node kernel is shown below.




**Ex.1005, FIG. 2 (annotated)**

**408.** Accordingly, Menon teaches the use of IBM's POE to start all the MATLAB processes (*receive a command to start a cluster initialization process*) on

the processors of the MultiMATLAB architecture (*for a computer cluster*). The MATLAB processes on each processor (each, a *node*) receive the start command via the communication layer in the set of components that provide interconnection with other modules (*user connection interface*) of the MATLAB process, starting the MATLAB process on that processor. Thus, Menon teaches that the processor has a communication layer in the set of components that provide interconnection with other modules (*user connection interface*) of the MATLAB process that receives a start command that causes the MATLAB process to start and await further commands, thereby rendering obvious *a user connection interface configured to receive a command to start a cluster initialization process for a computer cluster*.

#### e. [35.3] *a mechanism to communicate results of evaluation with other computer cluster nodes using a peer-to-peer architecture; and*

**409.** Menon and Trefethen render obvious *a mechanism to communicate results of evaluation with other computer cluster nodes using a peer-to-peer architecture* because Menon describes MEX routines, having code to implement MPI calls, that enable SPMD point-to-point communication so that each processor can communicate directly with each other over an interconnecting network, and Trefethen discloses that the SPMD point-to-point communication enables a peer-to-peer architecture where mathematical expression evaluation results from a first

MATLAB process on a first processor that is communicated to a second MATLAB process on a second processor, and the mathematical expression results obtained from the second MATLAB process on the second processor are communicated to a third MATLAB process on a third processor, and so on.

**410. First**, as shown in [26.2] and [26.6.2], Menon renders obvious *a mechanism to communicate results of evaluation with other computer cluster nodes* because it teaches MEX routines having code that implement standard variants of MPI calls to enable SPMD point-to-point communication so that the MATLAB processes can communicate tasks and data directly with each other.

**411.** Specifically, Menon discloses that MultiMATLAB uses "MEX routines" and that "[a]ll parallel functionality in the MultiMATLAB system is provided through MEX routines." Ex.1005, 3. For example, "the MultiMATLAB system provides users with an Eval routine for parallel evaluation. This routine allows users to execute commands on the other processes." Ex.1005, 3-4. "[T]he Eval routine is implemented [on the interactive process] as a MEX routine which accepts a vector of MATLAB process IDs and a MATLAB command" that the MEX Routine sends to the MATLAB processes identified by the MATLAB process IDs. Ex.1005, 4. "On non-interactive processes, a separate top-level MEX routine is immediately run upon initialization of the system. This MEX routine runs in a loop in which it waits for MATLAB commands to arrive from the interactive process

and executes them in its own MATLAB environment." Ex.1005, 4. The MEX Routines send the "commands over the underlying communication layer to the processes corresponding to the specified IDs." Ex.1005, 4.

| MultiMATLAB commands | |
|---|---|
| Starting and stopping MultiMATLAB (MPICH only) | |
| Start(n) | Initializes n remote MATLAB processes. |
| Quit | Closes remote MATLAB processes. |
| Running commands on multiple processors | |
| Nproc | Returns number of MATLAB processes. |
| ID | Returns ID of calling MATLAB process. |
| Eval(pid,'command') | Evaluates a command on one or more MATLAB process. |
| Master/Slave Communication | |
| Put(pid,dataname) | Put data from the master process onto one or more remote processes. |
| Get(pid,dataname) | Get data from a remote process onto the master process. |
| SPMD Communication | |
| Send(pid,data) | Send data from one process to another. |
| Recv(pid) | Receive data sent from another process. |
| Barrier | Synchronize processes. |
| Bcast(pid,data) | Broadcasts data from processor pid to all processes. |
| Sum(data) | Adds data across all processors to form a global sum. |
| Collect(data) | Collects local data segments into a single global one. |

Table 1: Selected commands in the MultiMATLAB system

Commands for point-to-point communication

**Ex.1005, Table 1 (annotated)**

**412.** The underlying communication layer of the MultiMATLAB architecture "runs over the parallel platform's interconnection network" and "provide[s] [a MATLAB process on a processor] with the ability to communicate with other processes." Ex.1005, 3. An example of the underlying communication layer is MPI. Ex.1005, 3. Table 1 of Menon below shows MultiMATLAB commands implemented as MEX Routines. Further, FIG. 2 and FIG. 3 of Menon

below show the MEX Routines of a MATLAB process and the interconnection network of the MultiMATLAB architecture linking the processors thereof, respectively.

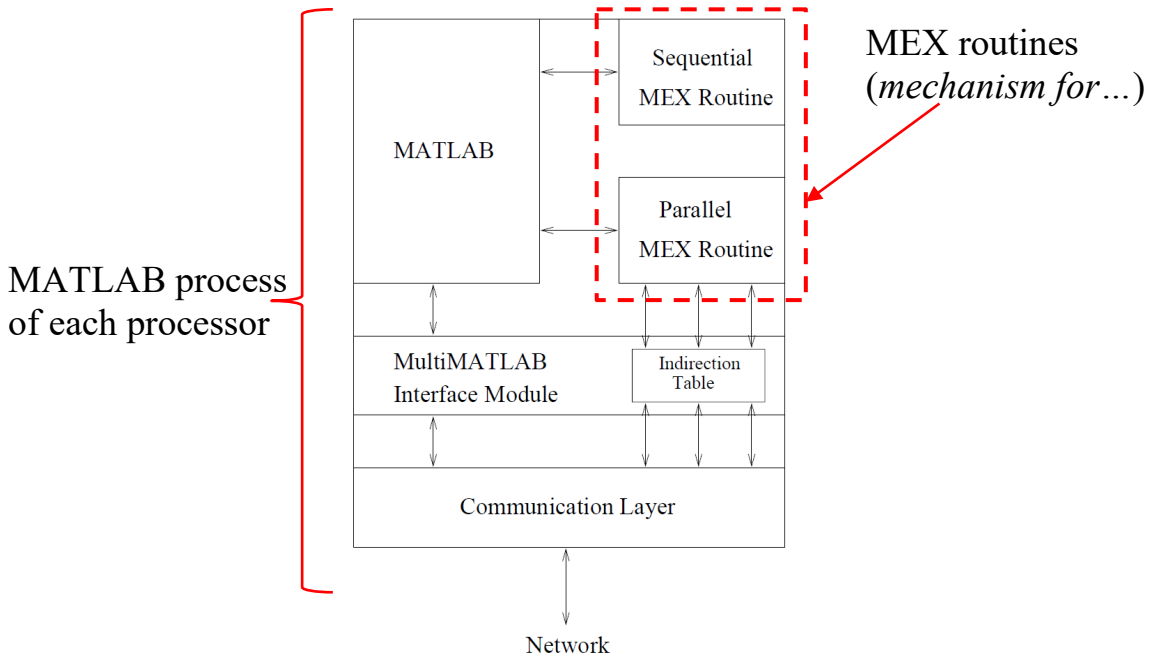
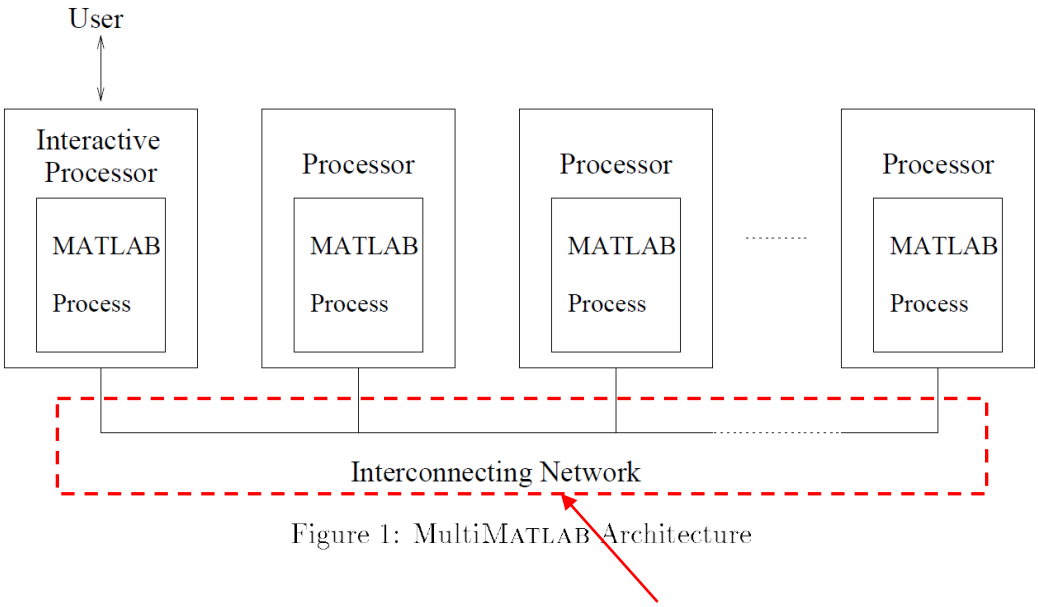



Figure 2: MATLAB Process Address Space

**Ex.1005, FIG. 2 (annotated)**




Interconnecting network connecting each MATLAB process on a processor to any other in the MultiMATLAB architecture

**Ex.1005, FIG. 1 (annotated)**

**413.** Trefethen discloses that the commands used in MultiMATLAB are "written as a C file" and "interfaced to MATLAB via the standard MATLAB Fortran/C/C++ interface known as MEX." Ex.1006, 10. The "[h]igher-level MultiMATLAB commands are usually built on higher-level MPI commands." Ex.1006, 10. Trefethen teaches that "MultiMATLAB is currently built upon the standard send and receive variants" of MPI. Ex.1006, 10. That is, "[m]ore general point-to-point communication is accomplished by send and receive commands, which can be executed on any of the MATLAB processes." Ex.1006, 5.

**414.** Trefethen also discloses sending tasks and data to other processors using these MultiMATLAB commands. Ex.1006, 3-7. "The standard

MultiMATLAB command for executing commands on one or more processors is Eval." Ex.1006, 3. For example,

> The command
>
> Eval( 'ID^ID' )
>
> might produce
>
> ans = 1
>
> ans = 1
>
> ans = 256
>
> ans = 3125
>
> ans = 27
>
> ans = 4.

Ex.1006, 4. That is, Eval( 'ID^ID' ) sends the task "ID^ID" to each processor ID to perform the task "ID^ID" using the data ID. Ex.1006, 4.

**415.** Trefethen discloses that a user can specify which MATLAB processes should perform the tasks, explaining that "one can select a subset of the processes by passing two arguments to the Eval command, the first being a vector of process IDs." Ex.1006, 4. For example,

> Eval( [4 5] , 'cond(hilb(ID))' )
>
> might return
>
> ans = 1.5514e+04

ans = 4.7661e+05

the condition numbers of the Hilbert matrices of

dimensions 4 and 5.

Ex.1006, 4. That is, the MATLAB processes on processor ID=4 and ID=5 each perform the task "cond(hilb(ID))" using the data ID, i.e., MATLAB process on processor ID=4 performs the task "cond(hilb(ID = 4))" using the data ID=4 and generates the result "ans = 1.5514e+04," "the condition number[] of the Hilbert [matrix] of dimension[] 4," and MATLAB process on processor ID=5 performs the task "cond(hilb(ID = 5))" using the data ID=5 and generates the result "ans = 4.7661e+05," "the condition number[] of the Hilbert [matrix] of dimension[] 5." Ex.1006, 4.

**416.** Thus, the MEX Routines that contain program code to implement the standard MPI send and receive calls to send commands (e.g., tasks) and data directly to other processes renders obvious an MPI module.

**417.** **Second**, Menon's MEX Routines are implemented using "point to point … communication" paradigm used to enable communication between the MATLAB processes operating in parallel in the MultiMATLAB architecture. Ex.1005, 9. Menon explains that, in contrast to a "master/slave paradigm" that "provid[es] a very simple mechanism for coarse-grain distributed computing" and "is often adequate in situations where little or no communication is required,"

Menon's SPMD/Message Passing paradigm "accommodate[s] applications requiring a finer degree of communication" and provides "point to point … communication." Ex.1005, 9. SPMD [single program multiple data] refers to a "style of parallel programming, where all processors use the same program, though each has its own data." Ex.1009, 1. Table 1 of Menon below shows example point-to-point commands for SPMD communication paradigm, such as the "Send(pid,data)" command that "send[s] data from one process to another," the "Recv(pid)" command that "receive[s] data sent from another process," and the "Bcast(pid,data)" command that "broadcasts data from processor pid to all processes." Ex.1005, Table 1. The SPMD point-to-point communication paradigm where each node can communicate tasks and data directly with each other node renders obvious *a peer-to-peer architecture*.

| MultiMATLAB commands | |
|---|---|
| Starting and stopping MultiMATLAB (MPICH only) | |
| Start(n) | Initializes n remote MATLAB processes. |
| Quit | Closes remote MATLAB processes. |
| Running commands on multiple processors | |
| Nproc | Returns number of MATLAB processes. |
| ID | Returns ID of calling MATLAB process. |
| Eval(pid,'command') | Evaluates a command on one or more MATLAB process. |
| Master/Slave Communication | |
| Put(pid,dataname) | Put data from the master process onto one or more remote processes. |
| Get(pid,dataname) | Get data from a remote process onto the master process. |
| SPMD Communication | |
| Send(pid,data) | Send data from one process to another. |
| Recv(pid) | Receive data sent from another process. |
| Barrier | Synchronize processes. |
| Bcast(pid,data) | Broadcasts data from processor pid to all processes. |
| Sum(data) | Adds data across all processors to form a global sum. |
| Collect(data) | Collects local data segments into a single global one. |

Table 1: Selected commands in the MultiMATLAB system

Commands for point to point communication

**Ex.1005, Table 1 (annotated)**

**418. Third**, Trefethen describes a MATLAB process on a processor of the MultiMATLAB architecture using the SPMD point-to-point commands to communicate *results of* mathematical expression *evaluation* to another MATLAB process on another processor. Ex.1006, 3-7. Specifically, Trefethen provides the following example code in which a MATLAB process communicates a result of a mathematical evaluation to a next MATLAB process, which in turn communicates the next mathematical result to the next MATLAB process, and so on:

```
if ID==0                % first process: send
   a = 1
   Send(ID+1,a)
elseif ID == Nproc-1    % last process: receive and double
   a = 2*Recv
else                    % middle processes: receive, double, and send
   a = 2*Recv
   Send(ID+1,a)
end;
```

Process 0 creates the variable a with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of a, and sends it along to process 2; and so on. If there are six processors the command `Eval( 'cycle' )` produces the output

```
a = 1
a = 2
a = 4
a = 8
a = 16
a = 32
```

Point-to-point communications of mathematical *evaluation results* between MATLAB processes on different processors

Ex.1006, 6 (annotated).

**419.** Accordingly, Menon describes MEX routines, having code to implement MPI calls, that enable SPMD point-to-point communication so that each processor can communicate directly with each other over an interconnecting network, and Trefethen discloses that the SPMD point-to-point communication enables a peer-to-peer architecture where mathematical expression evaluation results from a first MATLAB process on a first processor that is communicated to a second MATLAB process on a second processor, and the mathematical expression results obtained from the second MATLAB process on the second processor are communicated to a third MATLAB process on a third processor, and so on, rendering obvious *a mechanism to communicate results of evaluation with other computer cluster nodes using a peer-to-peer architecture*.

**f. [35.4.1] *program code that, when executed, is capable of causing the hardware processor to: receive calls from a second node comprising a second hardware processor configured to access a second memory comprising program code for a user interface and program code for a second single-node kernel, the second single-node kernel configured to interpret user instructions and distribute calls to at least one of a plurality of other nodes for execution;***

**420.** Menon discloses *program code that, when executed, is capable of causing the hardware processor to: receive calls from a second node comprising a second hardware processor configured to access a second memory comprising program code for a user interface and program code for a second single-node kernel, the second single-node kernel configured to interpret user instructions and distribute calls to at least one of a plurality of other nodes for execution* because it teaches (i) cluster node has program code executed by a processor, (ii) the cluster node receives calls from an interactive processor (*second node*) of the MultiMATLAB architecture, (iii) the interactive processor (*second node*) has a user interface, (iv) the MATLAB process on the interactive processor (*second node*) has MATLAB (*second single-node kernel*), and (v) the interactive processor (*second node*) receives user instructions, interprets them, and send commands to the other processors of the MultiMATLAB architecture for execution.

**421. First**, as discussed in limitation [35.1.1], Menon teaches each MATLAB process runs on a processor, explaining that "[i]n the MultiMATLAB

architecture, … each processor in a parallel platform individually runs a MATLAB process." Ex.1005, 3. The MultiMATLAB architecture is implemented on the IBM SP2, "a modern high performance distributed memory multiprocessor." Ex.1005, 5. RS6000 teaches that each node of the IBM SP2 can "have either a Symmetric MultiProcessor (SMP) configuration or a uniprocessor configuration." Ex.1007, 2. The SMP ("*hardware processor*") has "four, eight, twelve, or sixteen 375 MHz 630FP 64-bit processors." Ex.1007, 9. Accordingly, Menon and RS6000 disclose or at least render obvious *a hardware processor*.

**422.** Further, Menon describes the MATLAB process of each processor as being in an "address space" and further describes the building of "a table of pointers to all functions and data" that the parallel MEX Routines use "to access any function provided by the underlying communication layer" during the initialization of the MATLAB process. Ex.1005, FIG. 2, 5. Because an "address space" is a "a list of memory locations from some minimum (usually 0) to some maximum, which the process can read and write. The address space contains the executable program, the program's data, and its stack," a POSITA would recognize that a MATLAB process on a processor is software or computer instructions. Ex.1018, 67-68. Accordingly, Menon and RS6000 disclose or at least render obvious *program code that, when executed, is capable of causing the hardware processor to … .*

**423. Second**, as discussed in limitation [1.2], it was shown that Menon teaches MEX routines having code that implement standard variants of MPI calls to enable SPMD point-to-point communication so that the MATLAB processes can communicate tasks and data directly with each other. Ex.1005, 9. The "MEX routines" provide "[a]ll parallel functionality in the MultiMATLAB system." Ex.1005, 3. For example, the MEX routines send the "commands over the underlying communication layer to the processes corresponding to the specified IDs." Ex.1005, 4. Menon describes using "MPI as [the] underlying communication substrate" in its implementations of the MultiMATLAB architecture, although "a specific communication standard is not fundamental to the architecture." Ex.1005, 5. The choice of MPI is "due [to] its broad popularity, its suitability to high performance parallel computing, and the virtually universal availability of vendor-optimized versions on modern parallel computers." Ex.1005, 5. One implementation is "designed for the IBM SP2, a modern high performance distributed memory multiprocessor. This implementation is based upon MPI-F [], IBM's proprietary version of MPI specifically optimized for the SP2." Ex.1005, 5. The other implementation "is based upon MPICH [], a popular public domain version of MPI." Ex.1005, 5.

**424.** Menon discloses "point to point … communication" as one of the multiple paradigms used to enable communication between the MATLAB

processes operating in parallel in the MultiMATLAB architecture. Ex.1005, 9. Menon explains that SPMD/Message Passing paradigm provides "point to point … communication." Ex.1005, 9. SPMD [single program multiple data] refers to a "style of parallel programming, where all processors use the same program, though each has its own data." Ex.1009, 1. Table 1 of Menon below shows example point-to-point commands for SPMD communication paradigm, such as the "Send(pid,data)" command that "send[s] data from one process to another," the "Recv(pid)" command that "receive[s] data sent from another process," and the "Bcast(pid,data)" command that "broadcasts data from processor pid to all processes." Ex.1005, Table 1. Any MATLAB process on a processor of the MultiMATLAB architecture receiving communication from any other MATLAB process on another processor of the MultiMATLAB architecture teaches *receiv[ing] calls from* … .

| MultiMATLAB commands | |
|---|---|
| Starting and stopping MultiMATLAB (MPICH only) | |
| Start(n) | Initializes n remote MATLAB processes. |
| Quit | Closes remote MATLAB processes. |
| Running commands on multiple processors | |
| Nproc | Returns number of MATLAB processes. |
| ID | Returns ID of calling MATLAB process. |
| Eval(pid,'command') | Evaluates a command on one or more MATLAB process. |
| Master/Slave Communication | |
| Put(pid,dataname) | Put data from the master process onto one or more remote processes. |
| Get(pid,dataname) | Get data from a remote process onto the master process. |
| SPMD Communication | |
| Send(pid,data) | Send data from one process to another. |
| Recv(pid) | Receive data sent from another process. |
| Barrier | Synchronize processes. |
| Bcast(pid,data) | Broadcasts data from processor pid to all processes. |
| Sum(data) | Adds data across all processors to form a global sum. |
| Collect(data) | Collects local data segments into a single global one. |

Table 1: Selected commands in the MultiMATLAB system

Commands for point-to-point communication implemented in MEX

**Ex.1005, Table 1 (annotated)**

**425. Third**, as discussed in limitation [1.3.1], Menon and RS6000 describe a node having (i) a 64-bit processor that accesses and executes *program code* stored in *memory*, and (ii) the MATLAB code stored in the address space of, and accessed by, the node includes a toolkit with a graphical interface to allow users to provide equations and directly interact with the interactive processor node (*user interface*), rendering obvious *a second node comprising a second hardware processor configured to access a second memory comprising program code for a user interface and program code for a second single-node kernel*. FIG. 1 of Menon

below shows the MATLAB process with a user interface on the interactive processor (*second node*) of the MultiMATLAB architecture.

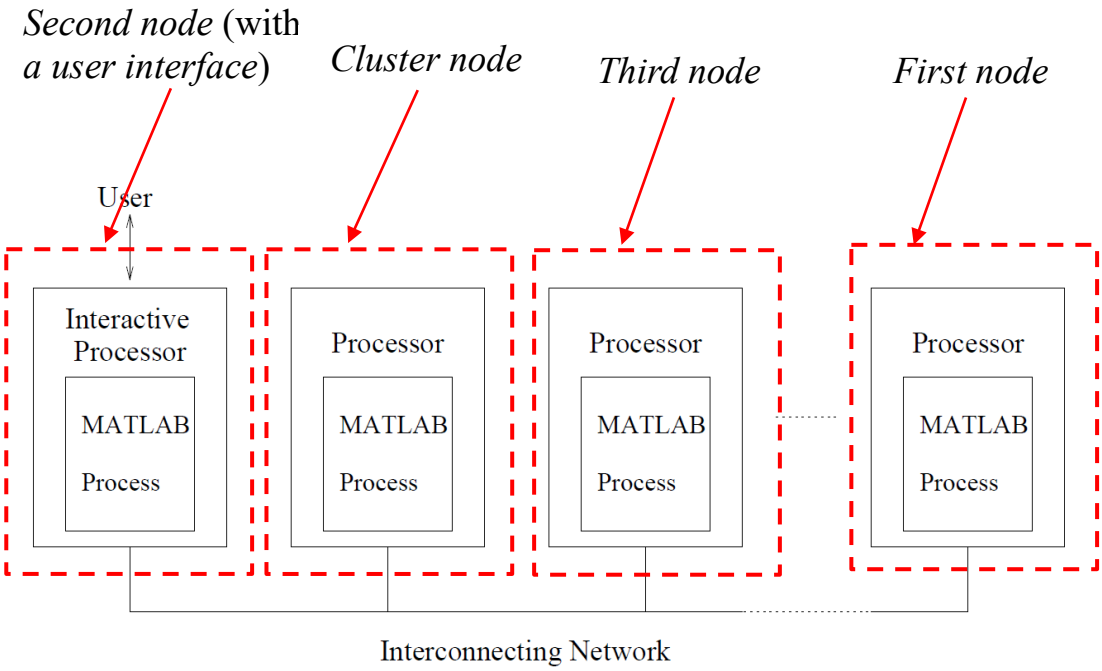



Figure 1: MultiMATLAB Architecture

**Ex.1005, FIG. 1 (annotated)**

**426. Fourth**, as discussed in limitation [1.3.2], Menon teaches that MATLAB receives user commands, interprets such commands into actions that MATLAB performs and then causes calls to be sent to the MATLAB processes on other nodes in accordance with the IDs specified by the user instructions, and Trefethen teaches that MATLAB interprets the commands in order to assign different actions to different processes, rendering obvious *the second single-node kernel configured to interpret user instructions and distribute calls to at least one of a plurality of other nodes for execution*.

**427.** Accordingly, Menon's (i) cluster node has program code executed by a processor, (ii) the cluster node receives calls from an interactive processor (*second node*) of the MultiMATLAB architecture, (iii) the interactive processor (*second node*) has a user interface, (iv) the MATLAB process on the interactive processor (*second node*) has MATLAB (*second single-node kernel*), and (v) the interactive processor (*second node*) receives user instructions, interprets them, and send commands to the other processors of the MultiMATLAB architecture for execution, rendering obvious *program code that, when executed, is capable of causing the hardware processor to: receive calls from a second node comprising a second hardware processor configured to access a second memory comprising program code for a user interface and program code for a second single-node kernel, the second single-node kernel configured to interpret user instructions and distribute calls to at least one of a plurality of other nodes for execution*.

**g. [35.4.2] [*program code that, when executed, is capable of causing the hardware processor to:*] *execute, using the hardware processor, at least a first mathematical expression evaluation; and***

**428.** Menon discloses *[program code that, when executed, is capable of causing the hardware processor to:] execute, using the hardware processor, at least a first mathematical expression evaluation* because it teaches that each cluster node executes MATLAB to evaluate mathematical expressions.

**429.** Menon discloses that a MATLAB process of each processor of the MultiMATLAB architecture evaluates mathematical expressions. Ex.1005, 11-13. Specifically, Menon teaches computing the conjugate gradients algorithm using the IBM SP2 system implementation of the MultiMATLAB architecture where the computations are evaluations of mathematical expressions, such as performing single matrix-vector multiplication, performed by MATLAB processes executing on multiple processors in parallel. Ex.1005, 11-13. Discussing "a parallel MATLAB implementation of conjugate gradients that uses the SPMD message passing," Menon describes the implementation as follows:

> The conjugate gradients algorithm is iterative method to solve a linear system of the form $Ax = b$, where $A$ is a symmetric positive definite matrix and $x$ and $b$ are vectors. The computational core of this method is a single matrix-vector multiplication at each iteration
>
> …
>
> the matrix A and the different vectors are each distributed by row over all processes as in Figure 6. In each iteration, a process only does the computation required for its local data. However, communication is required for two different operations: the dot-products needed to compute *rho* and *alpha* and the matrix-vector multiply needed to compute *w_local*. The dot products only require communication of a single value over all processes. On

> the other hand, the matrix-vector multiplication needed for *w_local* requires the global vector *p*, and, thus, more expensive communication.

Ex.1005, 11-13 (emphasis original). A POSITA would recognize that the computations or mathematical evaluations are performed by MATLAB (*single-node kernel*) in a MATLAB process.

Code to perform mathematical expression evaluation

```
r = b;
rho = r'*r;
k = 0;
while((k < kmax))
  k = k+1;
  if (k == 1)
    p = r;
  else
    beta = rho/oldrho;
    p = r + beta*p;
  end

  w = A * p;

  alpha = rho/(p'*w);
  x = x + alpha * p;
  r = r - alpha * w;
  oldrho = rho;
  rho = r'*r;
end
```

a. Sequential MATLAB

```
r_local = b_local;
rho = Sum(r_local'*r_local);
k = 0;
while((k < kmax))
  k = k+1;
  if (k == 1)
    p_local = r_local;
  else
    beta = rho/oldrho;
    p_local = r_local + beta*p_local;
  end

  p = Gather(p_local);
  w_local = A_local * p;

  alpha = rho/(Sum(p_local'*w_local));
  x_local = x_local + alpha * p_local;
  r_local = r_local - alpha * w_local;
  oldrho = rho;
  rho = Sum(r_local'*r_local);
end
```

b. Message Passing MATLAB

Figure 4: Conjugate Gradients

**Ex.1005, FIG. 4 (annotated)**

**430.** Accordingly, Menon's disclosure of each node having a processor and MATLAB process performing mathematical computations, such as single matrix-vector multiplication, renders obvious *[program code that, when executed, is*

*capable of causing the hardware processor to:] execute, using the hardware processor, at least a first mathematical expression evaluation*.

**h. [35.4.3.1] [*program code that, when executed, is capable of causing the hardware processor to:*] *communicate a result of the first mathematical expression evaluation to a third node comprising a third hardware processor with a plurality of processing cores,***

**431.** Menon discloses *[program code that, when executed, is capable of causing the hardware processor to:] communicate a result of the first mathematical expression evaluation to a third node comprising a third hardware processor with a plurality of processing cores* because Menon teaches (i) a cluster node having a processor and MATLAB process performing mathematical computations, where each node passes messages among nodes using identification numbers and (ii) a third node having a processing node with a symmetric multiprocessor having up to sixteen processors each having a core, as taught by RS6000, in view of Trefethen teaching passing mathematical results from a first MATLAB process at a first processor to a second MATLAB process at a second processor to a third MATLAB process at a third processor.

**432.** Menon discloses that the MATLAB processes of a MultiMATLAB architecture implemented on the IBM SP2 are started by a user utilizing IBM's POE, where the MATLAB processes are numbered starting with 0:

> In this implementation, all MATLAB processes are started via POE, IBM's Parallel Operating Environment.

> For each of *p* MATLAB processes, POE assigns an identification number between 0 and *p* - 1. The process numbered 0 is considered the interactive MATLAB. All other processes wait for commands from the interactive MATLAB process.

Ex.1005, 6 (emphasis original). The interactive MATLAB is the MATLAB process with which the user interacts. Ex.1005, 3. POEref teaches that the user invokes the initialization process using the "poe" command:

> When you invoke **poe**, the Partition Manager allocates processor nodes for each task and initializes the local environment. It then loads your program, and reproduces your local environment, on each processor node. The Partition Manager also passes the option list to each remote node. … If you are using the dynamic message passing interface, the appropriate communication subsystem library implementation (IP or US) is automatically loaded at this time.

Ex.1008, 28 (emphasis original).

**433.** In other words, when a user on an IBM SP2 invokes **poe**, a local environment, i.e., a MATLAB process, is initiated on the IBM SP2 processor with which the user is interacting and reproduced on each of the other IBM SP2 processors (called "remote nodes" by POEref), and the MATLAB processes (called "non-interactive processes" by Menon) on these remote nodes await for commands

from the interactive MATLAB process. Ex.1005, 6, 4 (calling the MATLAB processes on the remote nodes "non-interactive processes"); Ex.1008, 28, 3 (discussing "remote nodes"). Specifically, "[o]n non-interactive processes, a separate top-level MEX Routine is immediately run upon initialization of the system. This MEX Routine runs in a loop in which it waits for MATLAB commands to arrive from the interactive process and executes them in its own MATLAB environment." Ex.1005, 4.

**434.** Menon further discloses that, after the initialization of the MultiMATLAB architecture, a "user interacts directly with one MATLAB process, called the interactive process, and operates within that process's MATLAB environment." Ex.1005, 3. The user can use "an Eval routine for parallel evaluation" to provide user instructions to the interactive MATLAB process so that the user instructions are evaluated in parallel fashion by the processors of the MultiMATLAB architecture. Ex.1005, 4. "This routine allows users to execute commands on the other processes. On the interactive process, the Eval routine is implemented as a MEX Routine which accepts a vector of MATLAB process IDs and a MATLAB command. This routine sends the commands over the underlying communication layer to the processes corresponding to the specified IDs." Ex.1005, 4.

**435.** Trefethen provides an example of such user interaction with the MultiMATLAB architecture, where a user is "connected to a node of the IBM SP2, running MATLAB" and operates in a MultiMATLAB architecture where "6 MultiMATLAB processes [are] running." Ex.1006, 3. "If the user now types:

> Eval( 'sqrt(2)' )
>
> then the MATLAB command sqrt(2) is executed on all six processors. The result is six repetitions of 1.4142. … On the other hand the command
>
> Eval( 'ID' )
>
> calls the MultiMATLAB command ID on each of the processors running. This command returns the number of the current process, an integer from 0 to Nproc-1. Running it on all nodes might give the result
>
> ans = 0
>
> ans = 1
>
> ans = 5
>
> ans = 2
>
> ans = 3
>
> ans = 4

Ex.1006, 3-4. That is, when the user enters its commands at the interactive MATLAB process of the IBM SP2 node, of the 6 processors or IBM SP2 nodes that make up the MultiMATLAB architecture, "the MATLAB command sqrt(2) is

executed on all six processors" and "the MultiMATLAB command ID" is executed "on each of the processors running," the result of which shows that the six processors are numbered staring with 0 (the interactive processor). Ex.1006, 3-4.

**436.** Trefethen also provides an example where the processors of the MultiMATLAB architecture perform mathematical evaluations and communicate the results to each other. Ex.1006, 3-7. Specifically, Trefethen provides the following example code in which a MATLAB process at processor 0 (*second node*) creates a variable and sends that value to a MATLAB process at processor 1 (*cluster node*), which receives the transmitted value and doubles (*first mathematical expression evaluation*) it, and then sends (*communicate a result of the*) it (*first mathematical expression evaluation*) to the MATLAB process at processor 2 (*third node*), and so on:

```
if ID==0                % first process: send
   a = 1
   Send(ID+1,a)
elseif ID == Nproc-1    % last process: receive and double
   a = 2*Recv
else                    % middle processes: receive, double, and send
   a = 2*Recv
   Send(ID+1,a)
end;
```

Process 0 creates the variable a with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of a, and sends it along to process 2; and so on. If there are six processors the command `Eval( 'cycle' )` produces the output

```
a = 1
a = 2
a = 4
a = 8
a = 16
a = 32
```

Point-to-point communications of mathematical evaluation results from a MATLAB process at processor 0 (*second node*) to one at processor 1 (*cluster node*) to another processor 2 (*third node*)

Ex.1006, 6.

**437.** Processor 2 (*third node*) comprises *a hardware processor with a plurality of processing cores* because RS6000 teaches that each node of the IBM SP2, on which the MultiMATLAB architecture is implemented as disclosed by Menon, has a "Symmetric MultiProcessor (SMP)" ("*hardware processor*") and that the SMP includes "four, eight, twelve, or sixteen 375 MHz 630FP 64-bit processors," where each of these processors has a processor core. Ex.1005, 6; Ex.1007, 2, 9.

**438.** Accordingly, Menon teaches (i) a cluster node having a processor and MATLAB process performing mathematical computations, where each node passes messages among nodes using identification numbers and (ii) a third node having a processing node with up to sixteen processors each having a core, as taught by RS6000, in view of Trefethen teaching passing mathematical results from a MATLAB process at processor 0 (*second node*) to a MATLAB process at processor 1 (*cluster node*) to a MATLAB process at processor 2 (*third node*), renders obvious *communicate a result of the first mathematical expression evaluation to a third node comprising a third hardware processor with a plurality of processing cores*.

**i. [35.4.3.2] *wherein the third node is configured to receive the result of mathematical expression evaluation from the computer cluster node, execute at least a second mathematical expression evaluation using the result of the first mathematical expression evaluation, and***

***communicate a result of the second mathematical expression evaluation to the first node;***

**439.** Menon in view of Trefethen render obvious *wherein the third node is configured to receive the result of mathematical expression evaluation from the computer cluster node, execute at least a second mathematical expression evaluation using the result of the first mathematical expression evaluation, and communicate a result of the second mathematical expression evaluation to the first node* because Menon teaches a MATLAB environment using point to point communication among the processors of the MultiMATLAB architecture and Trefethen teaches passing mathematical results from a MATLAB process at processor 0 (*second node*) to a MATLAB process at processor 1 (*cluster node*) to a MATLAB process at processor 2 (*third node*) to a MATLAB process at processor 3 (*first node*) or a MATLAB process at processor 0 (*second node*).

**440.** As discussed in limitation [35.4.3.1], Trefethen provides a sample program where MATLAB process at processor 0 (*second node*) creates a variable and sends that value to a MATLAB process at processor 1 (*cluster node*), which receives the transmitted value and doubles (*first mathematical expression evaluation*) it, and then sends (*communicate a result of the*) it (*first mathematical expression evaluation*) to the MATLAB process at processor 2 (*third node*).

**441.** Trefethen further describes that the MATLAB process at processor 2 (*third node*), after receiving the value from the previous MATLAB process, doubles

(*second mathematical expression evaluation*) the received value (*using the result of the first mathematical expression evaluation*) and sends its result (*communicate a result of the second mathematical expression evaluation to*) to the MATLAB process at the next processor, processor 3 (*first node*), and this process could continue ("… and so on") and provides an example for when the MultiMATLAB architecture comprises six processors:

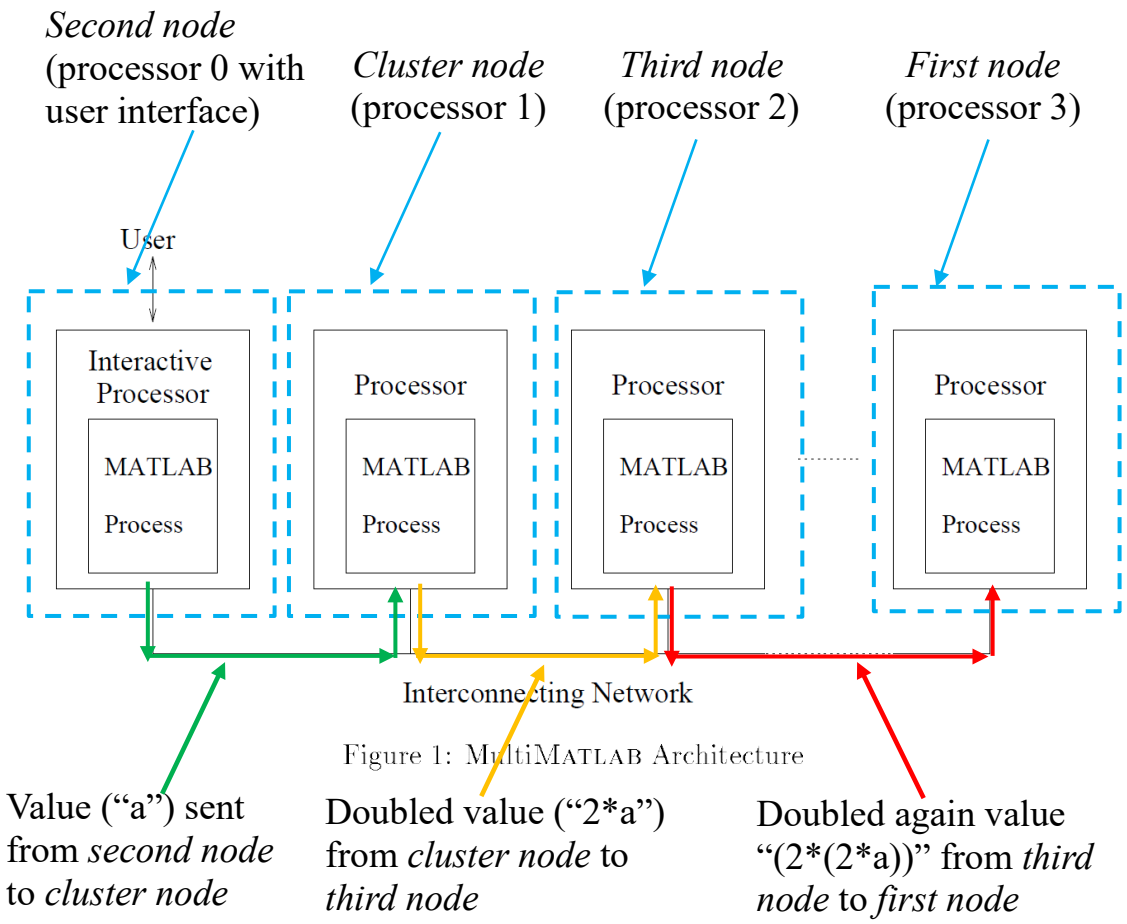
```
if ID==0                % first process: send
   a = 1
   Send(ID+1,a)
elseif ID == Nproc-1    % last process: receive and double
   a = 2*Recv
else                    % middle processes: receive, double, and send
   a = 2*Recv
   Send(ID+1,a)
end;
```

Process 0 creates the variable a with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of a, and sends it along to process 2; and so on. If there are six processors the command `Eval( 'cycle' )` produces the output

```
a = 1
a = 2
a = 4
a = 8
a = 16
a = 32
```

Point-to-point communications of mathematical evaluation results from a MATLAB process at processor 0 (*second node*) to one at processor 1 (*cluster node*) to another processor 2 (*third node*) and yet another at processor 3 (*first node*), each doubling the received value

Ex.1006, 6 (annotated). FIG. 1 of Menon below illustrates the execution of the sample program in Trefethen by processors of the MultiMATLAB architecture.




**Ex.1005, FIG. 1 (annotated).**

**442.** Further, as discussed in limitation [1.5.2], Trefethen discloses that each MATLAB process transmits its output to the interactive processor that has the user interface as soon as that MATLAB process's computation is completed. Ex.1006, 4. Specifically, Trefethen discloses that running the command Eval( 'ID' ) on all six nodes "give[s] the result:

ans = 0

ans = 1

> ans = 5
>
> ans = 2
>
> ans = 3
>
> ans = 4
>
> The ordering of these numbers is arbitrary, since the processors are not synchronized and **output is sent to the master process as soon as it is ready**.

Ex.1006, 4. In other words, as soon as a MATLAB process at a processor completes its computation, it sends the result of its computation to the processor with the user interface, i.e., the *second node*. A POSITA would recognize that the "master process" described by Trefethen would be the interactive process of Menon, because the interactive process is the MATLAB process in the MultiMATLAB architecture with which the user interacts.[4] Ex.1005, 3.

[4] In the context of SPMD point-to-point communication, a "master process" is a "pseudo-master [that] is arbitrarily chosen to be the task [that] all the peers know a priori who to send the results back to." Ex.1011, 64. That is, here, the "master" process is the process with which the user is directly interacting in a peer-to-peer paradigm, and not a "master" in a master/slave paradigm.

**443.** Accordingly, Menon teaching a point to point system for passing messages among MATLAB processes on processors of the MultiMATLAB architecture with identification numbers in view of Trefethen teaching passing mathematical results from a MATLAB process at processor 0 (*second node*) to a MATLAB process at processor 1 (*cluster node*) to a MATLAB process at processor 2 (*third node*) to a MATLAB process at processor 3 (*first node*) or the processor with the user interface (*second node*), renders obvious *wherein the third node is configured to receive the result of mathematical expression evaluation from the computer cluster node, execute at least a second mathematical expression evaluation using the result of the first mathematical expression evaluation, and communicate a result of the second mathematical expression evaluation to the first node*.

**j. [35.5] *wherein the user connection interface is configured to return at least one result of mathematical expression evaluation to a user interface or a script; and***

**444.** Menon and Trefethen render obvious *wherein the user connection interface is configured to return at least one result of mathematical expression evaluation to a user interface or a script* because Menon discloses (i) a cluster node having a MultiMATLAB interface module and communication layer for communicating mathematical results to the other nodes and (ii) a node with an interactive processor having a user interface for interfacing with the user, and

Trefethen discloses returning the results of mathematical expressions from each processor to the user interface.

**445.** Menon discloses that each processor of "the MultiMATLAB architecture … individually runs a MATLAB process" that includes a set of components to provide interconnection with other modules, including the "MEX Routines," "MultiMATLAB interface module," an "indirection table" and "communication layer." Ex.1005, 3, 5. Menon describes that the MultiMATLAB interface module on each processor is responsible for the underlying communication layer with other processors, explaining that "[t]he interface module, shown in Figure 2, is responsible for initializing an underlying communication layer, such as MPI …, or any other package available on the platform, and exposing it to the rest of the system." Ex.1005, 3. FIG. 2 of Menon shows the relationship between the MultiMATLAB interface module, the communication layer, and the single-node kernel is shown below.

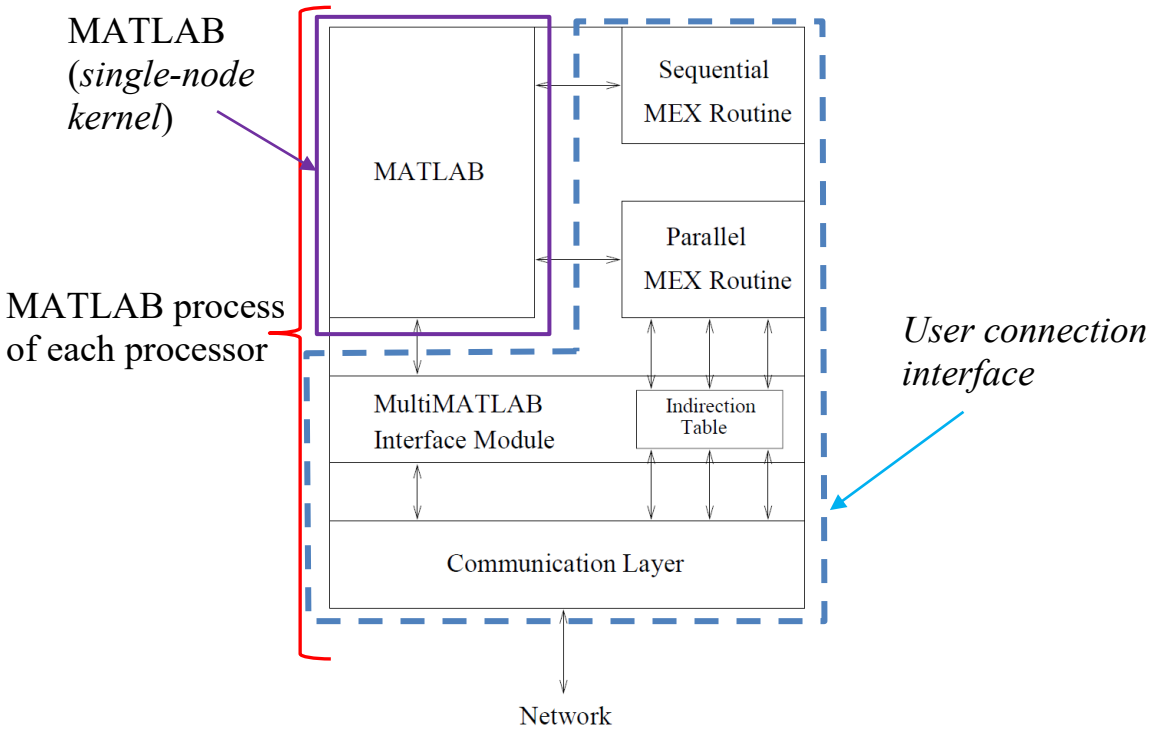
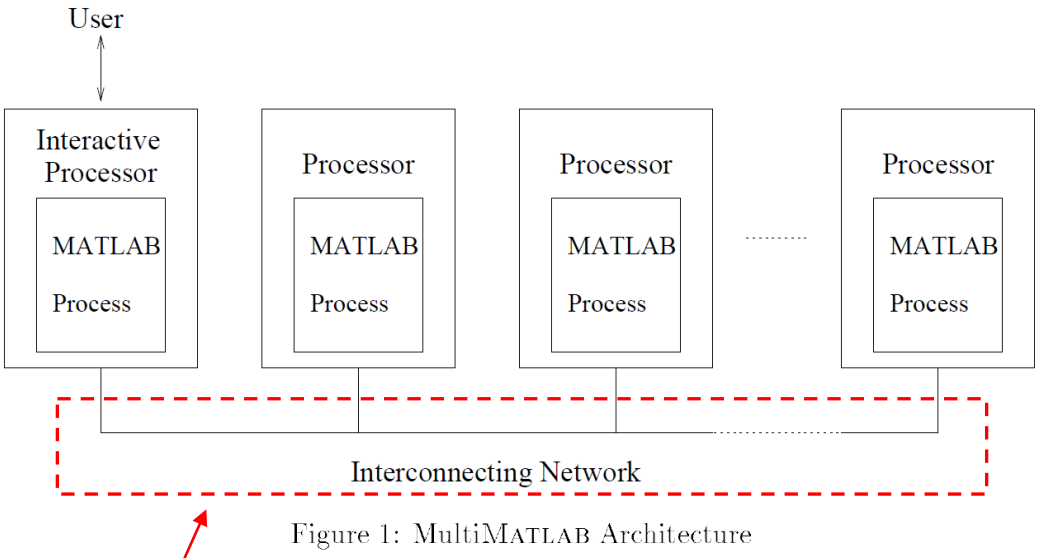



Figure 2: MATLAB Process Address Space

**Ex.1005, FIG. 2 (annotated)**

**446.** Menon further explains that the "MEX routines" provide "[a]ll parallel functionality in the MultiMATLAB system" and the communication layer provides "the ability to communicate with other processes." Ex.1005, 3. The communication layer, initialized and "expos[ed] to the rest of the [MultiMATLAB] system" by the MultiMATLAB interface module, uses the MPI communication standard as its underlying communication substrate. Ex.1005, 3. In operation, the MEX routines (such as Send() and Recv) are run directly from the user interface. Ex.1005, 3; *see also* Ex.1006, 3, 6. The MEX routines (such as Send() and Recv() are high-level calls that are mapped (using the indirection table) to the analogous MPI calls (such

as MPI_SEND() and MPI_RECV()) found in the communication layer. Ex.1005, 3, 6. The communication layer, using the MPI calls, communicates over the interconnection network with the communication layer in the other nodes. Ex.1005, 3. FIG. 3 of Menon below shows the interconnection network of the MultiMATLAB architecture linking the processors. Ex.1005, FIG. 3.




**Ex.1005, FIG. 2 (annotated).**

Interconnecting network connecting each MATLAB process on a processor to any other in the MultiMATLAB architecture

**447.** Further, as discussed in limitation [1.5.2], Trefethen discloses that each MATLAB process transmits its output to the interactive processor that has the user interface as soon as that MATLAB process's computation is completed. Ex.1006, 4. Specifically, Trefethen discloses that running the command Eval( 'ID' ) on all six nodes "give[s] the result:

> ans = 0
>
> ans = 1
>
> ans = 5
>
> ans = 2
>
> ans = 3
>
> ans = 4
>
> The ordering of these numbers is arbitrary, since the processors are not synchronized and **output is sent to the master process as soon as it is ready**.

Ex.1006, 4. In other words, as soon as a MATLAB process at a processor completes its computation, it sends the result of its computation to the processor with the user interface. A POSITA would recognize that the "master process" described by Trefethen would be the interactive process of Menon, because the interactive process is the MATLAB process in the MultiMATLAB architecture with which the user interacts.[5] Ex.1005, 3. Trefethen further teaches "produc[ing] plots in a

[5] In the context of SPMD point-to-point communication, a "master process" is a "pseudo-master [that] is arbitrarily chosen to be the task [that] all the peers know a priori who to send the results back to." Ex.1011, 64. That is, here, the "master"

distributed fashion that are then sent to the user's screen," which allows for "the progress of computations on several processors," such as the results of computations performed by the processors of the MultiMATLAB architecture and returned to the processor with the user interface, to be presented to the user at the user's screen. Ex.1006, 7.

**448.** Accordingly, Menon's set of components ( "MEX Routines," the "MultiMATLAB interface module," an "indirection table" and "communication layer") that is responsible for the underlying communication layer with other nodes (*user connection interface)* in view of Trefethen's teaching that each MATLAB process can transmit its mathematical results back to the processor with the user interface for presenting the results to the user, renders obvious *wherein the user connection interface is configured to return at least one result of mathematical expression evaluation to a user interface or a script*.

**k. [35.6.0] *wherein the computer cluster node is configured to:***

**449.** Limitation [35.6.0] is disclosed or rendered obvious for the same reasons presented above for limitation [35.0].

---

process is the process with which the user is directly interacting in a peer-to-peer paradigm, and not a "master" in a master/slave paradigm.

**l. [35.6.1] *accept user instructions;***

**450.** Limitation [35.6.1] is disclosed or rendered obvious for the same reasons presented above for limitation [1.7.1].

**m. [35.6.2] *after accepting user instructions, communicate at least some of the user instructions using the mechanism for the nodes to communicate with each other; and***

**451.** Limitation [35.6.2] is disclosed or rendered obvious for the same reasons presented above for limitation [1.7.2].

**n. [35.6.3] *after communicating at least some of the user instructions using the mechanism, communicate at least some of the user instructions to the single-node kernel.***

**452.** Limitation [35.6.3] is disclosed or rendered obvious for the same reasons presented above for limitation [1.7.3].

**33. Claim 36**

**a. [36.0] *The computer cluster node of claim 35, wherein the one or more non-transitory memory devices comprise program code for performing a parallel fast Fourier transform, wherein the program code causes the hardware processor to evaluate a command to perform a Fourier transform on an array comprising a first data portion that is stored on the computer cluster node and a second data portion that is not stored on the computer cluster node.***

**453.** Menon and Trefethen render obvious *wherein the one or more non-transitory memory devices comprise program code for performing a parallel fast Fourier transform, wherein the program code causes the hardware processor to*

*evaluate a command to perform a Fourier transform on an array comprising a first data portion that is stored on the computer cluster node and a second data portion that is not stored on the computer cluster node* because Menon teaches that (i) data is stored in a matrix distributed across the different processors of the MultiMATLAB architecture for parallel processing using MATLAB, (ii) the MATLAB processes compute Fourier transforms, and (iii) each MATLAB process performs computation for its local data, and Trefethen teaches that MATLAB of the MATLAB processes can perform fast Fourier transforms.

**454. First**, Menon teaches MATLAB is capable of causing a processor of the MultiMATLAB architecture to evaluate mathematical expressions having the data "distributed by row over all processes," as illustrated by the "MultiMATLAB implementations of the conjugate gradients algorithm on the IBM SP2" discussed in Menon. Ex.1005, 11-13. The "parallel MATLAB implementation of conjugate gradients" includes the step of "matrix A and the different vectors [] each [being] distributed by row over all processes." Ex.1005, 11. FIG. 6 of Menon shown below illustrates this step where "[m]atrix A and vector x [are] distributed by row over *P* processors" when the MultiMATLAB implementation includes "*p* MATLAB processes." Ex.1005, FIG. 6, 11.

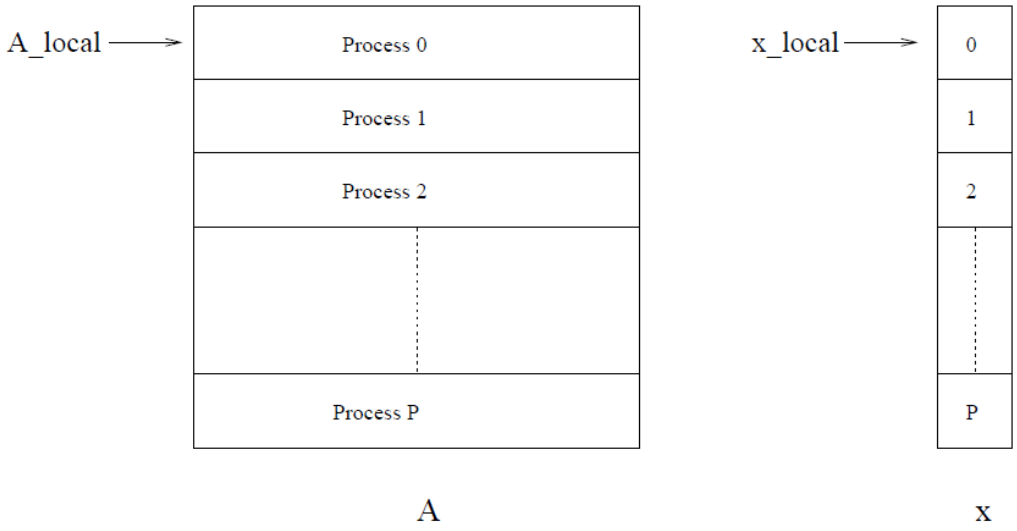



Figure 6: Matrix $A$ and vector $x$ distributed by row over $P$ processors

**Ex.1005, FIG. 6**

**455.** Menon teaches that each MATLAB process on a processor performs "computation required for its local data" (while communicating with other MATLAB processes to enable global conjugate gradient operations), explaining that:

> the matrix A and the different vectors are each distributed by row over all processes as in Figure 6. In each iteration, a process only does the computation required for its local data. However, communication is required for two different operations: the dot-products needed to compute *rho* and *alpha* and the matrix-vector multiply needed to compute *w_local*. The dot products only require communication of a single value over all processes. On the other hand, the matrix-vector multiplication needed

> for *w_local* requires the global vector *p*, and, thus, more expensive communication.

Ex.1005, 11-13 (emphasis original). A POSITA would recognize that each MATLAB process, when computing "local data," would only use the data on its local node and would not compute on data on another node.

**456. Second**, Trefethen discloses that MATLAB (in the MATLAB processes of the MultiMATLAB architecture) can perform fast Fourier transform, explaining that "MATLAB® is a high-level language, and a problem-solving environment, for mathematical and scientific calculations. It … contain[s] dozens of high-level commands such as svd (singular value decomposition), fft (fast Fourier transform), and roots (polynomial zerofinding)." Ex.1006, 2. Menon also teaches that the "parallel MEX routines" of MATLAB processes of the MultiMATLAB architecture "would perform matrix multiplications, solve for eigenvalues, compute Fourier transforms, and so on." Ex.1005, 10.

**457.** Accordingly, Menon teaching that (i) data is stored in a matrix distributed across the different processors of the MultiMATLAB architecture for parallel processing using MATLAB, (ii) the MATLAB processes compute Fourier transforms, and (iii) each MATLAB process performs computation for its local data, in view of Trefethen teaching that MATLAB of the MATLAB processes can perform fast Fourier transforms, renders obvious *wherein the one or more non-*

*transitory memory devices comprise program code for performing a parallel fast Fourier transform, wherein the program code causes the hardware processor to evaluate a command to perform a Fourier transform on an array comprising a first data portion that is stored on the computer cluster node and a second data portion that is not stored on the computer cluster node*.

**34. Claim 39**

**a. [39.0] *The computer cluster node of claim 35, wherein the hardware processor comprises a special purpose microprocessor.***

**458.** RS6000 discloses *wherein the hardware processor comprises a special purpose microprocessor* because it teaches that the symmetric multiprocessor hardware processors in each node are digital signal processors.

**459.** RS6000 discloses that each processor node of the IBM SP2 can be "375 MHz POWER3 SMP High Node (F/C 2058)," where the SMP ("*hardware processor*") has "four, eight, twelve, or sixteen 375 MHz 630FP 64-bit processors." Ex.1007, 2, 9. A POSITA would recognize then that the SMP is an integrated circuit designed for high-speed data manipulation and renders obvious a *special purpose microprocessor*.

**460.** Accordingly, RS6000's SMP renders obvious *wherein the hardware processor comprises a special purpose microprocessor*.

### B. Ground 2: Claims 27-28, 31-34, 37, and 38 are obvious over Menon in view of Trefethen, RS6000, and POEref, further in view of MPIref

**461.** The combination of Menon, Trefethen, RS6000, POEref, and MPIref renders obvious claims 27-28, 31-34, 37, and 38 as discussed below.

#### 35. Combining MPIref with the MultiMATLAB system of Menon, Trefethen, RS6000 and POEref

##### a. ***MPIref is Analogous Art***

**462.** MPIref is analogous to the '768 patent at least because MPIref is directed to the same field of endeavor as the '768 patent. The '768 patent is directed to "the field of cluster computing." Ex.1001, 1:14-15. Likewise, MPIref describes MPI which is for writing "portable message-passing programs" such as "software designed to run on parallel machines." Ex.1017, 2. In particular, "MPI provides many features intended to improve performance on scalable parallel computers." Ex.1017, 3.

**463.** Furthermore, MPIref is analogous to the '768 patent because MPIref is reasonably pertinent to a problem the inventors of the '768 patent attempted to solve. The '768 patent purports to provide a solution for "send[ing] messages between nodes in a computer cluster" by utilizing MPI. Ex.1001, 6:6-8. Likewise, MPIref describes "all the technical features" of the MPI. Ex.1017, ii.

**464.** Accordingly, in my opinion, a POSITA would understand that MPIref is a prior art that is analogous to the '768 patent.

**b. *Motivation to Combine MPIref with Menon***

**465.** A POSITA would have considered and combined the teachings of MPIref with Menon because Menon suggests the combination via its teaching that "the MPI communication standard" is used as the "underlying communication substrate" that allows all the MATLAB processes of the MultiMATLAB architecture to communicate with each other. Ex.1005, 5. For example, Menon discloses "two implementations of the MultiMATLAB architecture," both of which "use MPI as their underlying communication substrate" because of "its broad popularity, its suitability to high performance parallel computing, and the virtually universal availability of vendor-optimized versions on modern parallel computers." Ex.1005, 5.

**466.** MPIref defines the MPI communication standard. Ex.1017, 1 ("MPI: A Message-Passing Interface Standard."). MPIref defines a "set of library interface standards for message passing." Ex.1017, 2. The goal of the MPIref publication was to "develop a widely used standard for writing message-passing programs" and "should establish a practical, portable, efficient and flexible standard for message passing." Ex.1017, 2.

**467.** A POSITA would also have considered and combined MPIref with Menon because it is a combination of known elements (Menon describing an implementation of the MultiMATLAB architecture using MPI as its communication

layer and MPIref describing "all the technical features" of MPI) according to known methods, as taught by Menon, to yield the predictable result of an implementation of the MultiMATLAB architecture utilizing various technical features of MPI disclosed by MPIref. Ex.1005, 3-10; Ex.1017, ii.

**468.** The results of the combination would have been predictable and there would have been a reasonable expectation of success at least because Menon expressly discusses using MPI, which is the subject of MPIref, to establish communication between all the MATLAB processes of the MultiMATLAB architecture, and MPIref provides a "practical, portable, efficient and flexible standard for message passing." Ex.1005, 3; Ex.1017, 2.

**36. Claim 27**

**a. [27.0] *The computer cluster of claim 26, wherein the asynchronous calls comprise a first command to create a first packet containing:***

**469.** Menon, Trefethen, and MPIref render obvious *wherein the asynchronous calls comprise a first command to create a first packet* because Menon describes the calls of Send and Recv commands and Trefethen describes those calls as asynchronous, in view of Menon describing the execution of the command "Send (ID+1,a)" (*a first command*) by MATLAB (*single-node kernel*) in a MATLAB process of processor ID of the MultiMATLAB architecture would generate a collection or package (*first packet*) of information that is needed by the

corresponding MPI call (MPI_SEND), as described by MPIref, including (*containing*) (i) the variable "a" (*expression*), the data part that is associated with the send buffer information in MPI_SEND and that MPI_SEND transmits to the next MATLAB process at the next/destination processor ID+1 of the MultiMATLAB architecture as part of its message, and (ii) an identification of the destination processor ("ID+1") (*target node*).

**470. First**, it was shown in limitation [4.0] that each node of the MultiMATLAB architecture has a MATLAB process which in turn has a set of components to provide interconnection with other modules, namely, the "MEX Routines," the "MultiMATLAB interface module," an "indirection table" and "communication layer." (*a local cluster node module*). Ex.1005, 3, 5. Figure 2 of Menon below shows the *local cluster node module*.

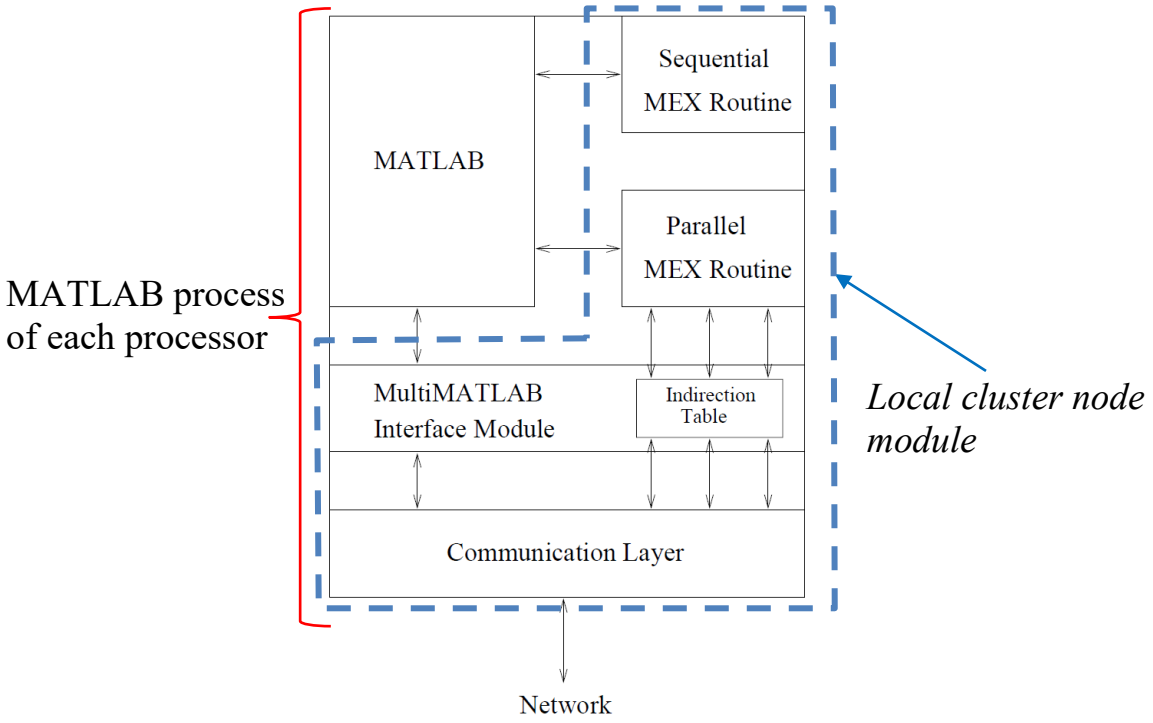
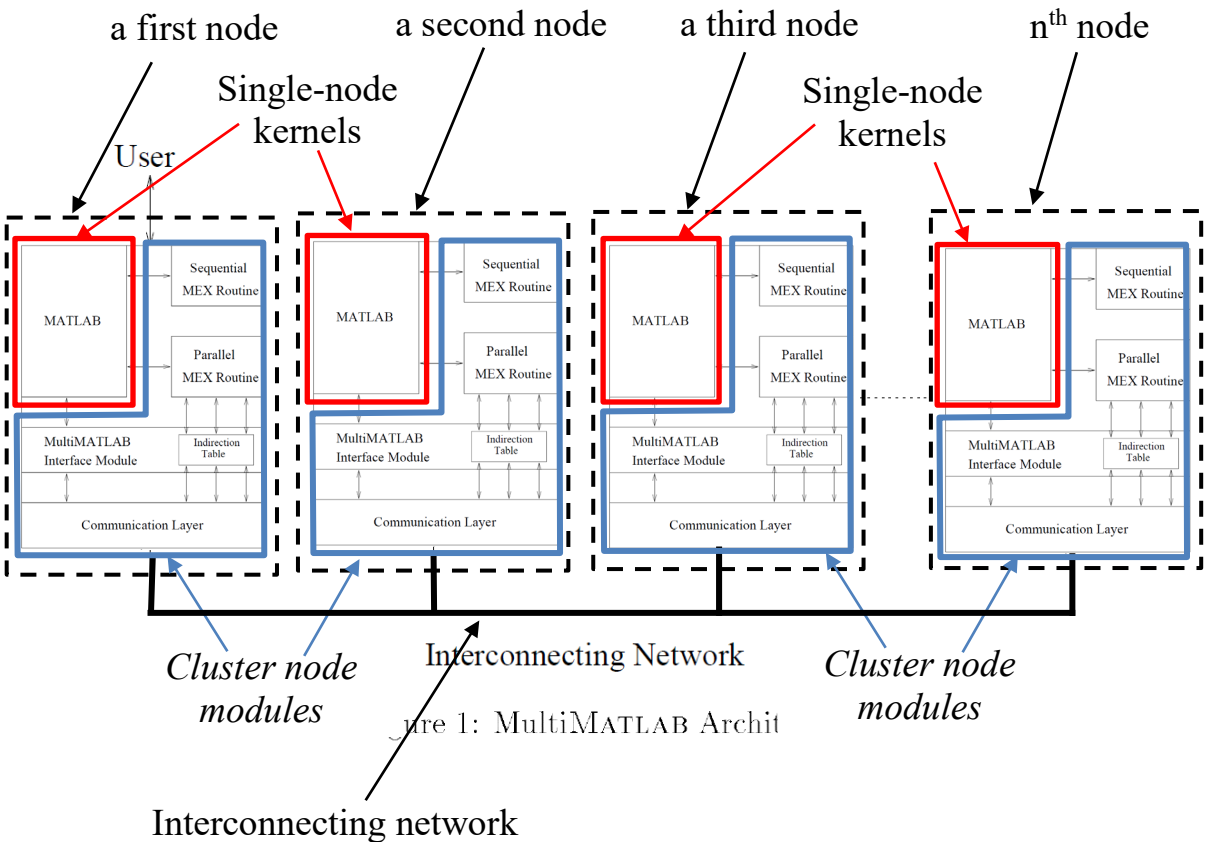



Figure 2: MATLAB Process Address Space

**Ex.1005, FIG. 2 (annotated)**

**471.** A combination of FIG. 1 and FIG. 2 of Menon below shows the relationship between the communication components (i.e., MEX routines, MultiMATLAB interface module, indirection table, the communication layer), and MATLAB (*the single-node kernel*) for each node of the MultiMATLAB architecture.




**Ex.1005, FIGs. 1 and 2 (combined and annotated)**

**472. Second**, it was shown in limitation [1.2] that Menon teaches MEX routines having code that implement standard variants of MPI calls to enable SPMD point-to-point communication so that the MATLAB processes can communicate tasks and data directly with each other. Ex.1005, 9. The "MEX routines" provide "[a]ll parallel functionality in the MultiMATLAB system." Ex.1005, 3. For example, the MEX routines send the "commands over the underlying communication layer to the processes corresponding to the specified IDs." Ex.1005, 4. Menon describes using "MPI as [the] underlying communication substrate" in its

implementations of the MultiMATLAB architecture, although "a specific communication standard is not fundamental to the architecture." Ex.1005, 5. The choice of MPI is "due [to] its broad popularity, its suitability to high performance parallel computing, and the virtually universal availability of vendor-optimized versions on modern parallel computers." Ex.1005, 5. One implementation is "designed for the IBM SP2, a modern high performance distributed memory multiprocessor. This implementation is based upon MPI-F [], IBM's proprietary version of MPI specifically optimized for the SP2." Ex.1005, 5. The other implementation "is based upon MPICH [], a popular public domain version of MPI." Ex.1005, 5.

**473.** As discussed in limitation [26.2], Menon describes example point-to-point commands that the MATLAB processes in the MultiMATLAB architecture use to communicate with each other, including SPMD calls such as the "Send(pid,data)" command that "send[s] data from one process to another" and the "Recv(pid)" command that "receive[s] data sent from another process," shown in Table 1 of Menon below.

| MultiMATLAB commands | |
|---|---|
| Starting and stopping MultiMATLAB (MPICH only) | |
| Start(n) | Initializes n remote MATLAB processes. |
| Quit | Closes remote MATLAB processes. |
| Running commands on multiple processors | |
| Nproc | Returns number of MATLAB processes. |
| ID | Returns ID of calling MATLAB process. |
| Eval(pid,'command') | Evaluates a command on one or more MATLAB process. |
| Master/Slave Communication | |
| Put(pid,dataname) | Put data from the master process onto one or more remote processes. |
| Get(pid,dataname) | Get data from a remote process onto the master process. |
| SPMD Communication | |
| Send(pid,data) | Send data from one process to another. |
| Recv(pid) | Receive data sent from another process. |
| Barrier | Synchronize processes. |
| Bcast(pid,data) | Broadcasts data from processor pid to all processes. |
| Sum(data) | Adds data across all processors to form a global sum. |
| Collect(data) | Collects local data segments into a single global one. |

Table 1: Selected commands in the MultiMATLAB system

Commands for point-to-point communication (including Send and Recv)

**Ex.1005, Table 1 (annotated).**

Ex.1005, 9.

**474.** Also as discussed in limitation [26.2], Trefethen teaches the processors of the MultiMATLAB architecture communicate with each other *using asynchronous calls*, such as the "collection of [MultiMATLAB] commands such as Send" and Recv:

```
if ID==0                    % first process: send
   a = 1
   Send(ID+1,a)
elseif ID == Nproc-1        % last process: receive and double
   a = 2*Recv
else                        % middle processes: receive, double, and send
   a = 2*Recv
   Send(ID+1,a)
end;
```

Process 0 creates the variable a with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of a, and sends it along to process 2; and so on. If there are six processors the command `Eval( 'cycle' )` produces the output

```
a = 1
a = 2
a = 4
a = 8
a = 16
a = 32
```

*Asynchronous calls*

Send command executed by single-node kernel at processor ID to send data or variable "a" to processor ID+1

The processes run asynchronously, but since each `Send` command is only executed after the corresponding `Recv` has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning.

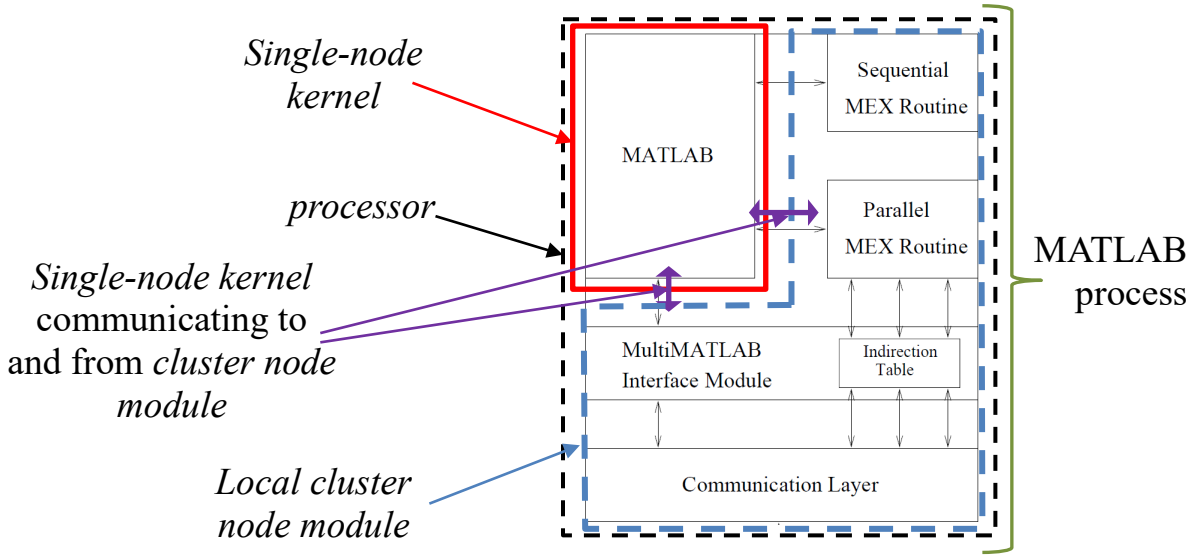
Ex.1006, 6, (annotated). For example, processor ID communicates with processor ID+1 *using asynchronous calls* Send and Recv commands. Ex.1006, 6. For instance, MATLAB process at processor ID uses the Send command to send the variable "a" to MATLAB process at processor ID+1 asynchronously, as shown in the above example from Trefethen.

**475. Third**, Menon describes how MATLAB calls the Send() command and communicates with the *local cluster node module*, explaining that "Send(pid,data)" "[s]end[s] data from one process to another" and includes the "pid" value as the destination and the "data" value as the expression. Ex.1005, Table 1. MATLAB provides that packet of information (the pid value and data value) to the MEX routine, which then decodes that packet by mapping the information to the corresponding MPI call (e.g., MPI_SEND). Ex.1005, 6. Menon describes that the

MultiMATLAB interface module acts as an interpretation layer that maps MPI calls (e.g., MPI_SEND) to an indirection table for the MEX routines, explaining that:

> The MultiMATLAB interface module provides parallel MEX Routines with access to the underlying communication layer. In our implementations, MEX Routines are granted access to MPI routines. Moreover, the indirection imposed by the interface module is transparent to the programmer. There are macros, provided to all parallel MEX Routines, which map MPI calls directly to the appropriate table offset while maintaining the syntax of MPI.

Ex.1005, 6. The MEX routine decodes that packet by mapping the information to the corresponding MPI call (e.g., MPI_SEND). Ex.1005, 6. Figure 2 of Menon below illustrates MATLAB (*single-node kernel*) communicating messages with the communication components of the MATLAB process, namely the MEX Routines, the MultiMATLAB interface module, the indirection table, and communication layer, (*local cluster node module*).




**Ex.1005, FIG. 2 (annotated)**

**476. Fourth**, as discussed in limitation [10.0], Menon discloses that the MultiMATLAB interface module of each MATLAB process acts as an interpretation layer that maps MPI calls to the indirection table for the MEX routines. Specifically, Menon teaches that:

> The MultiMATLAB interface module provides parallel MEX Routines with access to the underlying communication layer. In our implementations, MEX Routines are granted access to MPI routines. Moreover, the indirection imposed by the interface module is transparent to the programmer. There are macros, provided to all parallel MEX Routines, which map MPI calls directly to the appropriate table offset while maintaining the syntax of MPI.

Ex.1005, 6.

**477. Fifth**, "MPI: A Message-Passing Interface Standard" to the Message Passing Interface Forum ("MPIref," Ex.1017) describes how the MPI_SEND call uses the data (i.e., the data value in the send call) and the identification of the destination node (i.e., the pid value in the send call) and forwards the data to the destination node. Ex.1017, 15-19. MPIref discloses the details of what the parallel MEX Routines would be mapped to for the MPI calls; particularly, MPIref describes the "MPI_SEND" call, which needs or expects the send buffer information (the data to be transmitted) and the dest value (the identification of the destination node). Ex.1017, 15-21. MPIref teaches that "the **send** operation MPI_SEND" "specifies a **send buffer** in the sender memory from which the message data is taken" and that "[t]he send buffer specified by the MPI_SEND operation consists of **count** successive entries of the type indicated by **datatype**, starting with the entry at address **buf**." Ex.1017, 15, 16 (emphasis original). MPIref further discloses that, "[i]n addition to the data part" that "consists of a sequence of **count** values, each of the type indicated by **datatype,**"

> messages carry information that can be used to distinguish messages and selectively receive them. This information consists of a fixed number of fields, which we collectively call the **message envelope**. These fields are

> source
>
> destination
>
> tag
>
> communicator
>
> …
>
> The message destination is specified by the **dest** argument.

Ex.1017, 17, 18 (emphasis original). That is, MPIref teaches that MPI_SEND sends a message containing a data part (i.e., a message data) and a message envelope, where the data part includes the message data to be sent or transmitted and the message envelope includes an identification of the destination for the message data, as shown below.

Send buffer information

The syntax of the blocking send operation is given below.

Message envelope

MPI_SEND(buf, count, datatype, dest, tag, comm)

| | | |
|---|---|---|
| IN | buf | initial address of send buffer (choice) |
| IN | count | number of elements in send buffer (nonnegative integer) |
| IN | datatype | datatype of each send buffer element (handle) |
| IN | dest | rank of destination (integer) |
| IN | tag | message tag (integer) |
| IN | comm | communicator (handle) |

Ex.1017, 16 (annotated).

**478.** A POSITA would then recognize that execution of the Send command "Send (ID+1,a)" (*a first command*) by MATLAB in a MATLAB process generates a collection or package (*first packet*) of information that is needed by the corresponding MPI call (MPI_SEND), including (*containing*) (i) the variable "a" (*expression*), the data part that is associated with the send buffer information in MPI_SEND and that MPI_SEND transmits to the next MATLAB process at the next/destination processor of the MultiMATLAB architecture as part of its message, and (ii) an identification of the destination processor ("ID+1") (*target node*). In operation, the MEX routines (such as Send() and Recv) are run directly from the user interface. Ex.1005, 3; s*ee also* Ex.1006, 3, 6. The MEX routines (such as Send() is mapped (using the indirection table) to the analogous MPI calls (such as MPI_SEND()) found in the communication layer. Ex.1005, 3, 6. The communication layer, using the MPI calls, communicates over the interconnection network with the communication layer in the other nodes. Ex.1005, 3

**479.** Accordingly, Menon describing the calls of Send, Recv, and Bcast commands and Trefethen describing those calls as asynchronous, in view of Menon describing that the execution of the command "Send (ID+1,a)" (*a first command*) by MATLAB (*single-node kernel*) in a MATLAB process of processor ID of the MultiMATLAB architecture would generate a collection or package (*first packet*) of information that is needed by the corresponding MPI call (MPI_SEND), as

described by MPIref, including (*containing*) (i) the variable "a" (*expression*), the data part that is associated with the send buffer information in MPI_SEND and that MPI_SEND transmits to the next MATLAB process at the next/destination processor ID+1 of the MultiMATLAB architecture as part of its message, and (ii) an identification of the destination processor ("ID+1") (*target node*), renders obvious *wherein the asynchronous calls comprise a first command to create a first packet.*

#### b. [27.1] [*a first packet*] *containing: an expression to be sent as payload; and*

**480.** Menon, Trefethen, and MPIref render obvious *a first packet containing … an expression to be sent as payload* because Menon, Trefethen, and MPIref disclose MATLAB (*single-node kernel*) in a MATLAB process of processor ID executes the command "Send (ID+1,a)" (*a first command*) and generates (*to create*) a collection or package (*first packet*) of information including (*containing*) the variable "a" (*expression*), that MPI_SEND transmits (*to be sent*) as the data part (*as payload*) of its message to the next MATLAB process at the next/destination processor ID+1 of the MultiMATLAB architecture.

**481.** As was discussed in limitation [27.0], Menon describes the calls of Send and Recv commands and Trefethen describes those calls as asynchronous. Further, it was also shown in limitation [27.0] that Menon describes the execution of the command "Send (ID+1,a)" (*a first command*) by MATLAB (*single-node*

*kernel*) in a MATLAB process of processor ID of the MultiMATLAB architecture would generate a collection or package (*first packet*) of information that is needed by the corresponding MPI call (MPI_SEND), as described by MPIref, including (*containing*) (i) the variable "a" (*expression*), the data part that is associated with the send buffer information in MPI_SEND and that MPI_SEND transmits to the next MATLAB process at the next/destination processor ID+1 of the MultiMATLAB architecture as part of its message, and (ii) an identification of the destination processor ("ID+1") (*target node*).

**482.** Accordingly, Menon, Trefethen, and MPIref disclose MATLAB (*single-node kernel*) in a MATLAB process of processor ID executes the command "Send (ID+1,a)" (*a first command*) and generates (*to create*) a collection or package (*first packet*) of information including (*containing*) the variable "a" (*expression*), that MPI_SEND transmits (*to be sent*) as the data part (*as payload*) of its message to the next MATLAB process at the next/destination processor ID+1 of the MultiMATLAB architecture, rendering obvious [*a first packet containing*] *an expression to be sent as payload*.

#### c. [27.2] [*a first packet*] *containing … a target node where the expression should be sent;*

**483.** Menon, Trefethen, and MPIref render obvious [*a first packet containing …*] *a target node where the expression should be sent* because Menon, Trefethen, and MPIref disclose MATLAB (*single-node kernel*) in a MATLAB

process of processor ID executes the command "Send (ID+1,a)" (*a first command*) and generates (*to create*) a collection or package (*first packet*) of information including (*containing*) the variable "a" (*expression*), that MPI_SEND transmits (*to be sent*) as the data part (*as payload*) of its message to the next MATLAB process at the next/destination processor ID+1 of the MultiMATLAB architecture.

**484.** As was discussed in limitation [27.0], Menon describes the calls of Send and Recv commands and Trefethen describes those calls as asynchronous. Further, Menon describes the execution of the command "Send (ID+1,a)" (*a first command*) by MATLAB (*single-node kernel*) in a MATLAB process of processor ID of the MultiMATLAB architecture would generate a collection or package (*first packet*) of information that is needed by the corresponding MPI call (MPI_SEND), as described by MPIref, including (*containing*) (i) the variable "a" (*expression*), the data part that is associated with the send buffer information in MPI_SEND and that MPI_SEND transmits to the next MATLAB process at the next/destination processor ID+1 of the MultiMATLAB architecture as part of its message, and (ii) an identification of the destination processor ("ID+1") (*target node*).

**485.** Accordingly, Menon, Trefethen, and MPIref disclose MATLAB (*single-node kernel*) in a MATLAB process of processor ID executes the command "Send (ID+1,a)" (*a first command*) and generates (*to create*) a collection or package (*first packet*) of information including (*containing*) an identification of the

destination processor ("ID+1") (*target node*) to which MPI_SEND transmits (*where … should be sent*) the variable "a" (*expression*), the data part of its message, rendering obvious [*a first packet containing …*] *a target node where the expression should be sent*.

#### d. [27.3] *wherein the first command is configured to be called from within a single-node kernel;*

**486.** Menon, Trefethen, and MPIref render obvious *wherein the first command is configured to be called from within a single-node kernel* because Menon, Trefethen, and MPIref disclose the command "Send (ID+1,a)" (*a first command*) is executed (*is configured to be called*) by MATLAB (*from within a single-node kernel*) in a MATLAB process of processor ID to generate (i) the variable "a" (*expression*) and (ii) an identification of the destination processor ("ID+1") (*target node*).

**487.** As was discussed in limitation [27.0], Menon describes the calls of Send and Recv commands and Trefethen describes those calls as asynchronous. Further, Menon describes the execution of the command "Send (ID+1,a)" (*a first command*) by MATLAB (*single-node kernel*) in a MATLAB process of processor ID of the MultiMATLAB architecture would generate a collection or package (*first packet*) of information that is needed by the corresponding MPI call (MPI_SEND), as described by MPIref, including (*containing*) (i) the variable "a" (*expression*), the data part that is associated with the send buffer information in MPI_SEND and that

MPI_SEND transmits to the next MATLAB process at the next/destination processor ID+1 of the MultiMATLAB architecture as part of its message, and (ii) an identification of the destination processor ("ID+1") (*target node*).

**488.** Accordingly, Menon, Trefethen, and MPIref disclose the command "Send (ID+1,a)" (*a first command*) is executed (*is configured to be called*) by MATLAB (*from within a single-node kernel*) in a MATLAB process of processor ID to generate (i) the variable "a" (*expression*) and (ii) an identification of the destination processor ("ID+1") (*target node*), rendering obvious *wherein the first command is configured to be called from within a single-node kernel*.

**e. [27.4] *wherein the single-node kernel is configured to send the first packet to a local cluster node module; and***

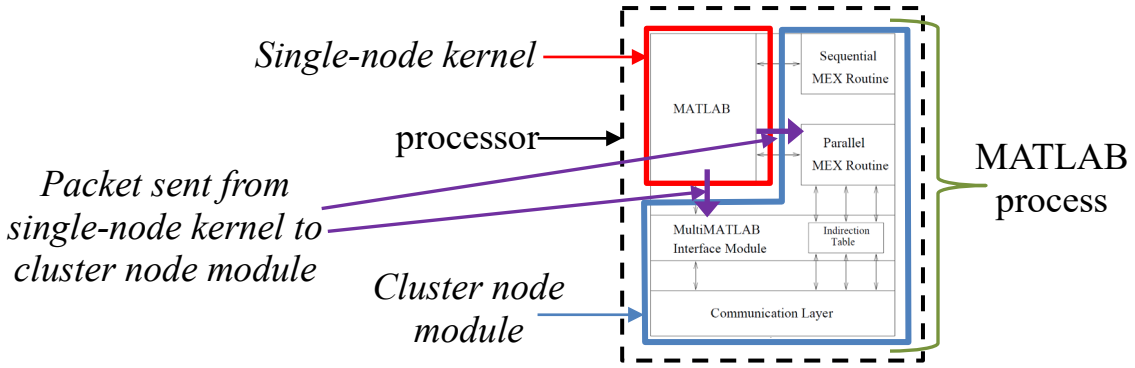
**489.** Menon, Trefethen, and MPIref render obvious *wherein the single-node kernel is configured to send the first packet to a local cluster node module* because Menon, Trefethen, and MPIref disclose MATLAB (*single-node kernel*) in a MATLAB process of processor ID executing the command Send (ID+1,a), generating the collection or package of information, and transmitting the collection or package of information to the communication components of the MATLAB process, namely the MEX Routines, the MultiMATLAB interface module, the indirection table, and communication layer, (*cluster node module*) of the same (*local*) MATLAB process.

**490.** As was discussed in limitation [27.0], Menon describes the calls of Send and Recv commands and Trefethen describes those calls as asynchronous. Menon further describes the execution of the command "Send (ID+1,a)" (*a first command*) by MATLAB (*single-node kernel*) in a MATLAB process of processor ID of the MultiMATLAB architecture would generate a collection or package (*first packet*) of information that is needed by the corresponding MPI call (MPI_SEND), as described by MPIref, including (*containing*) (i) the variable "a" (*expression*), the data part that is associated with the send buffer information in MPI_SEND and that MPI_SEND transmits to the next MATLAB process at the next/destination processor ID+1 of the MultiMATLAB architecture as part of its message, and (ii) an identification of the destination processor ("ID+1") (*target node*).

**491.** It was also shown in limitation [27.0] that MATLAB (*single-node kernel*) communicates messages with the communication components of the MATLAB process, namely the MEX Routines, the MultiMATLAB interface module, the indirection table, and communication layer, (*local cluster node module*) using point-to-point commands such as the Send command.

**492.** A POSITA would recognize then that the collection or package of information including the variable "a" and an identification of the destination processor ("ID+1") that is generated by MATLAB in a MATLAB process of processor ID executing the command Send (ID+1,a) is transmitted to the *local*

*cluster node module* of the MATLAB process so that the collection or package of information is sent to MATLAB processes on the destination processor ID+1 using the MPI call MPI_SEND, as shown below.




**Ex.1005, FIG. 2 (annotated)**

**493.** Accordingly, Menon, Trefethen, and MPIref disclose MATLAB (*single-node kernel*) in a MATLAB process of processor ID executing the command Send (ID+1,a) and generating the collection or package of information and transmitting the collection or package of information to the *cluster node module* of the same (*local*) MATLAB process, rendering obvious *wherein the single-node kernel is configured to send the first packet to a local cluster node module*.

**f. [27.5] *wherein the local cluster node module is configured to forward the expression to the target node.***

**494.** Menon, MPIref, and Trefethen render obvious *wherein the local cluster node module is configured to forward the expression to the target node* because Menon, Trefethen, and MPIref disclose that the communication layer

(*cluster node module*) of the same (*local*) MATLAB process transmits the variable "a" (*expression*) to the destination processor ("ID+1") (*target node*).

**495.** As was discussed in limitation [27.0], Menon describes the calls of Send and Recv commands and Trefethen describes those calls as asynchronous. Further, Menon describes the execution of the command "Send (ID+1,a)" (*a first command*) by MATLAB (*single-node kernel*) in a MATLAB process of processor ID of the MultiMATLAB architecture would generate a collection or package (*first packet*) of information that is needed by the corresponding MPI call (MPI_SEND), as described by MPIref, including (*containing*) (i) the variable "a" (*expression*), the data part that is associated with the send buffer information in MPI_SEND and that MPI_SEND transmits to the next MATLAB process at the next/destination processor ID+1 of the MultiMATLAB architecture as part of its message, and (ii) an identification of the destination processor ("ID+1") (*target node*).

**496.** Also, as was discussed in limitation [27.4], the collection or package of information including the variable "a" and an identification of the destination processor ("ID+1") that are generated by MATLAB in a MATLAB process of processor ID executing the command Send (ID+1,a) is transmitted to the *local cluster node module* of the MATLAB process so that the collection or package of information is sent to the MATLAB processes on processor ID+1.

**497.** A POSITA would recognize then that the collection or package of information including the variable "a" and an identification of the destination processor ("ID+1") is transmitted, by the *local cluster node module* of the MATLAB process using the MPI call MPI_SEND, to the next MATLAB process at the destination processor ("ID+1").

**498.** Accordingly, Menon, Trefethen, and MPIref disclose that the communication layer (*cluster node module*) of the same (*local*) MATLAB process transmits the variable "a" (*expression*) to the destination processor ("ID+1") (*target node*), rendering obvious *wherein the local cluster node module is configured to forward the expression to the target node*.

**37. Claim 28**

**a. [28.0] *The computer cluster of claim 27, wherein the asynchronous calls comprise a second command to create a second packet containing:***

**499.** Menon, Trefethen, and MPIref render obvious *wherein the asynchronous calls comprise a second command to create a second packet* because Menon describes the calls of Send and Recv commands and Trefethen describes those calls as asynchronous, in view of Menon describing that the execution of the command "Recv" (*a second command*) by MATLAB (*single-node kernel*) in a MATLAB process of processor ID of the MultiMATLAB architecture would generate a collection or package (*second packet*) of information that is needed by

the corresponding MPI call (MPI_RECV), as described by MPIref, including (*containing*) (i) the receive buffer (*a location where … is expected to be received*) for storing the variable "a" (*the expression*) and (ii) a source processor ID-1 (*a sending node from which … is expected to be received*) of the variable "a" (*the expression*).

**500. First**, it was shown in limitation [4.0] that each node of the MultiMATLAB architecture has a MATLAB process which in turn has a set of components to provide interconnection with other modules, namely, the "MEX Routines," the "MultiMATLAB interface module," an "indirection table" and "communication layer." (*a local cluster node module*). Ex.1005, 3, 5. Figure 2 of Menon below shows the *local cluster node module*.

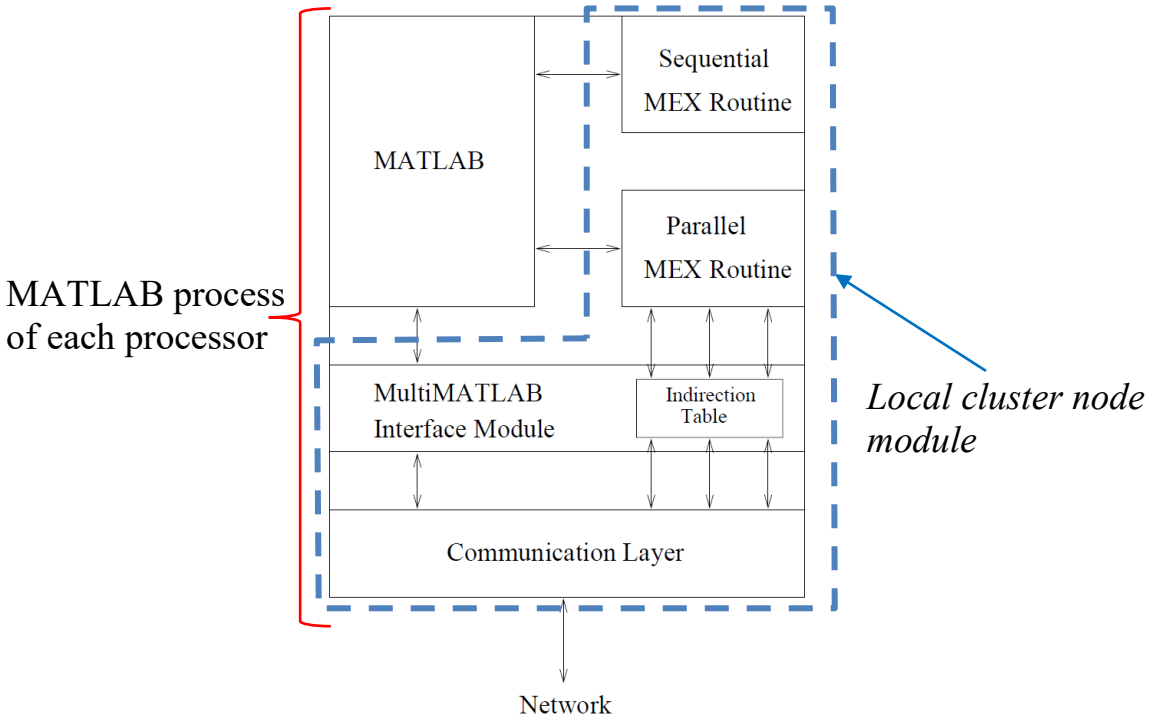
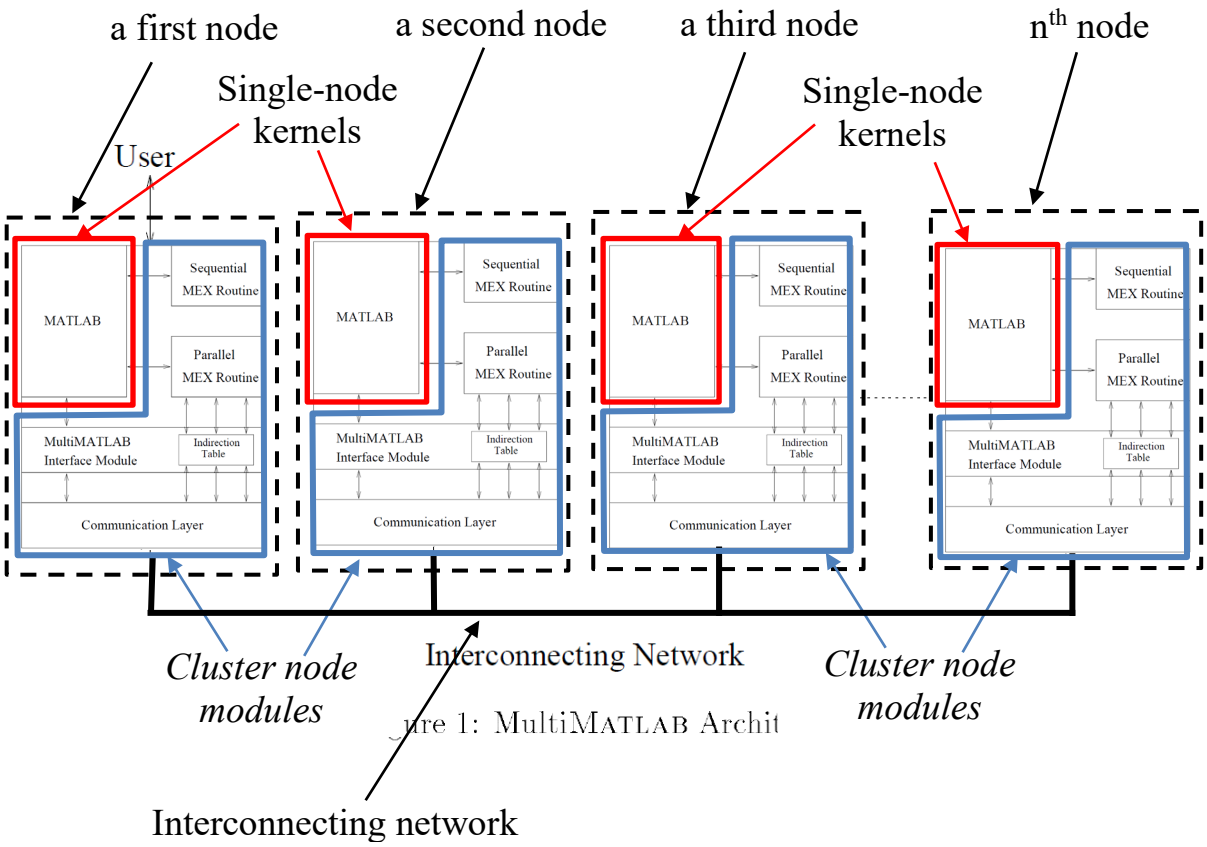



Figure 2: MATLAB Process Address Space

**Ex.1005, FIG. 2 (annotated)**

**501.** A combination of FIG. 1 and FIG. 2 of Menon below shows the relationship between the communication components (i.e., MEX routines, MultiMATLAB interface module, the communication layer), and MATLAB (*the single-node kernel*) for each node of the MultiMATLAB architecture.




**Ex.1005, FIGs. 1 and 2 (combined and annotated)**

**502. Second**, it was shown in limitation [1.2] that Menon teaches MEX routines having code that implement standard variants of MPI calls to enable SPMD point-to-point communication so that the MATLAB processes can communicate tasks and data directly with each other. Ex.1005, 9. The "MEX routines" provide "[a]ll parallel functionality in the MultiMATLAB system." Ex.1005, 3. For example, the MEX routines send the "commands over the underlying communication layer to the processes corresponding to the specified IDs." Ex.1005, 4. Menon describes using "MPI as [the] underlying communication substrate" in its

implementations of the MultiMATLAB architecture, although "a specific communication standard is not fundamental to the architecture." Ex.1005, 5. The choice of MPI is "due [to] its broad popularity, its suitability to high performance parallel computing, and the virtually universal availability of vendor-optimized versions on modern parallel computers." Ex.1005, 5. One implementation is "designed for the IBM SP2, a modern high performance distributed memory multiprocessor. This implementation is based upon MPI-F [], IBM's proprietary version of MPI specifically optimized for the SP2." Ex.1005, 5. The other implementation "is based upon MPICH [], a popular public domain version of MPI." Ex.1005, 5.

**503.** As discussed in limitation [26.2], Menon describes example point-to-point commands that the MATLAB processes in the MultiMATLAB architecture use to communicate with each other, including SPMD calls such as the "Send(pid,data)" command that "send[s] data from one process to another" and the "Recv(pid)" command that "receive[s] data sent from another process," shown in Table 1 of Menon below.

| MultiMATLAB commands | |
|---|---|
| Starting and stopping MultiMATLAB (MPICH only) | |
| Start(n) | Initializes n remote MATLAB processes. |
| Quit | Closes remote MATLAB processes. |
| Running commands on multiple processors | |
| Nproc | Returns number of MATLAB processes. |
| ID | Returns ID of calling MATLAB process. |
| Eval(pid,'command') | Evaluates a command on one or more MATLAB process. |
| Master/Slave Communication | |
| Put(pid,dataname) | Put data from the master process onto one or more remote processes. |
| Get(pid,dataname) | Get data from a remote process onto the master process. |
| SPMD Communication | |
| Send(pid,data) | Send data from one process to another. |
| Recv(pid) | Receive data sent from another process. |
| Barrier | Synchronize processes. |
| Bcast(pid,data) | Broadcasts data from processor pid to all processes. |
| Sum(data) | Adds data across all processors to form a global sum. |
| Collect(data) | Collects local data segments into a single global one. |

Table 1: Selected commands in the MultiMATLAB system

Commands for point-to-point communication (including Send and Recv)

**Ex.1005, Table 1 (annotated).**

Ex.1005, 9.

**504.** Also as discussed in limitation [26.2], Trefethen teaches the processors of the MultiMATLAB architecture communicate with each other *using asynchronous calls*, such as the "collection of [MultiMATLAB] commands such as Send" and Recv:

```
if ID==0                  % first process: send
   a = 1
   Send(ID+1,a)
elseif ID == Nproc-1      % last process: receive and double
   a = 2*Recv
else                      % middle processes: receive, double, and send
   a = 2*Recv
   Send(ID+1,a)
end;
```

Process 0 creates the variable a with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of a, and sends it along to process 2; and so on. If there are six processors the command `Eval( 'cycle' )` produces the output

```
a = 1
a = 2
a = 4
a = 8
a = 16
a = 32
```

*Asynchronous calls*

Recv command executed by single-node kernel at processor ID to receive data or variable "a" from processor ID-1

The processes run asynchronously, but since each `Send` command is only executed after the corresponding `Recv` has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning.

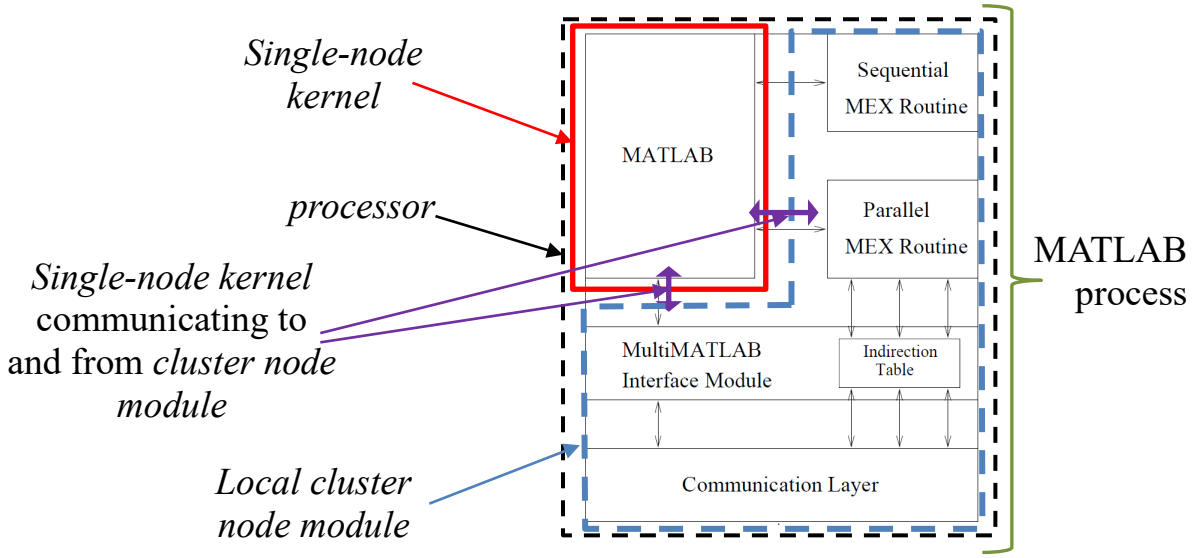
Ex.1006, 6, 10 (annotated). For example, processor ID communicates with processor ID+1 *using asynchronous calls* Send and Recv commands. Ex.1006, 6. For instance, MATLAB process at processor ID uses the Recv command to receive the variable "a" from MATLAB process at processor ID-1 asynchronously, as shown in the above example from Trefethen.

**505. Third**, Menon describes how MultiMATLAB interprets the Recv() command, explaining that "Recv(pid)" "[r]eceive[s] data sent from another process" and includes the "pid" value as the source of the expected data. Ex.1005, Table 1. MATLAB provides that packet of information (the pid value) to the MEX routine, which then decodes that packet by mapping the packet's information to the corresponding MPI call (e.g., MPI_RECV). Ex.1005, 6. Menon describes that the

MultiMATLAB interface module acts as an interpretation layer that maps MPI calls (e.g., MPI_RECV) to an indirection table for the MEX routines, explaining that:

> The MultiMATLAB interface module provides parallel MEX Routines with access to the underlying communication layer. In our implementations, MEX Routines are granted access to MPI routines. Moreover, the indirection imposed by the interface module is transparent to the programmer. There are macros, provided to all parallel MEX Routines, which map MPI calls directly to the appropriate table offset while maintaining the syntax of MPI.

Ex.1005, 6. The MEX routine decodes that packet, mapping the packet's information to the corresponding MPI call (e.g., MPI_RECV). Ex.1005, 6. Figure 2 of Menon below illustrates MATLAB (*single-node kernel*) communicating messages with the communication components of the MATLAB process, namely the MEX Routines, the MultiMATLAB interface module, the indirection table, and communication layer, (*local cluster node module*).




**Ex.1005, FIG. 2 (annotated)**

**506. Fourth**, as discussed in limitation [10.0], Menon discloses that the MultiMATLAB interface module of each MATLAB process acts as an interpretation layer that maps MPI calls to the indirection table for the MEX routines. Specifically, Menon teaches that:

> The MultiMATLAB interface module provides parallel MEX Routines with access to the underlying communication layer. In our implementations, MEX Routines are granted access to MPI routines. Moreover, the indirection imposed by the interface module is transparent to the programmer. There are macros, provided to all parallel MEX Routines, which map MPI calls directly to the appropriate table offset while maintaining the syntax of MPI.

Ex.1005, 6.

**507. Fifth**, MPIref discloses the details of what the parallel MEX Routines would be mapped to for the MPI calls; particularly, MPIref describes the "MPI_RECV" call, which needs or expects the receive buffer information (the data transmitted) and the source (the identification of the source node). Ex.1017, 15-21. MPIref teaches that a process receives a message that is transmitted by MPI_SEND call "with the **receive** operation MPI_RECV." Ex.1017, 16 (emphasis original). "The message to be received is selected according to the value of its envelope, and the message data is stored into the **receive buffer**." Ex.1017, 16 (emphasis original). "The receive buffer consists of the storage containing **count** consecutive elements of the type specified by **datatype**, starting at address **buf**." Ex.1017, 16 (emphasis original).

**508.** With reference to the message envelope, MPIref explains that "messages carry information that can be used to distinguish messages and selectively receive them. This information consists of a fixed number of fields, which we collectively call the **message envelope**. These fields are

source

destination

tag

communicator

…

> The message destination is specified by the **dest** argument."

Ex.1017, 18 (emphasis original). That is, MPIref teaches that MPI_RECV is used to receive messages associated with a message envelope having fields for the source and destination, where the message data is stored into the **receive buffer** that consists of a storage containing **count** consecutive elements of the type specified by **datatype**, starting at address **buf**, as shown below.

Recv buffer information  Message envelope

MPI_RECV (buf, count, datatype, source, tag, comm, status)

| | | |
|---|---|---|
| OUT | buf | initial address of receive buffer (choice) |
| IN | count | number of elements in receive buffer (integer) |
| IN | datatype | datatype of each receive buffer element (handle) |
| IN | source | rank of source (integer) |
| IN | tag | message tag (integer) |
| IN | comm | communicator (handle) |
| OUT | status | status object (Status) |

Ex.1017, 19 (annotated).

MPIref also describes that the MPI_RECV call "en_queues" (i.e., stores in a queue) the information received by the MPI_RECV call. Ex.1017, 164.

**509.** A POSITA would then recognize that execution of the Recv command "Recv" (*a second command*) by MATLAB in a MATLAB process at processor ID+1 generates a collection or package (*second packet*) of information that is

needed by the corresponding MPI call (MPI_RECV) for receiving the message transmitted by the MPI_SEND call from processor ID, including (*containing*) (i) the starting address of the receive buffer for storing the variable "a" (*expression*) that is transmitted by the processor ID-1 using the MPI call MPI_SEND, and (ii) an identification of the source processor ID-1 (*a sending node*) of the variable "a".

**510.** Accordingly, Menon describing the calls of Send, Recv, and Bcast commands and Trefethen describing those calls as asynchronous, in view of Menon describing that the execution of the command "Recv" (*a second command*) by MATLAB (*single-node kernel*) in a MATLAB process of processor ID of the MultiMATLAB architecture would generate a collection or package (*second packet*) of information that is needed by the corresponding MPI call (MPI_RECV), as described by MPIref, including (*containing*) (i) the receive buffer (*a location where … is expected to be received*) for storing the variable "a" (*the expression*) and (ii) a source processor ID-1 (*a sending node from which … is expected to be received*) of the variable "a" (*the expression*), renders obvious *wherein the asynchronous calls comprise a second command to create a second packet.*

**b. [28.1] [*a second packet*] *containing: a location where the expression is expected to be received; and***

**511.** Menon, Trefethen, and MPIref render obvious [*a second packet*] *containing: a location where the expression is expected to be received* because Menon, Trefethen, and MPIref disclose MATLAB (*single-node kernel*) in a

MATLAB process of processor ID executes the command "Recv" and generates a collection or package (*second packet*) of information that includes (*containing*) the receive buffer (*a location where … is expected to be received*) for storing the variable "a" (*the expression*).

**512.** As was discussed in limitation [28.0], Menon describes the calls of Send and Recv commands and Trefethen describes those calls as asynchronous. Further, Menon describes that the execution of the command "Recv" (*a second command*) by MATLAB (*single-node kernel*) in a MATLAB process of processor ID of the MultiMATLAB architecture would generate a collection or package (*second packet*) of information that is needed by the corresponding MPI call (MPI_RECV), as described by MPIref, including (*containing*) (i) the receive buffer (*a location where … is expected to be received*) for storing the variable "a" (*the expression*) and (ii) a source processor ID-1 (*a sending node from which … is expected to be received*) of the variable "a" (*the expression*).

**513.** Accordingly, Menon, Trefethen, and MPIref disclose MATLAB (*single-node kernel*) in a MATLAB process of processor ID executing the command "Recv" and generating a collection or package (*second packet*) of information that includes (*containing*) the receive buffer (*a location where … is expected to be received*) for storing the variable "a" (*the expression*), rendering

obvious [*a second packet*] *containing: a location where the expression is expected to be received*.

**c. [28.2] [*a second packet*] *containing … a sending node from which the expression is expected to be received;***

**514.** Menon, Trefethen, and MPIref render obvious [*a second packet*] *containing … a sending node from which the expression is expected to be received* because Menon, Trefethen, and MPIref disclose MATLAB (*single-node kernel*) in a MATLAB process of processor ID executes the command "Recv" and generates a collection or package (*second packet*) of information that includes (*containing*) a source processor ID-1 (*a sending node from which … is expected to be received*) of the variable "a" (*the expression*).

**515.** As was discussed in limitation [28.0], Menon describes the calls of Send and Recv commands and Trefethen describes those calls as asynchronous. Further, Menon describes that the execution of the command "Recv" (*a second command*) by MATLAB (*single-node kernel*) in a MATLAB process of processor ID of the MultiMATLAB architecture would generate a collection or package (*second packet*) of information that is needed by the corresponding MPI call (MPI_RECV), as described by MPIref, including (*containing*) (i) the receive buffer (*a location where … is expected to be received*) for storing the variable "a" (*the expression*) and (ii) a source processor ID-1 (*a sending node from which … is expected to be received*) of the variable "a" (*the expression*).

**516.** Accordingly, Menon, Trefethen, and MPIref disclose MATLAB (*single-node kernel*) in a MATLAB process of processor ID executing the command "Recv" and generating a collection or package (*second packet*) of information that includes (*containing*) a source processor ID-1 (*a sending node from which … is expected to be received*) of the variable "a" (*the expression*), rendering obvious [*a second packet*] *containing … a sending node from which the expression is expected to be received*.

**d. [28.3] *wherein the second command is configured to be called from within a single-node kernel;***

**517.** Menon, Trefethen, and MPIref render obvious *wherein the second command is configured to be called from within a single-node kernel* because Menon, Trefethen, and MPIref disclose the command Recv (*a second command*) is executed (*is configured to be called*) by MATLAB (*from within a single-node kernel*) in a MATLAB process of processor ID to generate a collection or package (*second packet*) including (*containing*) (i) the receive buffer (*a location where … is expected to be received*) for storing the variable "a" (*the expression*) and (ii) a source processor ID-1 (*a sending node from which … is expected to be received*) of the variable "a" (*the expression*).

**518.** As was discussed in limitation [28.0], Menon describes the calls of Send and Recv commands and Trefethen describes those calls as asynchronous. Further, Menon describes that the execution of the command "Recv" (*a second*

*command*) by MATLAB (*single-node kernel*) in a MATLAB process of processor ID of the MultiMATLAB architecture would generate a collection or package (*second packet*) of information that is needed by the corresponding MPI call (MPI_RECV), as described by MPIref, including (*containing*) (i) the receive buffer (*a location where … is expected to be received*) for storing the variable "a" (*the expression*) and (ii) a source processor ID-1 (*a sending node from which … is expected to be received*) of the variable "a" (*the expression*).

**519.** Accordingly, Menon, Trefethen, and MPIref disclose the command Recv (*a second command*) is executed (*is configured to be called*) by MATLAB (*from within a single-node kernel*) in a MATLAB process of processor ID to generate a collection or package (*second packet*) including (*containing*) (i) the receive buffer (*a location where … is expected to be received*) for storing the variable "a" (*the expression*) and (ii) a source processor ID-1 (*a sending node from which … is expected to be received*) of the variable "a" (*the expression*), rendering obvious *wherein the second command is configured to be called from within a single-node kernel*.

**e. [28.4] *wherein the single-node kernel is configured to send the second packet to a local cluster node module; and***

**520.** Menon, Trefethen, and MPIref render obvious *wherein the single-node kernel is configured to send the second packet to a local cluster node module*

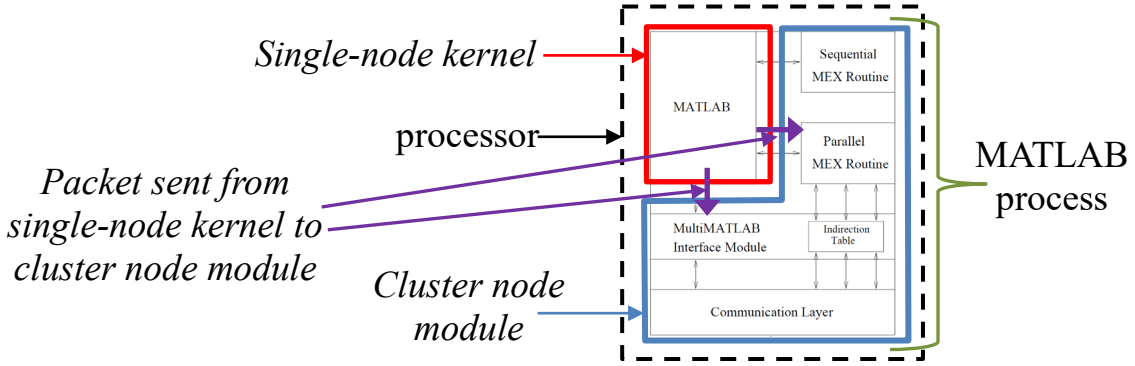
because Menon, Trefethen, and MPIref disclose MATLAB (*single-node kernel*) in a MATLAB process of processor ID executes the command Recv, generates the collection or package of information, and transmits the collection or package of information to the communication components of the MATLAB process, namely the MEX Routines, the MultiMATLAB interface module, the indirection table, and communication layer, (*cluster node module*) of the same (*local*) MATLAB process.

**521.** As was discussed in limitation [28.0], Menon describes the calls of Send and Recv commands and Trefethen describes those calls as asynchronous. Further, Menon describes that the execution of the command "Recv" (*a second command*) by MATLAB in a MATLAB process (*single-node kernel*) of processor ID of the MultiMATLAB architecture would generate a collection or package (*second packet*) of information that is needed by the corresponding MPI call (MPI_RECV), as described by MPIref, including (*containing*) (i) the receive buffer (*a location where … is expected to be received*) for storing the variable "a" (*the expression*) and (ii) a source processor ID-1 (*a sending node from which … is expected to be received*) of the variable "a" (*the expression*). In operation, the MEX routines (such as and Recv) are run directly from the user interface. Ex.1005, 3; s*ee also* Ex.1006, 3, 6. The MEX routines (such Recv() are mapped (using the indirection table) to the analogous MPI calls (such as MPI_RECV()) found in the communication layer. Ex.1005, 3, 6. The communication layer, using the MPI calls,

communicates over the interconnection network with the communication layer in the other nodes. Ex.1005, 3

**522.** It was also shown in limitation [28.0] that MATLAB (*single-node kernel*) communicates messages with the communication components of the MATLAB process, namely the MEX Routines, the MultiMATLAB interface module, the indirection table, and communication layer, (collectively, *local cluster node module*) using point-to-point commands such as the Recv command.

**523.** A POSITA would recognize then that the collection or package of information including the receive buffer for storing the variable "a" and a source processor ID-1 of the variable "a" that is generated by MATLAB in a MATLAB process of processor ID executing the command Recv is transmitted to *local cluster node module* of the MATLAB process so that the collection or package of information is received from the MATLAB process on processor ID-1 via the MPI call MPI_RECV, as shown below.




**Ex.1005, FIG. 2 (annotated)**

**524.** Accordingly, Menon, Trefethen, and MPIref disclose MATLAB (*single-node kernel*) in a MATLAB process of processor ID executes the command Recv to generate the collection or package of information and then transmits the collection or package of information to the *cluster node module* of the same (*local*) MATLAB process, rendering obvious *wherein the single-node kernel is configured to send the second packet to a local cluster node module*.

**f. [28.5] *wherein the local cluster node module is configured to store the second packet contents in a message receiving queue.***

**525.** Menon, Trefethen, and MPIref render obvious *wherein the local cluster node module is configured to store the second packet contents in a message receiving queue* because Menon, Trefethen, and MPIref disclose that MATLAB in a MATLAB process of processor ID executes the command Recv and generates a collection or package (*second packet*) of information including (i) the starting address of the receive buffer and (ii) a source processor ID-1 of the variable "a," and then transmits to the corresponding MPI call (MPI_RECV), where the contents of the package (*the second packet contents*) are received and placed in a queue (*store in a message receiving queue*).

**526.** As discussed in limitation [28.4], Menon, Trefethen, and MPIref disclose MATLAB (*single-node kernel*) in a MATLAB process of processor ID executes the command Recv, generates a collection or package (*second packet*) of

information that is needed by the corresponding MPI call (MPI_RECV), and then transmits the collection or package of information to the *cluster node module* of the same (*local*) MATLAB process, the package of information including (i) the starting address of the receive buffer and (ii) a source processor ID-1 of the variable "a".

**527.** Also, as was discussed in limitation [28.4], the collection or package of information including the starting address of the receive buffer and the source processor ID-1 of the variable "a" is transmitted to the *local cluster node module* of the MATLAB process so that the collection or package of information is received at the MATLAB processes on processor ID.

**528.** A POSITA would recognize then that the collection or package of information is received from the MATLAB process at processor ID-1 by the *local cluster node module* of the MATLAB process using the MPI call MPI_RECV.

**529.** MPIref provides example code showing the MPI_RECV call in operation. Ex.1017, 163-164. In a snippet of this code, the MPI_RECV call is executed and the data obtained from the call is stored in variables named "client_tag," "client_source," and "client_rank_in_new_world":

```
int Do_server(server_comm)
MPI_Comm server_comm;
{
    void init_queue();
    int en_queue(), de_queue(); /* keep triplets of integers
                                   for later matching (fns not shown) */

    MPI_Comm comm;
    MPI_Status status;
    int client_tag, client_source;
    int client_rank_in_new_world, pairs_rank_in_new_world;
    int buffer[10], count = 1;

    void *queue;
    init_queue(&queue);

    for (;;)
    {
        MPI_Recv(buffer, count, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG,
                 server_comm, &status); /* accept from any client */

        /* determine client: */
        client_tag = status.MPI_TAG;
        client_source = status.MPI_SOURCE;
        client_rank_in_new_world = buffer[0];
```

MPI_RECV call in the communication layer (*cluster node module*)

Source (*sending node*)

Contents of collection or package of information from MPI_RECV (*second packet contents*)

Receive buffer (*location for receiving expression*)

Ex.1017, 163.

**530.** Subsequently, the code "en_queues" (i.e., stores in a queue) the information received by the MPI_RECV call (the information in the variables named "client_tag," "client_source," and "client_rank_in_new_world):

```
en_queue(queue, client_tag, client_source,
                            client_rank_in_new_world);
```

Ex.1017, 164.

**531.** Accordingly, Menon, Trefethen, and MPIref disclose that MATLAB in a MATLAB process of processor ID executes the command Recv, generates a collection or package (*second packet*) of information including (i) the starting address of the receive buffer and (ii) a source processor ID-1 of the variable "a," and then transmits to the corresponding MPI call (MPI_RECV), where the contents of the package (*the second packet contents*) are received and placed in a queue (*store in a message receiving queue*), rendering obvious *wherein the local cluster node module is configured to store the second packet contents in a message receiving queue*.

**38.Claim 31**

**a. [31.0] *The computer cluster of claim 4, wherein intercommunication among the plurality of single-node kernels during thread execution is enabled by the plurality of cluster node modules, and wherein the computer cluster is configured to permit exchange of information between nodes during the course of a parallel computation.***

**532.** Menon, Trefethen, and MPIref render obvious *wherein intercommunication among the plurality of single-node kernels during thread*

*execution is enabled by the plurality of cluster node modules, and wherein the computer cluster is configured to permit exchange of information between nodes during the course of a parallel computation* because Menon teaches that the processors of MultiMATLAB have a set of components for communication with the other nodes (*the cluster node module*) that includes (i) MEX routines that enable intercommunication among the processors and (ii) a communication layer that uses MPI as the communication substrate, MPIref teaches that MPI communication enables threaded execution, and Trefethen teaches that MultiMATLAB's commands are executed simultaneously (*parallel*) and kernels are configured to exchange data during the parallel computation.

**533. First**, as discussed in limitation [4.0], Menon discloses that each node has communication components, i.e., MEX Routines, MultiMATLAB interface module, indirection table, and communication layer, that teach a *cluster node module* of that node, as shown in Figure 2 of Menon below.

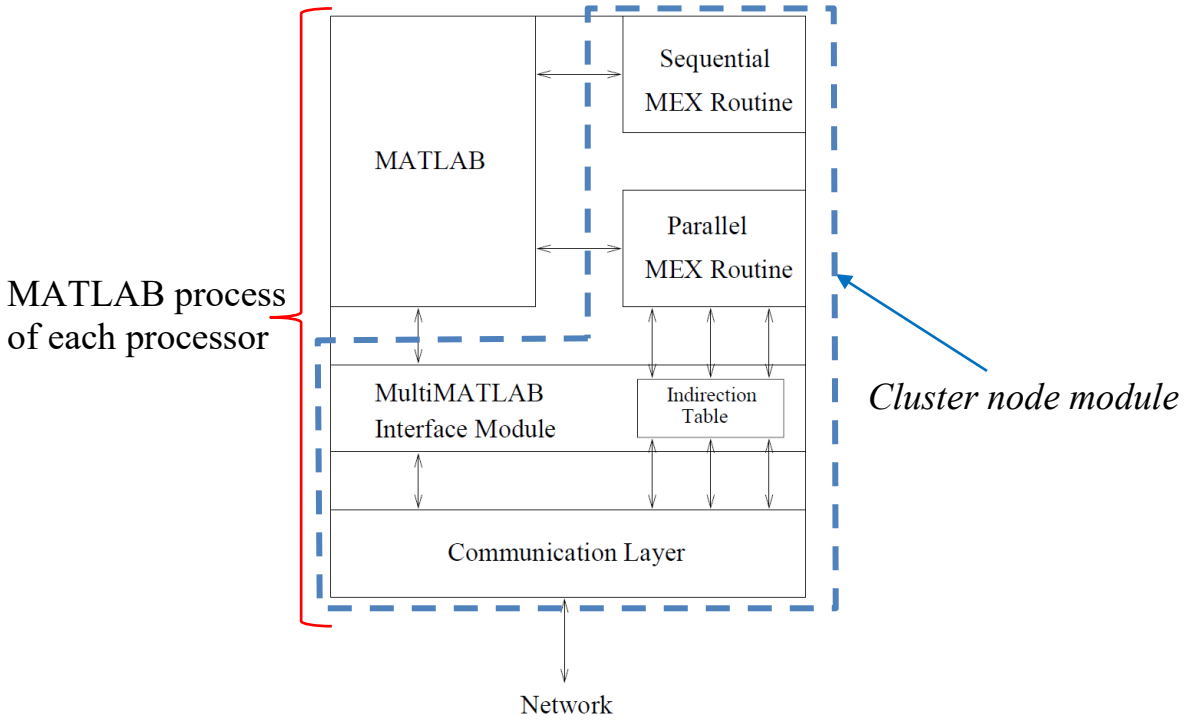



Figure 2: MATLAB Process Address Space

**Ex.1005, FIG. 2 (annotated)**

**534. Second**, as discussed in limitation [1.2], it was shown that Menon teaches MEX routines having code that implement standard variants of MPI calls to enable SPMD point-to-point communication so that the MATLAB processes can communicate tasks and data directly with each other. Ex.1005, 9. The "MEX routines" provide "[a]ll parallel functionality in the MultiMATLAB system." Ex.1005, 3. For example, the MEX Routines send the "commands over the underlying communication layer to the processes corresponding to the specified IDs." Ex.1005, 4. Menon describes using "MPI as [the] underlying communication substrate" in its implementations of the MultiMATLAB architecture, although "a

specific communication standard is not fundamental to the architecture." Ex.1005, 5. The choice of MPI is "due [to] its broad popularity, its suitability to high performance parallel computing, and the virtually universal availability of vendor-optimized versions on modern parallel computers." Ex.1005, 5. One implementation is "designed for the IBM SP2, a modern high performance distributed memory multiprocessor. This implementation is based upon MPI-F [], IBM's proprietary version of MPI specifically optimized for the SP2." Ex.1005, 5. The other implementation "is based upon MPICH [], a popular public domain version of MPI." Ex.1005, 5.

**535.** Table 1 of Menon below shows MultiMATLAB commands implemented as MEX Routines, including SPMD calls such as "Send(pid,data)" command that "send[s] data from one process to another," the "Recv(pid)" command that "receive[s] data sent from another process," and the "Bcast(pid,data)" command that "broadcasts data from processor pid to all processes":

| MultiMATLAB commands | |
|---|---|
| Starting and stopping MultiMATLAB (MPICH only) | |
| Start(n) | Initializes n remote MATLAB processes. |
| Quit | Closes remote MATLAB processes. |
| Running commands on multiple processors | |
| Nproc | Returns number of MATLAB processes. |
| ID | Returns ID of calling MATLAB process. |
| Eval(pid,'command') | Evaluates a command on one or more MATLAB process. |
| Master/Slave Communication | |
| Put(pid,dataname) | Put data from the master process onto one or more remote processes. |
| Get(pid,dataname) | Get data from a remote process onto the master process. |
| SPMD Communication | |
| Send(pid,data) | Send data from one process to another. |
| Recv(pid) | Receive data sent from another process. |
| Barrier | Synchronize processes. |
| Bcast(pid,data) | Broadcasts data from processor pid to all processes. |
| Sum(data) | Adds data across all processors to form a global sum. |
| Collect(data) | Collects local data segments into a single global one. |

Table 1: Selected commands in the MultiMATLAB system

Commands implemented in MEX Routines

**Ex.1005, Table 1 (annotated)**

**536. Third**, Menon discloses that a MATLAB process of each processor of the MultiMATLAB architecture evaluates mathematical expressions while simultaneously communicating to distribute the results of the evaluations to the other MATLAB processes. Ex.1005, 11-12. Discussing "a parallel MATLAB implementation of conjugate gradients that uses the SPMD message passing," Menon explains that:

> the matrix A and the different vectors are each distributed by row over all processes as in Figure 6. In each iteration, a process only does the computation required for its local data. However, communication is required for two different operations: the dot-products needed to compute

> *rho* and *alpha* and the matrix-vector multiply needed to compute *w_local*. The dot products only require communication of a single value over all processes. On the other hand, the matrix-vector multiplication needed for *w_local* requires the global vector *p*, and, thus, more expensive communication.

Ex.1005, 11-13 (emphasis original). That is, each MATLAB process performs its own computations while communicating values that may be needed for the computations with other MATLAB processes, as shown in FIG. 4b of Menon shown below. Ex.1005, 11, FIG. 4. A POSITA would recognize that the computations or mathematical evaluations are performed by MATLAB (*single-node kernel*) in a MATLAB process.

Multiple processors evaluating or computing mathematical expressions

```
r = b;
rho = r'*r;
k = 0;
while((k < kmax))
  k = k+1;
  if (k == 1)
    p = r;
  else
    beta = rho/oldrho;
    p = r + beta*p;
  end

  w = A * p;

  alpha = rho/(p'*w);
  x = x + alpha * p;
  r = r - alpha * w;
  oldrho = rho;
  rho = r'*r;
end
```

a. Sequential MATLAB

```
r_local = b_local;
rho = Sum(r_local'*r_local);
k = 0;
while((k < kmax))
  k = k+1;
  if (k == 1)
    p_local = r_local;
  else
    beta = rho/oldrho;
    p_local = r_local + beta*p_local;
  end

  p = Gather(p_local);
  w_local = A_local * p;

  alpha = rho/(Sum(p_local'*w_local));
  x_local = x_local + alpha * p_local;
  r_local = r_local - alpha * w_local;
  oldrho = rho;
  rho = Sum(r_local'*r_local);
end
```

b. Message Passing MATLAB

Figure 4: Conjugate Gradients

Inter-processor communications take place for these computations

**Ex.1005, FIG. 4 (annotated)**

**537. Fourth**, Trefethen provides further details of the SPMD communication commands and the evaluation of the MultiMATLAB's commands. Ex.1006, 5-7. Trefethen describes a MATLAB process on a processor of the MultiMATLAB architecture using the SPMD point-to-point commands to communicate mathematical results to another MATLAB process on another processor. Ex.1006, 3-7. Trefethen also teaches that MultiMATLAB's commands are "evaluated simultaneously on all processes." Ex.1006, 6. Specifically, Trefethen

provides the following example code where (i) the cluster is configured to exchange information between nodes during the course of parallel computation, and (ii) a MATLAB process communicates a mathematical result to a next MATLAB process, which in turn communicates the next mathematical result to the next MATLAB process, and so on, using SPMD calls such as Send() and Recv():

Instruction for MATLAB process ID=0 to create a=1 and send "a" to MATLAB process ID=1

Instruction for the last MATLAB process to receive the variable a and double it

```
if ID==0                    % first process: send
   a = 1
   Send(ID+1,a)
elseif ID == Nproc-1        % last process:
   a = 2*Recv
else                        % middle proces
   a = 2*Recv
   Send(ID+1,a)
end;
```

Instruction for a middle MATLAB process ID to receive the variable "a," double it, and send the doubled variable to the next MATLAB process ID=ID+1

Process 0 creates the variable a with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of a, and sends it along to process 2; and so on. If there are six processors the command `Eval( 'cycle' )` produces the output

```
a = 1
a = 2
a = 4
a = 8
a = 16
a = 32
```

SPMD point to point communication among processors of the MultiMATLAB architecture

*user instructions*

Ex.1006, 6 (annotated).

**538. Fifth**, MPIref discloses that MPI (which is implemented in Menon's communication layer of *the cluster node module*) enables "sequential" or "multi-threaded" processing and that "[c]are has been taken to make MPI 'thread-safe.'"

Ex.1017, 12. The "desired interaction of MPI with threads is that concurrent threads be allowed to execute MPI calls." Ex.1017, 12.

**539.** Accordingly, Menon teaching that the processors of MultiMATLAB have a set of components for communication with the other nodes (*the cluster node module*) that includes (i) MEX routines that enable intercommunication among the processors and (ii) a communication layer that uses MPI as the communication substrate, in view of MPIref's disclosure that MPI communication enables threaded execution, further in view of Trefethen teaching that MultiMATLAB's commands are executed simultaneously (*parallel*) and kernels are configured to exchange data during the parallel computation, renders obvious *wherein intercommunication among the plurality of single-node kernels during thread execution is enabled by the plurality of cluster node modules, and wherein the computer cluster is configured to permit exchange of information between nodes during the course of a parallel computation*.

**39. Claim 32**

**a. [32.0] *The computer cluster of claim 4, wherein each of the single-node kernels are configured to call a send command involving creating a packet containing:***

**540.** Limitation [32.0] is disclosed or rendered obvious for the same reasons presented above for limitation [27.0].

**b. [32.1] *an expression to be sent as payload; and***

**541.** Limitation [32.1] is disclosed or rendered obvious for the same reasons presented above for limitation [27.1].

**c. [32.2] *where the expression should be sent;***

**542.** Limitation [32.2] is disclosed or rendered obvious for the same reasons presented above for limitation [27.2].

**d. [32.3] *wherein, upon reception of the send command by a local cluster node module associated with the single-node kernel, the local cluster node module is configured to decode the packet and forward a payload to the cluster node module specified in the packet;***

**543.** Menon, Trefethen, and MPIref render obvious *wherein, upon reception of the send command by a local cluster node module associated with the single-node kernel, the local cluster node module is configured to decode the packet and forward a payload to the cluster node module specified in the packet* because Menon teaches using a Send command from MATLAB (*a single-node kernel*) of a MATLAB process that sends information to the corresponding MPI call (MPI_Send) in the communication layer of the same MATLAB process (*local cluster node module*), in view of MPIref teaching the MPI_SEND call that takes in the information and then transmits the information to the destination node, as taught by Trefethen, the transmission being to the communication layer (*cluster node module specified in the packet*) of that destination node as taught by Menon.

**544.** As discussed in limitation [1.2], Menon teaches MEX routines having code that implement standard variants of MPI calls to enable SPMD point-to-point communication so that the MATLAB processes can communicate tasks and data directly with each other. Table 1 of Menon below shows MultiMATLAB commands implemented as MEX routines, including SPMD calls such as "Send(pid,data)" command that "send[s] data from one process to another" and the "Recv(pid)" command that "receive[s] data sent from another process":

| MultiMATLAB commands | |
|---|---|
| Starting and stopping MultiMATLAB (MPICH only) | |
| Start(n) | Initializes n remote MATLAB processes. |
| Quit | Closes remote MATLAB processes. |
| Running commands on multiple processors | |
| Nproc | Returns number of MATLAB processes. |
| ID | Returns ID of calling MATLAB process. |
| Eval(pid,'command') | Evaluates a command on one or more MATLAB process. |
| Master/Slave Communication | |
| Put(pid,dataname) | Put data from the master process onto one or more remote processes. |
| Get(pid,dataname) | Get data from a remote process onto the master process. |
| SPMD Communication | |
| Send(pid,data) | Send data from one process to another. |
| Recv(pid) | Receive data sent from another process. |
| Barrier | Synchronize processes. |
| Bcast(pid,data) | Broadcasts data from processor pid to all processes. |
| Sum(data) | Adds data across all processors to form a global sum. |
| Collect(data) | Collects local data segments into a single global one. |

Table 1: Selected commands in the MultiMATLAB system

Commands for point-to-point communication implemented in MEX (including Send() and Recv())

**Ex.1005, Table 1 (annotated)**

Ex.1005, 9.

**545.** As discussed in limitation [27.4], MATLAB (*single-node kernel*) in a MATLAB process of processor ID executes the command Send (ID+1,a), generates a collection or package of information, and transmits the collection or package of information to the communication components (i.e., MEX routines, MultiMATLAB interface module, the communication layer) (*cluster node module*) of the same MATLAB process (*local … associated with the single-node kernel*), teaching *upon reception of the send command by a local cluster node module associated with the single-node kernel*.

**546.** In limitation [27.1], it was shown that the collection or package (*packet*) of information includes the variable "a" that MPI_SEND transmits (*forward*) to the communication layer of the next MATLAB process at the destination processor ID+1 as the data part (*a payload*) of its message and an identification of the destination processor ("ID+1"), teaching *the local cluster node module is configured to … forward a payload to the cluster node module specified in the packet*. The collection or package is sent to the communication layer of the next MATLAB process because, as discussed in limitation [27.0], each MATLAB process uses its own communication layer for communication with every other MATLAB process. As taught by Trefethen, the communications are "asynchronous." Ex.1006, 6.

**547.** Menon describes how MATLAB calls the Send() command and communicates with the *local cluster node module*, explaining that "Send(pid,data)" "[s]end[s] data from one process to another" and includes the "pid" value as the destination and the "data" value as the expression. Ex.1005, Table 1. As discussed in limitation [27.4], MATLAB provides that packet of information (the pid value and data value) to the MEX routine, which then decodes that packet by mapping the information to the corresponding MPI call (e.g., MPI_SEND). Ex.1005, 6. Menon describes that the MultiMATLAB interface module acts as an interpretation layer that maps MPI calls (e.g., MPI_SEND) to an indirection table for the MEX routines, explaining that:

> The MultiMATLAB interface module provides parallel MEX Routines with access to the underlying communication layer. In our implementations, MEX Routines are granted access to MPI routines. Moreover, the indirection imposed by the interface module is transparent to the programmer. There are macros, provided to all parallel MEX Routines, which map MPI calls directly to the appropriate table offset while maintaining the syntax of MPI.

Ex.1005, 6.

**548.** The *local cluster node module* accepting the package (*packet*) of information from the Send command (as sent by MATLAB (*the single-node*

*kernel*)) and mapping that data to the MPI_SEND call teaches *the local cluster node module is configured to decode the packet*.

**549.** MPIref describes how the MPI_SEND call uses the data (i.e., the data value in the send call) and the identification of the destination node (i.e., the pid value in the send call) and forwards the data to the destination node. Ex.1017, 15-19. MPIref discloses that "the **send** operation MPI_SEND" "specifies a **send buffer** in the sender memory from which the message data is taken" and that "[t]he send buffer specified by the MPI_SEND operation consists of **count** successive entries of the type indicated by **datatype**, starting with the entry at address **buf**." Ex.1017, 15. 16 (emphasis original). MPIref further discloses that, "[i]n addition to the data part" that "consists of a sequence of **count** values, each of the type indicated by **datatype,**"

> messages carry information that can be used to distinguish messages and selectively receive them. This information consists of a fixed number of fields, which we collectively call the **message envelope**. These fields are
>
> source
>
> destination
>
> tag

> communicator
>
> …
>
> The message destination is specified by the **dest** argument.

Ex.1017, 17, 18 (emphasis original). That is, MPIref teaches that MPI_SEND sends a message containing a data part (i.e., a message data) and a message envelope, where the data part includes the message data to be sent or transmitted and the message envelope includes an identification of the destination for the message data, as shown below.

Send buffer information

The syntax of the blocking send operation is given below.

Message envelope

MPI_SEND(buf, count, datatype, dest, tag, comm)

| | | |
|---|---|---|
| IN | buf | initial address of send buffer (choice) |
| IN | count | number of elements in send buffer (nonnegative integer) |
| IN | datatype | datatype of each send buffer element (handle) |
| IN | dest | rank of destination (integer) |
| IN | tag | message tag (integer) |
| IN | comm | communicator (handle) |

Ex.1017, 16 (annotated).

**550.** The data is sent to the communication layer of the destination node (*the cluster node module specified in the packet*) because, as discussed in limitation

[27.0], each MATLAB process uses its own communication layer for communication with every other MATLAB process.

**551.** Accordingly, Menon using a Send command from MATLAB (*a single-node kernel*) of a MATLAB process that sends information to the corresponding MPI call (MPI_Send) in the communication layer of the same MATLAB process (*local cluster node module*), and MPIref teaching the MPI_SEND call that takes in the information and then transmits the information to the destination node, as taught by Trefethen, the transmission being to the communication layer (*cluster node module specified in the packet*) of that destination node as taught by Menon, renders obvious *wherein, upon reception of the send command by a local cluster node module associated with the single-node kernel, the local cluster node module is configured to decode the packet and forward a payload to the cluster node module specified in the packet*.

**e. [32.4] *wherein each of the single-node kernels is configured to call a receipt command involving creating a packet specifying which message to test for completion and to then wait for a reply expression to evaluate;***

**552.** Menon, Trefethen and MPIref render obvious *wherein each of the single-node kernels is configured to call a receipt command involving creating a packet specifying which message to test for completion and to then wait for a reply expression to evaluate* because Menon teaches that MATLAB (*single-node kernel*) of a MATLAB process executes the Recv call to communicate a packet of

information (*a receipt command involving creating a packet*) to its communication components (MEX Routines, MultiMATLAB interface module, indirection table, and communication layer) to execute the corresponding MPI call (MPI_RECV), Trefethen teaches using the Recv call to wait for an expression to evaluate, and MPIref teaches the MPI_RECV call, which specifies a message to test for completion and wait for data from the specified node.

**553. First**, as discussed in limitation [4.0], Menon discloses that each node has communication components, i.e., MEX Routines, MultiMATLAB interface module, indirection table, and communication layer, that teach a *cluster node module* of that node, as shown in Figure 2 of Menon below.

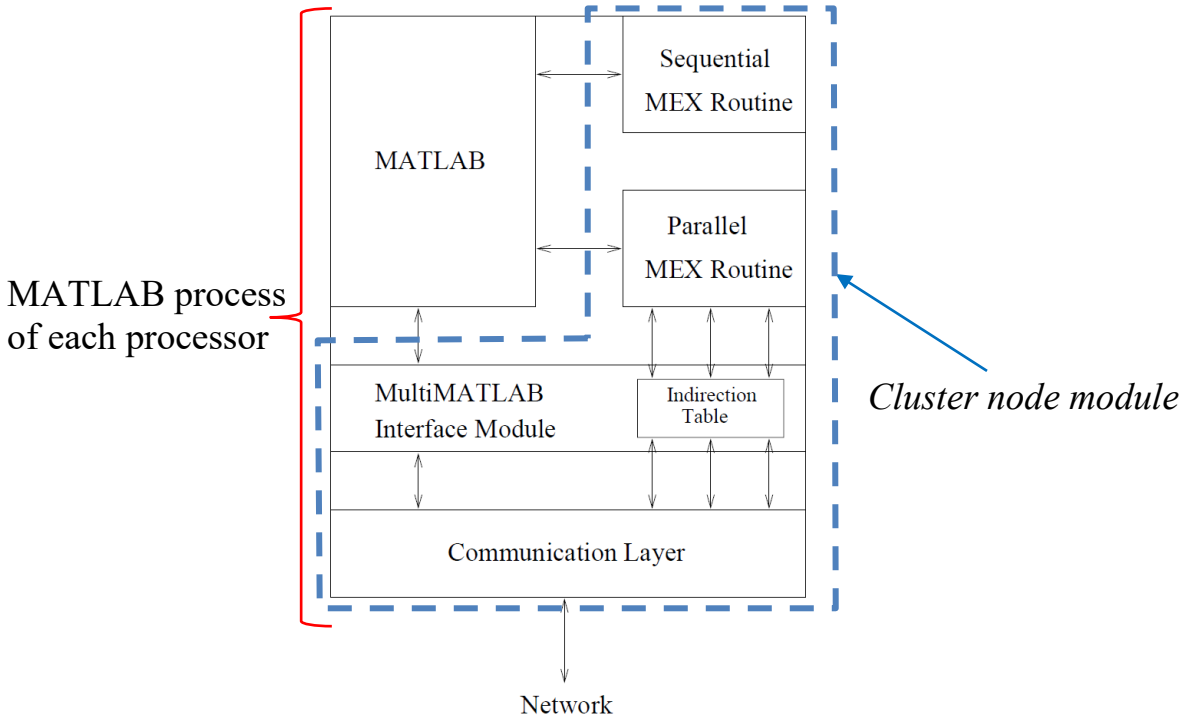



Figure 2: MATLAB Process Address Space

**Ex.1005, FIG. 2 (annotated)**

**554. Second**, as discussed in limitation [1.2], Menon teaches MEX routines having code that implement standard variants of MPI calls to enable SPMD point-to-point communication so that the MATLAB processes can communicate tasks and data directly with each other. Ex.1005, 9. The "MEX routines" provide "[a]ll parallel functionality in the MultiMATLAB system." Ex.1005, 3. For example, the MEX routines send the "commands over the underlying communication layer to the processes corresponding to the specified IDs." Ex.1005, 4. Menon describes using "MPI as [the] underlying communication substrate" in its implementations of the MultiMATLAB architecture, although "a specific communication standard is not

fundamental to the architecture." Ex.1005, 5. The choice of MPI is "due [to] its broad popularity, its suitability to high performance parallel computing, and the virtually universal availability of vendor-optimized versions on modern parallel computers." Ex.1005, 5. One implementation is "designed for the IBM SP2, a modern high performance distributed memory multiprocessor. This implementation is based upon MPI-F [], IBM's proprietary version of MPI specifically optimized for the SP2." Ex.1005, 5. The other implementation "is based upon MPICH [], a popular public domain version of MPI." Ex.1005, 5.

**555.** Table 1 of Menon below shows MultiMATLAB commands implemented as MEX Routines, including SPMD calls such as "Send(pid,data)" command that "send[s] data from one process to another" and the "Recv(pid)" command that "receive[s] data sent from another process":

| MultiMATLAB commands | |
|---|---|
| Starting and stopping MultiMATLAB (MPICH only) | |
| Start(n) | Initializes n remote MATLAB processes. |
| Quit | Closes remote MATLAB processes. |
| Running commands on multiple processors | |
| Nproc | Returns number of MATLAB processes. |
| ID | Returns ID of calling MATLAB process. |
| Eval(pid,'command') | Evaluates a command on one or more MATLAB process. |
| Master/Slave Communication | |
| Put(pid,dataname) | Put data from the master process onto one or more remote processes. |
| Get(pid,dataname) | Get data from a remote process onto the master process. |
| SPMD Communication | |
| Send(pid,data) | Send data from one process to another. |
| Recv(pid) | Receive data sent from another process. |
| Barrier | Synchronize processes. |
| Bcast(pid,data) | Broadcasts data from processor pid to all processes. |
| Sum(data) | Adds data across all processors to form a global sum. |
| Collect(data) | Collects local data segments into a single global one. |

Table 1: Selected commands in the MultiMATLAB system

Commands for point-to-point communication implemented in MEX (including Send() and Recv())

**Ex.1005, Table 1 (annotated)**

**556. Third**, Menon describes how MultiMATLAB interprets the Recv() command, explaining that "Recv(pid)" "[r]eceive[s] data sent from another process" and includes the "pid" value as the source of the expected data. Ex.1005, Table 1. MATLAB provides that packet of information (the pid value) to the MEX routine, which then decodes that packet by mapping the packet's information to the corresponding MPI call (e.g., MPI_RECV). Ex.1005, 6. Menon describes that the MultiMATLAB interface module acts as an interpretation layer that maps MPI calls (e.g., MPI_RECV) to an indirection table for the MEX routines, explaining that:

> The MultiMATLAB interface module provides parallel MEX Routines with access to the underlying communication layer. In our implementations, MEX Routines are granted access to MPI routines. Moreover, the indirection imposed by the interface module is transparent to the programmer. There are macros, provided to all parallel MEX Routines, which map MPI calls directly to the appropriate table offset while maintaining the syntax of MPI.

Ex.1005, 6.

**557. Fourth**, also as discussed in limitation [26.2], Trefethen provides further details of the SPMD communication commands. Ex.1006, 5-7. Trefethen describes a MATLAB process on a processor of the MultiMATLAB architecture using the SPMD point-to-point commands to communicate mathematical results to another MATLAB process on another processor. Ex.1006, 3-7. Specifically, Trefethen provides the following example code in which a MATLAB process communicates a mathematical result to a next MATLAB process, which in turn communicates the next mathematical result to the next MATLAB process, and so on, where the communications of the mathematical results from one MATLAB process to another use calls such as Send and Recv:

```
if ID==0                  % first process: send
   a = 1
   Send(ID+1,a)
elseif ID == Nproc-1      % last process: receive and double
   a = 2*Recv
else                      % middle processes: receive, double, and send
   a = 2*Recv
   Send(ID+1,a)
end;
```

Process ID issues RECV to wait for a message from process ID-1, then doubles the received value

Process 0 creates the variable a with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of a, and sends it along to process 2; and so on. If there are six processors the command `Eval( 'cycle' )` produces the output

```
a = 1
a = 2
a = 4
a = 8
a = 16
a = 32
```

SPMD point to point communication among processors of the MultiMATLAB architecture

Ex.1006, 6 (annotated). During the communication, a first MATLAB process (e.g., MATLAB process 0 at processor 0) creates a variable and sends that value to a second MATLAB process (e.g., MATLAB process 1 at processor 1), where MATLAB process 1 issues the Recv command to wait for the message and after receiving the message, doubles the value, teaching *wait for a reply expression to evaluate*.

**558. Fifth**, MPIref describes how the MPI_RECV call uses the information (i.e., the pid value in the recv call) to receive the incoming data. Ex.1017, 15-21. MPIref discloses the details of what the parallel MEX Routines would be mapped to for the MPI calls; particularly, MPIref describes the "MPI_RECV" call, which needs or expects the receive buffer information (the data transmitted) and the source (the identification of the source node). Ex.1017, 15-21. MPIref teaches that a process receives a message that is transmitted by MPI_SEND call "with the **receive**

operation MPI_RECV." Ex.1017, 16 (emphasis original). "The message to be received is selected according to the value of its envelope, and the message data is stored into the **receive buffer**." Ex.1017, 16 (emphasis original). "The receive buffer consists of the storage containing **count** consecutive elements of the type specified by **datatype**, starting at address **buf**." Ex.1017, 16 (emphasis original).

**559.** With reference to the message envelope, MPIref explains that "messages carry information that can be used to distinguish messages and selectively receive them. This information consists of a fixed number of fields, which we collectively call the **message envelope**. These fields are

> source
>
> destination
>
> tag
>
> communicator
>
> …
>
> The message destination is specified by the **dest** argument."

Ex.1017, 18 (emphasis original). That is, MPIref teaches that MPI_RECV is used to receive messages associated with a message envelope having fields for the source and destination, where the message data is stored into the **receive buffer** that

consists of a storage containing **count** consecutive elements of the type specified by **datatype**, starting at address **buf**, as shown below:

Recv buffer information  Message envelope

MPI_RECV (buf, count, datatype, source, tag, comm, status)

| | | |
|---|---|---|
| OUT | buf | initial address of receive buffer (choice) |
| IN | count | number of elements in receive buffer (integer) |
| IN | datatype | datatype of each receive buffer element (handle) |
| IN | source | rank of source (integer) |
| IN | tag | message tag (integer) |
| IN | comm | communicator (handle) |
| OUT | status | status object (Status) |

Ex.1017, 19 (annotated).

**560.** A POSITA would then recognize that execution of the Recv command "Recv" (*receipt command*) by MATLAB in a MATLAB process at processor ID+1 generates a collection or package (*a packet*) of information that is needed by the corresponding MPI call (MPI_RECV) for receiving the message transmitted by the MPI_SEND call from processor ID, including (i) the receive buffer for storing the variable "a" that is transmitted by the processor ID-1 using the MPI call MPI_SEND, and (ii) an identification of the source processor ID-1 of the variable "a".

**561.** **Sixth**, Trefethen teaches using a message specifier to search for matching expressions that have been received by using an "argument" to "specify a

message tag so as to ensure that sends and receives are properly matched and to aid in error checking." Ex.1006, 6. This "argument" can be "added in both Send and Recv." Ex.1006, 6. Trefethen also describes a "Probe" command that returns "1 (true) if a message has arrived from the indicated source, otherwise (0) false." Ex.1006, 6. Correspondingly, MPIref discloses MPI_TEST call that uses a request handle (message specifier) to find the corresponding message to determine if the operation identified by the request (such as a send or receive) is complete. Ex.1017, 40. Specifically, MPIref discloses using MPI_TEST call that determines if an operation is complete:

MPI_TEST(request, flag, status)

| | | |
|---|---|---|
| INOUT | request | communication request (handle) |
| OUT | flag | true if operation completed (logical) |
| OUT | status | status object (Status) |

```
int MPI_Test(MPI_Request *request, int *flag, MPI_Status *status)
```

```
MPI_TEST(REQUEST, FLAG, STATUS, IERROR)
    LOGICAL FLAG
    INTEGER REQUEST, STATUS(MPI_STATUS_SIZE), IERROR
```

A call to MPI_TEST returns flag = true if the operation identified by request is complete. In such a case, the status object is set to contain information on the completed operation; if the communication object was created by a nonblocking send or receive, then it is deallocated and the request handle is set to MPI_REQUEST_NULL. The call returns flag = false, otherwise. In this case, the value of the status object is undefined. MPI_TEST is a local operation.

Ex.1017, 40. The handle **request** is used "to identify communication operations," i.e., the handle **request** is a message specifier, and MPI_TEST call uses **request** to determine "if the operation identified by **request** is complete." Ex.1017, 36, 40.

**562.** Accordingly, Menon teaches that MATLAB (*single-node kernel*) of a MATLAB process executes the Recv call to communicate a packet of information (*a receipt command involving creating a packet*) to its communication components (MEX Routines, MultiMATLAB interface module, indirection table, and communication layer) to execute the corresponding MPI call, Trefethen teaches using the Recv call to wait for an expression to evaluate, and MPIref teaches the MPI_RECV call, which specifies a message to test for completion and wait for data from the specified node, renders obvious *wherein each of the single-node kernels is configured to call a receipt command involving creating a packet specifying which message to test for completion and to then wait for a reply expression to evaluate*.

**f. [32.5] *wherein, upon reception of the receipt command by the local cluster node module associated with the single-node kernel, the local cluster node module is configured to decode the packet and use a message specifier to search for any matching expressions listed as completed in a received message queue;***

**563.** Menon, Trefethen and MPIref render obvious *wherein, upon reception of the receipt command by the local cluster node module associated with the single-node kernel, the local cluster node module is configured to decode the packet and use a message specifier to search for any matching expressions listed as completed*

*in a received message queue* because Menon teaches that MATLAB (*single-node kernel*) in a MATLAB process executes the command Recv to communicate a collection or package (*packet*) of information to its communication components (MEX Routines, MultiMATLAB interface module, indirection table, and communication layer) (*local cluster node module*) to execute the corresponding MPI call, and MPIref teaches the MPI_RECV call, which specifies a message to test for completion and then checking for a matching value in the received message queue.

**564. First**, as discussed in limitation [28.4], MATLAB (*single-node kernel*) in a MATLAB process of processor ID executes the command Recv, generates a collection or package (*packet*) of information, and transmits the collection or package of information to the communication components (MEX Routines, MultiMATLAB interface module, indirection table, and communication layer) (*cluster node module*) of the same (*local*) MATLAB process, where the collection or package (*packet*) of information includes the receive buffer for storing the variable "a" and a source processor ID-1 of the variable "a," teaching *upon reception of the receipt command by the local cluster node module associated with the single-node kernel*.

**565. Second**, as discussed in limitation [32.4], MPIref discloses the details of what the parallel MEX Routines would be mapped to for the MPI calls;

particularly, MPIref describes the "MPI_RECV" call, which needs or expects the receive buffer information (the data transmitted) and the source (the identification of the source node). Ex.1017, 15-21. As such, *the local cluster node module* accepting the package (*packet*) of information from the Recv call (as sent from the *single-node kernel*) and mapping that data to the MPI_RECV call teaches *configured to decode the packet*.

**566.** MPIref provides example code showing the MPI_RECV call in operation. Ex.1017, 163-164. In a snippet of this code, the MPI_RECV call is executed and the data obtained from the call is stored in variables named "client_tag," "client_source," and "client_rank_in_new_world":

```
int Do_server(server_comm)
MPI_Comm server_comm;
{
    void init_queue();
    int en_queue(), de_queue(); /* keep triplets of integers
                                   for later matching (fns not shown) */

    MPI_Comm comm;
    MPI_Status status;
    int client_tag, client_source;
    int client_rank_in_new_world, pairs_rank_in_new_world;
    int buffer[10], count = 1;

    void *queue;
    init_queue(&queue);

    for (;;)
    {
        MPI_Recv(buffer, count, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG,
                 server_comm, &status); /* accept from any client */

        /* determine client: */
        client_tag = status.MPI_TAG;
        client_source = status.MPI_SOURCE;
        client_rank_in_new_world = buffer[0];
```

MPI_RECV call in the communication layer (*cluster node module*)

Source

Contents of collection or package of information from MPI_RECV (*packet* contents)

Receive buffer (*location for receiving expression*)

Ex.1017, 163.

**567.** Subsequently, the code checks for a match of an expression in the message receiving queue using the information received by the MPI_RECV call (the information in the variables named "client_tag," "client_source," and

"client_rank_in_new_world," teaching *use a message specifier to search for any matching expressions listed as completed in a received message queue*:

```
if (de_queue(queue, client_tag, &pairs_rank_in_new_world,
                &pairs_rank_in_server))
{
    /* matched pair with same tag, tell them
       about each other! */
    buffer[0] = pairs_rank_in_new_world;
    MPI_Send(buffer, 1, MPI_INT, client_src, client_tag,
                                            server_comm);
```

Ex.1017, 163.

**568.** Accordingly, Menon teaches that MATLAB (*single-node kernel*) in a MATLAB process executes the command Recv to communicate a collection or package (*packet*) of information to its communication components (MEX Routines, MultiMATLAB interface module, indirection table, and communication layer) (*local cluster node module*) to execute the corresponding MPI call, and MPIref teaches the MPI_RECV call, which specifies a message to test for completion and then checking for a matching value in the received message queue, rendering obvious *wherein, upon reception of the receipt command by the local cluster node module associated with the single-node kernel, the local cluster node module is configured to decode the packet and use a message specifier to search for any matching expressions listed as completed in a received message queue*.

**g. [32.6] *wherein, upon determining that such a completed expression is found, the local cluster node module is***

***configured to send the completed expression to a local single-node kernel associated with the cluster node module in response to the receipt command, and***

**569.** Menon, Trefethen and MPIref render obvious *wherein, upon determining that such a completed expression is found, the local cluster node module is configured to send the completed expression to a local single-node kernel associated with the cluster node module in response to the receipt command* because Menon teaches that MATLAB (*single-node kernel*) of a MATLAB process executes the Recv command to receive data for further processing, MPIref teaches the MPI_RECV call, and Trefethen teaches that the Recv command causes the communication components (MEX routines, a MultiMATLAB interface module, an indirection table, and a communication layer) (*cluster node module*) of the same (*local*) MATLAB process to provide the information to MATLAB (*single-node kernel*).

**570.** **First**, it was shown in limitation [32.5] that Menon, Trefethen and MPIref render obvious using a message specifier to search for any matching expressions listed as completed in a received message queue, teaching *upon determining that such a completed expression is found*.

**571.** **Second**, as discussed in limitation [26.2], Menon describes example point-to-point commands that the MATLAB processes in the MultiMATLAB architecture use to communicate with each other, such as the "Send(pid,data)"

command that "send[s] data from one process to another" identified by the value "pid," the "Recv(pid)" command that "receive[s] data sent from another process" identified by the value "pid," and the "Bcast(pid,data)" command that "broadcasts data from processor pid to all processes," shown in Table 1 of Menon below. For example, each of MATLAB (*single-node kernel*) executes the Recv command using the "pid" value to identify the source of the data to be received at the MATLAB process:

| MultiMATLAB commands | |
|---|---|
| Starting and stopping MultiMATLAB (MPICH only) | |
| Start(n) | Initializes n remote MATLAB processes. |
| Quit | Closes remote MATLAB processes. |
| Running commands on multiple processors | |
| Nproc | Returns number of MATLAB processes. |
| ID | Returns ID of calling MATLAB process. |
| Eval(pid,'command') | Evaluates a command on one or more MATLAB process. |
| Master/Slave Communication | |
| Put(pid,dataname) | Put data from the master process onto one or more remote processes. |
| Get(pid,dataname) | Get data from a remote process onto the master process. |
| SPMD Communication | |
| Send(pid,data) | Send data from one process to another. |
| Recv(pid) | Receive data sent from another process. |
| Barrier | Synchronize processes. |
| Bcast(pid,data) | Broadcasts data from processor pid to all processes. |
| Sum(data) | Adds data across all processors to form a global sum. |
| Collect(data) | Collects local data segments into a single global one. |

Table 1: Selected commands in the MultiMATLAB system

Commands for point to point communication
(including Send, Recv, and Bcast)

**Ex.1005, Table 1 (annotated).**

Ex.1005, 9.

**572. Third**, also as discussed in limitation [26.2], Trefethen provides further details of the SPMD communication commands. Ex.1006, 5-7. Trefethen describes a MATLAB process on a processor of the MultiMATLAB architecture using the SPMD point-to-point commands to communicate mathematical results to another MATLAB process on another processor. Ex.1006, 3-7. Specifically, Trefethen provides the following example code in which a MATLAB process communicates a mathematical result to a next MATLAB process, which in turn communicates the next mathematical result to the next MATLAB process, and so on, where the communications of the mathematical results from one MATLAB process to another use calls such as Send and Recv:

```
if ID==0                 % first process: send
   a = 1
   Send(ID+1,a)
elseif ID == Nproc-1     % last process: receive and double
   a = 2*Recv
else                     % m
   a = 2*Recv
   Send(ID+1,a)
end;
```

Process ID issues RECV to wait for a message from process ID-1, then doubles the received value

Process 0 creates the variable a with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of a, and sends it along to process 2; and so on. If there are six processors the command `Eval( 'cycle' )` produces the output

```
a = 1
a = 2
a = 4
a = 8
a = 16
a = 32
```

SPMD point to point communication among processors of the MultiMATLAB architecture

Ex.1006, 6 (annotated). During the communication, a first MATLAB process (e.g., MATLAB process 0 at processor 0) creates a variable and sends that value to a second MATLAB process (e.g., MATLAB process 1 at processor 1), where

MATLAB process 1 issues the RECV command to wait for the message and after receiving the message, doubles the value.

**573. Fourth**, MPIref discloses the details of what the parallel MEX Routines would be mapped to for the MPI calls; particularly, MPIref describes the "MPI_RECV" call, which needs or expects the receive buffer information (the data transmitted) and the source (the identification of the source node). Ex.1017, 15-21. MPIref teaches that a process receives a message that is transmitted by MPI_SEND call "with the **receive** operation MPI_RECV." Ex.1017, 16 (emphasis original). "The message to be received is selected according to the value of its envelope, and the message data is stored into the **receive buffer**." Ex.1017, 16 (emphasis original). "The receive buffer consists of the storage containing **count** consecutive elements of the type specified by **datatype**, starting at address **buf**." Ex.1017, 16 (emphasis original). The message envelope has fields for the source and the destination of the message. Ex.1017, 18.

**574.** That is, MPIref teaches that MPI_RECV is used to receive messages associated with a message envelope having fields for the source and destination, where the message data is stored into the **receive buffer** that consists of a storage containing **count** consecutive elements of the type specified by **datatype**, starting at address **buf**, as shown below:

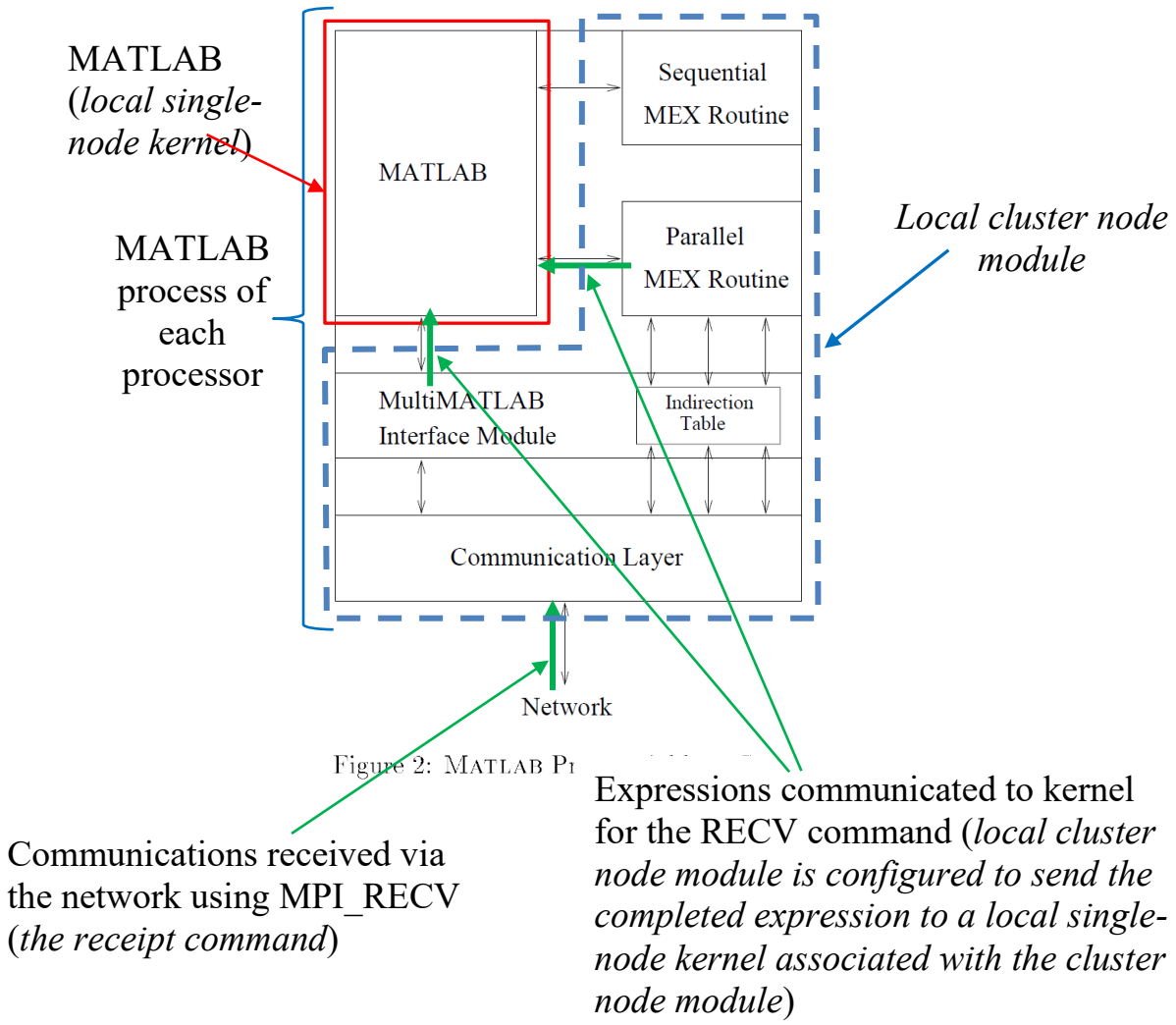
Recv buffer information  Message envelope

MPI_RECV (buf, count, datatype, source, tag, comm, status)

| | | |
|---|---|---|
| OUT | buf | initial address of receive buffer (choice) |
| IN | count | number of elements in receive buffer (integer) |
| IN | datatype | datatype of each receive buffer element (handle) |
| IN | source | rank of source (integer) |
| IN | tag | message tag (integer) |
| IN | comm | communicator (handle) |
| OUT | status | status object (Status) |

Ex.1017, 19 (annotated).

**575. Fifth**, as discussed in limitation [6.0], each MATLAB process has a set of communication components (MEX routines, a MultiMATLAB interface module, an indirection table, and a communication layer) that, when executed by the processor of the MATLAB process, communicates with MATLAB (*local single-node kernel*) of the MATLAB process and the communication components of the other nodes, as shown below in FIG. 2 of Menon.




**Ex.1005, FIG. 2 (annotated)**

**576.** Accordingly, Menon teaches that MATLAB (*single-node kernel*) of a MATLAB process execute the Recv command to receive data for further processing, MPIref teaches the MPI_RECV call, and Trefethen teaches that the Recv command causes the communication components (MEX routines, a MultiMATLAB interface module, an indirection table, and a communication layer) (*cluster node module*) of the same (*local*) MATLAB process to provide the

information to MATLAB (*single-node kernel*), rendering obvious *wherein, upon reception of the receipt command by the local cluster node module associated with the single-node kernel, the local cluster node module is configured to decode the packet and use a message specifier to search for any matching expressions listed as completed in a received message queue*.

**h. [32.7] *wherein each of the single-node kernels is configured to receive the completed expression and to update variables used by the single-node kernel during evaluation of expressions.***

**577.** Menon and Trefethen render obvious *wherein each of the single-node kernels is configured to receive the completed expression and to update variables used by the single-node kernel during evaluation of expressions* because Menon teaches that the Recv command is used by MATLAB (*single-node kernel*) to perform evaluation of expressions and Trefethen teaches that the Recv command can be used to update variables used by MATLAB (*single-node kernel*) to evaluate expressions.

**578. First**, as discussed in limitation [26.2], Menon describes example point-to-point commands that the MATLAB processes in the MultiMATLAB architecture use to communicate with each other, such as the "Send(pid,data)" command that "send[s] data from one process to another" identified by the value "pid," the "Recv(pid)" command that "receive[s] data sent from another process" identified by the value "pid," and the "Bcast(pid,data)" command that "broadcasts

data from processor pid to all processes," shown in Table 1 of Menon below. For example, each of MATLAB (*single-node kernel*) executes the Recv command using the "pid" value to identify the source of the data to be received at the MATLAB process:

| MultiMATLAB commands | |
|---|---|
| Starting and stopping MultiMATLAB (MPICH only) | |
| Start(n) | Initializes n remote MATLAB processes. |
| Quit | Closes remote MATLAB processes. |
| Running commands on multiple processors | |
| Nproc | Returns number of MATLAB processes. |
| ID | Returns ID of calling MATLAB process. |
| Eval(pid,'command') | Evaluates a command on one or more MATLAB process. |
| Master/Slave Communication | |
| Put(pid,dataname) | Put data from the master process onto one or more remote processes. |
| Get(pid,dataname) | Get data from a remote process onto the master process. |
| SPMD Communication | |
| Send(pid,data) | Send data from one process to another. |
| Recv(pid) | Receive data sent from another process. |
| Barrier | Synchronize processes. |
| Bcast(pid,data) | Broadcasts data from processor pid to all processes. |
| Sum(data) | Adds data across all processors to form a global sum. |
| Collect(data) | Collects local data segments into a single global one. |

Table 1: Selected commands in the MultiMATLAB system

Commands for point to point communication

**Ex.1005, Table 1 (annotated)**

**579. Second**, also as discussed in limitation [26.2], Trefethen provides further details of the SPMD communication commands. Ex.1006, 5-7. Trefethen describes a MATLAB process on a processor of the MultiMATLAB architecture using the SPMD point-to-point commands to communicate mathematical results to another MATLAB process on another processor. Ex.1006, 3-7. Specifically, Trefethen provides the following example code in which a MATLAB process

communicates a mathematical result to a next MATLAB process, which in turn communicates the next mathematical result to the next MATLAB process, and so on, where the communications of the mathematical results from one MATLAB process to another use calls such as Send and Recv:

```
if ID==0                % first process: send
   a = 1
   Send(ID+1,a)
elseif ID == Nproc-1    % last process: receive and double
   a = 2*Recv
else                    % mi
   a = 2*Recv
   Send(ID+1,a)
end;
```

Process ID issues RECV to wait for a message from process ID-1, then doubles the received value

Process 0 creates the variable a with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of a, and sends it along to process 2; and so on. If there are six processors the command `Eval( 'cycle' )` produces the output

```
a = 1
a = 2
a = 4
a = 8
a = 16
a = 32
```

SPMD point to point communication among processors of the MultiMATLAB architecture

Ex.1006, 6 (annotated). During the communication, a first MATLAB process (e.g., MATLAB process 0 at processor 0) creates a variable and sends that value to a second MATLAB process (e.g., MATLAB process 1 at processor 1), where MATLAB process 1 issues the RECV command to wait for the message and after receiving the message, doubles the value.

**580.** Accordingly, Menon teaches that the Recv command is used by MATLAB (*single-node kernel*) to perform evaluation of expressions and Trefethen teaches that the Recv command can be used to update variables used by the MATLAB (*single-node kernel*) to evaluate expressions, rendering obvious *wherein*

*each of the single-node kernels is configured to receive the completed expression and to update variables used by the single-node kernel during evaluation of expressions*.

**40. Claim 33**

**a. [33.0] *The computer cluster of claim 1, wherein the plurality of nodes are configured to permit exchange of information between nodes during the course of parallel computation.***

**581.** Limitation [33.0] is disclosed or rendered obvious for the same reasons presented above for limitation [31.0].

**41. Claim 34**

**a. [34.0] *The computer cluster of claim 1, wherein each of the plurality of nodes comprises instructions executable by the hardware processor and configured to implement asynchronous behavior, wherein the instructions comprise:***

**582.** Menon and Trefethen render obvious *wherein each of the plurality of nodes comprises instructions executable by the hardware processor and configured to implement asynchronous behavior* because Menon teaches that the nodes include software instructions to perform message passing of mathematical results to the other processors, and Trefethen teaches that a MATLAB process (i) uses an asynchronous send call to send a payload that (ii) is received using an asynchronous receive call, and (iii) uses an argument to specify a message to ensure that a payload is properly sent and received.

**583. First**, as discussed in limitation [1.1.3], a POSITA would recognize that the MATLAB process and MATLAB of a node are in a computer-readable medium of the node, and that MATLAB is accessed from the computer-readable medium, i.e., a *hardware processor* accesses MATLAB from the computer-readable medium.

**584. Second**, as discussed in limitation [1.2], Menon teaches MEX routines having code that implement standard variants of MPI calls to enable SPMD point-to-point communication so that the MATLAB processes can communicate tasks and data directly with each other. The "MEX routines" provide "[a]ll parallel functionality in the MultiMATLAB system." Ex.1005, 3. For example, the MEX routines send the "commands over the underlying communication layer to the processes corresponding to the specified IDs." Ex.1005, 4. Table 1 of Menon below shows MultiMATLAB commands implemented as MEX routines, including SPMD calls such as "Send(pid,data)" command that "send[s] data from one process to another," the "Recv(pid)" command that "receive[s] data sent from another process," and the "Bcast(pid,data)" command that "broadcasts data from processor pid to all processes":

| MultiMATLAB commands | |
|---|---|
| Starting and stopping MultiMATLAB (MPICH only) | |
| Start(n) | Initializes n remote MATLAB processes. |
| Quit | Closes remote MATLAB processes. |
| Running commands on multiple processors | |
| Nproc | Returns number of MATLAB processes. |
| ID | Returns ID of calling MATLAB process. |
| Eval(pid,'command') | Evaluates a command on one or more MATLAB process. |
| Master/Slave Communication | |
| Put(pid,dataname) | Put data from the master process onto one or more remote processes. |
| Get(pid,dataname) | Get data from a remote process onto the master process. |
| SPMD Communication | |
| Send(pid,data) | Send data from one process to another. |
| Recv(pid) | Receive data sent from another process. |
| Barrier | Synchronize processes. |
| Bcast(pid,data) | Broadcasts data from processor pid to all processes. |
| Sum(data) | Adds data across all processors to form a global sum. |
| Collect(data) | Collects local data segments into a single global one. |

Table 1: Selected commands in the MultiMATLAB system

Commands implemented in MEX Routines

**Ex.1005, Table 1 (annotated)**

**585. Third**, as discussed in limitation [26.2], Trefethen provides further details of the SPMD communication commands. Ex.1006, 5-7. Specifically, Trefethen teaches that the Send() and Recv() calls are "asynchronous." Ex.1006, 6. Trefethen describes a MATLAB process on a processor of the MultiMATLAB architecture using the SPMD point-to-point commands to communicate mathematical results to another MATLAB process on another processor asynchronously. Ex.1006, 3-7. Trefethen provides the following example code in which a MATLAB process communicates a mathematical result to a next MATLAB process, which in turn communicates the next mathematical result to the

next MATLAB process, and so on, where the communications of the mathematical results from one MATLAB process to another use *asynchronous calls*:

SPMD programs can be built upon `Send` and `Recv` commands. Typically the program contains `if` and `else` commands that specify different actions for different processes. For example, suppose the m-file `cycle.m` consists of the following program:

```
if ID==0                    % first process: send
   a = 1
   Send(ID+1,a)
elseif ID == Nproc-1        % last process: receive and double
   a = 2*Recv
else                        % middle processes: receive, double, and send
   a = 2*Recv
   Send(ID+1,a)
end;
```

Process 0 creates the variable `a` with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of `a`, and sends it along to process 2; and so on. If there are six processors the command `Eval( 'cycle' )` produces the output

```
a = 1
a = 2
a = 4
a = 8
a = 16
a = 32
```

*Asynchronous calls*

The processes run asynchronously, but since each `Send` command is only executed after the corresponding `Recv` has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning.

Ex.1006, 3-6.

**586. Fourth**, as discussed in limitation [10.0], Menon discloses that the MultiMATLAB interface module of each MATLAB process acts as an interpretation layer that maps MPI calls to the indirection table so tasks and data can be provided to MATLAB for computation. Menon describes the indirection table as containing "pointers to all functions and data in the underlying communication layer that needed to be exposed to parallel MEX Routines," explaining:

> The MultiMATLAB interface module provides MEX Routines access to the underlying communication layer via an indirection table. Upon initialization, the [MultiMATLAB] interface module builds a table of pointers to all functions and data in the underlying communication layer that need to be exposed to parallel MEX Routines. When a parallel MEX Routine is first loaded and executed, it accesses the MultiMATLAB interface module through a function call to MATLAB and is given the location of this table. … Once the table is obtained, the MEX Routine has the capability to access any function provided by the underlying communication layer through its corresponding offset in the table.

Ex.1005, 5.

**587.** A POSITA would recognize that the MATLAB processes on the nodes includes instructions for interpreting the Send command "Send (ID+1,a)" to generate a collection of information that is used by the MEX routine to map to a corresponding MPI call (MPI_SEND). In this example, the MEX routine for the asynchronous Send command causes the variable "a" (*payload*) to be transmitted to process ID+1 (*asynchronously send to another node*).

**588.** Furthermore, a POSITA would recognize that the MATLAB processes on the nodes includes instructions for interpreting the Recv command in a MATLAB process at processor ID+1 to receive a message. In this example, the MEX Routine for the asynchronous Recv command receives the variable "a" (*payload*) transmitted from another processor.

**589.** **Fifth**, Trefethen teaches using a message specifier to search for matching expressions that have been received by using an "argument" to "specify a message tag so as to ensure that sends and receives are properly matched and to aid in error checking." Ex.1006, 6. This "argument" can be "added in both Send and Recv." Ex.1006, 6. Trefethen also describes a "Probe" command that returns "1 (true) if a message has arrived from the indicated source, otherwise (0) false." Ex.1006, 6.

**590.** Accordingly, Menon's teaching that the nodes include software instructions to perform message passing of mathematical results to the other processors, in view of Trefethen's teaching that a MATLAB process (i) uses an asynchronous send call to send a payload that (ii) is received using an asynchronous receive call, and (iii) uses an argument to specify a message to ensure that a payload is properly sent and received, renders obvious *wherein each of the plurality of nodes comprises instructions executable by the hardware processor and configured to implement asynchronous behavior*.

**b. [34.1]** ***a first instruction to asynchronously send a payload to another node;***

**591.** Limitation [34.1] is disclosed or rendered obvious for the same reasons presented above for limitations [27.0]-[27.5].

**c. [34.2]** ***a second instruction to asynchronously receive a payload from another node; and***

**592.** Limitation [34.2] is disclosed or rendered obvious for the same reasons presented above for limitations [28.0]-[28.5].

**d. [34.3]** ***a third instruction to search for a payload matching a message specifier.***

**593.** Limitation [34.3] is disclosed or rendered obvious for the same reasons presented above for limitations [32.0]-[32.7].

**42. Claim 37**

**a. [37.0]** ***The computer cluster node of claim 35, wherein the computer cluster node is configured to permit exchange of information with other computer cluster nodes during the course of parallel computation.***

**594.** Limitation [37.0] is disclosed or rendered obvious for the same reasons presented above for limitations [31.0].

**43. Claim 38**

**a. [38.0]** ***The computer cluster node of claim 35, wherein the computer cluster node comprises instructions executable by the hardware processor and configured to***

***implement asynchronous behavior, wherein the instructions comprise:***

**595.** Limitation [38.0] is disclosed or rendered obvious for the same reasons presented above for limitation [34.0].

**b. [38.1]** ***a first instruction to asynchronously send a payload to another node;***

**596.** Limitation [38.1] is disclosed or rendered obvious for the same reasons presented above for limitations [34.1].

**c. [38.2]** ***a second instruction to asynchronously receive a payload from another node; and***

**597.** Limitation [38.2] is disclosed or rendered obvious for the same reasons presented above for limitations [34.2].

**d. [38.3]** ***a third instruction to search for a payload matching a message specifier.***

**598.** Limitation [38.3] is disclosed or rendered obvious for the same reasons presented above for limitations [34.3].

## X. DECLARATION

**599.** I declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true, and that these statements were made with knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under section 1001 of Title 18 of the United States Code.

Dated: March 28, 2025

Chandrajit L. Bajaj, Ph.D.