

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

INTEL CORPORATION,
ADVANCED MICRO DEVICES, INC.,
Petitioners

IPR2025-00916
U.S. Patent No. 11,811,582

**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.	RELEVANT LEGAL STANDARDS.....	14
IV.	FOCUSED ANALYSIS.....	16
A.	Level Of Ordinary Skill in the Art	16
B.	Background	17
C.	The '582 Patent	33
1.	Summary of the '582 Patent	33
2.	Prosecution History.....	38
D.	Claim Construction	38
1.	“peer-to-peer architecture”.....	39
E.	Key Teachings of the Prior Art	39
1.	Kernel.....	39
2.	Cluster node module	41
3.	User commands received by the first cluster node module without first passing through the first kernel.....	45
4.	Peer-to-peer mathematical computation	48
F.	Reasons to Combine.....	50
1.	Combining Trefethen with Menon	50
2.	Combining RS6000 with Menon	55
3.	Combining MPIref with Menon	58
4.	Combining POEref with Menon	61
V.	DETAILED ANALYSIS	63
A.	PRIOR ART	64
1.	Menon	65

2.	Trefethen	73
3.	RS6000	77
4.	POEref	79
5.	MPIref	82
B. Ground 1: Claims 1-3 and 5-19 are obvious over Menon in view of		
Trefethen, RS6000, and MPIref		85
1.	Combining Trefethen with Menon	86
2.	Combining RS6000 with Menon	91
3.	Combining MPIref with Menon	94
4.	Claim 1	96
5.	Claim 2	162
6.	Claim 3	164
7.	Claim 5	172
8.	Claim 6	180
9.	Claim 7	188
10.	Claim 8	190
11.	Claim 9	195
12.	Claim 10	197
13.	Claim 11	205
14.	Claim 12	207
15.	Claim 13	210
16.	Claim 14	211
17.	Claim 15	214
18.	Claim 16	218
19.	Claim 17	228
20.	Claim 18	229
21.	Claim 19	231

C. Ground 2: Claims 4 and 20 are obvious over Menon in view of Trefethen, RS6000, and MPIref further in view of POEref	239
1. Combining POEref with Menon	239
2. Claim 4.....	242
3. Claim 20.....	247
VI. DECLARATION	253

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. 11,811,582 (“the ’582 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 20 (the “Challenged Claims”) of the ’582 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 ’582 patent, Ex.1001;
- (2) the prosecution history of the ’582 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) SC ‘97 Conference Materials, Ex.1026;
- (20) Declaration of Gordon MacPherson, IEEE, Ex.1027;
- (21) A copy of Menon with list of publications citing Menon, Ex.1028;
- (22) Affidavit of Mina Ching, Internet Archive, Ex.1029;
- (23) “Visual Supercomputing to be Demonstrated at SC97,” HPC wire, 1997, Ex.1030;
- (24) SC1997 flyer, 1997, Ex.1031;
- (25) IEEE Xplore’s “About IEEE Xplore” page, Ex.1032;
- (26) “As Cornell Theory Center winds up Microsoft pact, it seeks faculty advice on future, direction,” Cornell Chronicle, 2005; Ex.1033;
- (27) ACM’s description of “ACM Digital Library,” Ex.1034;
- (28) A copy of Trefethen with list of publications citing Trefethen, Ex.1035;
- (29) “A Beginner’s Guide to the IBM SP,” Stephanie Wolf et al., University of Tennessee, 2000, Ex.1036;
- (30) List of publications citing MPIref, Ex.1037;
- (31) IBM website’s documentation library for RS6000, Ex.1038
- (32) IBM website’s documentation library for POEref, Ex.1039;
- (33) ACS Preliminary Response, IPR2021-00019, Ex.1040;

- (34) Affidavit of Tanya Zeif, Internet Archive, Ex.1048; and
- (35) MPIref at ANL, Ex.1049.

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 keynote talks on these topics. I am a Fellow of the American Association for the Advancement of Science (AAAS), 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 (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. RELEVANT LEGAL STANDARDS

18. I am not an attorney. In preparing and expressing my opinions and considering the subject matter of the '582 patent, I am relying on certain legal principles that counsel has explained to me.

19. I understand that prior art to the '582 patent includes patents and printed publications in the relevant art that predate the priority date of the '582 patent. For purposes of this Declaration, I am applying the earliest claimed priority date of June 13, 2006, as the priority date of the '582 patent.

20. 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.

21. 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.

22. 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.

23. 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.

IV. FOCUSED ANALYSIS

A. Level Of Ordinary Skill in the Art

24. 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.

25. A POSITA in the field of the '582 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.

26. 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.

27. 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 '582 patent.

B. 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. Cluster Computing

30. The '582 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, 1:22-25, 6:11-13, 11:46-58. Computer clusters (and cluster computing techniques), however, were well known in the art prior to the earliest priority date of the '582 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 direct, 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 '582 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.

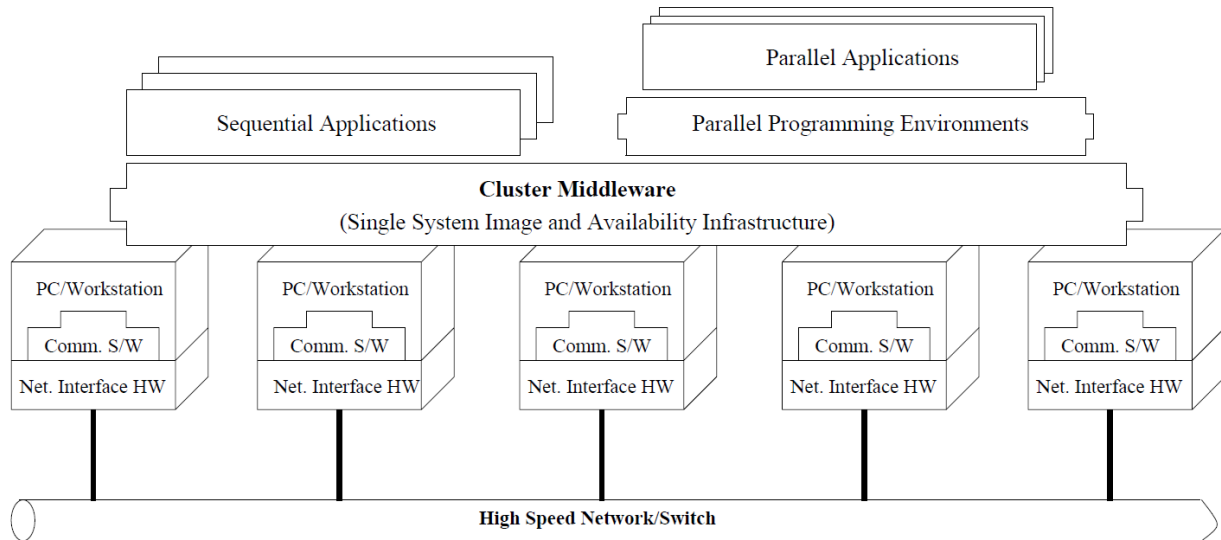


Figure 1.1 Cluster computer architecture.

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. Parallel Mathematical Processing

34. At the relevant time frame of the ’582 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 ’582 patent identifies a Mathematica kernel as a single-node kernel of a cluster node. Ex.1001, 1:42-45. However, this was nothing new, as it was known prior to the ’582 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 '582 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. Peer-to-peer Architecture

39. Cluster nodes each having single-node programs were known in the art prior to the ’582 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.

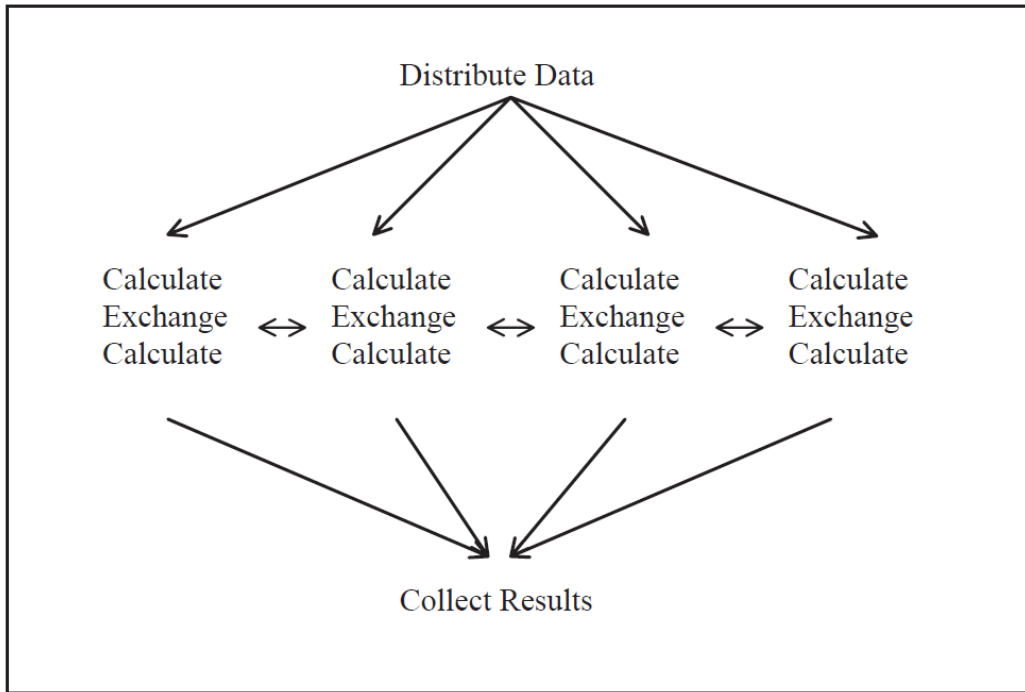


Figure 1.5 Basic structure of a SPMD program.

Ex.1011, FIG. 1.5

40. The '582 patent states that “first node module, the second node module, and the third node module are configured to communicate with one another using a peer-to-peer architecture” and that “MPI may be used to send messages between nodes in a computer cluster.” Ex.1001, 3:51-54, 6:11-13. 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 '582 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.

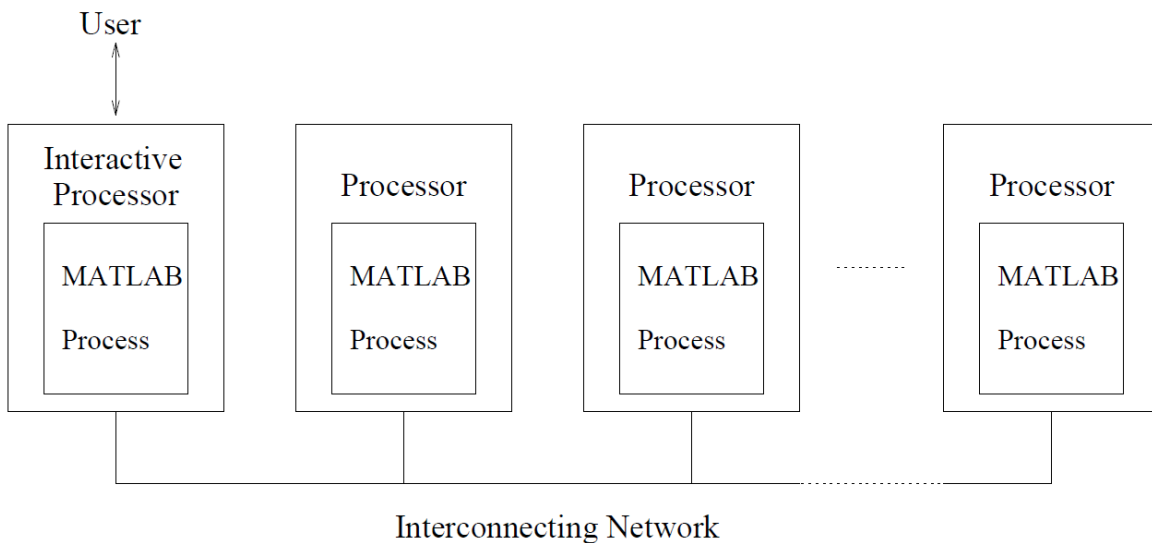


Figure 1: MultiMATLAB Architecture

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. **Sequential Processing from One Node to the Next**

46. Also known in the art prior to the '582 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.

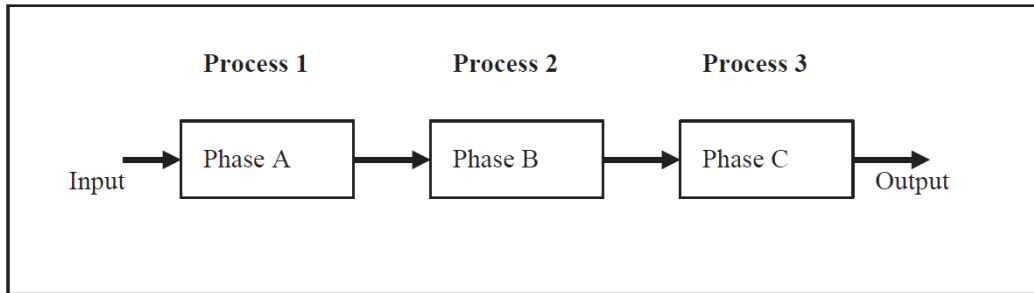


Figure 1.6 Data pipeline structure.

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.

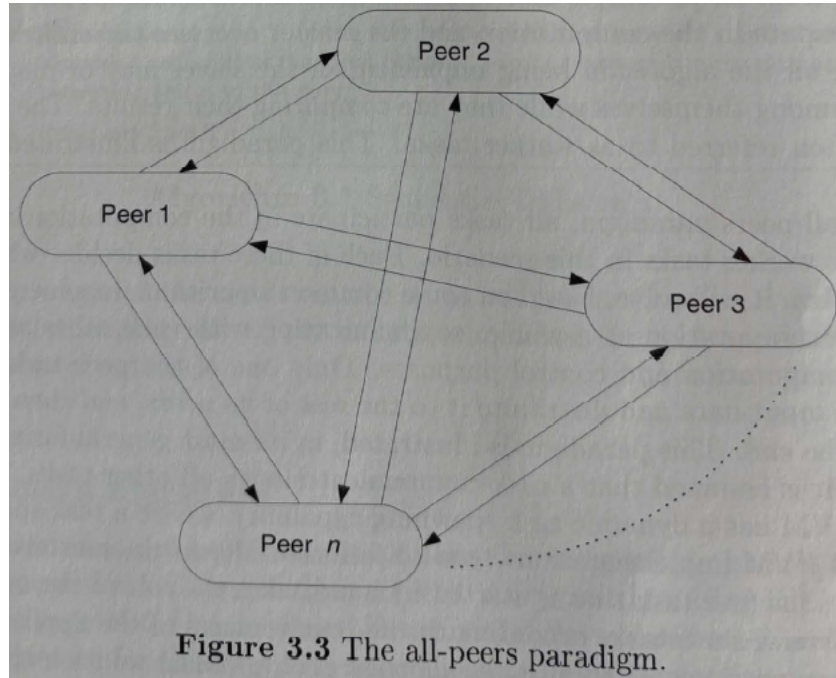
48. Buyya teaches combining paradigms (e.g., such as the SPMD cluster computing paradigm and the data pipeline structure for data flows) when discussing “the need, [in some applications,] to mix elements of different paradigms,” which may involve “situations where it makes sense to mix data and task 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:

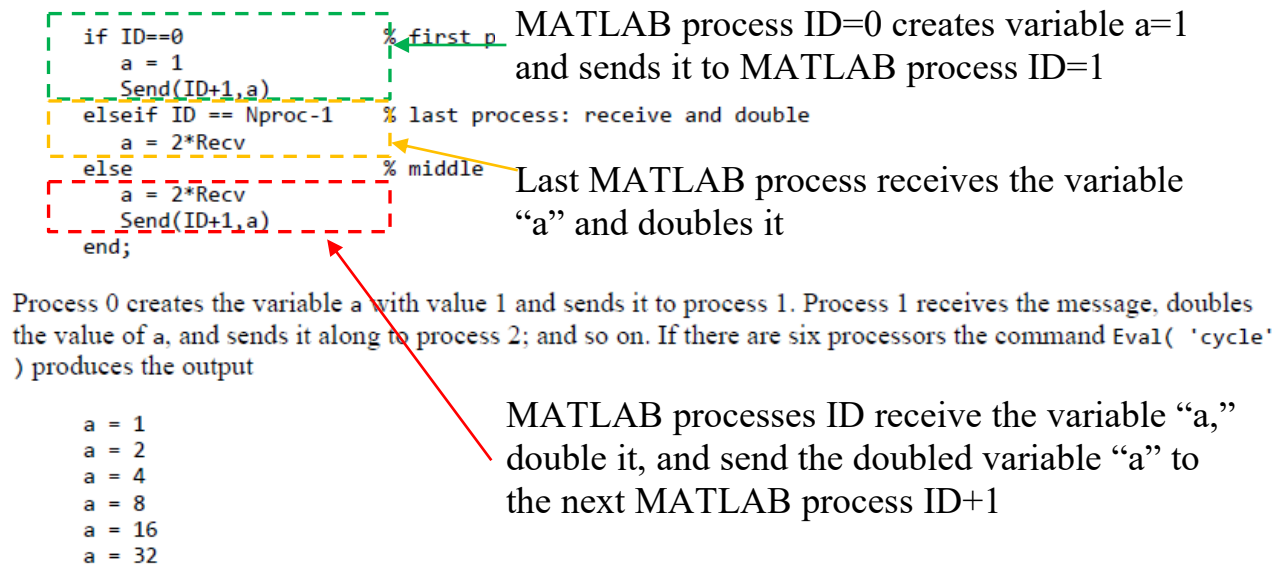
In the all-peers paradigm, all tasks participate in the computation and are referred to as worker tasks in this scenario. Each of these tasks decides what portion of the problem it will solve, based on some common algorithm. In general, all tasks control the computation as a whole, communicating with 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.



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:



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 constitute 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 ’582 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.

C. The '582 Patent

1. Summary of the '582 Patent

53. The '582 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 '582 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, 1:22-29. The purported novelty of the '582 patent is that it discloses “systems and methods for adding cluster computing functionality to a computer program.” Ex.1001, 1:16-17. Besides the “Mathematica software,” the '582 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, 4:19-26.

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, 5:24-25, 38-40. “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, 6:20-25. 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, 11:46-52. 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, 25:17-18, 24:34-37.

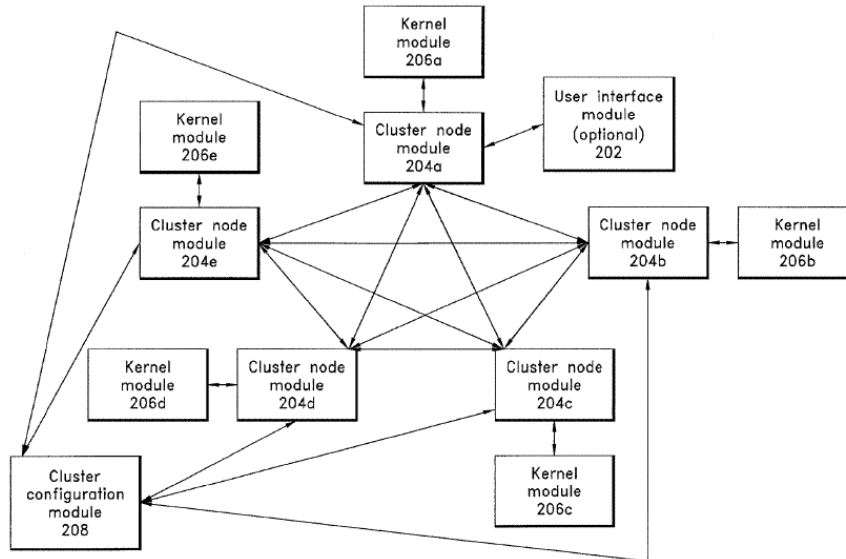


FIG. 2

Ex.1001, FIG. 2

55. Representative Claim 1 of the '582 patent is reproduced below:

1. A computer cluster comprising:
 - a first processor;
 - a second processor;
 - a third processor;
 - at least one computer-readable medium in communication at least one of the first processor, the second processor, or the third processor;
 - a first kernel residing in the at least one computer-readable medium, said first kernel configured to translate user commands into calls

for execution of matrix operations on an array of
data elements;

a first cluster node module residing in the at
least one computer-readable medium, said first
cluster node module configured to communicate
with the first kernel and receive commands from a
user interface;

a second kernel residing in the at least one
computer-readable medium, said second kernel
configured to execute of the matrix operations on
the array of data elements and return a first result
using the second processor;

a second cluster node module residing in the
at least one computer-readable medium, said
second cluster node module configured to send
commands to the second kernel and communicates
with the first cluster node module;

a third kernel residing in the at least one
computer-readable medium, said third kernel
configured to receive the first result and execute

additional matrix operations on the first and return
a second result to one or more of the first, second,
or third cluster node modules using the third
processor; and

a third cluster node module residing in the at
least one computer-readable medium, said third
cluster node module configured to send commands
to the third kernel and configured to communicate
with the first cluster node module and the second
cluster node module;

wherein the first cluster node module is
configured to return, based on the second result, a
final result.

56. As I explain in detail below, there is nothing novel about the alleged invention disclosed and claimed in the '582 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 '582 patent. Ex.1005, Abstract; Ex.1006, Abstract.

2. Prosecution History

57. The '582 patent was filed on January 23, 2023 and issued on November 7, 2023. Ex.1002, 229, 11-13. No Office Action was mailed during prosecution. Ex.1002, 1-343. The Examiner requested the Applicant file a terminal disclaimer over Patent Owner's U.S. Patent No. 11,563,621, which the Applicant did. Ex.1002, 33, 202. The Examiner then allowed the claims; however, the Examiner's Statement of Reasons for Allowance did not particularly identify any limitations of the allowed claims as the basis for allowance. Ex.1002, 25-32. A Certificate of Correction was issued that changed the ultimate wording of the claims. Ex.1002, 4-6.

D. Claim Construction

58. It is my understanding that in order to properly consider whether the '582 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.

1. “peer-to-peer architecture”

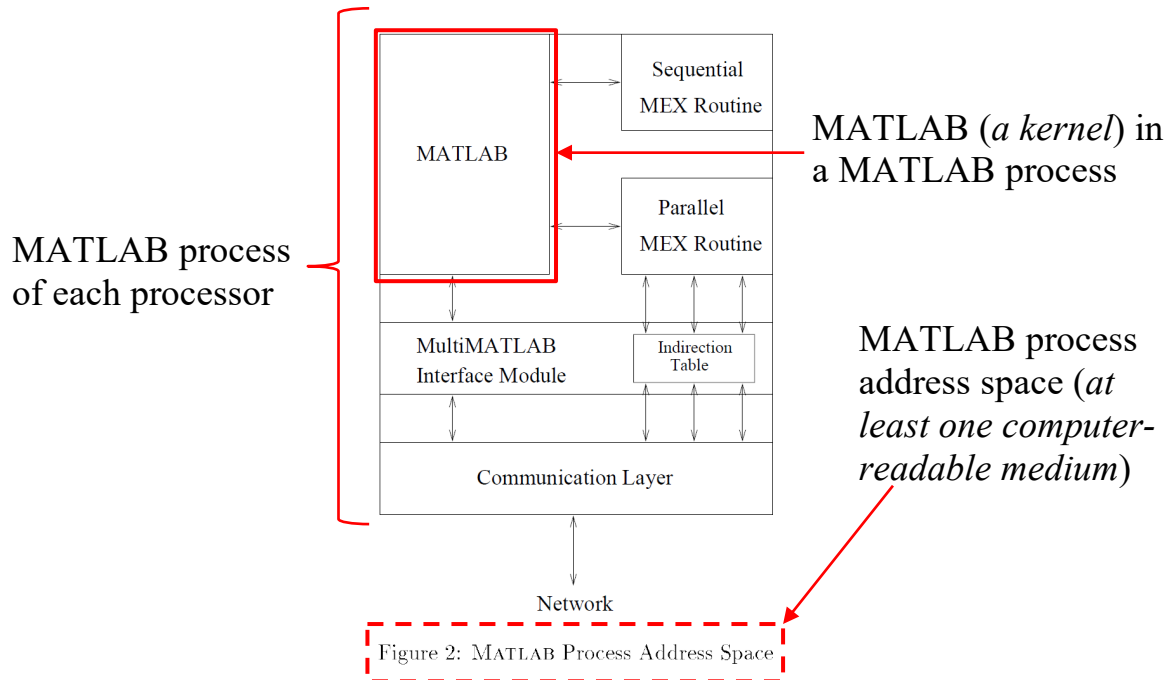
59. The limitation “peer-to-peer architecture” appears in claim 15.

60. 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 claim.

E. Key Teachings of the Prior Art

1. Kernel

61. Menon explains that each MATLAB process on the processors of the MultiMATLAB architecture 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). The ’621 patent identifies MATLAB as an example of a *kernel*. Ex.1001, 1:16-17, 4:21-26.



Ex.1005, FIG. 2 (annotated)

62. FIG. 1 of Menon below shows a MATLAB process running on each processor:

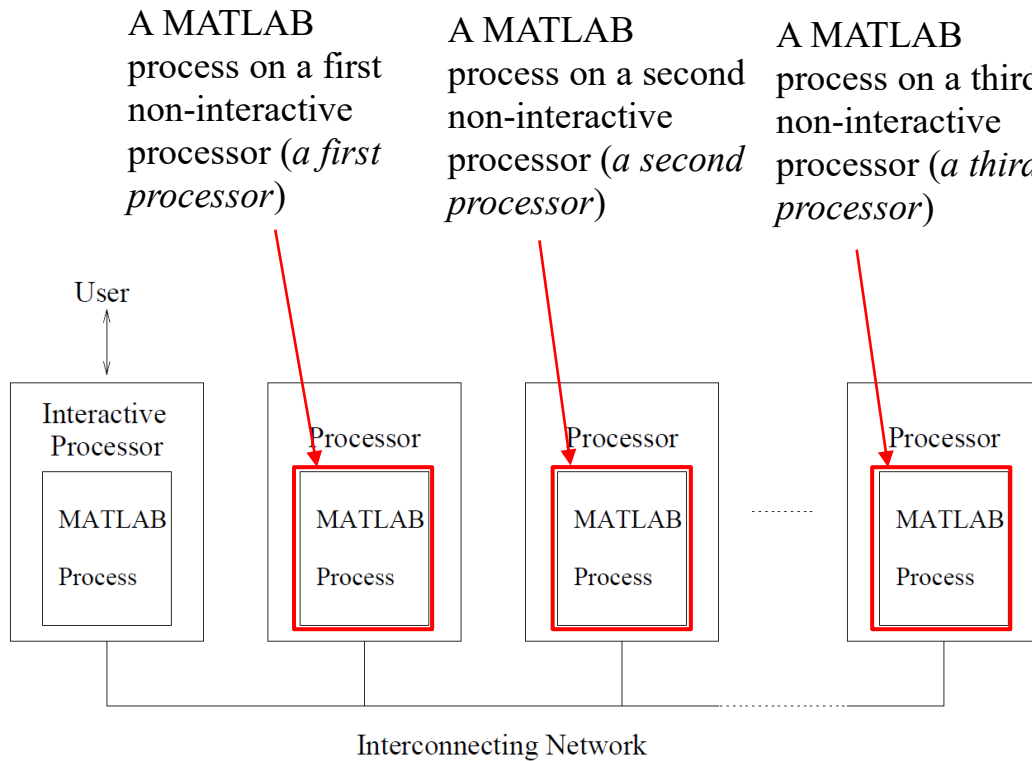


Figure 1: MultiMATLAB Architecture

Ex.1005, FIG. 1 (annotated)

63. 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.

2. Cluster node module

64. Menon discloses that each processor includes a set of components in the MATLAB process address space (*computer-readable medium*) to provide interconnection with other modules: “MEX Routines,” “MultiMATLAB interface module,” “indirection table” and “communication layer” (collectively, the *cluster node module*). Ex.1005, 3, 5. “MEX routines” provide “[a]ll parallel functionality

in the MultiMATLAB system.” Ex.1005, 3. “The interface module ... is responsible for initializing an underlying communication layer, such as MPI ..., and exposing it to the rest of the system.” Ex.1005, 3.

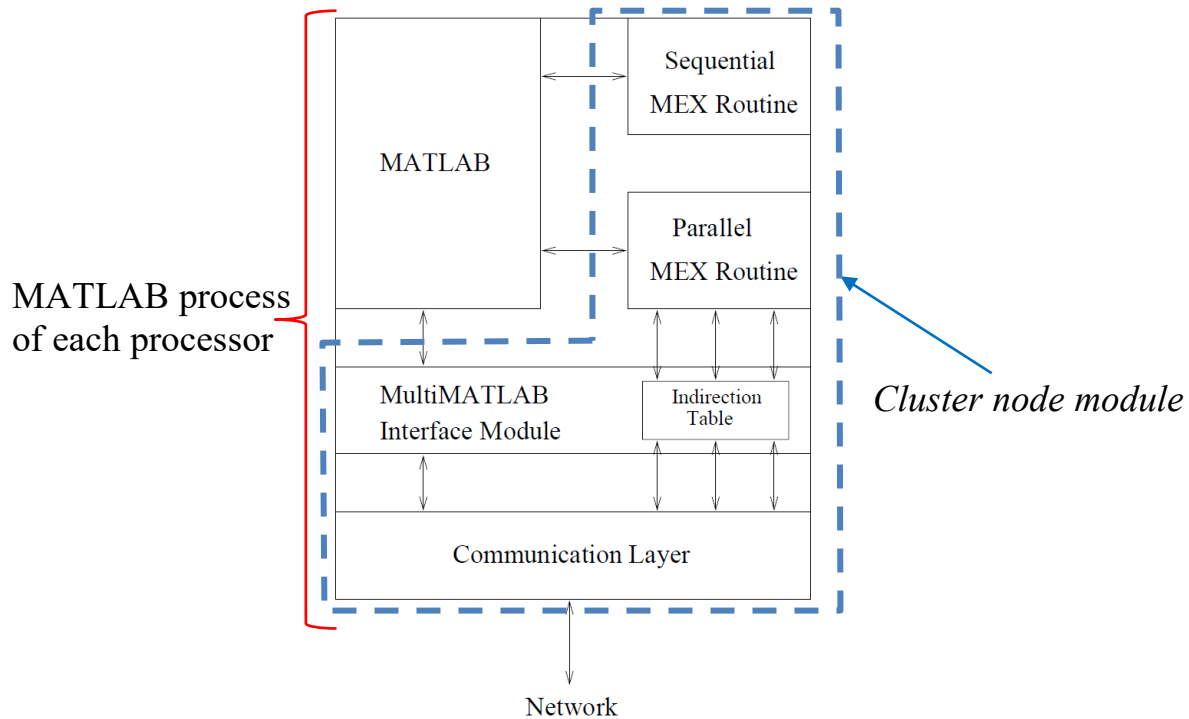
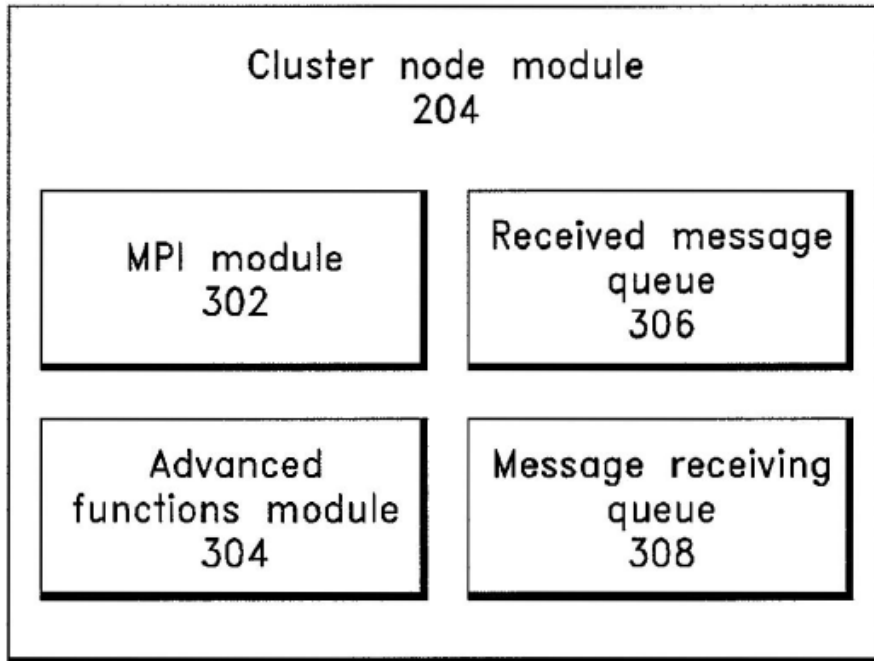


Figure 2: MATLAB Process Address Space

Ex.1005, FIG. 2 (annotated)

65. Correspondingly, the '621 patent depicts its cluster node module as a set of components to provide communication among the nodes:



Ex.1001, FIG. 3

66. In the cluster node module, 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.

67. MPIref discloses the standard for MPI send and receive. For example, MPIref describes that MPI_RECV() includes an MPI **receive buffer** for storing received data having the type specified by **datatype**. Ex.1017, 15-21. Menon’s MEX routine Recv(), which “receive[s] data sent from another process,” “do[es] any allocation necessary for received data.” Ex.1005, 9-10, Ex.1006, 10. Menon teaches that MEX routines (such as Recv()) are mapped (using the indirection table) to the analogous MPI calls (such as MPI_RECV() described in MPIref) found in the

communication layer (part of the *cluster node module*). Ex.1005, 3, 6. Menon's communication layer, using the MPI calls, communicates over the interconnection network with the communication layer in the other processor. Ex.1005, 3.

68. MPIref describes the details of what the parallel MEX routines such as Recv() 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 "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 basic datatypes that can be specified for message data values correspond to the basic datatypes of the host language." Ex.1017, 17.

69. 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:

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)(emphasis original).

70. 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.

3. User commands received by the first cluster node module without first passing through the first kernel

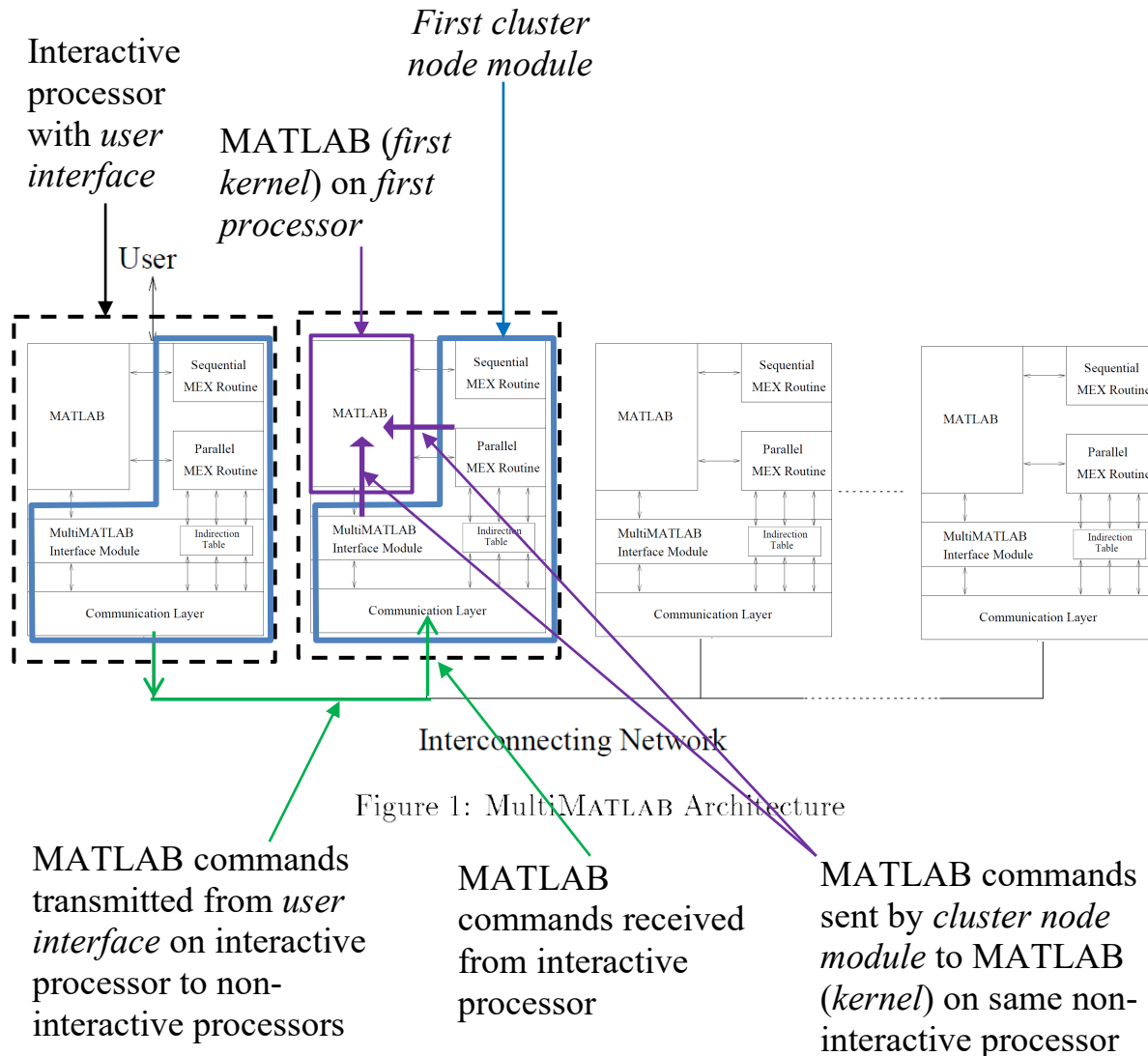
71. Trefethen provides an example illustration where a user directly interacts with a MATLAB process and “Eval” is “[t]he standard MultiMATLAB command for executing commands on one or more processors[.]” Ex.1006, 3. “[T]he user now types Eval(‘sqrt(2)’)” in an interactive MATLAB process at “a node of the IBM SP2” implementing a MultiMATLAB architecture having “6 MultiMATLAB processes running.” Ex.1006, 3. “[T]hen the MATLAB command sqrt(2) is executed on all six processors. The result is six repetitions of 1.4142.”

Ex.1006, 3. Because “sqrt(2)” is a “MATLAB command,” a POSITA would recognize and understand that it is MATLAB in the MATLAB process on each processor that executes the MATLAB command sqrt(2). Trefethen also discloses sending tasks and data to other processor nodes and provides the example “Eval([4 5] , ‘cond(hilb(ID))’)”. Ex.1006, 6. MATLAB on each of node ID=4 and ID=5 then perform the task “cond(hilb(ID))” using the data ID. Ex.1006, 4.

72. In other words, Menon and Trefethen teach that when a user, interacting with a MATLAB process on the interactive processor via a user interface, the “MEX routine [Eval] accepts ... [the] MATLAB command [and] sends the command[s]” to the MATLAB process on each non-interactive processor of the MultiMATLAB architecture. Ex.1005, 3-4, 6; Ex.1006, 3. The MEX routines are “run directly from MATLAB.” Ex.1005, 3. The MATLAB command arrives at each MATLAB process’s communication layer (part of the *cluster node module*). Ex.1005, 3. Then, “[o]n non-interactive processes, a separate top-level MEX routine” (part of *cluster node module*) of each MATLAB process receives the MATLAB command for execution “in [the separate top-level MEX routine’s] own MATLAB environment.” Ex.1005, 4.

73. A POSITA would thus recognize and understand that “the MATLAB command” is executed by MATLAB in the MATLAB process on each processor and that the MEX routines (part of *cluster node modules*) in a MATLAB process on

each non-interactive processor sends the MATLAB command to MATLAB (*kernel*) on that MATLAB process for execution by that MATLAB, as shown in the combined figure of Figures 1 and 2 of Menon below. Ex.1006, 3.



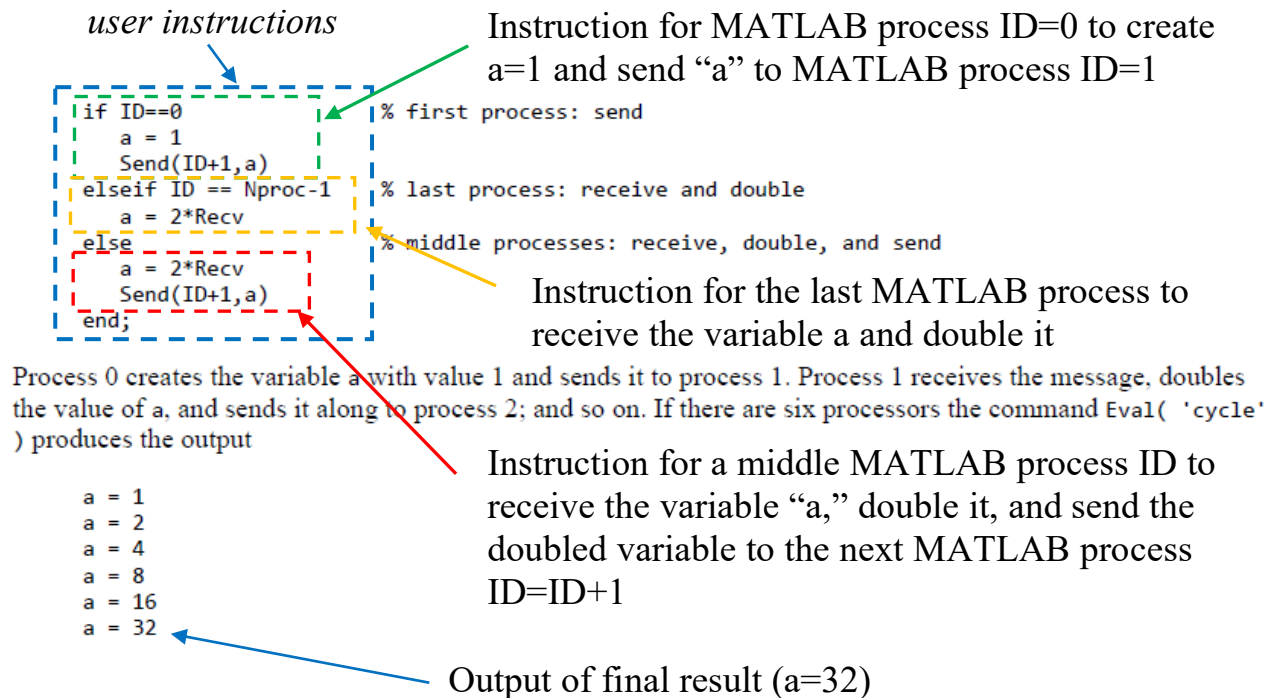
Ex.1005, FIGs. 1 and 2 (combined and annotated)

74. That is, (i) the user enters a command via the user interface on the interactive processor, (ii) that command is provided “directly” to the MEX routine of the cluster node module on the interactive processor, (iii) the cluster node

module on the interactive processor communicates the command to the cluster node modules on the first processor, second processor and third processor, (iv) the cluster node module of the first processor receives the command via the interconnecting network, and (v) the cluster node module of the first processor communicates the command to the first kernel. Thus, Menon and Trefethen teach that the command entered by the user is received by the first kernel **without the command first passing through the first kernel.**

4. Peer-to-peer mathematical computation

75. Trefethen 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:



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.

76. 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.”).

F. Reasons to Combine

1. Combining Trefethen with Menon

a. Menon and Trefethen Are Analogous Art

77. Menon and Trefethen are each analogous art at least because Menon and Trefethen are each directed to the same field of endeavor as the '582 patent. The '582 patent is directed to “the field of cluster computing.” Ex.1001, 1:15-16. 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.

78. Furthermore, Menon and Trefethen are each analogous art to the '582 patent because each is reasonably pertinent to a problem the inventors of the '582

patent attempted to solve. The '582 patent addresses the problem of “adding cluster computing functionality to a computer program” such as Mathematica, MATLAB, etc. Ex.1001, 1:16-17, 4:24-28. As an example, the '582 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, 2:22-24. The '582 patent describes using “Message Passing Interface (‘MPI’) calls directly from within a user interface, such as, for example, the Mathematica programming environment.” Ex.1001, 2:28-31.

79. 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.

80. 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.

81. 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 '582 patent.

b. Motivation to Combine Menon with Trefethen

82. 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.

83. 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.

84. 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.

85. 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

86. RS6000 is analogous art at least because it is directed to the same field of endeavor as the '582 patent. The '582 patent is directed to “the field of cluster computing.” Ex.1001, 1:15-16. Similarly, RS6000 is directed to “provid[ing] a state-of-the-art parallel computing system.” Ex.1007, 1.

87. Furthermore, RS6000 is analogous art to the '582 patent because RS6000 is reasonably pertinent to a problem the inventors of the '582 patent attempted to solve. The '582 patent purports to provide a solution “for adding cluster computing functionality to a computer program.” Ex.1001, 1:16-17. The '582 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.” Ex.1001, 1:22-25. The '582 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, 2:14-17.

88. 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.

89. Accordingly, in my opinion, a POSITA would understand that RS6000 is a prior art that is analogous to the ’582 patent.

b. Motivation to Combine RS6000 with Menon

90. 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 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

¹ As discussed in Section VIII.C.1, IBM SP2 is the same as IBM RS/6000 SP.

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.

91. 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.

92. 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 processors and memory devices disclosed by RS6000. Ex.1005, Abstract; Ex.1007, 1, 12-13.

93. 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 MPIref with Menon

a. MPIref is Analogous Art

94. MPIref is analogous to the ’582 patent at least because MPIref is directed to the same field of endeavor as the ’582 patent. The ’582 patent is directed to “the field of cluster computing.” Ex.1001, 1:15-16. 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.

95. Furthermore, MPIref is analogous to the '582 patent because MPIref is reasonably pertinent to a problem the inventors of the '582 patent attempted to solve. The '582 patent purports to provide a solution for “send[ing] messages between nodes in a computer cluster” by utilizing MPI. Ex.1001, 6:11-13. Likewise, MPIref describes “all the technical features” of the MPI. Ex.1017, ii.

96. Accordingly, in my opinion, a POSITA would understand that MPIref is a prior art that is analogous to the '582 patent.

b. Motivation to Combine MPIref with Menon

97. 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.

98. 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.

99. Additionally, Menon suggests the combination by expressly citing to MPIref and explaining that its MultiMATLAB architecture is implemented “based upon the MPI communication standard” taught by MPIref. Ex.1005, 1, 3, 16.

100. 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.

101. 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.

4. Combining POEref with Menon

a. POEref is Analogous Art

102. POEref is analogous art at least because it is directed to the same field of endeavor as the '582 patent. The '582 patent is directed to “the field of cluster computing.” Ex.1001, 1:15-16. Similarly, POEref is directed to “compil[ing], execut[ing], and analyz[ing] parallel programs.” Ex.1008, ix.

103. Furthermore, POEref is analogous art to the '582 patent because it is reasonably pertinent to a problem the inventors of the '582 patent attempted to solve. The '582 patent purports to provide a solution “for adding cluster computing functionality to a computer program.” Ex.1001, 1:16-17. The '582 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, 2:14-17. The '582 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, 1:15-18, 29-31, 2:13-14.

104. 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.

105. Accordingly, in my opinion, a POSITA would understand that POEref is a prior art that is analogous to the '582 patent.

b. Motivation to Combine POEref with Menon

106. 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.

107. 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.

108. 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.

109. 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, 14; Ex.1008, ix.

V. DETAILED ANALYSIS

110. I have been asked to provide my opinion as to whether the Challenged Claims of the '582 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 '582 patent.

111. 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 '582 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. PRIOR ART

112. Below is a detailed discussion of the prior art references I considered as part of my analysis of '582 patent claims 1-20:

- (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).

1. Menon

a. Summary of Menon

113. Like the '582 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.

114. 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.

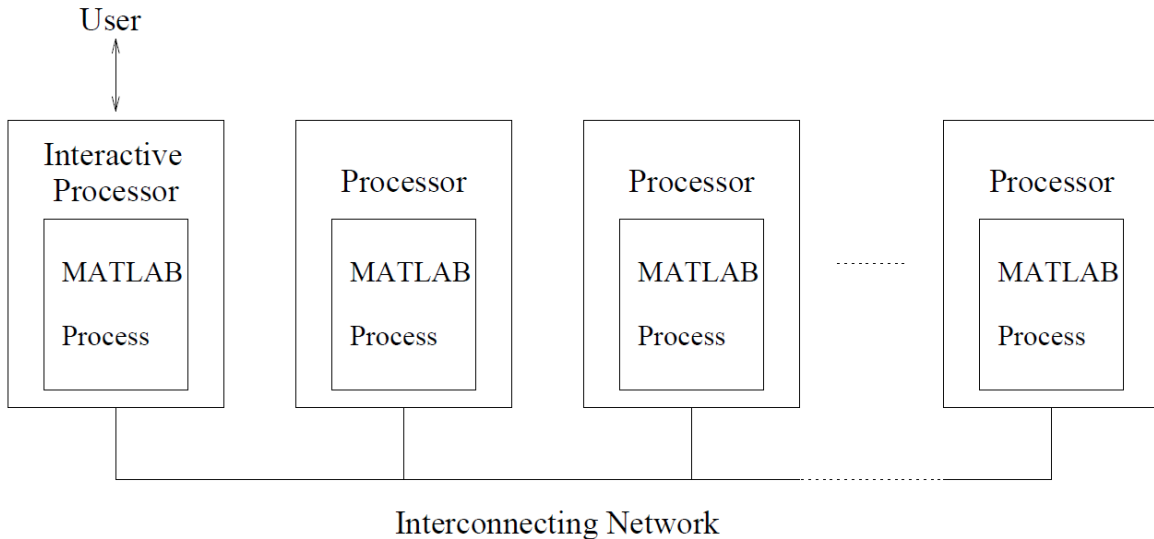


Figure 1: MultiMATLAB Architecture

Ex.1005, FIG. 1.

115. 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.

116. 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.

117. 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.

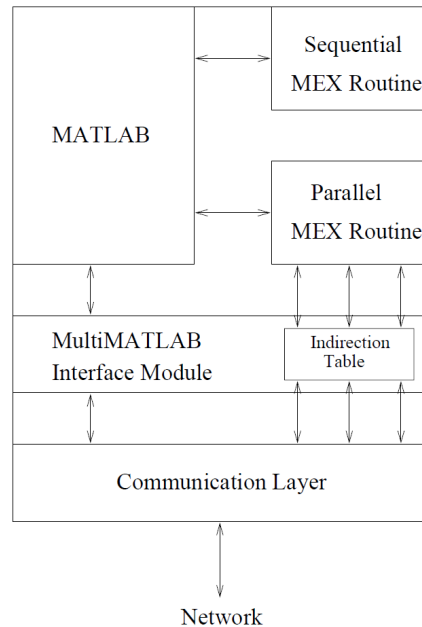


Figure 2: MATLAB Process Address Space

Ex.1005, FIG. 2

118. 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.

119. 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) and “sends the commands over the underlying communication layer to the processes corresponding to the specified IDs.” 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.

120. 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.

121. 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. Menon’s Table 1 describes commands for two distinct paradigms: a master/slave paradigm and SPMD/Message Passing. The SPMD/Message Passing paradigm uses Eval(), Send() and Recv() for peer-to-peer communication and that is the focus of my analysis.

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)

122. 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.

b. Availability of Menon

123. 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 '582 patent. Ex.1028, 4-9.

124. 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.

125. 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 '582 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 '582 patent. Ex.1028, 4-9.

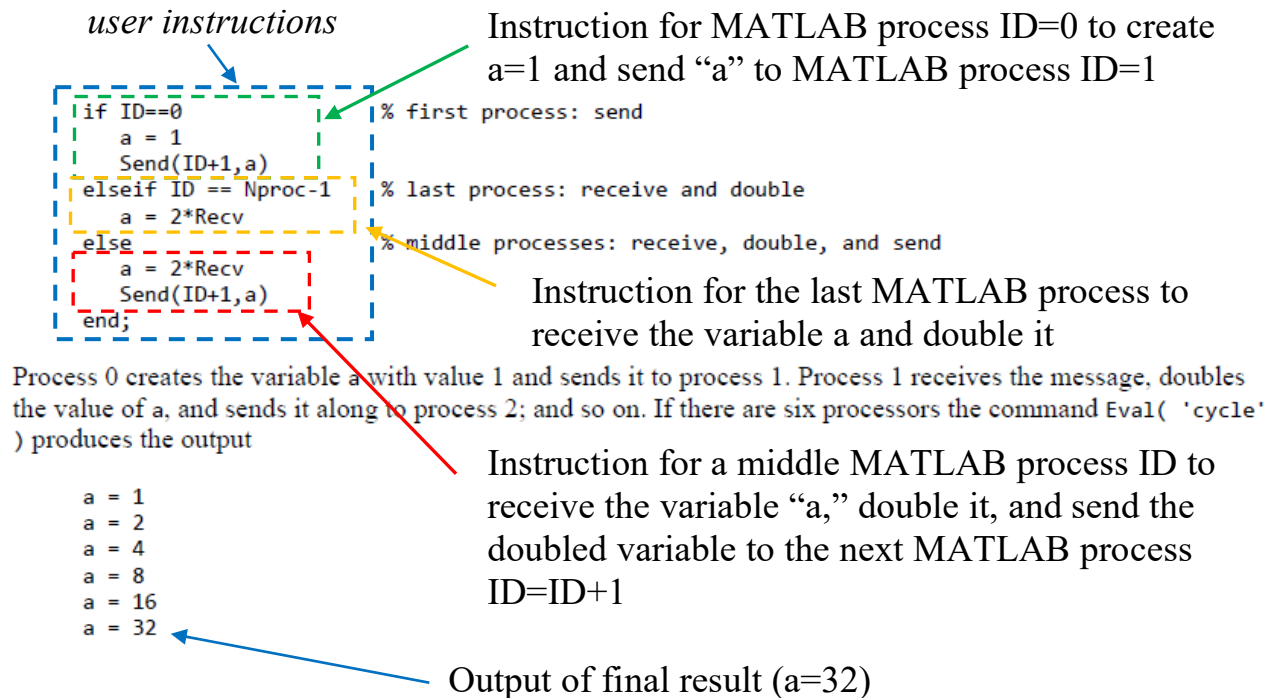
2. Trefethen

a. Summary of Trefethen

126. 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.

127. 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:



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.

128. 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.”).

b. Availability of Trefethen

129. 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 '582 patent, including by Menon. Ex.1035, 2-3.

130. 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 ’582 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.

3. RS6000

a. Summary of RS6000

131. 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.

b. Availability of RS6000

132. 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.

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

4. POEref

a. Summary of POEref

134. 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.

135. 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.

b. Availability of POEref

136. 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.

137. 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.

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

5. MPIref

a. Summary of MPIref

139. 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.

140. 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).

141. 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).

b. Availability of MPIref

142. 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 '582 patent. Ex.1037, 2-16.

143. Furthermore, Menon, while citing to MPIref, provided a website address of Argonne National Laboratory (<http://www.mcs.anl.gov/mpi>) for retrieving MPIref. Ex.1005, 16; *see also* Section IX.A.3.b. A copy of MPIref was available at that website as of June 24, 1997 and accessible using the following link chain starting at the provided website address: "MPI Standard" → "Postscript for MPI draft standard Version 1.0 of May 5, 1994." Ex.1048, 1-3; Ex.1049, 11-247.

144. Accordingly, an interested POSITA exercising reasonable diligence would have located MPIref at the MPI Forum website and/or Argonne National Laboratory, 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 '582 patent. Ex.1037, 2-16.

B. Ground 1: Claims 1-3 and 5-19 are obvious over Menon in view of Trefethen, RS6000, and MPIref

145. The combination of Menon, Trefethen, RS6000, and MPIref renders obvious claims 1-3 and 5-19 as discussed below.

1. Combining Trefethen with Menon

a. Menon and Trefethen Are Analogous Art

146. Menon and Trefethen are each analogous art at least because Menon and Trefethen are each directed to the same field of endeavor as the '582 patent. The '582 patent is directed to “the field of cluster computing.” Ex.1001, 1:15-16. 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.

147. Furthermore, Menon and Trefethen are each analogous art to the '582 patent because each is reasonably pertinent to a problem the inventors of the '582 patent attempted to solve. The '582 patent addresses the problem of “adding cluster computing functionality to a computer program” such as Mathematica, MATLAB, etc. Ex.1001, 1:16-17, 4:24-28. As an example, the '582 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, 2:22-24. The '582 patent describes using “Message Passing Interface ('MPI') calls directly from within a user interface, such as, for example, the Mathematica programming environment.” Ex.1001, 2:28-31.

148. 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.

149. 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.

150. 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 ’582 patent.

b. Motivation to Combine Menon with Trefethen

151. 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.

152. 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.

153. 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.

154. 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

155. RS6000 is analogous art at least because it is directed to the same field of endeavor as the '582 patent. The '582 patent is directed to “the field of cluster computing.” Ex.1001, 1:15-16. Similarly, RS6000 is directed to “provid[ing] a state-of-the-art parallel computing system.” Ex.1007, 1.

156. Furthermore, RS6000 is analogous art to the '582 patent because RS6000 is reasonably pertinent to a problem the inventors of the '582 patent attempted to solve. The '582 patent purports to provide a solution “for adding cluster computing functionality to a computer program.” Ex.1001, 1:16-17. The '582 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.” Ex.1001, 1:22-25. The '582 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, 2:14-17.

157. 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.

158. Accordingly, in my opinion, a POSITA would understand that RS6000 is a prior art that is analogous to the ’582 patent.

b. Motivation to Combine RS6000 with Menon

159. 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.”² Ex.1005, 5; Ex.1007, 1. For example, RS6000 discloses that “[t]he 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.

160. 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

² As discussed in Section VIII.C.1, IBM SP2 is the same as IBM RS/6000 SP.

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.

161. 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.

162. 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 MPIref with Menon

a. MPIref is Analogous Art

163. MPIref is analogous to the ’582 patent at least because MPIref is directed to the same field of endeavor as the ’582 patent. The ’582 patent is directed to “the field of cluster computing.” Ex.1001, 1:15-16. 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.

164. Furthermore, MPIref is analogous to the ’582 patent because MPIref is reasonably pertinent to a problem the inventors of the ’582 patent attempted to solve. The ’582 patent purports to provide a solution for “send[ing] messages between nodes in a computer cluster” by utilizing MPI. Ex.1001, 6:11-13. Likewise, MPIref describes “all the technical features” of the MPI. Ex.1017, ii.

165. Accordingly, in my opinion, a POSITA would understand that MPIref is a prior art that is analogous to the ’582 patent.

b. Motivation to Combine MPIref with Menon

166. 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.

167. 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.

168. Additionally, Menon suggests the combination by expressly citing to MPIref and explaining that its MultiMATLAB architecture is implemented “based upon the MPI communication standard” taught by MPIref. Ex.1005, 1, 3, 16.

169. 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.

170. 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.

4. Claim 1

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

171. Menon in combination with RS6000 renders obvious the preamble. The '582 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, 1:22-25. Likewise, Menon discloses a MultiMATLAB

cluster with a series of MATLAB processes, implemented on an IBM SP2 that uses multiple processors, 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*.

172. 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:

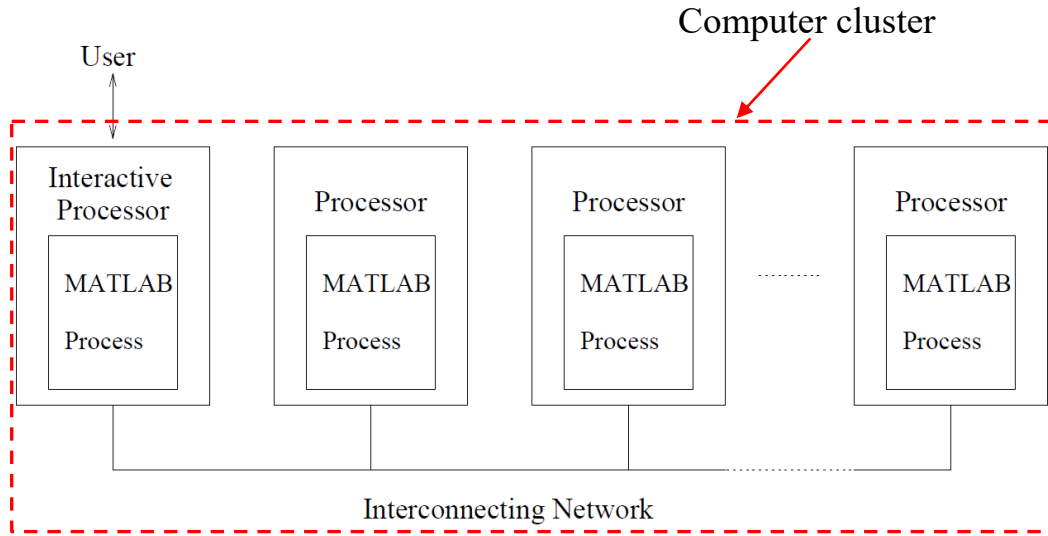


Figure 1: MultiMATLAB Architecture

Ex.1005, FIG. 1 (annotated)

173. 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.

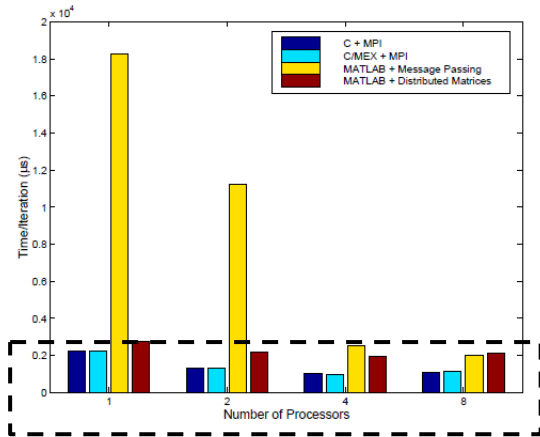


Figure 7: Parallel Conjugate Gradients on the IBM SP-2: 256×256 matrix

8 processors

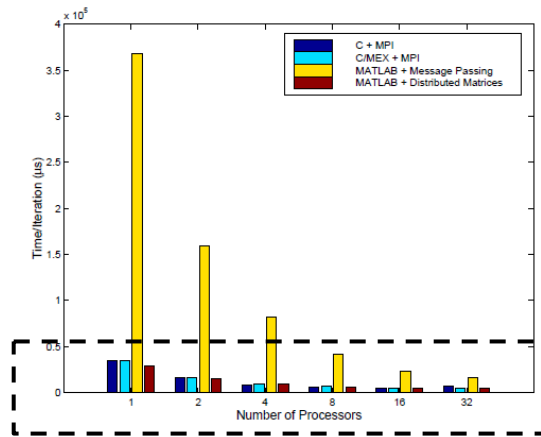


Figure 8: Parallel Conjugate Gradients on the IBM SP-2: 1024×1024 matrix

32 processors

Ex.1005, FIGs. 7 and 8 (annotated)

174. 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.

175. 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 interactive processor of the IBM SP2 and then the non-interactive processors 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.

176. 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 processors, and the IBM SP2’s parallel computing system (as described in RS6000) has a series of processors 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 processors would result in Menon’s MATLAB processes each running on a processor of the IBM SP2, as exemplified below:

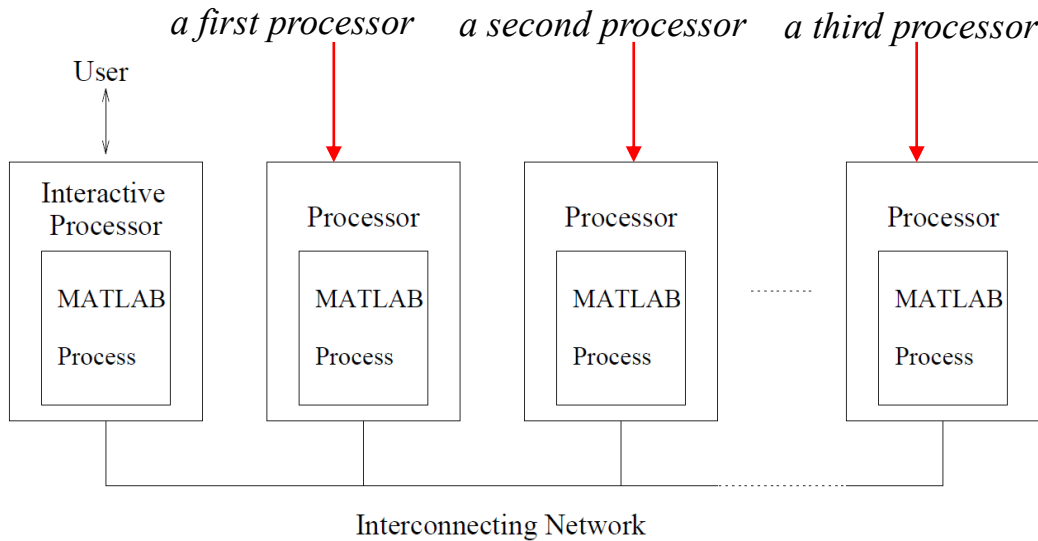


Figure 1: MultiMATLAB Architecture

Ex.1005, FIG. 1 (annotated)

177. 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 processors, switches, network connectivity, and disk drives, for executing software, renders obvious *a computer cluster*.

b. [1.1.1] *a first processor*;

178. Menon and RS6000 render obvious *a first processor* because Menon teaches a first non-interactive processor of the MultiMATLAB architecture that is implemented in IBM SP2 and RS6000 teaches that IBM SP2 has at least four, eight, twelve, or sixteen processors.

179. In limitation [1.0], it was shown that Menon’s MultiMATLAB architecture, implemented on an IBM SP2 using 32 processors, teaches *a computer cluster*. 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 teaches that each processor node with a symmetric multiprocessor has “four, eight, twelve, or sixteen 375 MHz 630FP 64-bit processors.” Ex.1007, 9. A POSITA would recognize and understand that Menon’s first non-interactive processor is one of the four processors on the IBM SP2, rendering obvious *a first processor*. Ex.1005, 3 (“each processor in a parallel platform individually runs a MATLAB process”).

180. Figure 1 of Menon below shows an example illustration of a MultiMATLAB architecture (*computer cluster*) having one interactive processor with an interactive MATLAB process and a plurality of non-interactive processors (e.g., a first non-interactive processor, a second non-interactive processor, a third non-interactive processor, etc.) each having a non-interactive MATLAB process. Ex.1005, 3-4.

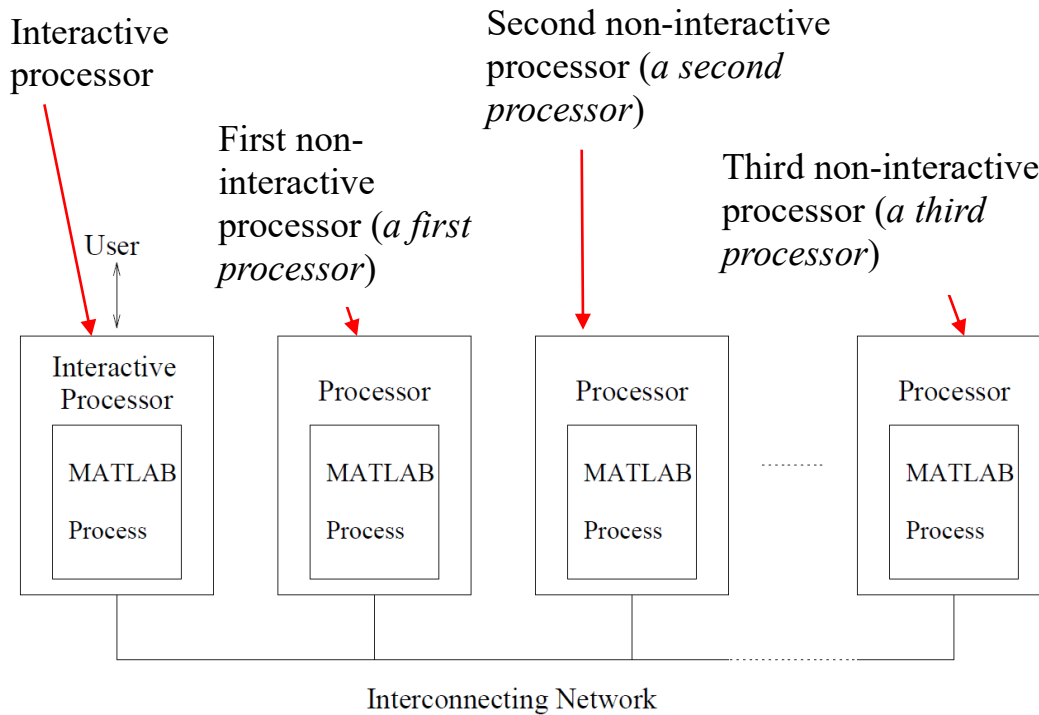


Figure 1: MultiMATLAB Architecture

Ex.1005, FIG. 1 (annotated)

181. Accordingly, Menon teaching the first non-interactive processor of the MultiMATLAB architecture that is implemented in IBM SP2 in view of RS6000 teaching a symmetric multiprocessor of the IBM SP2 having “four, eight, twelve, or sixteen” processors renders obvious *a first processor*. Ex.1007, 9.

c. [1.1.2] *a second processor*;

182. Menon and RS6000 render obvious *a second processor* because Menon teaches a second non-interactive processor of the MultiMATLAB architecture that is implemented in IBM SP2 and RS6000 teaches that IBM SP2 has a symmetric multiprocessor (SMP) that has four, eight, twelve, or sixteen

processors. A POSITA would recognize and understand that Menon's second non-interactive processor is one of the four processors on the IBM SP2, rendering obvious *a second processor*. Ex.1005, 3 ("each processor in a parallel platform individually runs a MATLAB process").

183. In limitation [1.1.1], it was shown that Menon's MultiMATLAB architecture (*computer cluster*) has one interactive processor and a plurality of non-interactive processors (e.g., a first non-interactive processor, a second non-interactive processor, a third non-interactive processor, etc., as shown in Figure 1 of Menon there. Ex.1005, 3-4. Accordingly, Menon teaching the second non-interactive processor of the MultiMATLAB architecture that is implemented in IBM SP2 in view of RS6000 teaching a symmetric multiprocessor of the IBM SP2 having "four, eight, twelve, or sixteen" processors renders obvious *a second processor*. Ex.1007, 9.

d. [1.1.3] *a third processor*;

184. Menon and RS6000 render obvious *a third processor* because Menon teaches a third non-interactive processor of the MultiMATLAB architecture that is implemented in IBM SP2 and RS6000 teaches that IBM SP2 has a symmetric multiprocessor (SMP) that has four, eight, twelve, or sixteen processors. A POSITA would recognize and understand that Menon's third non-interactive processor is one of the four processors on the IBM SP2, rendering obvious *a third processor*.

Ex.1005, 3 (“each processor in a parallel platform individually runs a MATLAB process”).

185. In limitation [1.1.1], it was shown that Menon’s MultiMATLAB architecture (*computer cluster*) has one interactive processor and a plurality of non-interactive processors (e.g., a first non-interactive processor, a second non-interactive processor, a third non-interactive processor, etc., as shown in Figure 1 of Menon there. Ex.1005, 3-4. Accordingly, Menon teaching the third non-interactive processor of the MultiMATLAB architecture that is implemented in IBM SP2 in view of RS6000 teaching a symmetric multiprocessor of the IBM SP2 having “four, eight, twelve, or sixteen” processors renders obvious *a third processor*. Ex.1007, 9.

- e. **[1.2] *at least one computer-readable medium in communication with at least one of the first processor, the second processor, or the third processor;***

186. Menon in view of RS6000 renders obvious *at least one computer-readable medium in communication with at least one of the first processor, the second processor, or the third processor* because Menon discloses that each processor has a MATLAB process with a MATLAB process address space that the processor reads and writes (*in communication with at least one of the first processor, the second processor, or the third processor*) and RS6000 teaches that each processor node of the IBM SP2 has hard drives and up to 64 GB memory accessible by that processor node.

187. 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. Figure 1 of Menon below shows a MATLAB process executing on each processor of the MultiMATLAB architecture, i.e., a MATLAB process on an interactive processor and MATLAB processes each on additional non-interactive processors (e.g., a first non-interactive processor (*a first processor*), a second non-interactive processor, a third non-interactive processor, etc.), of the MultiMATLAB architecture.

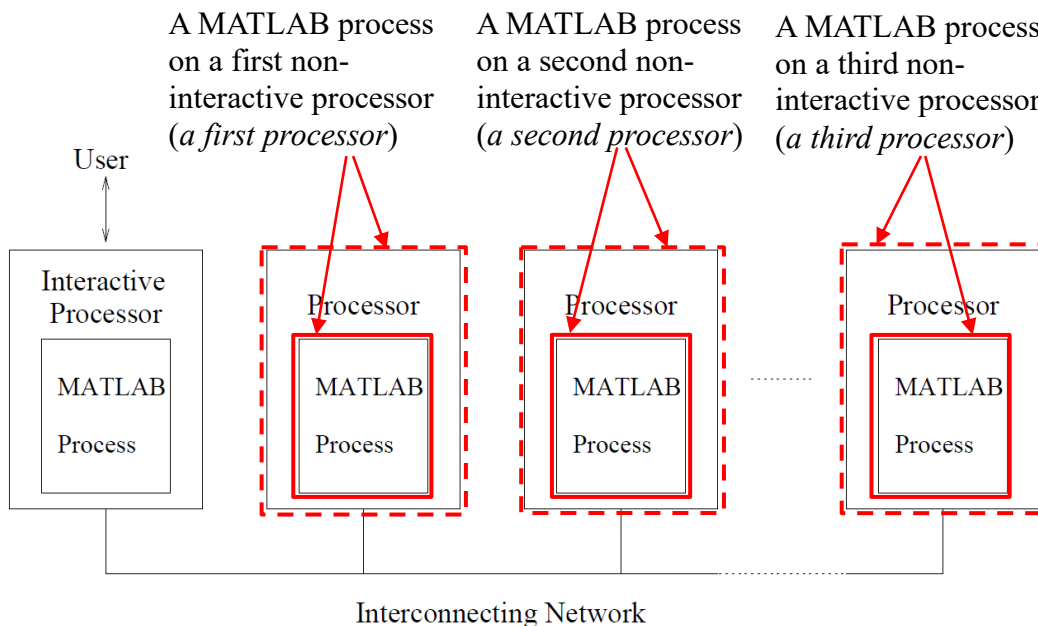
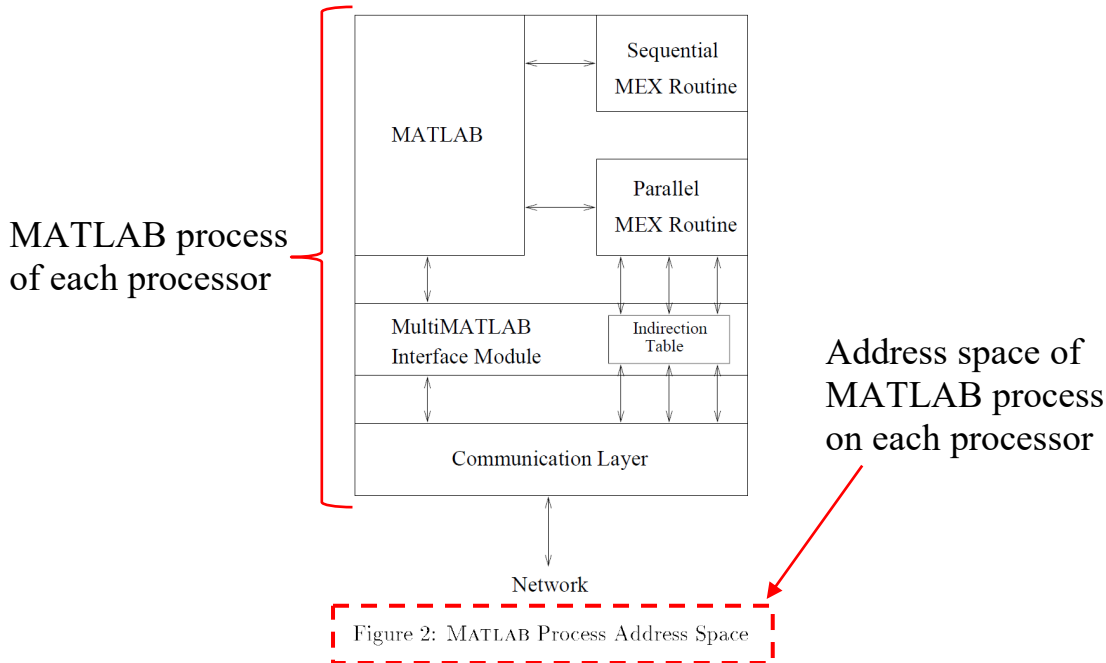


Figure 1: MultiMATLAB Architecture

Ex.1005, FIG. 1 (annotated)

188. Second, Menon teaches *at least one computer-readable medium* that is *in communication with* a processor, because Menon describes each MATLAB process, which is run by a processor, as being in an “address space” that the MATLAB process reads and writes. Ex.1005, FIG. 2; Ex.1018, 67-68. An “address space” refers 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.” Ex.1018, 67-68. A POSITA would recognize and understand that a process that is being run by a processor and that reads and writes an address space uses that processor to read and write the address space. Accordingly, because each processor of the MultiMATLAB architecture has a MATLAB process, run by that processor, in a MATLAB process address space that the processor reads and writes and is in a computer’s memory, a POSITA would recognize and understand that each MATLAB process’s address space is *a computer-readable medium* that is *in communication with* the processor of that MATLAB process. FIG. 2 of Menon below shows a MATLAB process address space for a MATLAB process on a processor.



Ex.1005, FIG. 2 (annotated)

189. In limitation [1.0], it was shown that Menon’s MultiMATLAB architecture is implemented on an IBM SP2 using 32 processors. The MATLAB process address space (*a computer-readable medium*) of each processor of the IBM SP2 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 processor nodes have internal and external hard disk drives), 10 (disclosing IBM SP2 processor nodes have “one to four memory cards” that support a “maximum of 64 GB”).

190. Accordingly, Menon’s disclosure of each processor having a MATLAB process that has a MATLAB process address space (*a computer-readable medium*) that the processor reads and writes (*in communication with at least one of the first processor, the second processor, or the third processor*) in

view of RS6000 teaching that each processor node of the IBM SP2 has hard drives and up to 64 GB memory accessible by that processor node, renders obvious *at least one computer-readable medium in communication with at least one of the first processor, the second processor, or the third processor.*

f. [1.3.1] *a first kernel residing in the at least one computer-readable medium,*

191. Menon and RS6000 render obvious *a first kernel residing in the at least one computer-readable medium* because Menon and RS6000 disclose MATLAB in a MATLAB process of the first non-interactive processor of the MultiMATLAB architecture, where the MATLAB process is contained in a MATLAB process address space.

192. First, Menon discloses that each processor of the MultiMATLAB architecture 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 running on each processor of the MultiMATLAB architecture.

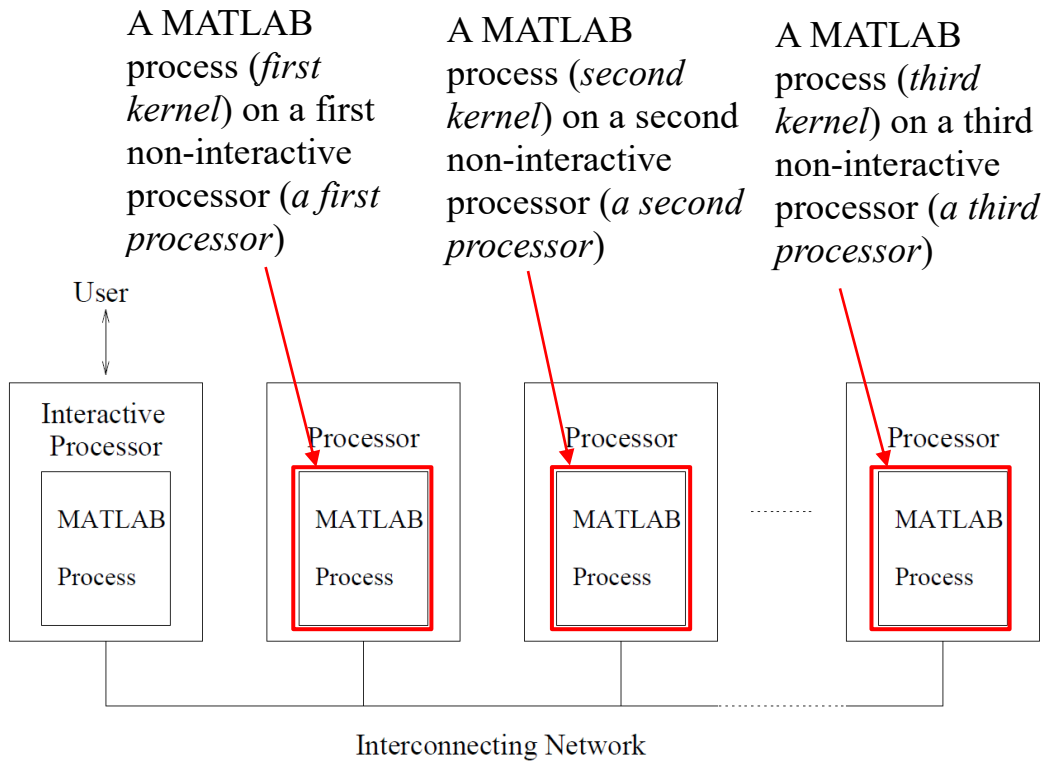
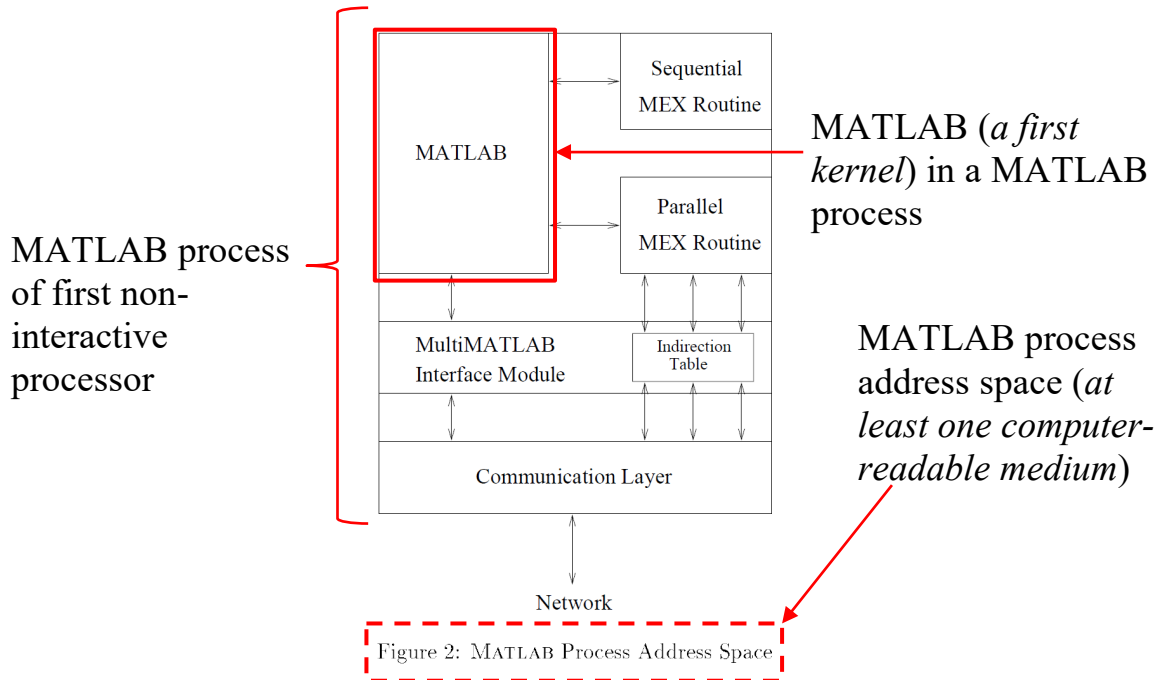


Figure 1: MultiMATLAB Architecture

Ex.1005, FIG. 1 (annotated)

193. Menon explains that each MATLAB process on the processors of the MultiMATLAB architecture 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. MATLAB of a MATLAB process on a processor

teaches *a kernel*, as shown below in Figure 2 of Menon below. Accordingly, MATLAB of the MATLAB process on the first non-interactive processor of the MultiMATLAB architecture teaches *a first kernel*. I also note that the '582 patent identifies MATLAB as an example of a *kernel*. Ex.1001, 1:16-17, 4:21-26.



Ex.1005, FIG. 2 (annotated)

194. Second, in limitation [1.2], it was shown that the MATLAB process of each processor of Menon's MultiMATLAB architecture is in a MATLAB process address space in memory (*computer-readable medium*). Accordingly, Menon teaches a MATLAB (*a first kernel*) in the MATLAB process on the first non-interactive processor (*first processor*), where the MATLAB process is contained in (*residing in*) a MATLAB process address space (*the at least one computer-readable medium*), as shown in Figure 2 of Menon above.

195. Accordingly, Menon and RS6000 disclose MATLAB in a MATLAB process of the first non-interactive processor of the MultiMATLAB architecture, where the MATLAB process is contained in a MATLAB process address space, rendering obvious *a first kernel residing in the at least one computer-readable medium.*

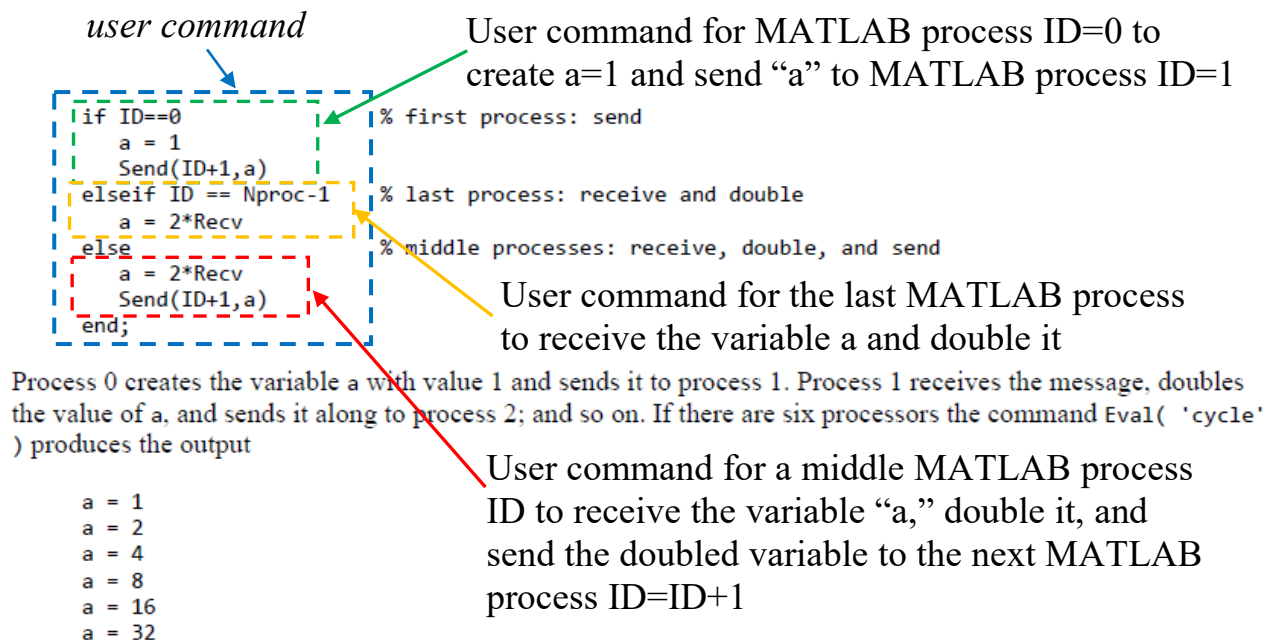
g. [1.3.2] *said first kernel configured to translate user commands into calls for execution of matrix operations on an array of data elements;*

196. Menon and Trefethen render obvious *said first kernel configured to translate user commands into calls for execution of matrix operations on an array of data elements* because Menon teaches MATLAB in the MATLAB process of the first non-interactive processor, in view of Trefethen teaching an m-file where that MATLAB of the MATLAB process on the first non-interactive processor executing an m-file to send a variable to a next MATLAB process for execution, and in further view of Menon and Trefethen teaching that each MATLAB process performs matrix-related mathematical evaluations of matrices.

197. First, in limitation [1.3.1], it was shown that MATLAB in the MATLAB process of the first non-interactive processor teaches *said first kernel.*

198. Second, Trefethen describes a user starting MATLAB on six processors to have “6 MultiMATLAB processes running” and then typing “Eval(‘cycle’)” in the MATLAB process on the interactive processor, where Eval is the

“standard MultiMATLAB command for executing commands on one or more processors.” Ex.1006, 3, 6. The program or m-file “cycle.m” instructs: (i) the MATLAB process on the interactive processor 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, as shown below:



Ex.1006, 6 (annotated).

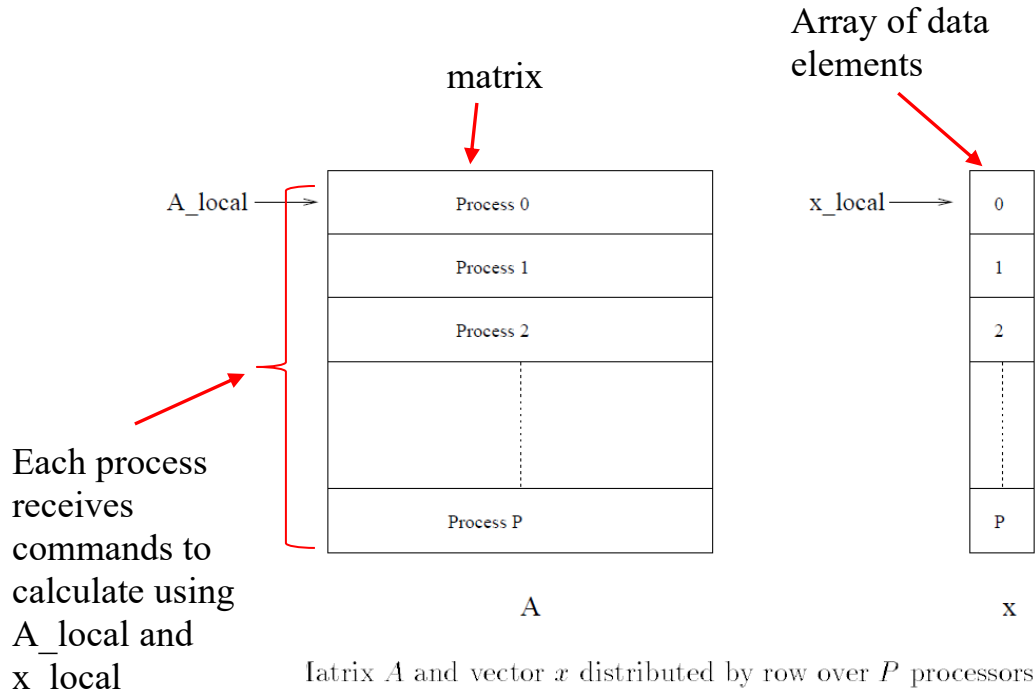
199. Because “each command passed to Eval was executed on all MATLAB processes,” the user commands of m-file cycle.m are translated into the mathematical computations and calls executed by each MATLAB process,

including the MATLAB process (i.e., MATLAB process ID=1) on the first non-interactive processor. Ex.1006, 4. The user commands of m-file cycle.m are translated into the mathematical computations and calls executed by each MATLAB process, including the MATLAB process (i.e., MATLAB process ID=1) on the first non-interactive processor. Ex.1006, 4. For example, MATLAB process ID=1 receives from MATLAB process ID=0 the variable “a” having the value “1,” the command is translated into multiplication that is executed (“a=2*Recv”), and then the command is translated into a call that sends the doubled variable “a=2” to MATLAB process ID+1, i.e., MATLAB process 2 (“Send(ID+1,a)”). Ex.1006, 6. In turn, MATLAB process 2 receives the variable a, translates the command into multiplication, and then translates the command into a call that sends the value of the variable “a” to the next MATLAB process, MATLAB process ID=3. Ex.1006, 4. Because an m-file is a script file that “allows [a user] to place MATLAB commands in a simple text file and then tells MATLAB to open the file and evaluate the commands exactly as it would have if [the user] had typed the commands at the MATLAB prompt,” a POSITA would recognize and understand that the m-file cycle.m that Trefethen teaches is executed by a MATLAB process is executed by MATLAB of that MATLAB process. Ex.1016, 47.

200. Accordingly, Trefethen teaches that MATLAB (*said first kernel*) of the MATLAB process ID=1 on the first non-interactive processor receives the m-file

cycle.m (*user commands*) and executes cycle.m (*configured to translate user commands into*) to send the doubled variable “a=2” using Send() (*calls*) to MATLAB process ID=2 on the second non-interactive processor so that MATLAB process ID=2 further doubles the received variable “a” and sends it to the next MATLAB process (*for execution*), rendering obvious *said first kernel configured to translate user commands into calls for execution*.

201. Third, Menon and Trefethen teach that MATLAB processes, and as such MATLABs in the MATLAB processes, perform mathematical evaluations of matrices and their elements. Ex.1005, 10-13; Ex.1006, 4, 5, 7. Menon discloses “high-level MATLAB operations” such as “matrix multiplications [and] solve for eigenvalues,” and provides example “MultiMATLAB implementations of the conjugate gradients algorithm” that “consists of a matrix-vector multiplication at every iteration.” Ex.1005, 11. 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” and “communication is required for two different operations.” 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).



Ex.1005, FIG. 6 (annotated)

202. Trefethen also provides examples of MATLAB processes performing mathematical evaluations of matrices and their elements such as condition numbers of Hilbert matrices and eigenvalue computations. Ex.1006, 4-5. For example, Trefethen describes the computation of the user command ‘`cond(hilb(ID))`’ at MATLAB process IDs=4 and 5 to return the results “`ans = 1.5514e+04` [and] `ans = 4.7661e+05`, the condition numbers of the Hilbert matrices of dimensions 4 and 5.” Ex.1006, 4. As another example, Trefethen describes the computation of the user command ‘`max(abs(eig(randn(400))))`’ by the MATLAB processes “on all processors” “to find the spectral radii (maximum of the absolute values of the

eigenvalues) of six matrices of dimension 400,” with the results being “six copies of the number 20.8508, since the random number generators deliver identical results on all processors.” Ex.1006, 4-5.

203. A POSITA would then find it obvious for Trefethen’s m-file, cycle.m, to include user commands for mathematical evaluations of matrices such as multiplying matrices, computing condition numbers of Hilbert matrices, computing eigenvalues of matrices, etc., and for MATLAB in each MATLAB process to execute matrix-related mathematical evaluations (*matrix operations on*) of matrices (*an array of data elements*) when receiving and executing the m-file cycle.m.

204. Accordingly, Menon teaching MATLAB in the MATLAB process of the first non-interactive processor translates user commands into matrix-related mathematical evaluations of matrices and an array of data elements, in view of Trefethen teaching an m-file where that MATLAB of the MATLAB process on the first non-interactive processor executes an m-file for performing matrix-related mathematical evaluations of matrices, renders obvious *said first kernel configured to translate user commands into calls for execution of matrix operations on an array of data elements*.

h. [1.4.1] a first cluster node module residing in the at least one computer-readable medium,

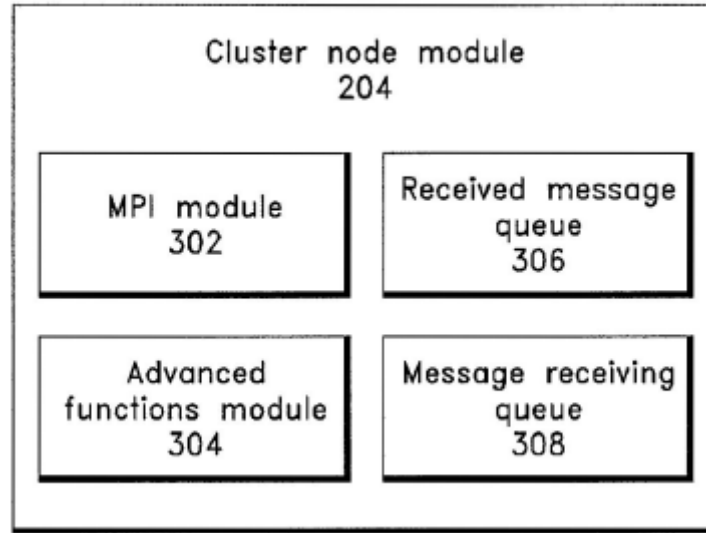
205. Menon and RS6000 render obvious *a first cluster node module residing in the at least one computer-readable medium* because Menon and RS6000

disclose MEX routines, MultiMATLAB interface module, indirection table, and communication layer in the MATLAB process address space of the MATLAB process on the first non-interactive processor of the MultiMATLAB architecture.

206. The '582 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, 11:46-58. Fig. 3 of the '582 patent depicts a single cluster node module comprising multiple sub-components:



Ex.1001, FIG. 3

207. FIG. 2 of the '582 patent depicting the cluster node modules 204a-e is shown below:

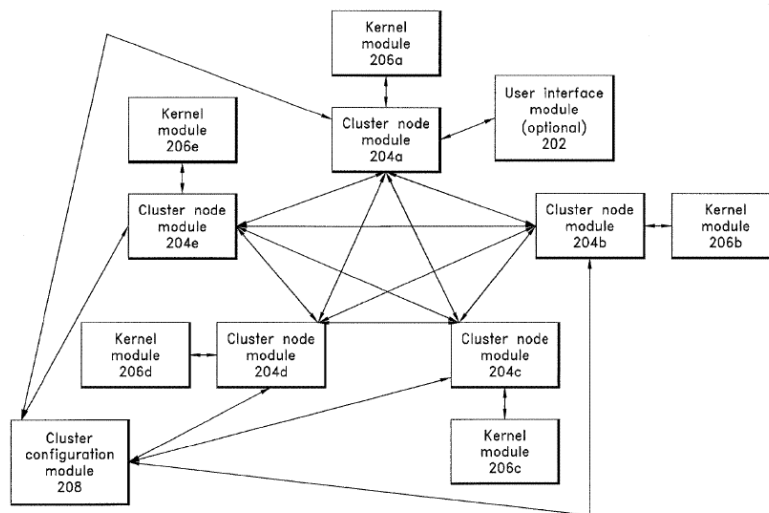


FIG. 2

Ex.1001, FIG. 2

208. Similarly, Menon discloses that each processor of “the MultiMATLAB architecture ... individually runs a MATLAB process” that includes “MEX routines,” a “MultiMATLAB interface module” (a “key component in [the MultiMATLAB] architecture”), a “communication layer,” and an “indirection table” via which “[t]he MultiMATLAB interface module provides MEX routines access to the underlying 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. In operation, the MEX routines (that implement calls such as Send and Recv) are run directly from the user interface. Ex.1005, 3; *see also* Ex.1006, 3, 6. The MEX routines 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 MATLAB processes. Ex.1005, 3. Menon 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.” Ex.1005, 3.

209. A POSITA would recognize that the MEX routines, MultiMATLAB interface module, indirection table, and communication layer of a MATLAB process on a processor constitute a set of components of that processor that handle communications with other processors. Further, a POSITA would recognize that the MEX routines, MultiMATLAB interface module, indirection table, and communication layer correspond to a *cluster node module* of the '582 patent because the '582 patent describes its cluster node module as a grouping of subcomponents, as shown in its Fig. 3 depicted above.

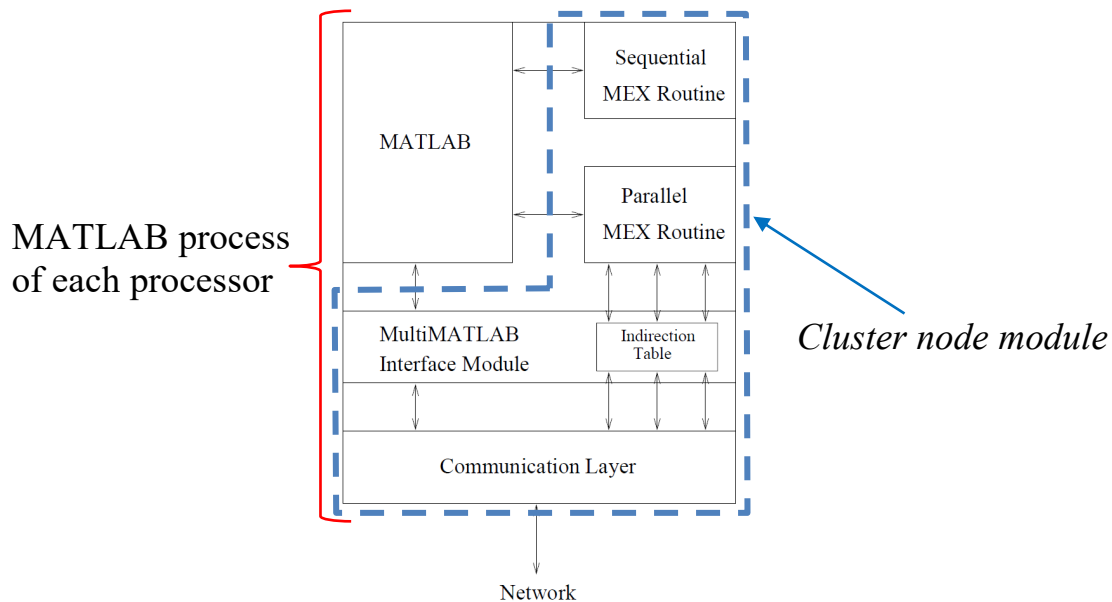


Figure 2: MATLAB Process Address Space

Ex.1005, FIG. 2 (annotated)

210. In limitation [1.2], it was shown that each MATLAB process of a processor is in a MATLAB process address space (*at least one computer-readable medium*). Therefore, the MEX routines, MultiMATLAB interface module,

indirection table, and communication layer (*cluster node module*) of each MATLAB process on a processor of the MultiMATLAB architecture are in (*residing in*) the MATLAB process address space (*the at least one computer-readable medium*) of that MATLAB process. Accordingly, the MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*a first cluster node module*) of the MATLAB process of the first non-interactive processor are in (*residing in*) the MATLAB process address space (*the at least one computer-readable medium*) of the MATLAB process on the first non-interactive processor.

- i. **[1.4.2] *said first cluster node module configured to communicate with the first kernel and receive commands from a user interface;***

211. Menon and Trefethen render obvious *said first cluster node module configured to communicate with the first kernel and receive commands from a user interface* because Menon and Trefethen teach that MEX routines in a MATLAB process on the first non-interactive processor of the MultiMATLAB architecture receive MATLAB commands from a user interface such as a PDE toolkit graphical interface on the interactive processor and send the MATLAB commands to MATLAB in the MATLAB process on the first non-interactive processor for execution by that MATLAB.

212. First, it was shown in limitation [1.4.1] that a POSITA would recognize and understand the MEX routines, MultiMATLAB interface module, indirection table, and communication layer of the first non-interactive processor (*first processor*) teach the *said first cluster node module*. Further, it was shown in limitation [1.3.1] that MATLAB in the MATLAB process on the first non-interactive processor (*first processor*) teaches *the first kernel*.

213. Second, Menon discloses that a “user interacts directly with one MATLAB process, called the interactive process, and operates within that process’s MATLAB environment” to have user instructions executed in parallel by the MultiMATLAB architecture. Ex.1005, 3. Specifically,

the MultiMATLAB system provides users with 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 over the underlying communication layer 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, 3-4.

214. Menon discloses an example user interface, 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. That is, MATLAB’s partial differential equation (PDE) toolkit has a
graphical interface that allow users to directly interact with a MATLAB process.

Ex.1005, 6.

215. Trefethen provides an example illustration where a user directly
interacts with a MATLAB process and utilizes “[t]he standard MultiMATLAB
command for executing commands on one or more processors[,]Eval.” Ex.1006, 3.
“[T]he user now types Eval(‘sqrt(2)’)” in an interactive MATLAB process at “a
node of the IBM SP2” implementing a MultiMATLAB architecture having “6
MultiMATLAB processes running.” Ex.1006, 3. “[T]hen the MATLAB command
sqrt(2) is executed on all six processors. The result is six repetitions of 1.4142.”

Ex.1006, 3. Because “sqrt(2)” is a “MATLAB command,” a POSITA would

recognize and understand that it is MATLAB in the MATLAB process on each processor that executes the MATLAB command `sqrt(2)`. Trefethen also provides an example where the Eval command receives the processor IDs “[4 5],” and a MATLAB command. The interactive processor sends the MATLAB command to the non-interactive processors (i.e., processor IDs = 4 and 5):

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

might return

```
ans = 1.5514e+04
```

```
ans = 4.7661e+05
```

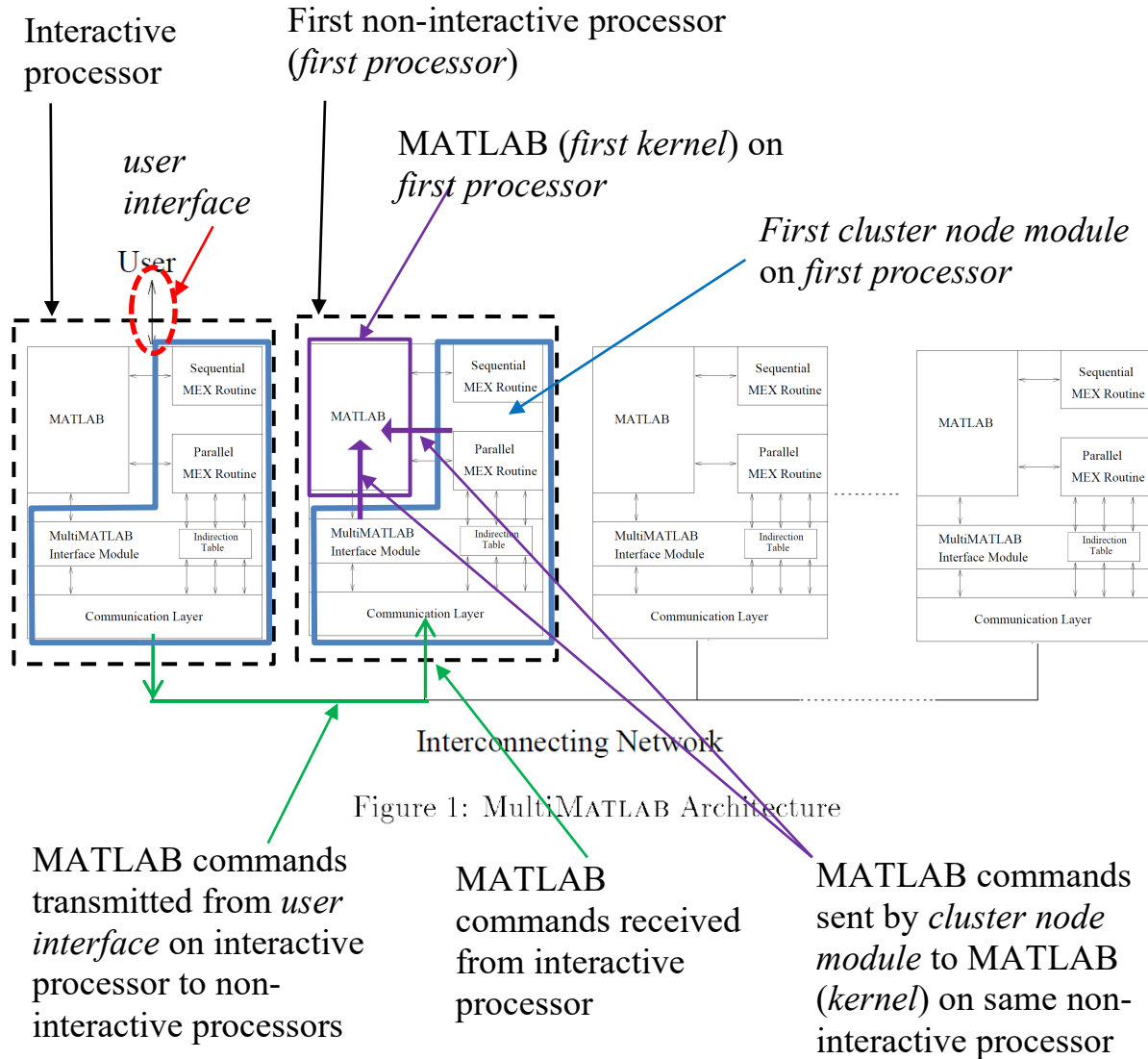
the condition numbers of the Hilbert matrices of
dimensions 4 and 5.

Ex.1006, 4.

216. In other words, Menon and Trefethen teach that when a user, interacting with a MATLAB process on the interactive processor via a user interface, the “MEX routine [Eval] accepts ... [the] MATLAB command [and] sends the command[s]” to the MATLAB process on each non-interactive processor of the MultiMATLAB architecture. Ex.1005, 3-4, 6; Ex.1006, 3. The MEX routines are “run directly from MATLAB.” Ex.1005, 3. The MATLAB command arrives at each MATLAB process’s communication layer (part of the *cluster node module*) because “[e]ach 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. Then, "[o]n non-interactive processes, a separate top-level MEX routine" (part of the *cluster node module*) of each MATLAB process receives the MATLAB command for execution "in [the separate top-level MEX routine's] own MATLAB environment." Ex.1005, 4.

217. Because Trefethen teaches that "sqrt(2)" or "cond(hilb(ID))" are each an example of a "MATLAB command" that is executed by the MATLAB process on each processor, a POSITA would recognize and understand that "the MATLAB command" is executed by MATLAB in the MATLAB process on each processor and that the MEX routines (part of each *cluster node module*) in a MATLAB process on each non-interactive processor sends the MATLAB command to MATLAB (*kernel*) on that MATLAB process for execution by that MATLAB, as shown in the combined figure of Figures 1 and 2 of Menon below. Ex.1006, 3.



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

218. That is, (i) the user enters a command via the user interface on the interactive processor, (ii) that command is provided “directly” to the MEX routine of the cluster node module on the interactive processor, (iii) the cluster node module on the interactive processor communicates the command to the cluster node modules on the first processor, second processor and third processor, (iv) the cluster node module of the first processor receives the command via the interconnecting

network, and (v) the cluster node module of the first processor communicates the command to the first kernel.

219. Accordingly, Menon and Trefethen teach MEX routines in a MATLAB process on the first non-interactive processor of the MultiMATLAB architecture receive MATLAB commands from a user interface such as a PDE toolkit graphical interface on the interactive processor and send the MATLAB commands to MATLAB in the MATLAB process on the same first non-interactive processor for execution by that MATLAB, rendering obvious *said first cluster node module configured to communicate with the first kernel and receive commands from a user interface.*

j. [1.5.1] *a second kernel residing in the at least one computer-readable medium,*

220. Menon and RS6000 render obvious *a second kernel residing in the at least one computer-readable medium* because Menon and RS6000 disclose MATLAB in a MATLAB process of the second non-interactive processor of the MultiMATLAB architecture, where the MATLAB process is contained in a MATLAB process address space.

221. In limitation [1.3.1], it was shown that MATLAB of a MATLAB process on a processor teaches *a kernel*. Accordingly, MATLAB of the MATLAB process on the second non-interactive processor teaches *a second kernel*.

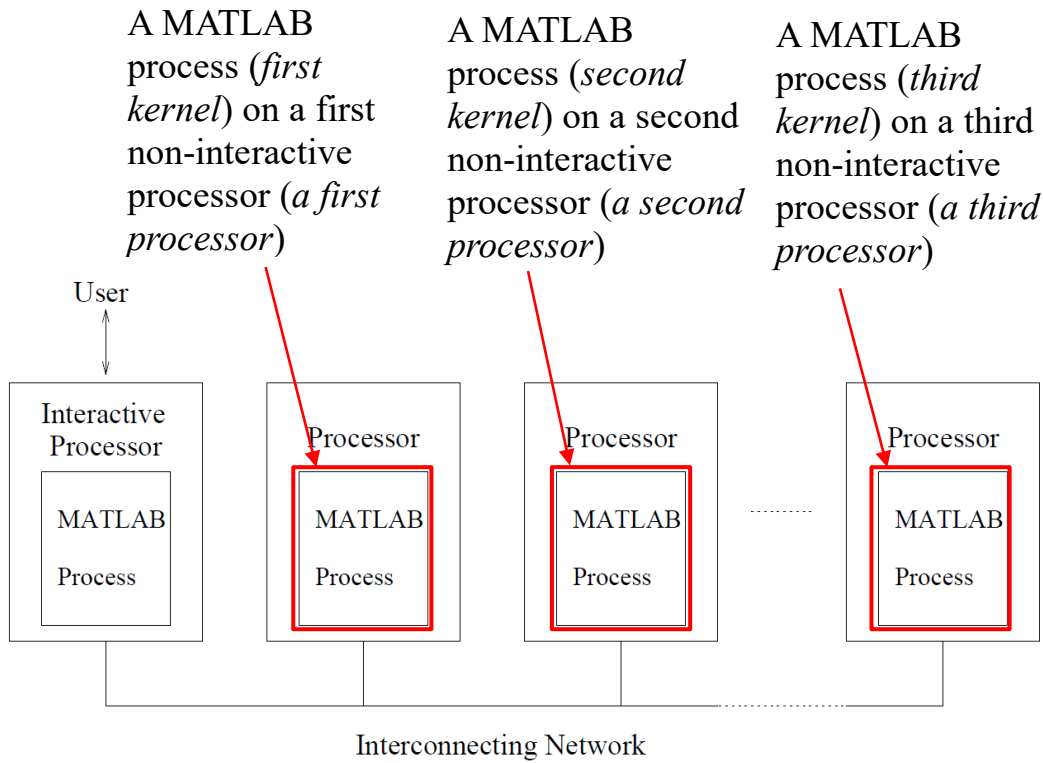
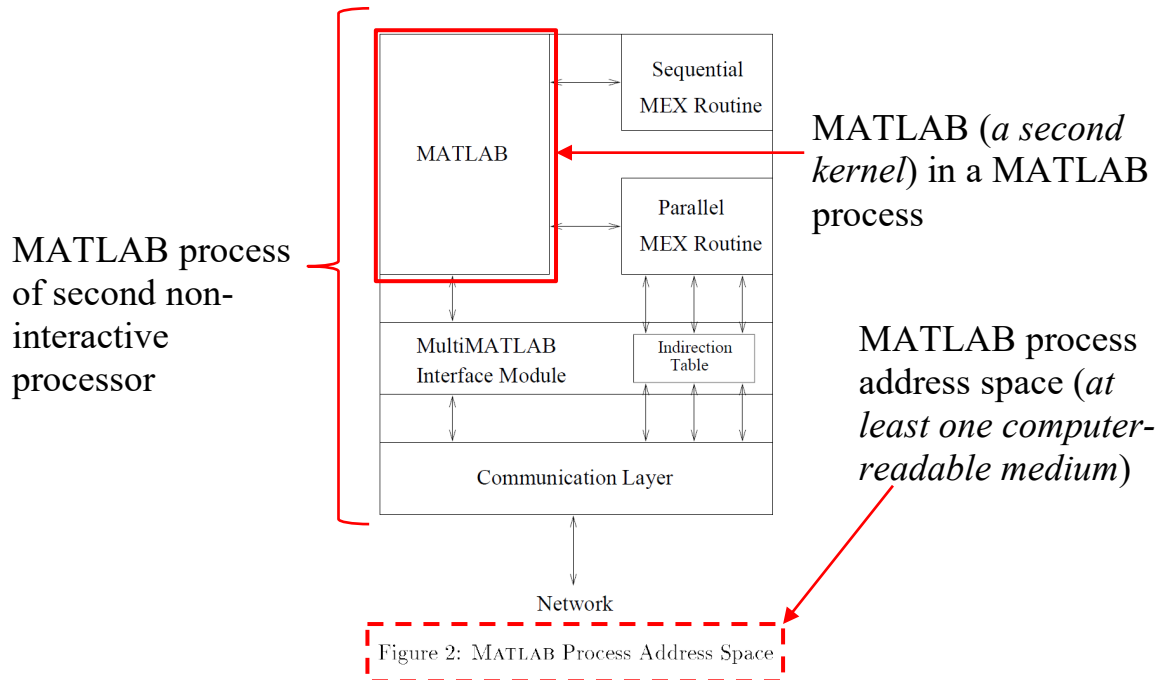


Figure 1: MultiMATLAB Architecture

Ex.1005, FIG. 1 (annotated)



Ex.1005, FIG. 2 (annotated)

222. Further, in limitation [1.2], it was shown that the MATLAB process of each processor of Menon's MultiMATLAB architecture is in a MATLAB process address space (*computer-readable medium*). Accordingly, Menon teaches a MATLAB (*a second kernel*) in the MATLAB process on the second non-interactive processor (*second processor*), where the MATLAB process is contained in (*residing in*) a MATLAB process address space (*the at least one computer-readable medium*).

- k. [1.5.2] *said second kernel configured to execute the matrix operations on the array of data elements and return a first result using the second processor;*

223. Menon and Trefethen render obvious *said second kernel configured to execute the matrix operations on the array of data elements and return a first result*

using the second processor because Menon teaches MATLAB in the MATLAB process of the second non-interactive processor, in view of Trefethen teaching an m-file where that MATLAB of the MATLAB process on the second non-interactive processor receives the variable “a=2,” doubles it to “a=4,” outputs the result “a=4,” and then sends it to the next MATLAB process, in further view of Menon and Trefethen teaching that each MATLAB process performs matrix-related mathematical evaluations of matrices.

224. First, it was shown in limitation [1.1.2] that the second non-interactive processor teaches *the second processor*. Further, it was shown in limitation [1.5.1] that MATLAB in the MATLAB process on the second non-interactive processor teaches *said second kernel*.

225. Second, as discussed in limitation [1.3.2], Trefethen teaches that MATLAB in MATLAB process ID=1 on the first non-interactive processor receives the m-file cycle.m and executes the m-file to send the variable “a” (which MATLAB process ID=1 doubles to “a=2”) to MATLAB process ID=2 on the second non-interactive processor so that MATLAB in MATLAB process ID=2 further doubles the received variable “a” (to “a=4”) and sends it to the next MATLAB process, i.e., MATLAB process ID=3. Trefethen also discloses that MATLAB process ID=2 produces an output, as shown below:

```
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;
```

← user command

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 in MATLAB process ID=2 receives variable “a=2” from MATLAB process ID=1, doubles it, outputs the result “a=4,” and then sends it to the next MATLAB process ID=3

Result (*a first result*) output or returned from MATLAB process ID=2 on second non-interactive processor

Ex.1006, 6 (annotated).

226. Because MATLAB is “MATLAB software,” a POSITA would recognize that MATLAB executes the m-file (e.g., including the user commands therein instructing the doubling of the variable “a”) using a processor. Ex.1006, 2; Ex.1005, 3. For example, MATLAB (*said second kernel*) in the MATLAB process ID=2 on the second non-interactive processor executes (*configured to execute*) the m-file `cycle.m` and outputs the result “a=4” (*return a first result*) using the second non-interactive processor (*using the second processor*). Ex.1005, 3.

227. Third, it was also shown in limitation [1.3.2] that a POSITA would find it obvious for Trefethen’s m-file, `cycle.m`, to include user commands for mathematical evaluations of matrices such as multiplying matrices, computing condition numbers of Hilbert matrices, computing eigenvalues of matrices, etc., and

for MATLAB in each MATLAB process to execute matrix-related mathematical evaluations (*the matrix operations on*) of matrices (*an the array of data elements*) when receiving and executing the m-file cycle.m.

228. Accordingly, Menon teaching MATLAB in the MATLAB process of the second non-interactive processor, in view of Trefethen teaching an m-file where that MATLAB in the MATLAB process on the second non-interactive processor receives the variable “a=2,” doubles it to “a=4,” outputs the result “a=4,” and then sends it to the next MATLAB process, in further view of Menon and Trefethen teaching that each MATLAB process performs matrix-related mathematical evaluations on the array of data elements, renders obvious *said second kernel configured to execute the matrix operations on the array of data elements and return a first result using the second processor.*

I. [1.6.1] a second cluster node module residing in the at least one computer-readable medium,

229. Menon and RS6000 render obvious *a second cluster node module residing in the at least one computer-readable medium* because Menon and RS6000 disclose MEX routines, MultiMATLAB interface module, indirection table, and communication layer in the MATLAB process address space of the MATLAB process on the second non-interactive processor of the MultiMATLAB architecture.

230. It was shown in limitation [1.4.1] that the MEX routines, MultiMATLAB interface module, indirection table, and communication layer

(*cluster node module*) of each MATLAB process on a processor of the MultiMATLAB architecture are in the MATLAB process address space of that MATLAB process. Accordingly, the MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*a second cluster node module*) of the MATLAB process on the second non-interactive processor are in (*residing in*) the MATLAB process address space (*the at least one computer-readable medium*) of that MATLAB process.

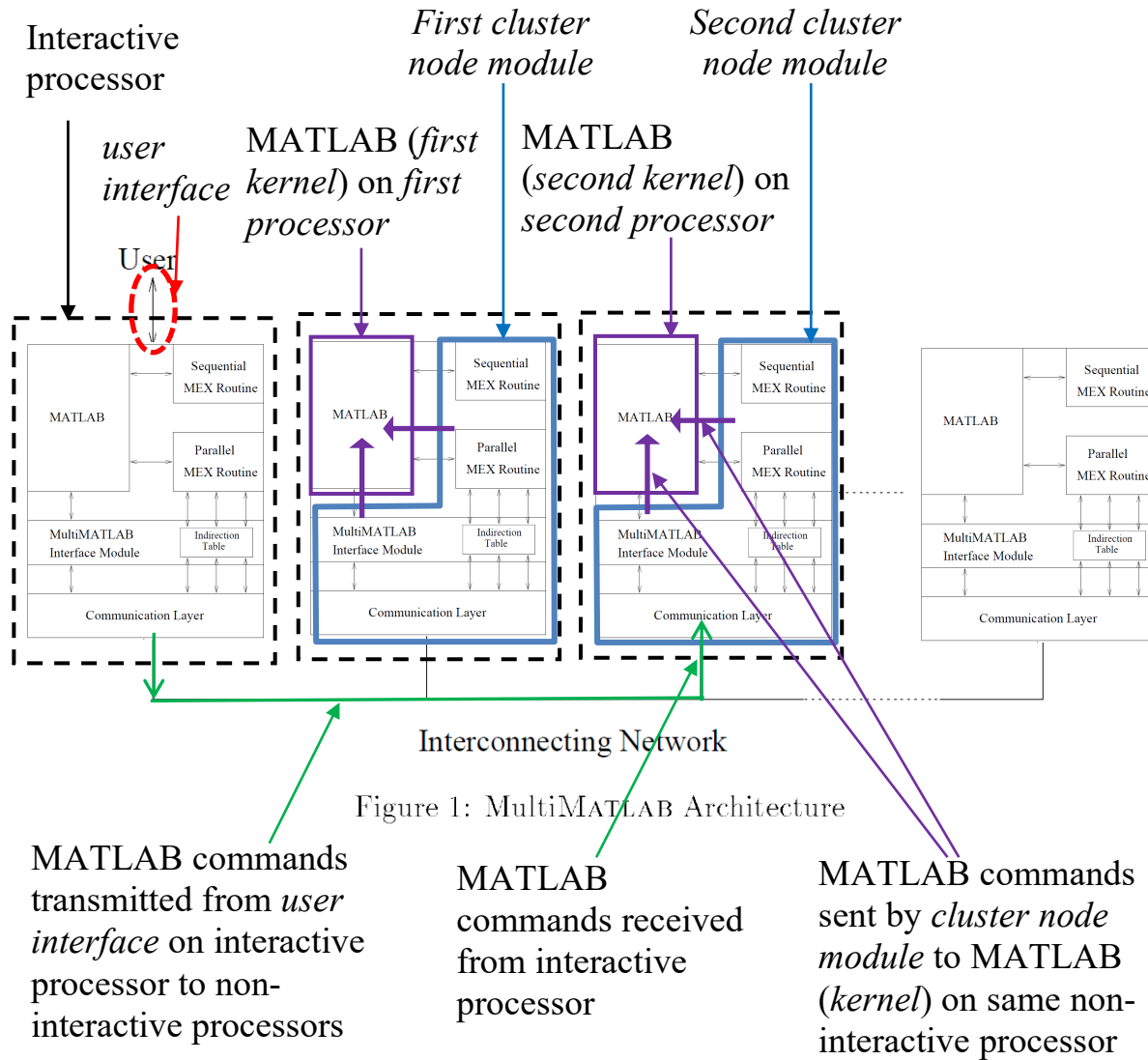
m. [1.6.2] *said second cluster node module configured to send commands to the second kernel*

231. Menon and Trefethen render obvious *said second cluster node module configured to send commands to the second kernel* because Menon and Trefethen teach that the MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*said second cluster node module*) receive a MATLAB command from a user interface on the interactive processor and send the MATLAB command to the MATLAB in that MATLAB process for execution by that MATLAB.

232. First, it was shown in limitation [1.6.1] that a POSITA would recognize and understand that the MEX routines, MultiMATLAB interface module, indirection table, and communication layer of the second non-interactive processor (*second processor*) teach *said second cluster node module*. Further, it was shown in

limitation [1.5.1] that MATLAB in the MATLAB process on the second non-interactive processor (*second processor*) teaches *the second kernel*.

233. Second, it was shown in limitation [1.4.2] that the MEX routines (part of the *cluster node module*) in the MATLAB process of each non-interactive processor receive a MATLAB command from the PDE toolkit graphical interface via the communication layer (part of the *cluster node module*) of the MATLAB process and send the MATLAB command to the MATLAB (*kernel*) in that MATLAB process for execution by that MATLAB. Accordingly, the MEX routines (part of *said second cluster node module*) in the MATLAB process of the second non-interactive processor receive a MATLAB command from a user interface on the interactive processor and send the MATLAB command (*configured to send commands*) to the MATLAB (*to the second kernel*) in that MATLAB process for execution by that MATLAB, as shown in the combined figure of Figures 1 and 2 of Menon below.



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

n. [1.6.3] *[said second cluster node module configured to ...] communicate with the first cluster node module;*

234. Menon and Trefethen render obvious *[said second cluster node module configured to ...] communicate with the first cluster node module* because Menon and Trefethen teach that the MATLAB processes on the second non-interactive processor and the first non-interactive processor use their respective MEX routines,

MultiMATLAB interface module, indirection table, and communication layer (*said second cluster node module* and *the first cluster node module*) to communicate mathematical evaluation results, commands (e.g., tasks), and data with each other over the MultiMATLAB architecture's interconnection network.

235. First, it was shown in limitations [1.4.1] and [1.6.1] that a POSITA would recognize and understand that the MEX routines, MultiMATLAB interface module, indirection table, and communication layer of the first non-interactive processor (*first processor*) and the second non-interactive processor (*second processor*) teach *the first cluster node module* and *said second cluster node module*.

236. Second, it was shown in limitation [1.4.1] that the MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*cluster node module*) of a MATLAB process on a processor constitute a set of components of that processor that handle communications with other processors. Specifically, the “MEX routines” provide “[a]ll parallel functionality in the MultiMATLAB system” and that “[e]ach 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. Further, “[a]ll parallel MEX routines access the underlying communication layer and, hence, the network, through the MultiMATLAB interface module.” Ex.1005, 4. “The MultiMATLAB interface module provides MEX routines access to the underlying communication

layer via an indirection table.” Ex.1005, 5. An example of the communication layer is MPI. Ex.1005, 3. The MultiMATLAB architecture’s interconnection network and the MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*cluster node module*) of a MATLAB process on a processor are depicted in Figure 1 and Figure 2 of Menon shown below.

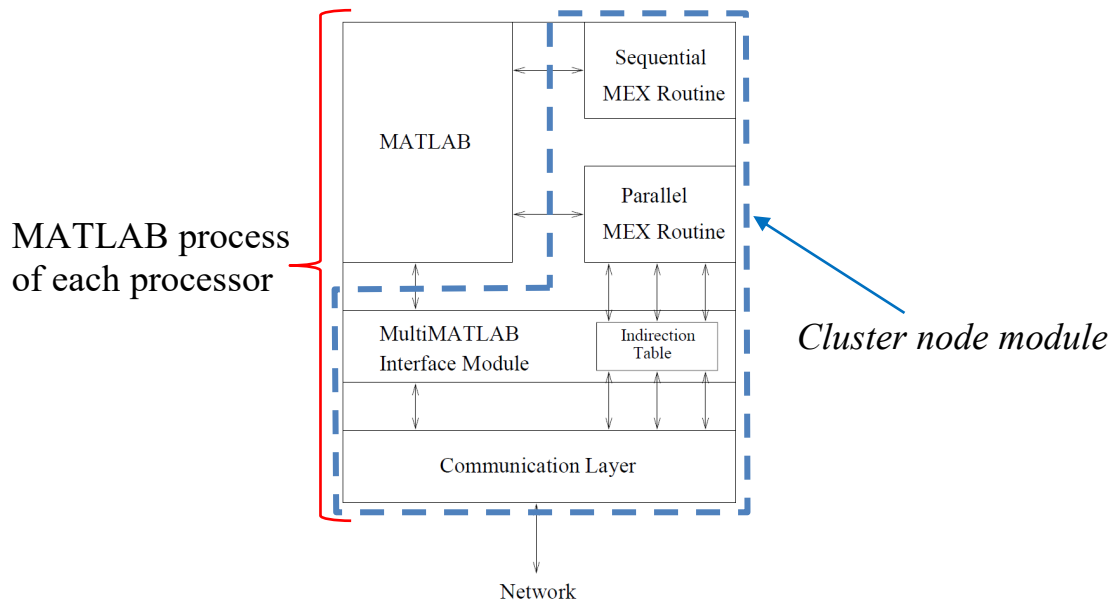


Figure 2: MATLAB Process Address Space

Ex.1005, FIG. 2 (annotated)

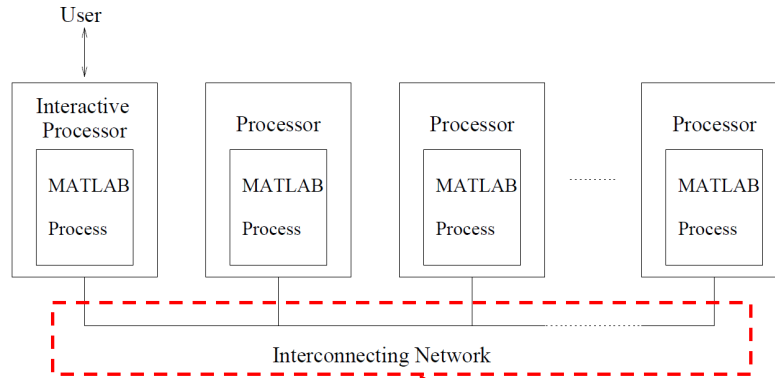


Figure 1: MultiMATLAB Architecture

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

Ex.1005, FIG. 1 (annotated)

237. “MEX routines” provide “[a]ll parallel functionality in the MultiMATLAB system” because communication commands used in MultiMATLAB are “written as [] C file[s]” and “interfaced to MATLAB via the standard MATLAB Fortran/C/C++ interface known as MEX.” Ex.1006, 10. For example, the “command [Send] is written as a C file called Send.c, which is interfaced to MATLAB via” MEX. Ex.1006, 10. “More 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. Trefethen teaches that “MultiMATLAB is [] built upon the standard send and receive variants” of MPI. Ex.1006, 10. “Higher-level MultiMATLAB commands are [] built on higher-level MPI commands,” such as the broadcast command “Bcast ... built on MPI collective communication routines.” Ex.1006, 10.

238. Menon also discloses that MEX routines are implemented using “point to point ... communication” paradigm to enable communication between the MATLAB processes operating in parallel in the MultiMATLAB architecture. Ex.1005, 6-9. 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. The “SPMD paradigm,” which “permit[s] finer grain communication of MATLAB data structures between any [MATLAB] processes,” “is facilitated by MEX routines that expose some functionality of the underlying communication layer [i.e., MPI] to the MATLAB language.” Ex.1005, 8, 3 (“MPI”).

239. 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 that are implemented as MEX routines, such as the “Eval(pid, ‘command’)” command that “[e]valuates a command on one or more MATLAB process[es],” 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, 8, Table 1.

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.

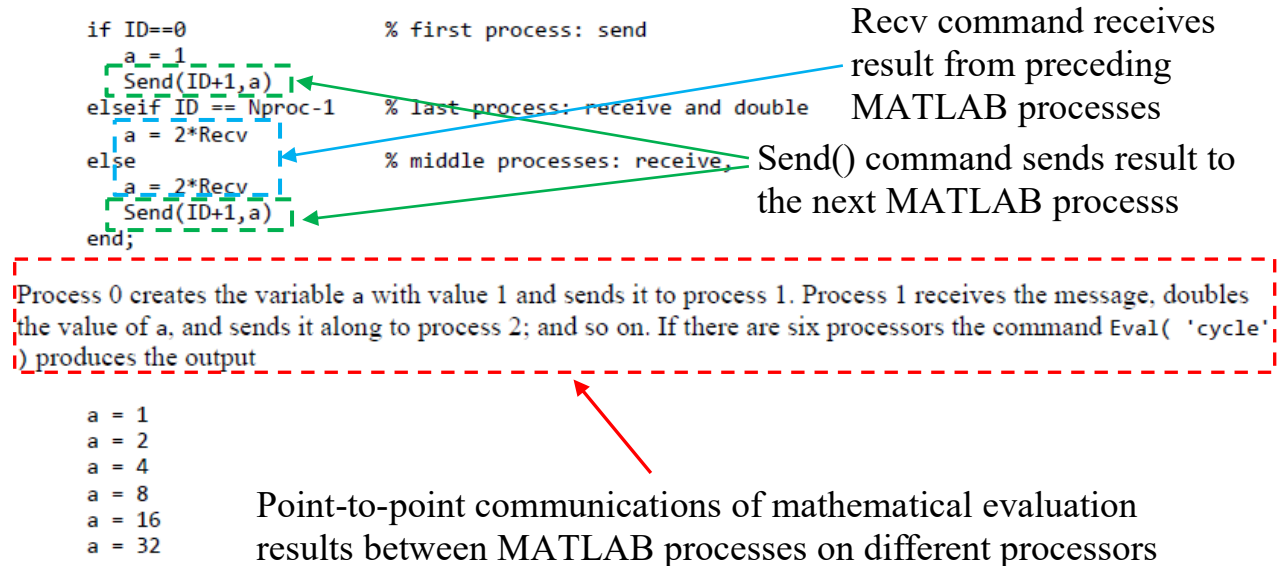
Table 1: Selected commands in the MultiMATLAB system

Commands for point to point communication

Ex.1005, Table 1 (annotated)

240. Trefethen provides an example where MATLAB processes on processors of the MultiMATLAB architecture use the MEX routines Send() and Recv(), implemented as SPMD point-to-point commands, to communicate results of mathematical expression evaluation to other MATLAB processes on other processors. Ex.1006, 6; Ex.1005, 9. 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, using the MEX routines Send() and Recv():



Ex.1006, 6 (annotated). In particular, the above example code shows MATLAB process ID=2 using its MEX routines (part of *said second cluster node module*) to receive (*configured to communicate with*) a result that MATLAB process ID=1 sent using its own MEX routines (part of *the first cluster node module*).

241. Trefethen provides another example where a MATLAB process uses the MEX routine Bcast, implemented as an SPMD point-to-point command, to directly communicate with every other MATLAB process in the MultiMATLAB architecture. Ex.1006, 6-7. Trefethen teaches that “[t]he Bcast command is used to broadcast a matrix from one process to all other processes ... For example, Eval (‘Bcast(1,ID)’) returns the number 1 on all processes.” Ex.1006, 7; Ex.1005, 9.

242. Trefethen provides yet another example where the MEX routines Send() and Recv, implemented as SPMD point-to-point commands, are used to send and/or receive variables from one MATLAB process directly to another in the MultiMATLAB architecture. Ex.1006, 5-6. Trefethen describes a user, at a user interface on an interactive processor of the MultiMATLAB architecture, running “the [following] sequence,” which “passes a message containing a 2-vector from the [MATLAB process at the interactive processor] to process 3”:

```
“x = [pi pi^2];  
Send(3,x)  
Eval(3, ‘Recv’ ),”
```

“leading to the output

```
3.1416 9.8696.”
```

Ex.1006, 3, 5-6. That is, the MATLAB process at the interactive processor uses the MEX routine Send() to send the 2-vector [pi pi^2] to the MATLAB process at the third non-interactive processor, which in turn uses the MEX routine Recv to receive the 2-vector [pi pi^2] and then outputs 3.1416 9.8696 upon execution. Ex.1006, 5-6.

243. Menon discloses that each MATLAB process on a processor of the MultiMATLAB architecture has MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*cluster node module*), and Menon and Trefethen disclose that the MEX routines, implemented using standard MPI send,

receive, and broadcast calls, enable SPMD point-to-point communication so that each MATLAB process can communicate, i.e., send, receive, and broadcast, mathematical evaluation results, commands (e.g., tasks), and data directly with each other over the MultiMATLAB architecture's interconnection network the using their respective *cluster node module*, as depicted in the combination figure of Figures 1 and 2 of Menon below.

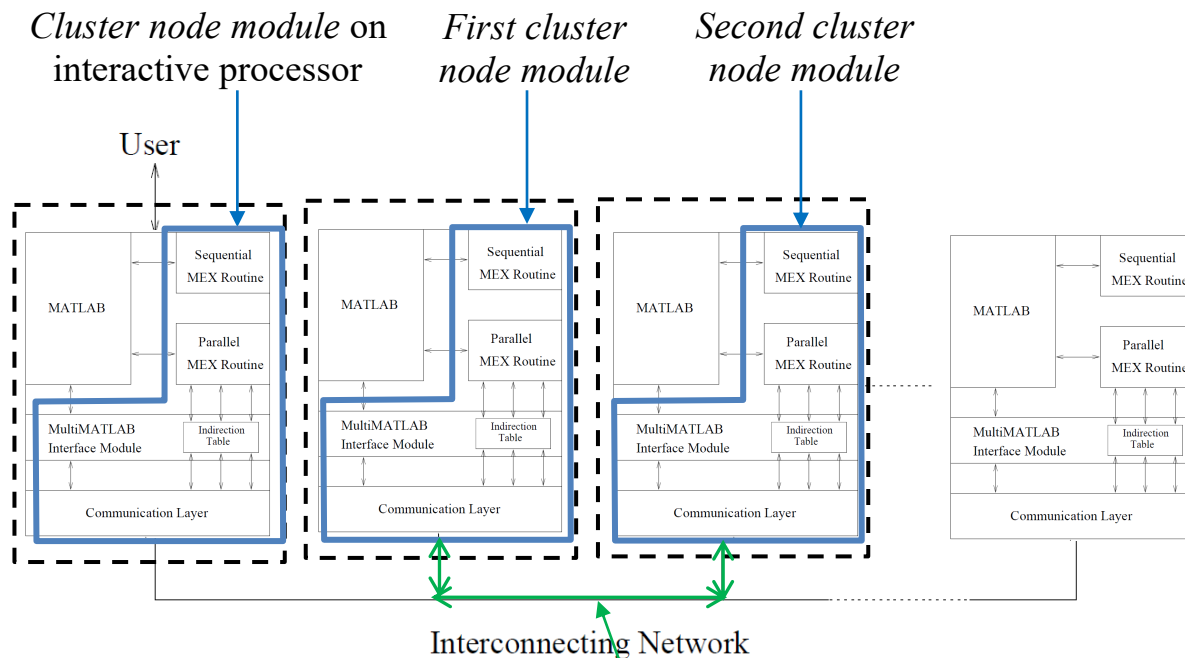


Figure 1: MultiMATLAB Architecture

Cluster node modules communicating to and from each other

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

Accordingly, Menon and Trefethen teach that the MATLAB processes on the second non-interactive processor and the first non-interactive processor use their

respective MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*said second cluster node module* and *the first cluster node module*) to communicate commands (e.g., tasks) and data directly with each other, rendering obvious [*said second cluster node module configured to ...*] *communicate with the first cluster node module*.

o. [1.7.1] *a third kernel residing in the at least one computer-readable medium,*

244. Menon and RS6000 render obvious *a third kernel residing in the at least one computer-readable medium* because Menon and RS6000 disclose MATLAB in a MATLAB process of the third non-interactive processor of the MultiMATLAB architecture, where the MATLAB process is contained in a MATLAB process address space.

245. In limitation [1.3.1], it was shown that MATLAB of a MATLAB process on a processor teaches *a kernel*. Accordingly, MATLAB of the MATLAB process on the third non-interactive processor teaches *a third kernel*. Further, in limitation [1.2], it was shown that the MATLAB process of each processor of Menon's MultiMATLAB architecture is in a MATLAB process address space (*computer-readable medium*).

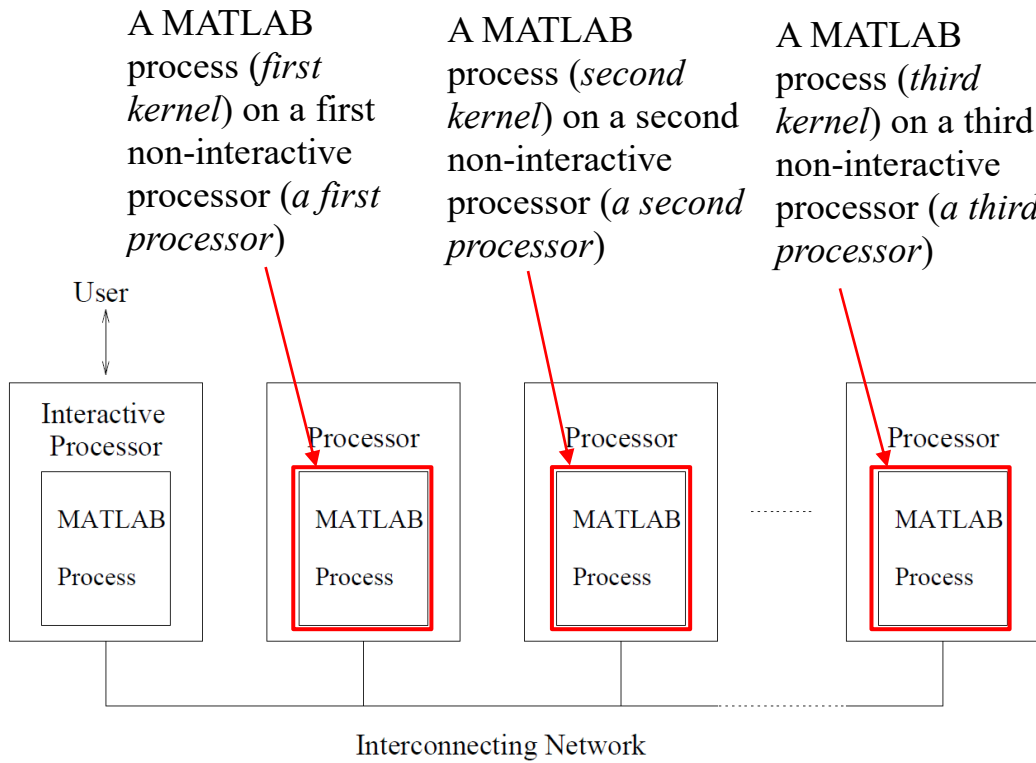
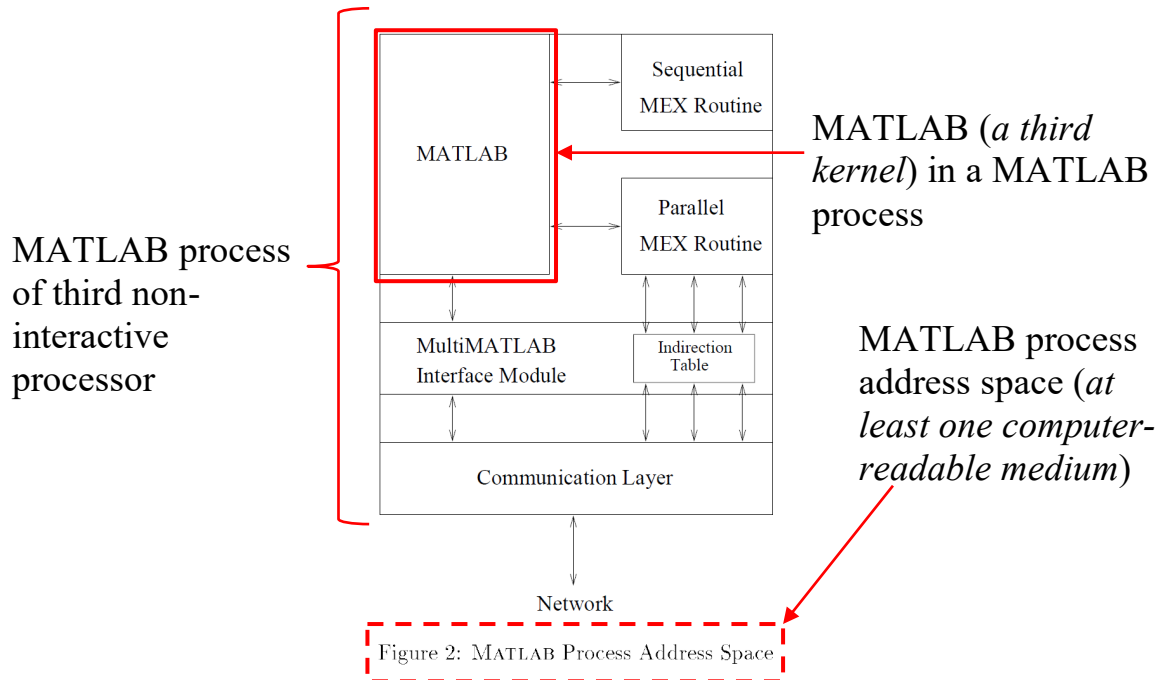


Figure 1: MultiMATLAB Architecture

Ex.1005, FIG. 1 (annotated)



Ex.1005, FIG. 2 (annotated)

246. Accordingly, Menon teaches a MATLAB (*a third kernel*) in the MATLAB process on the third non-interactive processor (*third processor*), where the MATLAB process is contained in (*residing in*) a MATLAB process address space (*the at least one computer-readable medium*).

- p. [1.7.2] *said third kernel configured to receive the first result and execute additional matrix operations on the first result and return a second result to one or more of the first, second, or third cluster node modules using the third processor; and*

247. Menon and Trefethen render obvious *said third kernel configured to receive the first result and execute additional matrix operations on the first result and return a second result to one or more of the first, second, or third cluster node modules using the third processor* because Menon teaches MATLAB in the

MATLAB process on the third non-interactive processor, in view of Trefethen teaching an m-file where that MATLAB of the MATLAB process on the third non-interactive processor receives the variable “a=4,” doubles it to “a=8,” and provides the result “a=8” to the *cluster node module* in the MATLAB process on the third non-interactive processor, in further view of Menon and Trefethen teaching that each MATLAB process performs matrix-related mathematical evaluations of matrices.

248. First, it was shown in limitation [1.1.2] that the third non-interactive processor teaches *the third processor*. Further, it was shown in limitation [1.5.1] that MATLAB in the MATLAB process on the third non-interactive processor teaches *said third kernel*.

249. Second, it was disclosed in limitation [1.5.2] that MATLAB process ID=3 on the third non-interactive processor receives the result “a=4” (*the first result*) from MATLAB process ID=2. Further, as shown in limitation [1.3.2], Trefethen teaches that MATLAB in MATLAB process ID=3 executes the program or m-file “cycle.m” to double the received variable to “a=8” (*a second result*) and sends it to the next MATLAB process, i.e., MATLAB process ID=4, as shown below:

```
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;
```

user command

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 process ID=3 receives variable “a=4” (*the first result*) from MATLAB process ID=2, doubles it, outputs the result “a=8” (*a second result*), and then sends it to the next MATLAB process ID=4

Result (*a second result*) output or returned from MATLAB process ID=3 on third non-interactive processor

Ex.1006, 6 (annotated).

250. Third, in limitation [1.6.3], it was shown that each MATLAB process communicates, i.e., sends, receives, and broadcasts results, directly with each other using their respective *cluster node module*; that is, for example, MATLAB process ID=3 on the third non-interactive processor sends the result “a=8” (*a second result*) to the next MATLAB process ID=4 using its *third cluster node module* (for the result to arrive at the next MATLAB process ID=4 via its own *cluster node module*). A POSITA would recognize and understand that for *the third cluster node module* in MATLAB process ID=3 on the third non-interactive processor to send the result “a=8” (*a second result*) to the next MATLAB process, *the third cluster node module* receives the result “a=8” (*a second result*) from MATLAB (*third kernel*) in MATLAB process ID=3 that generated it. In other words, a POSITA

would recognize and understand that MATLAB (*third kernel*) in MATLAB process ID=3 provides the result “a=8” (*a second result*) to the *third cluster node module* so that the latter forwards it to the next MATLAB process ID=4, plus the result is returned to the interactive processor.

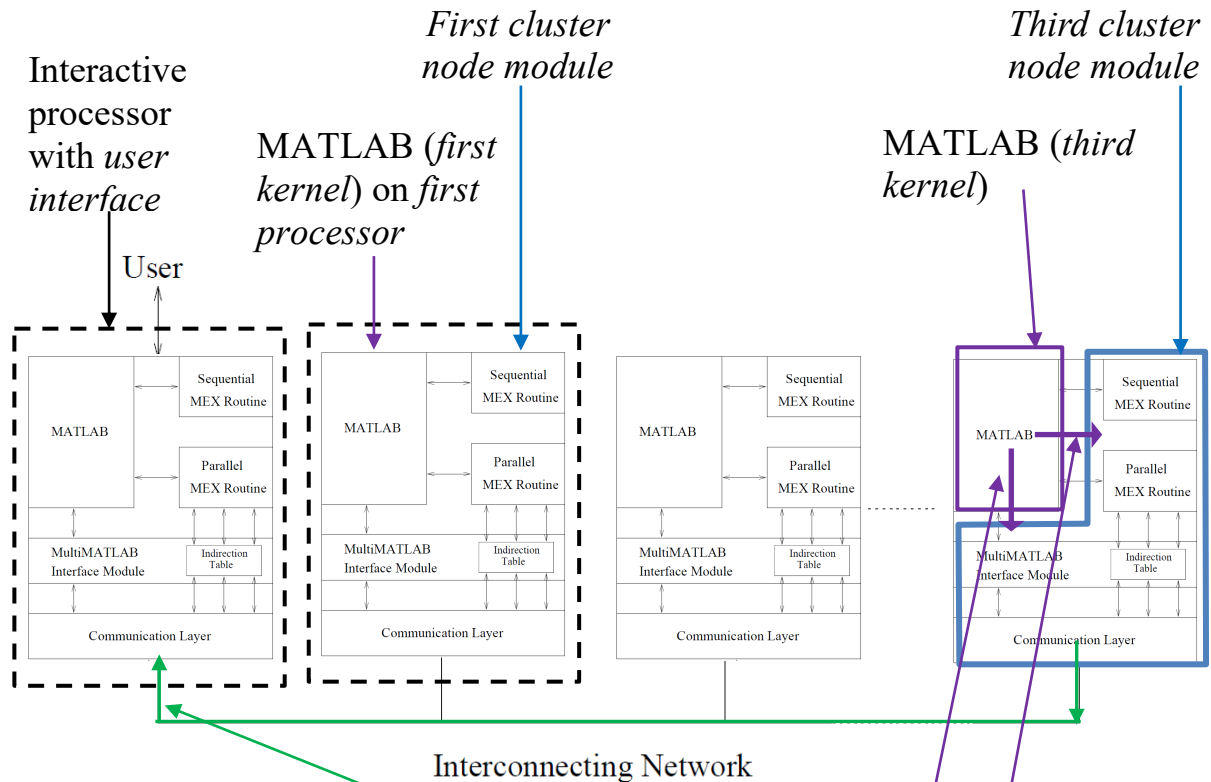


Figure 1. MultiMATLAB Architecture

MATLAB results
received from third
non-interactive
processor

MATLAB results sent to
cluster node module by
MATLAB (kernel) on
same non-interactive
processor

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

251. Fourth, because MATLAB is “MATLAB software,” a POSITA would recognize that MATLAB executes the m-file (e.g., including the user commands

therein instructing receiving the result from the preceding MATLAB process and doubling it) using a processor. Ex.1006, 2; Ex.1005, 3. That is, for example, MATLAB (*said third kernel*) in the MATLAB process ID=3 on the third non-interactive processor receives the result “a=4” (*configured to receive the first result*) from MATLAB process ID=2, doubles the received variable to “a=8” (*execute ... on the first result*), and provides the result “a=8” to (*return a second result to*) the *third cluster node module* in the MATLAB process ID=3 on the third non-interactive processor using the third non-interactive processor (*using the third processor*). Ex.1005, 3.

252. Fifth, it was also shown in limitation [1.3.2] that a POSITA would find it obvious for Trefethen’s m-file, cycle.m, to include user commands for mathematical evaluations of matrices such as multiplying matrices, computing condition numbers of Hilbert matrices, computing eigenvalues of matrices, etc., and for MATLAB in each MATLAB process to execute matrix-related mathematical evaluations of matrices when receiving and executing the m-file cycle.m. For example, Menon’s “parallel MATLAB implementation of conjugate gradients,” performs “matrix-vector multiplication” in “iteration” and “communication is required for two different operations.” Ex.1005, 11. That is, a POSITA would find it obvious for the result (*first result*) from MATLAB process ID=2 to be used by MATLAB (*said third kernel*) in the MATLAB process ID=3 to perform a next

iteration matrix-related mathematical evaluations such as computing its condition number or eigenvalue (*execute additional matrix operations*) to generate a matrix result (*second result*). For example, the conjugate gradients algorithm described in Menon has, at its “computational core” performing “matrix-vector multiplication” in “iteration” and “communication is required for two different operations.”

Ex.1005, 11.

253. Accordingly, Menon teaching MATLAB in the MATLAB process on the third non-interactive processor, in view of Trefethen teaching an m-file where that MATLAB of the MATLAB process on the third non-interactive processor receives tasks and data, performs mathematical computation and provides the result to the *cluster node module* in the MATLAB process on the third non-interactive processor for sending to another processor, in further view of Menon and Trefethen teaching that each MATLAB process performs matrix-related mathematical evaluations of matrices, renders obvious *said third kernel configured to receive the first result and execute additional matrix operations on the first result and return a second result to one or more of the first, second, or third cluster node modules using the third processor.*

q. [1.8.1] a third cluster node module residing in the at least one computer-readable medium,

254. Menon and RS6000 render obvious *a third cluster node module residing in the at least one computer-readable medium* because Menon and RS6000

disclose MEX routines, MultiMATLAB interface module, indirection table, and communication layer in the MATLAB process address space of the MATLAB process on the third non-interactive processor of the MultiMATLAB architecture.

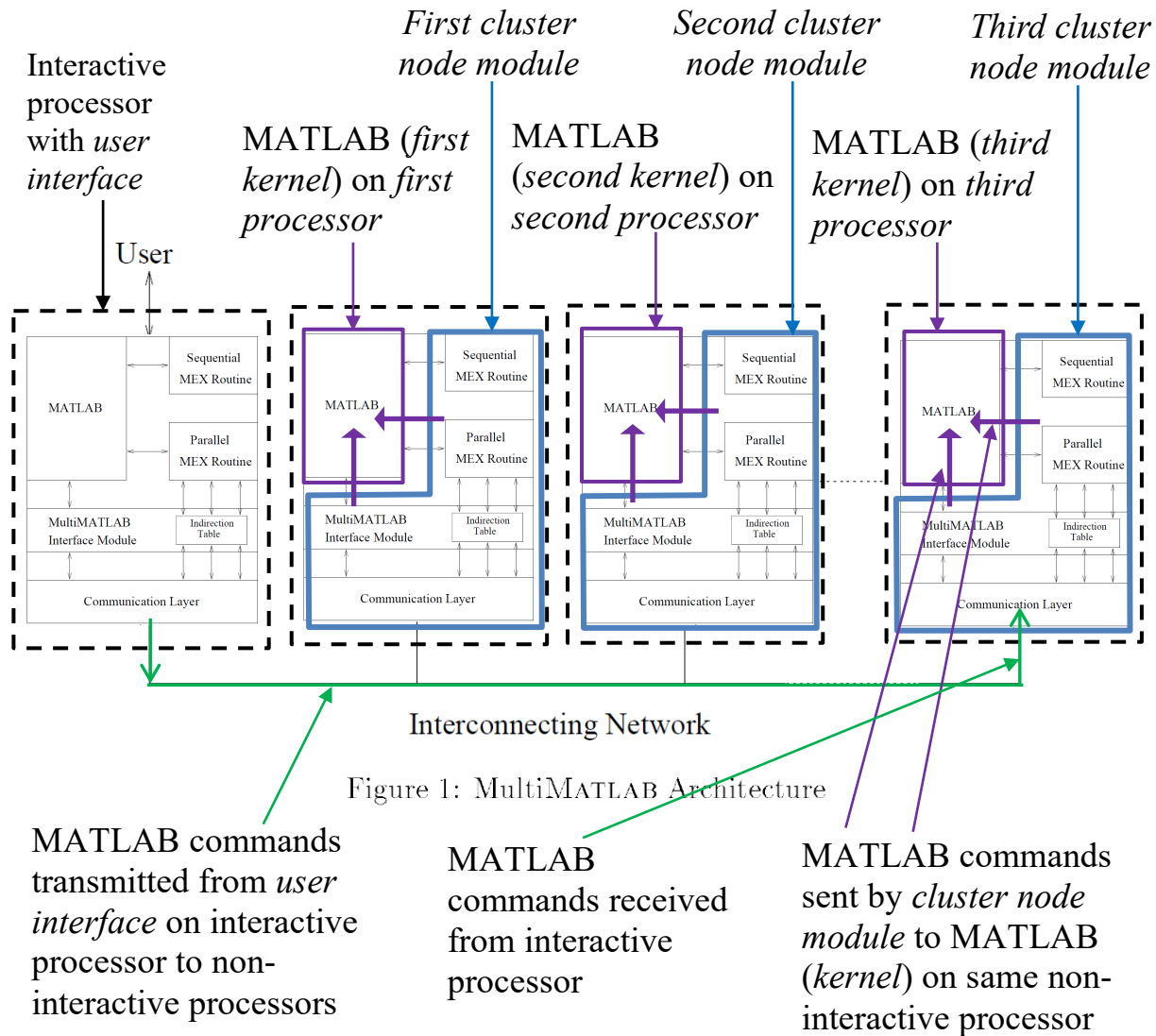
255. It was shown in limitation [1.4.1] that the MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*cluster node module*) of each MATLAB process on a processor of the MultiMATLAB architecture are in the MATLAB process address space of that MATLAB process. Accordingly, the MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*a third cluster node module*) of the MATLAB process on the third non-interactive processor are in (*residing in*) the MATLAB process address space (*the at least one computer-readable medium*) of that MATLAB process.

r. **[1.8.2]** *said third cluster node module configured to send commands to the third kernel and*

256. Menon and Trefethen render obvious *said third cluster node module configured to send commands to the third kernel* because Menon and Trefethen teach that the MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*said third cluster node module*) receive a MATLAB command from a user interface on the interactive processor and send the MATLAB command to the MATLAB in that MATLAB process for execution by that MATLAB.

257. First, it was shown in limitation [1.8.1] that a POSITA would recognize and understand that the MEX routines, MultiMATLAB interface module, indirection table, and communication layer of the third non-interactive processor (*third processor*) teach *said third cluster node module*. Further, it was shown in limitation [1.7.1] that MATLAB in the MATLAB process on the third non-interactive processor (*third processor*) teaches *the third kernel*.

258. Second, it was shown in limitation [1.4.2] that the MEX routines (part of the *cluster node module*) in the MATLAB process of each non-interactive processor receive a MATLAB command from the PDE toolkit graphical interface via the communication layer (part of the *cluster node module*) of the MATLAB process and send the MATLAB command to the MATLAB (*kernel*) in that MATLAB process for execution by that MATLAB. Accordingly, the MEX routines (part of *said third cluster node module*) in the MATLAB process of the third non-interactive processor receive a MATLAB command from a user interface on the interactive processor and send the MATLAB command (*configured to send commands*) to the MATLAB (*to the third kernel*) in that MATLAB process for execution by that MATLAB, as shown in the combined figure of Figures 1 and 2 of Menon below.



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

- s. [1.8.3] *[said third cluster node module configured to ...] communicate with the first cluster node module and the second cluster node module;*

259. Menon and Trefethen renders obvious *[said third cluster node module configured to ...] communicate with the first cluster node module and the second cluster node module* because Menon and Trefethen teach that the MATLAB processes on the third non-interactive processor, the second non-interactive

processor, and the first non-interactive processor use their respective MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*said third cluster node module, the first cluster node module and the second cluster node module*) to communicate mathematical evaluation results, commands (e.g., tasks), and data with each other over the MultiMATLAB architecture's interconnection network.

260. As shown in limitation [1.6.3], Menon discloses that each MATLAB process on a processor of the MultiMATLAB architecture has MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*cluster node module*), and Menon and Trefethen disclose that the MEX routines, implemented using standard MPI send, receive, and broadcast calls, enable SPMD point-to-point communication so that each MATLAB process can communicate, i.e., send, receive, and broadcast, mathematical evaluation results, commands (e.g., tasks), and data directly with each other over the MultiMATLAB architecture's interconnection network using their respective *cluster node module*, as depicted in the combination figure of Figures 1 and 2 of Menon below.

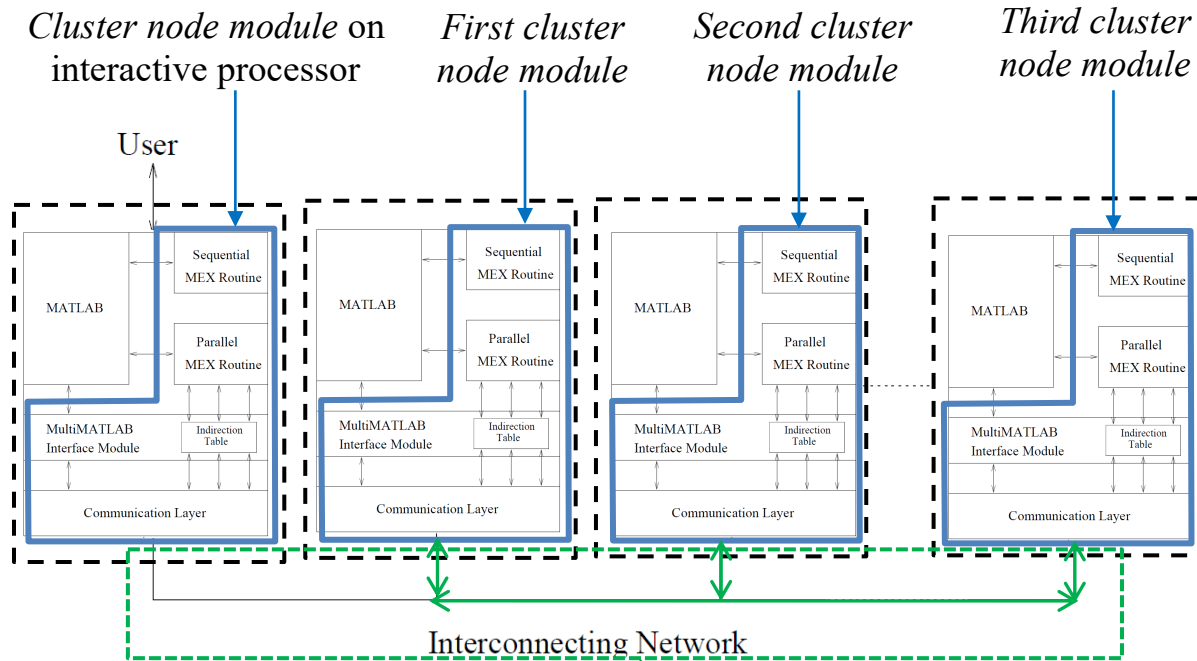


Figure 1: MultiMATLAB Architecture

Cluster node modules communicating to and from each other

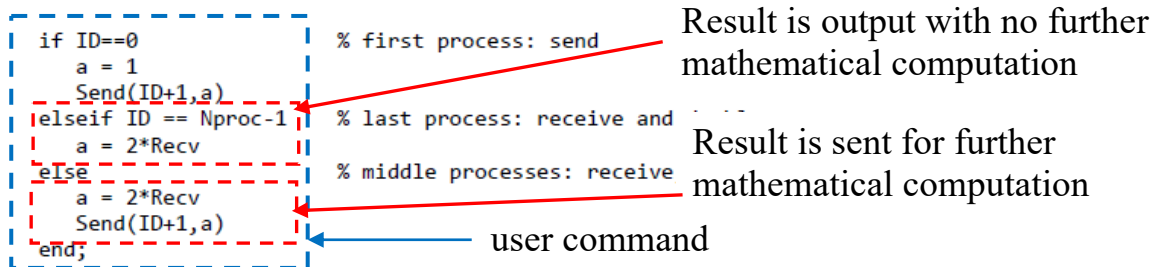
Ex.1005, FIGs. 1 and 2 (combined and annotated)

261. Accordingly, Menon and Trefethen teach that the MATLAB processes on the third non-interactive processor, the second non-interactive processor, and the first non-interactive processor use their respective MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*said third cluster node module, the first cluster node module and the second cluster node module*) to communicate commands (e.g., tasks) and data with each other, rendering obvious [*said third cluster node module configured to ...*] *communicate with the first cluster node module and the second cluster node module.*

t. [1.9] *wherein the first cluster node module is configured to return, based on the second result, a final result.*

262. Trefethen renders obvious *wherein the first cluster node module is configured to return, based on the second result, a final result* because Trefethen teaches that the MATLAB process on the first non-interactive processor uses its MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*first cluster node module*) to return an output (*is configured to return ... a final result*) that includes a result “a=8” (*based on the second result*) from the MATLAB process on the third non-interactive processor.

263. As discussed in limitation [1.7.2], Trefethen teaches that, when executing “the m-file cycle.m,” the final MATLAB process merely outputs the results to the screen of the interactive process.



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

Result from MATLAB process ID=1
Result (*a second result*) from MATLAB process ID=3

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

Ex.1006, 6 (annotated).

264. Trefethen's "cycle.m" example code is a simplistic mathematical algorithm showing that MultiMATLAB can (i) pass results from one process to another process and (ii) the result is used to perform subsequent mathematical computations. Trefethen describes MultiMATLAB as useful for running a "program for six different boundary conditions, or a dozen different parameter choices, or a hundred different initial guesses." Ex.1006, 3. Trefethen further notes that it is useful for parallel programming, for prototyping parallel algorithms, and for fast and convenient execution of easily parallelizable numerical computations." Ex.1006, Abstract. Menon describes using MultiMATLAB "as a linear system solver," calculating "discrete Fourier transform," using "using 64 x 64 real matrix," "matrix multiplications," "solve for eigenvalues," and calculating the "conjugate gradient algorithm." Ex.1005, 7, 10-11.

265. To enable MultiMATLAB to accomplish the more complicated mathematical algorithms contemplated by Menon and Trefethen, Trefethen describes that "any of the MATLAB processes" can perform "general point-to-point communication" using "send and receive commands." Ex.1006, 5. The Send(), Eval(), and Recv() MEX routines (built on MPI message passing) send results directly from one MATLAB process to another process. Ex.1006, 5-6. For example, the Send(), Eval(), and Recv() MEX routines can send a result to any

other process by simply specifying the process ID (in this example, the result is sent to process ID=3):

```
“x = [pi pi^2];  
Send(3,x)  
Eval(3, ‘Recv’ ),”
```

“leading to the output

3.1416 9.8696.”

Ex.1006, 3, 5-6.

266. Further, POSITAs knew that MPI message passing (such as used in Trefethen) can be used to transmit results to a “chosen” process (other than the master) that the other processes (“peers”) know to send the results to. Ex.1011, 64 (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”).

267. Accordingly, to implement the mathematical computations contemplated by Menon and Trefethen, a POSITA would use the MEX routine commands to send a result calculated by third kernel to the first cluster node module for subsequent processing by the first kernel. *See* Ex.1011, 64. For example, in the conjugate gradient algorithm described in Menon, the kernels are each processing, in parallel, and awaiting results from the other processes to iteratively calculate further results, which includes the third process using the results from the

second process and the first process using the results from the third node. Ex.1005, 11-14. As an additional example, the “cycle.m” algorithm of Trefethen would, as a final calculation by process id=1, add 1 (or any other mathematical computation) to the result received from the third process, for a final result of 9.

268. The processes each use its *cluster node module* to send the output to the other MATLAB processes because, in a MultiMATLAB architecture, each MATLAB process uses its MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*cluster node module*) to directly communicate with any other MATLAB process, as disclosed in limitation [1.6.3].

269. Further, it was shown in limitation [1.4.2], that the interactive processor has a user interface for providing intermediary and final results of the mathematical calculations to the user.

270. Accordingly, it would have been obvious to a POSITA that Menon and Trefethen teach that any MATLAB process (e.g., the first non-interactive process) is configured to receive a result from any other MATLAB process via its cluster node module, and that the result can be used for subsequent processing and then returned to interactive processor upon completion, rendering obvious *wherein the first cluster node module is configured to return, based on the second result, a final result.*

5. Claim 2

- a. [2.0] *The computer cluster of claim 1, wherein the first cluster node module is logically positioned between the user interface and the first kernel.***

271. Menon and Trefethen render obvious *wherein the first cluster node module is logically positioned between the user interface and the first kernel* because Menon and Trefethen teach that the MEX routines (*the first cluster node module*) in a MATLAB process on the first non-interactive processor are operationally positioned between (*is logically positioned between*) the user interface such as a PDE toolkit graphical interface (*the user interface*) and the MATLAB (*the first kernel*) in the MATLAB process on the same first non-interactive processor.

272. In limitation [1.4.2], it was shown that the MEX routines, which are part of the *cluster node module*, in the MATLAB process of each non-interactive processor receive a MATLAB command from the PDE toolkit graphical interface (*the user interface*) via the communication layer of the *cluster node module* of the MATLAB process and send the MATLAB command to the MATLAB (*kernel*) in that MATLAB process for execution by that MATLAB. Accordingly, as illustrated below in the combination figure of Figures 1 and 2 of Menon, the MEX routines of *the first cluster node module* on the first non-interactive processor of the MultiMATLAB architecture receive MATLAB commands from a user interface such as a PDE toolkit graphical interface (*the user interface*) on the interactive

processor and send the MATLAB commands to MATLAB (*the first kernel*) in the MATLAB process on the same first non-interactive processor for execution by that MATLAB, i.e., the MEX routines of *the first cluster node module* in a MATLAB process on the first non-interactive processor are operationally positioned between (*is logically positioned between*) the user interface on the interactive processor and the MATLAB on the first non-interactive processor (*the first kernel*), rendering obvious *wherein the first cluster node module is logically positioned between the user interface and the first kernel*.

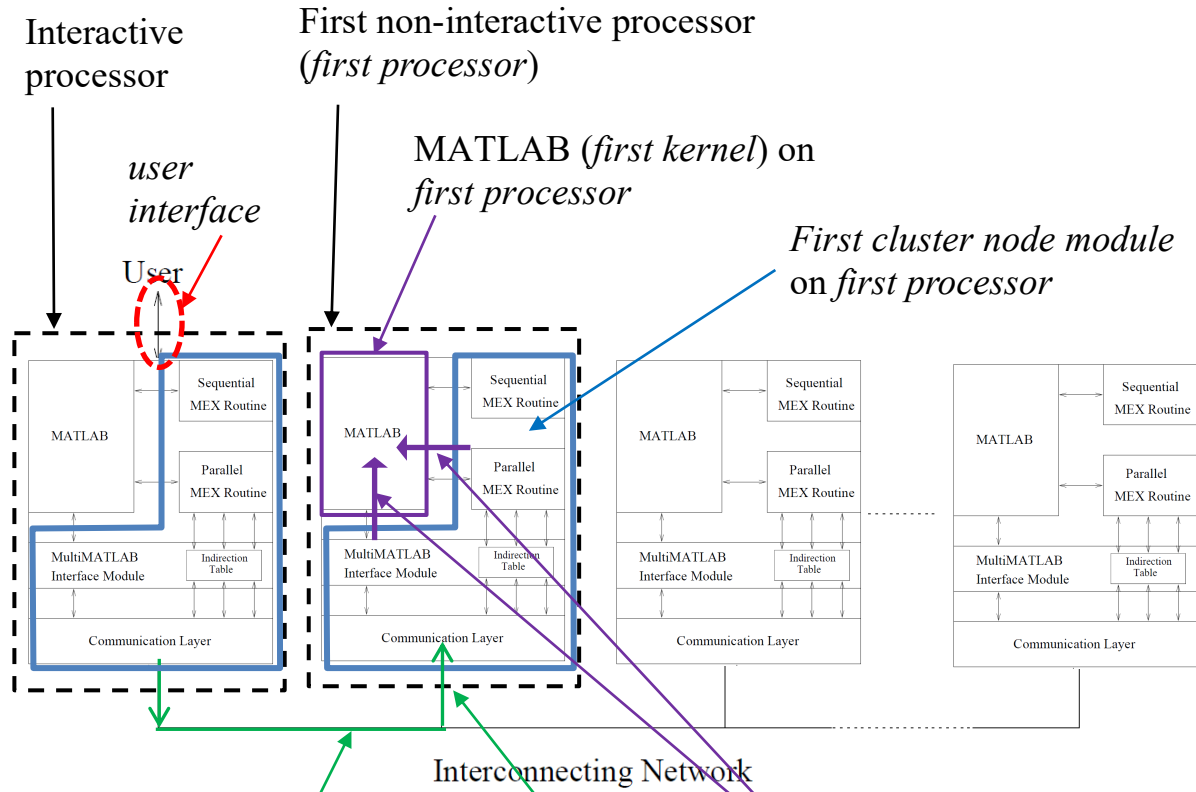


Figure 1: MultiMATLAB Architecture

MATLAB commands transmitted from *user interface* on interactive processor to non-interactive processors

MATLAB commands received from interactive processor

MATLAB commands sent by *cluster node module* to MATLAB (*kernel*) on same non-interactive processor

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

6. Claim 3

a. [3.0] *The computer cluster of claim 1,*

273. Limitation [3.0] is rendered obvious for the same reasons presented above for limitation [1.0].

- b. [3.1] *wherein the at least one computer-readable medium comprises a first computer-readable medium coupled to at least the first processor and a second computer-readable medium coupled to at least the second processor,*

274. Menon and RS6000 render obvious *wherein the at least one computer-readable medium comprises a first computer-readable medium coupled to at least the first processor and a second computer-readable medium coupled to at least the second processor* because Menon discloses the MATLAB process of each of the first non-interactive processor and the second non-interactive processor of the IBM SP2 is *a computer-readable medium* that is in the form of internal and external hard disk drives, or memory cards, and RS6000 teaches that the hard disk drives are attached to, and the memory cards are mounted on, the symmetric multiprocessor of the of the IBM SP2.

275. In limitations [1.1.1] and [1.1.2], it was shown that each of the first non-interactive processor (*the first processor*) and the second non-interactive processor (*the second processor*) of the MultiMATLAB architecture has a respective MATLAB process, as shown in Figure 1 of Menon below.

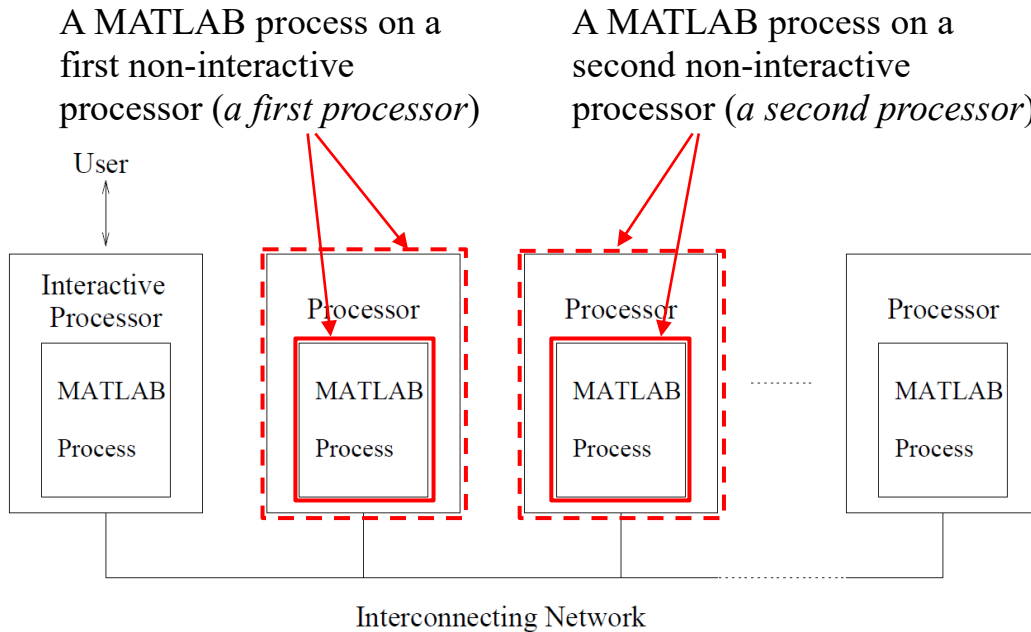


Figure 1: MultiMATLAB Architecture

Ex.1005, FIG. 1 (annotated)

276. Further, in limitation [1.2], it was shown that each MATLAB process's address space is *a computer-readable medium that is in communication with the processor of that MATLAB process*. That is, it was shown in limitation [1.2] that an address space (*a first computer-readable medium*) of the MATLAB process on the first non-interactive processor (*the first processor*) is *a computer-readable medium that is in communication with the first non-interactive processor (the first processor)* and an address space (*a second computer-readable medium*) of the MATLAB process on the second non-interactive processor (*the second processor*) is *a computer-readable medium that is in communication with the second non-interactive processor (the second processor)*, rendering obvious *wherein*

the at least one computer-readable medium comprises a first computer-readable medium ... [of] at least the first processor and a second computer-readable medium ... [of] at least the second processor.

277. It was shown in limitations [1.1.1] and [1.1.2] that Menon’s disclosure of the first non-interactive processor and the second non-interactive processor of the MultiMATLAB architecture that is implemented in IBM SP2 in view of RS6000 teaching a symmetric multiprocessor of the IBM SP2 having “four, eight, twelve, or sixteen” processors renders obvious *a first processor and a second processor*, respectively. Ex.1007, 9. Further, it was shown in limitation [1.2] that the MATLAB process address space (*a computer-readable medium*) of each processor of the IBM SP2 can be in the form of internal and external hard disk drives, or memory cards. RS6000 discloses that the symmetric multiprocessor can be attached to “one pair of internal hard disk drives” and to “[a]dditional hard disk drives,” and that “[m]emory is supplied” to the memory cards of the symmetric multiprocessor by “DIMMs [Dual In-line Memory Module] which must be mounted in banks of eight DIMMs.” Ex.1007, 11, 10.

278. Accordingly, Menon’s teaching that each MATLAB process is in an address space (*a computer-readable medium*) on an IBM SP2 in view of RS6000’s teaching that the IBM SP2’s processors are attached to disk drives and memory, renders obvious *a first computer-readable medium coupled to at least the first*

processor and a second computer-readable medium coupled to at least the second processor.

- c. **[3.2] *wherein the first kernel and the first cluster node module reside in the first computer-readable medium, and wherein the second kernel and the second cluster node module reside in the second computer-readable medium.***

279. Menon and RS600 render obvious *wherein the first kernel and the first cluster node module reside in the first computer-readable medium, and wherein the second kernel and the second cluster node module reside in the second computer-readable medium* because Menon and RS600 teach MATLAB in the MATLAB process on the first non-interactive processor and the MEX routines, MultiMATLAB interface module, indirection table, and communication layer of the MATLAB process on the first non-interactive processor are in the MATLAB process address space of the MATLAB process on the first non-interactive processor, and MATLAB in the MATLAB process on the second non-interactive processor and the MEX routines, MultiMATLAB interface module, indirection table, and communication layer of the MATLAB process on the second non-interactive processor are in the MATLAB process address space of the MATLAB process on the second non-interactive processor.

280. In limitation [1.3.1], it was shown that Menon teaches a MATLAB (*a first kernel*) in the MATLAB process on the first non-interactive processor (*first*

processor), where the MATLAB process is contained in (*residing in*) a MATLAB process address space. In limitation [3.1], it was shown that the MATLAB process address space of the MATLAB process on the first non-interactive processor teaches *a first computer-readable medium*.

281. In limitation [1.4.1], it was shown that the MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*a first cluster node module*) of the MATLAB process of the first non-interactive processor are in (*residing in*) the MATLAB process address space of the MATLAB process on the first non-interactive processor. In limitation [3.1], it was shown that the MATLAB process address space of the MATLAB process on the first non-interactive processor teaches *a first computer-readable medium*.

282. In limitation [1.5.1], it was shown that Menon teaches a MATLAB (*a second kernel*) in the MATLAB process of the second non-interactive processor (*a second processor*), where the MATLAB process is contained in (*residing in*) a MATLAB process address space. In limitation [3.1], it was shown that the MATLAB process address space of the MATLAB process on the second non-interactive processor teaches *a second computer-readable medium*.

283. In limitation [1.6.1], it was shown that the MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*a second cluster node module*) of the MATLAB process of the second non-interactive

processor are in (*residing in*) the MATLAB process address space of the MATLAB process on the second non-interactive processor. In limitation [3.1], it was shown that the MATLAB process address space of the MATLAB process on the second non-interactive processor teaches *a second computer-readable medium*.

284. Accordingly, a MATLAB (*a first kernel*) in the MATLAB process on the first non-interactive processor (*first processor*) and the MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*a first cluster node module*) of the MATLAB process on the first non-interactive processor (*first processor*) are in (*reside in*) the MATLAB process address space (*first computer-readable medium*) of the MATLAB process on the first non-interactive processor, and a MATLAB (*a second kernel*) in the MATLAB process on the second non-interactive processor (*second processor*) and the MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*a second cluster node module*) of the MATLAB process on the second non-interactive processor (*second processor*) are in (*reside in*) the MATLAB process address space (*second computer-readable medium*) of the MATLAB process on the second non-interactive processor, as shown in the combination figure of Figures 1 and 2 of Menon below.

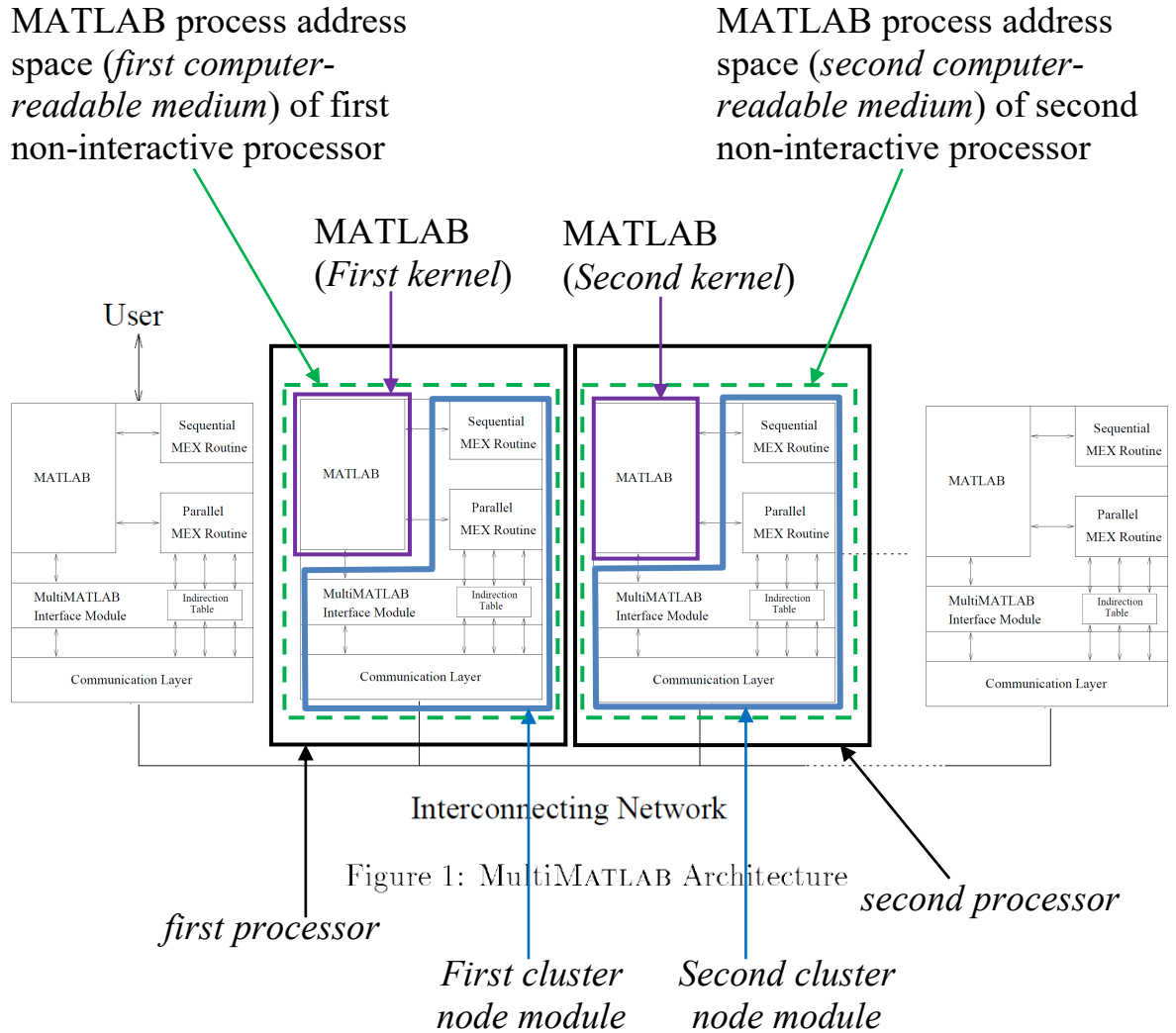


Figure 1: MultiMATLAB Architecture

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

7. Claim 5

- a. **[5.0]** *The computer cluster of claim 1, wherein the second cluster node module communicates with the first cluster node module using commands that implement at least a portion of a Message-Passing Interface configured to distribute tasks among at least one of the first, second, or third kernels without requiring user commands.*

285. Menon and Trefethen render obvious *wherein the second cluster node module communicates with the first cluster node module using commands that implement at least a portion of a Message-Passing Interface configured to distribute tasks among at least one of the first, second, or third kernels without requiring user commands* because Menon and Trefethen disclose that the *second cluster node module* of the MATLAB process on the second non-interactive processor directly communicates with the *first cluster node module* of the MATLAB process on the first non-interactive processor, where the MEX routines (part of the *second cluster node module*) in the MATLAB process on the second non-interactive processor use commands such as Send() and Recv() that implement standard variants of MPI calls, which send MATLAB commands (*configured to distribute tasks*) to MATLAB (*among at least one of the first, second, or third kernels*) in the MATLAB process on the second non-interactive processor without any further input or command from the user (*without requiring user commands*).

286. First, in limitation [1.6.3], it was shown that each MATLAB process in the MultiMATLAB architecture communicates with any other using its MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*cluster node module*), where MEX routines implement standard variants of MPI calls to enable SPMD point-to-point communication so that the MATLAB processes communicate tasks and data directly with each other. Examples point-to-point commands for SPMD communication paradigm that are implemented as MEX routines include “Send(pid,data)” command that “send[s] data from one process to another,” “Recv(pid)” command that “receive[s] data sent from another process,” and “Bcast(pid,data)” command that “broadcasts data from processor pid to all processes,” as shown in Table 1 of Menon shown below. Ex.1005, Table 1.

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)

287. Second, 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 without requiring user commands. Ex.1006, 3-7. Specifically, Trefethen provides the following example code where (i) the MultiMATLAB architecture is configured to exchange information between processors during the course of parallel computation without any command from the user, 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(), without any command from the user:

Instruction/command for MATLAB process ID=0 to create a=1 and send “a” to MATLAB process ID=1

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

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

Instruction/command 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/commands

Ex.1006, 6 (annotated).

288. Third, Menon describes how the MEX routines (such as Send and Recv) in a MATLAB process implement standard variants of MPI calls, explaining that the MultiMATLAB interface module in the MATLAB process acts as an interpretation layer that maps MPI calls (e.g., MPI_SEND, MPI_RECV) to an indirection table for the MEX routines:

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; Ex.1006, 10 (“MultiMATLAB is currently built upon the standard send and receive variants” of MPI); Ex.1017, 15-16 (disclosing MPI calls MPI_SEND, MPI_RECV).

289. Accordingly, each MATLAB process communicates with any other MATLAB process using its respective *cluster node module*, where the MEX routines of the respective *cluster node modules* use commands such as Send() and Recv to implement standard variants of MPI calls to enable SPMD point-to-point communication so that the MATLAB processes communicate directly with each other. That is, the *cluster node module* of one MATLAB process (e.g., the *second cluster node module* of the MATLAB process on the second non-interactive processor) communicates with the *cluster node module* of any other MATLAB process (e.g., the *first cluster node module* of the MATLAB process on the first non-interactive processor) using commands such as Send() and Recv that

implement standard variants of MPI calls (*using commands that implement at least a portion of a Message-Passing Interface*) to enable SPMD point-to-point communication between the MATLAB processes.

290. Fourth, Menon teaches that “[o]n the interactive process, the Eval routine ... accepts a vector of MATLAB process IDs and a MATLAB command” and “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. Trefethen provides an example where “a vector of process IDs,” “[4 5],” and a MATLAB command are passed to the Eval command at the interactive processor to have the MATLAB processes on the non-interactive processors (i.e., processor IDs = 4 and 5) execute the MATLAB command:

```
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.

291. A POSITA would recognize and understand that the MATLAB command (“cond(hilb(ID))”) that Trefethen teaches is executed by a MATLAB process is executed by MATLAB of that MATLAB process. In other words, a POSITA would recognize and understand that MEX routines in MATLAB processes on non-interactive processors, such as MEX routines (part of *the second cluster node module*) in the MATLAB process on the second non-interactive processor, which are “run[n]ing in a loop ... wait[ing] for MATLAB commands to arrive from the interactive process,” send the MATLAB command (e.g., “cond(hilb(ID))”) (*configured to distribute tasks*) to MATLAB (*among at least one of the first, second, or third kernels*) in that MATLAB process without any further input or command from the user (*without requiring user commands*) upon receiving the MATLAB command from the interactive processor. Ex.1005, 4.

292. Trefethen teaches 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” and that “MultiMATLAB is currently built upon the standard [MPI] send and receive variants, which employ buffered communication for most messages.” Ex.1006, 4, 10. Accordingly, a POSITA would recognize and understand that the MEX routines in the MATLAB process on the second non-interactive processor sends MATLAB commands to MATLAB in that MATLAB

process using standard variants of MPI calls implemented by send and receive commands.

293. Accordingly, Menon and Trefethen disclose that the *second cluster node module* of the MATLAB process on the second non-interactive processor directly communicates with the *first cluster node module* of the MATLAB process on the first non-interactive processor, where the MEX routines (part of the *second cluster node module*) in the MATLAB process on the second non-interactive processor use commands such as Send() and Recv() that implement standard variants of MPI calls, which send MATLAB commands (*configured to distribute tasks*) to MATLAB (*among at least one of the first, second, or third kernels*) in the MATLAB process on the second non-interactive processor without any input or command from the user (*without requiring user commands*), rendering obvious *wherein the second cluster node module communicates with the first cluster node module using commands that implement at least a portion of a Message-Passing Interface configured to distribute tasks among at least one of the first, second, or third kernels without requiring user commands.*

8. Claim 6

- a. [6.0] *The computer cluster of claim 1, further comprising a data structure in which messages originating from the second and third cluster node modules are stored.***

294. Menon and Trefethen, or alternatively Menon, Trefethen, and MPIref render obvious *further comprising a data structure in which messages originating from the second and third cluster node modules are stored* because Menon and Trefethen, or alternatively Menon, Trefethen, and MPIref, disclose that each MATLAB process in the MultiMATLAB architecture communicates tasks and data with any other MATLAB process using its MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*cluster node module*), where MEX routines implement standard variants of MPI calls to enable SPMD point-to-point communication so that the MATLAB processes communicate data directly with each other, where the data have MATLAB datatypes such as sparse matrices, cell arrays, and structures, and that the MEX routines store the received data in a memory allocated for storing data having datatype of the host language (i.e., data having the MATLAB datatypes).

295. First, in limitation [1.6.3], it was shown that each MATLAB process in the MultiMATLAB architecture communicates tasks and data directly with each other using their respective MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*cluster node module*). For example, as

shown in limitation [1.8.3], the MATLAB processes on the third non-interactive processor, the second non-interactive processor, and the first non-interactive processor use their respective MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*said third cluster node module, the first cluster node module and the second cluster node module*) to communicate commands (e.g., tasks) and data with each other. Menon teaches that “[e]ach 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. An example of the communication layer is MPI. Ex.1005, 3.

296. Second, also as discussed in limitation [1.6.3], 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. The “SPMD paradigm,” which “permit[ts] finer grain communication of MATLAB data structures between any [MATLAB] processes,” “is facilitated by MEX routines that expose some

functionality of the underlying communication layer [i.e., MPI] to the MATLAB language.” Ex.1005, 8, 3 (“MPI”).

297. 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. As shown in limitation [1.6.3], MEX routines 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. Table 1 of Menon below shows example point-to-point commands for SPMD communication paradigm that are implemented as MEX routines, 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.

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)

298. Third, Menon discloses that the SPMD communication paradigm “permit[s] finer grain communication of MATLAB data structures between any processes.” Ex.1005, 8. Referring to the above-mentioned standard variants of MPI calls that MEX routines implement to enable SPMD point-to-point communication, Menon states that “[i]n order to accommodate applications requiring a finer degree of communication, we provide a set of MATLAB message passing routines analogous to those in MPI. In particular, we provide point to point and collective communication MATLAB routines operating directly on MATLAB data

structures,” examples of which include Send(), Recv(), and Bcast() shown in Table 1 of Menon above. Ex.1005, 9.

299. Menon discloses that the “MATLAB data structure” that the MEX routines (part of the *cluster node module*) operate directly on are matrices, arrays, and structures, explaining that:

While the MATLAB communication routines are optimized for dense, real matrices that may be mapped almost directly to the underlying layer, they also work for arbitrary MATLAB data structures such as sparse matrices, cell arrays, and structures. The size and shape of communicated data structures are determined at runtime.

Ex.1005, 9.

300. Menon also teaches that the MEX routines, such as Recv(), allocate memory for storing received data, stating “MATLAB communication routines do not expose any memory management issues to users. Routines do any allocation necessary for received data. Routines block until any data to be sent can safely be overwritten and any data to be received has arrived.” Ex.1005, 9-10. Trefethen teaches that the “allocation necessary for received data” disclosed by Menon is “memory allocation for received matrices.” Ex.1005, 10; Ex.1006, 10.

301. Accordingly, a POSITA would recognize and understand that Menon and Trefethen’s combined teaching of a memory allocation of a MEX routine for received data for “MATLAB data structures such as sparse matrices, cell arrays, and structures” that are communicated renders obvious *further comprising a data structure in which messages originating from the second and third cluster node modules are stored.*

302. Alternatively, MPIref discloses an MPI **receive buffer** (of the MPI_RECV call) for storing received data having the type specified by **datatype**. Ex.1017, 15-21. The MEX routine Recv(), which “receive[s] data sent from another process,” “do[es] any allocation necessary for received data.” Ex.1005, 9-10, Ex.1006, 10. Menon teaches that MEX routines (such as Recv()) are mapped (using the indirection table) to the analogous MPI calls (such as MPI_RECV()) found in the communication layer (part of the *cluster node module*). Ex.1005, 3, 6. The communication layer, using the MPI calls, communicates over the interconnection network with the communication layer in the other processor. Ex.1005, 3.

303. MPIref describes the details of what the parallel MEX routines such as Recv() 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 “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 basic datatypes that can be specified for message data values correspond to the basic datatypes of the host language.” Ex.1017, 17.

304. 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:

address of memory allocated
for storing received data

type of the received data

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)(emphasis original).

305. 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.

306. Accordingly, Menon and Trefethen’s combined teaching of a memory allocation of a MEX routine for received data for “MATLAB data structures such as sparse matrices, cell arrays, and structures” that are communicated renders obvious *further comprising a data structure in which messages originating from the second and third cluster node modules are stored.*

307. Alternatively, Menon and Trefethen disclose that each MATLAB process in the MultiMATLAB architecture communicates tasks and data directly with any other MATLAB process using its MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*cluster node module*), where MEX routines implement standard variants of MPI calls to enable SPMD point-to-point communication so that the MATLAB processes communicate data directly with each other, where the data have MATLAB datatypes such as sparse matrices, cell arrays, and structures, and that the MEX routines store the received data in a memory allocated for storing data having datatype of the host language (i.e., data having the MATLAB datatypes) in view of MPIref disclosing a message buffer and queue for storing messages, rendering obvious *further comprising a data structure*

in which messages originating from the second and third cluster node modules are stored.

9. Claim 7

- a. [7.0] *The computer cluster of claim 6, wherein the data structure of the first cluster node module comprises a received message queue.***

308. The '582 patent states that “[t]he received message queue 306 includes a data structure for storing messages received from other cluster node modules. Related data pertaining to the messages received, such as whether an expression has been completed, may also be stored in the received message queue 306. The received message queue 306 may include a queue and/or another type of data structure such as, for example, a stack, a linked list, an array, a tree, etc.” Ex.1001, 21:51-60. Likewise, Menon, Trefethen, and MPIref teach that the memory allocated for storing received data having MATLAB datatypes such as matrices, arrays, structures, etc., includes a receive buffer that stores received data having matrix datatype, array datatype, structure datatype, etc., rendering obvious *wherein the data structure of the first cluster node module comprises a received message queue.*

309. In limitation [6.0], it was shown that the MEX routines of each MATLAB process (and as such, the MEX routines (part of the *first cluster node module*) of the MATLAB process on the first non-interactive processor) store the received data in a memory allocated for storing data having MATLAB datatypes

such as matrices, arrays, structures, etc. In particular, it was shown that the MPI call `MPI_RECV`, to which the MEX routine `Recv` that “receive[s] data sent from another process” is mapped, stores received data having MATLAB datatypes such as matrices, arrays, structures, etc., in a **receive buffer** that consists of a storage containing **count** consecutive elements of the type specified by **datatype**, starting at address **buf**, as shown below:

receive buffer (memory allocated for storing received data)



<code>MPI_RECV (buf, count, datatype, source, tag, comm, status)</code>		
OUT	<code>buf</code>	initial address of receive buffer (choice)
IN	<code>count</code>	number of elements in receive buffer (integer)
IN	<code>datatype</code>	datatype of each receive buffer element (handle)
IN	<code>source</code>	rank of source (integer)
IN	<code>tag</code>	message tag (integer)
IN	<code>comm</code>	communicator (handle)
OUT	<code>status</code>	status object (Status)

Ex.1005, 9; Ex.1017, 19 (annotated)(emphasis original).

310. 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.

311. Accordingly, Menon describes allocating memory for storing received data having MATLAB datatypes such as matrices, arrays, structures, using MPI and MPIref describes the `MPI_RECV` function that includes a receive buffer that stores

received data having matrix datatype, array datatype, structure datatype, and enqueues the message received by the MPI_RECV call, rendering obvious *wherein the data structure of the first cluster node module comprises a received message queue.*

10. Claim 8

- a. **[8.0]** *The computer cluster of claim 7, wherein the first cluster node module further comprises a message receiving queue, in which data specifying a location to which the first cluster node module can expect to receive an expression and an identifier of a processor from which the expression is expected to be sent are stored.*

312. Menon and MPIref render obvious *wherein the first cluster node module further comprises a message receiving queue, in which data specifying a location to which the first cluster node module can expect to receive an expression and an identifier of a processor from which the expression is expected to be sent are stored* because Menon discloses MEX routines (such as Recv()) (part of the *first cluster node module*) that implement standard variants of MPI calls to enable SPMD point-to-point communication between any MATLAB processes including the MATLAB process on the first non-interactive processor, and MPIref discloses that the MPI call MPI_RECV, found in the communication layer (i.e., MPI) (part of the *first cluster node module*) of the MATLAB process on the first non-interactive processor and to which the MEX routine Recv() is mapped, stores in a queue the information in the variables received by RECV_MPI and named “client_source,”

which contains information about the source of the received data, and
“client_rank_in_new_world,” which contains information about the initial address
of receive buffer for storing the received data.

313. In limitation [6.0], it was shown that the MEX routines of each
MATLAB process (and as such, the MEX routines (part of the *first cluster node*
module) of the MATLAB process on the first non-interactive processor) store the
received data in a memory allocated for storing data having MATLAB datatypes
such as matrices, arrays, structures, etc. In particular, it was shown that the MPI call
MPI_RECV, to which the MEX routine Recv that “receive[s] data sent from
another process” is mapped, stores received data having MATLAB datatypes such
as matrices, arrays, structures, etc., in a **receive buffer** that consists of a storage
containing **count** consecutive elements of the type specified by **datatype**, starting at
address **buf**, as shown below:

receive buffer (memory allocated for
storing received data)

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.1005, 9; Ex.1017, 19 (annotated)(emphasis original).

314. 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, i.e., MPI (cluster node module)

Source of received data

Receive buffer (address of memory allocated for storing received data)

Contents of information from MPI_RECV

Ex.1017, 163 (annotated).

315. 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. The variable “client_source” contains the source of the received data because “client_source = status.MPI_SOURCE” and “status.MPI_SOURCE ... contain[s] the source ... of the received message.” Ex.1017, 164, 20. The variable “client_rank_in_new_world” contains the “initial address of receive buffer” because “client_rank_in_new_world = buffer[0]” and the receive buffer “start[s] at address **buf**” which is the “initial address of receive buffer.” Ex.1017, 164, 19 (emphasis original).

316. Accordingly, Menon discloses MEX routines (such as Recv()) (part of the *cluster node module*) that implement standard variants of MPI calls to enable SPMD point-to-point communication between any MATLAB processes (and as such including the MATLAB process on the first non-interactive processor), and MPIref discloses that the MPI call MPI_RECV, found in the communication layer (i.e., MPI) (part of *first cluster node module*) of the MATLAB process on the first non-interactive processor and to which the MEX routine Recv() is mapped, stores in a queue (*a message receiving queue, in which data ... are stored*) the information in the variables received by RECV_MPI and named “client_source,” which contains information about the source of the received data, (*an identifier of a processor from which the expression is expected to be sent*) and “client_rank_in_new_world,”

which contains information about the initial address of receive buffer for storing the received data (*data specifying a location to which the first cluster node module can expect to receive an expression*), rendering obvious *wherein the first cluster node module further comprises a message receiving queue, in which data specifying a location to which the first cluster node module can expect to receive an expression and an identifier of a processor from which the expression is expected to be sent are stored.*

11. Claim 9

- a. **[9.0] *The computer cluster of claim 1, wherein the second cluster node module further comprises a received message queue in which messages originating from the first and third cluster node modules are stored.***

317. Menon, Trefethen, and MPIref render obvious *wherein the second cluster node module further comprises a received message queue in which messages originating from the first and third cluster node modules are stored* because Menon, Trefethen, and MPIref disclose that the MEX routines (*the second cluster node module*) of the MATLAB process on the second non-interactive processor store the data received from the *first cluster node module* and the *third cluster node module* in a receive buffer that stores received data having matrix datatype, array datatype, structure datatype, etc.

318. In limitation [1.6.3], it was shown that Menon and Trefethen, or alternatively Menon, Trefethen, and MPIref, disclose that each MATLAB process

in the MultiMATLAB architecture communicates tasks and data with any other MATLAB process using its MEX routines, MultiMATLAB interface module, indirection table, and communication layer, i.e., for example, the MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*second cluster node module*) of the MATLAB process on the second non-interactive processor communicates tasks and data with the MEX routines, MultiMATLAB interface module, indirection table, and communication layer of the MATLAB processes on the first non-interactive processor and the third non-interactive processor (*first cluster node module* and *third cluster node module*, respectively), where the MEX routines (part of each *cluster node module*) implement standard variants of MPI calls to enable SPMD point-to-point communication between the MATLAB processes. Further, it was also shown in limitation [6.0] that the MEX routines (e.g., part of *the second cluster node module*) of each MATLAB process store data received from the MEX routines of the other MATLAB processes (e.g., part of *the first cluster node module* and part of *the third cluster node module*) in a memory allocated for storing received data having MATLAB datatypes such as matrices, arrays, structures, etc.

319. In limitation [7.0], it was shown that each cluster node module allocates memory for storing received data having MATLAB datatypes such as matrices, arrays, structures, etc., includes a receive buffer that stores received data

having matrix datatype, array datatype, structure datatype, etc., rendering obvious that each cluster node module has a data structure with *a received message queue*.

320. Accordingly, Menon's MEX routines of *the second cluster node module* in the MATLAB process on the second non-interactive processor store the data received from the *first cluster node module* and *the third cluster node module* in a receive buffer that stores received data having matrix datatype, array datatype, structure datatype, etc., rendering obvious, *wherein the second cluster node module further comprises a received message queue in which messages originating from the first and third cluster node modules are stored.*

12. Claim 10

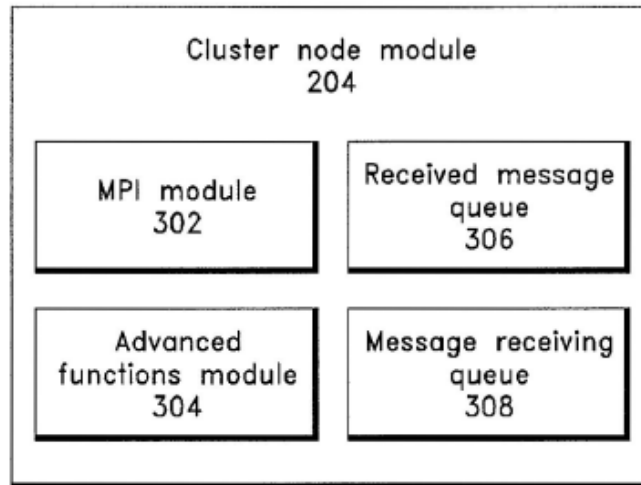
a. [10.0] *The computer cluster of claim 1,*

321. Limitation [10.0] is rendered obvious for the same reasons presented above for limitation [1.0].

b. [10.1] *wherein the first cluster node module comprises an advanced functions module, and*

322. Menon and Trefethen render obvious *wherein the first cluster node module comprises an advanced functions module* because Menon teaches MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*the first cluster node module*) of a MATLAB process on the first non-interactive processor and Menon and Trefethen teach that parallel MEX routines enable all parallel functionalities of the MultiMATLAB architecture.

323. Fig. 3 of the '582 patent depicts a single cluster node module comprising *an advanced functions module* as a set of components:



Ex.1001, FIG. 3

324. The '582 patent describes the advanced functions module 304 as including “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, 16:51-52, 16:44-47. “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, 16:48-50, 16:64 (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 (‘FFT’s’),” “parallel disk input and output calls,” and “[a]utomatic [l]oad [b]alancing.” Ex.1001, 17:5-9, 18:12-16, 19:18-21, 20:17-19, 21:1.

325. In limitation [1.4.1], it was shown that the MEX routines, MultiMATLAB interface module, indirection table, and communication layer of a MATLAB process on a processor correspond to the *cluster node module* of the ’582 patent, as shown in Figure 2 of Menon below.

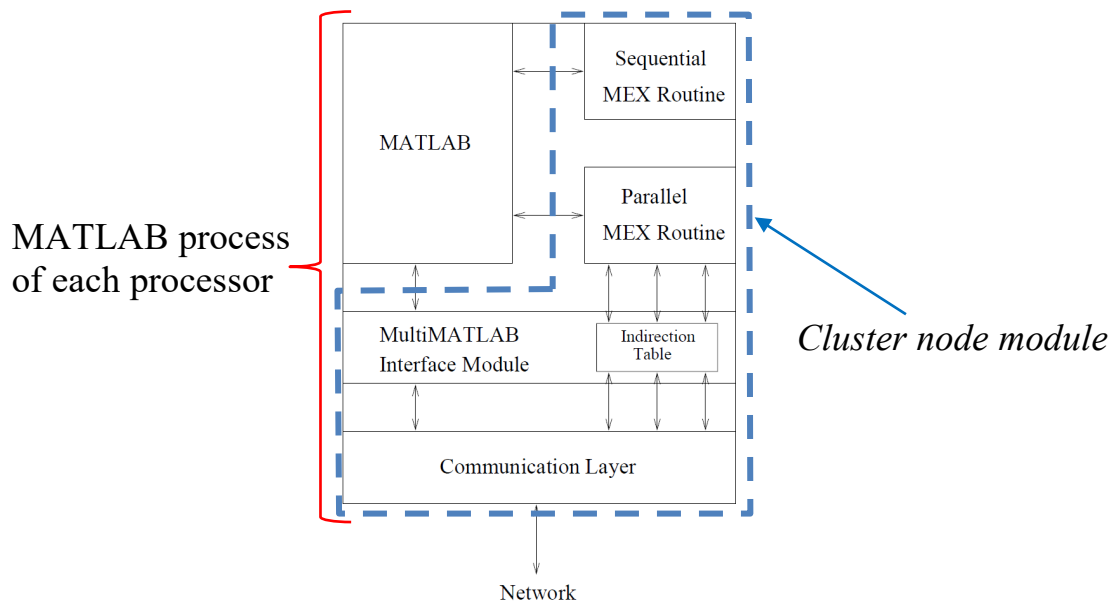


Figure 2: MATLAB Process Address Space

Ex.1005, FIG. 2 (annotated)

326. Menon explains that the MEX routines in the MATLAB processes are responsible for all the parallel functionalities of the MultiMATLAB architecture. Ex.1005, 3. Specifically, the parallel MEX routines “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. For example:

[T]he MultiMATLAB system provides users with 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 over the underlying communication layer to the processes corresponding to the specified IDs. 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. All parallel MEX routines access the underlying communication layer and, hence, the network, through the MultiMATLAB interface module.

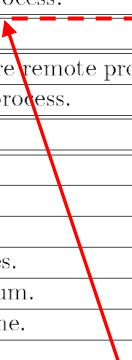
Ex.1005, 3-4. The “[p]arallel MEX routines may be run ... via the Eval command,” which “[e]valuates a [MATLAB] command on one or more MATLAB processes.”

Ex.1005, 6, Table 1.

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

Eval routine for
parallel evaluation



Ex.1005, Table 1 (annotated)

327. The MEX routines obtain “access [to] any function provided by the underlying communication layer” via the indirection table, where an example of the underlying communication layer is MPI. Ex.1005, 5, 3. Menon explains that “[p]arallel MEX routines may be run in a SPMD fashion on any subset of MATLAB processes via the Eval command.” Ex.1005, 6. Specifically, “MEX routines are granted access to MPI routines” and are mapped to MPI calls using the indirection table. Ex.1005, 6. That is, similar to the ’582 patent’s advanced

functions module 304, the MEX routines rely on MPI calls for their implementations. Ex.1001, 16:48-50, 16:64; Ex.1005, 5-6.

328. Menon also discloses that “parallel MEX routines would perform matrix multiplications, solve for eigenvalues, compute Fourier transforms, and so on.” Ex.1005, 10. Menon teaches that “parallel MEX routines” enable “parallelism” in applications “such as a linear system solver or a discrete Fourier transform.” Ex.1005, 7. Trefethen provides several examples where the MEX routine Eval is used to perform parallel linear algebra operations. Ex.1006, 4-10. For example, Trefethen describes the use of Eval to compute “the condition numbers of the Hilbert matrices” and matrix “eigenvalue computation” in parallel, as well as “comput[ation of] vectors obtained by reducing data over the copies of a vector or matrix located on all processors.” Ex.1006, 4, 5, 7.

329. Accordingly, Menon’s cluster node module (having parallel MEX routines), provide basic parallelization of operations on the MultiMATLAB architecture, where these operations include advanced functions operations such as linear algebra operations (e.g., matrix and vector operations), Fourier transform computations, etc., render obvious *an advanced functions module*. Because the MEX routines, MultiMATLAB interface module, indirection table, and communication layer of the MATLAB process on the first non-interactive processor that correspond to the *first cluster node module* include MEX routines (which

render obvious *an advanced functions module*), a POSITA would recognize and understand that *the first cluster node module* includes (*comprises*) *an advanced functions module*.

- c. [10.2] *wherein the advanced functions module comprises a call that evaluates an expression across the computer cluster in parallel.*

330. Menon and Trefethen render obvious *wherein the advanced functions module comprises a call that evaluates an expression across the computer cluster in parallel* because Menon and Trefethen disclose that the MEX routines include the routine Eval that causes parallel evaluation of tasks and commands by the MATLAB processes of the MultiMATLAB architecture.

331. First, as shown in limitation [10.1], the parallel MEX routines render obvious *an advanced functions module*.

332. Second, as discussed in limitation [10.1], the MEX routines are responsible for all the parallel functionalities of the MultiMATLAB architecture. Ex.1005, 3. The parallel MEX routines “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. Menon discloses that the “[p]arallel MEX routines may be run ... via the Eval command,” which is a “routine for parallel evaluation” and “[e]valuates a [MATLAB] command on one or more MATLAB processes.” Ex.1005, 6, 4, Table 1.

333. Trefethen provides several examples where the MEX routine Eval is used to perform parallel evaluations. Ex.1006, 4-10. For example, Trefethen discloses that “[r]unning [the command Eval(‘ID’)] on all nodes might give the result

ans = 0

ans = 1

ans = 5

ans = 2

ans = 3

ans = 4,”

where the MultiMATLAB architecture consists six MATLAB processes on six processors or nodes and “the MultiMATLAB command ID” “returns the number of the current process.” Ex.1006, 3-4. In another example, Trefethen discloses that the “Eval(‘sqrt(2)’),” run on the MultiMATLAB architecture with the six MATLAB processes, causes “the MATLAB command sqrt(2) [to be] executed on all six processors,” and “[t]he result is six repetitions of 1.4142.” Ex.1006, 3.

334. Accordingly, the parallel MEX routines (*the advanced functions module*) include the command Eval (*comprises a call*) that causes parallel evaluation of commands and mathematical expressions (*that evaluates an*

expression ... in parallel) by the MATLAB processes on the processors of the MultiMATLAB architecture (*across the computer cluster*).

13. Claim 11

- a. **[11.0]** *The computer cluster of claim 10, wherein the advanced functions module comprises a call that calculates a Fourier transform across the computer cluster in parallel.*

335. Menon and Trefethen render obvious *wherein the advanced functions module comprises a call that calculates a Fourier transform across the computer cluster in parallel.* because Menon and Trefethen disclose that MEX routines include a MEX routine that causes the parallel evaluation of discrete and/or fast Fourier transforms by the MATLAB processes of the MultiMATLAB architecture.

336. First, as shown in limitation [10.1], the parallel MEX routines render obvious *an advanced functions module.*

337. Second, Menon discloses that parallel MEX routines evaluate Fourier transforms in parallel, explaining that:

[P]arallel MEX routines [] 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. Trefethen discloses that an example of the “high-level [MATLAB] commands” is “fft (fast Fourier transform).” Ex.1006, 2. A POSITA would recognize and understand that parallel MEX routines compute discrete and/or fast Fourier transforms by distributing data among all the MATLAB processes for parallel computation and then collecting the results back from the MATLAB processes, *i.e.*, A POSITA would recognize and understand that all the MATLAB processes evaluate discrete and/or fast Fourier transforms in parallel. Ex.1005, 10; Ex.1006, 2.

338. Accordingly, Menon’s parallel MEX routines (*the advanced functions module*) include a MEX routine (*comprises a call*) that causes the parallel evaluation of (*that calculates ... in parallel*) discrete and/or fast Fourier transforms (*a Fourier transform*) by the MATLAB processes of the MultiMATLAB architecture (*across the computer cluster*), rendering obvious *wherein the advanced functions module comprises a call that calculates a Fourier transform across the computer cluster in parallel.*

14. Claim 12

- a. [12.0] *The computer cluster of claim 10, wherein the advanced functions module comprises one or more calls that evaluate matrix operations across the computer cluster in parallel.*

339. Menon and Trefethen render obvious *wherein the advanced functions module comprises one or more calls that evaluate matrix operations across the computer cluster in parallel* because Menon and Trefethen disclose that the MEX routines include a MEX routine (e.g., the routine Eval) that causes parallel evaluation of matrix and vector computations by the MATLAB processes of the MultiMATLAB architecture.

340. First, as shown in limitation [10.1], the parallel MEX routines render obvious *an advanced functions module*.

341. Second, Menon discloses that parallel MEX routines perform matrix operations in parallel, explaining that:

[P]arallel MEX routines [] 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. As is shown in [1.3.2], parallel MEX routines “perform matrix multiplications [and] solve for eigenvalues” by distributing data among all the MATLAB processes for parallel computation and then collecting the results back from the MATLAB processes. Ex.1005, 10; Ex.1006, 2. Accordingly, a POSITA would recognize and understand that parallel MEX routines (*the advanced functions module*) include a MEX routine (*comprises one or more calls*) that causes the parallel evaluation of (*that evaluate ... in parallel*) matrix multiplications (*matrix operations*) by the MATLAB processes of the MultiMATLAB architecture (*across the computer cluster*).

342. However, to the extent more is needed, Trefethen provides several examples where the MEX routine Eval is used to perform matrix operations. Ex.1006, 4-5. For example, Trefethen discloses the use of Eval routine in a MultiMATLAB architecture with six processors “to find the spectral radii (maximum of the absolute values of the eigenvalues) of six matrices of dimension 400.” Ex.1006, 4. Trefethen explains that “[t]he command

Eval(‘max(abs(eig(randn(400))))’)

does not do the trick; we get six copies of the number
20.8508, since the random number generators deliver

identical results on all processors. Preceding the
eigenvalue computation by

```
Eval( 'randn("seed",ID)' ),
```

however, leads to the result desired:

```
ans = 20.9729
```

```
ans = 20.8508
```

```
ans = 21.0364
```

```
ans = 21.0312
```

```
ans = 21.6540
```

```
ans = 20.4072.
```

Ex.1006, 4-5.

343. In another example, Trefethen describes the MultiMATLAB
architecture performing matrix operations using a subset of the six processors of the
MultiMATLAB architecture, explaining that the command

```
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.

344. Trefethen also provides an example use of the Eval routine to evaluate vector operations, explaining that “the commands Min, Max, and Sum compute vectors obtained by reducing data over the copies of a vector or matrix located on all processors. Thus the command

Eval(‘Sum(1,[1 ID Nproc])’)

executed on six processes will return the vector

[6 15 36]

to process 1.” Ex.1006, 7.

345. Accordingly, the MEX routines (*the advanced functions module*) include the routine Eval (*comprises one or more calls*) that causes parallel evaluation of matrix and vector computations (*that evaluate matrix operations ... in parallel*) by the MATLAB processes of the MultiMATLAB architecture (*across the computer cluster*).

15. Claim 13

a. [13.0] *The computer cluster of claim 1, wherein the final result comprises at least a portion of the second result.*

346. Trefethen renders obvious *wherein the final result comprises at least a portion of the second result* because Trefethen teaches that the output returned by the MATLAB process on the first non-interactive processor includes the result “a=8” from the MATLAB process on the third non-interactive processor.

347. As was shown in limitation [1.9], it is obvious that the MATLAB process on the first non-interactive processor receives a result (the second result) from the MATLAB process on the third non-interactive processor and performs further computations to achieve a final result (e.g., performing the conjugate gradient algorithm, eigenvalues or linear system solvers). Accordingly, the output of the first non-interactive processors further computations (the final result) includes the result from the third non-interactive processor (at least a portion of the second result).

16. Claim 14

- a.** [14.0] *The computer cluster of claim 1, wherein the first cluster node module is configured to return the final result to the user interface.*

348. Trefethen renders obvious *wherein the first cluster node module is configured to return the final result to the user interface* because Trefethen teaches that the MEX routines, MultiMATLAB interface module, indirection table, and communication layer in the MATLAB process on the first non-interactive processor return the output having the result “a=8” to a user interface such as a PDE toolkit graphical interface.

349. First, in limitation [1.9], it was shown that each MATLAB process (including the MATLAB process on the first non-interactive processor) is configured to use its MEX routines, MultiMATLAB interface module, indirection

table, and communication layer (first cluster node module) to return an output (is configured to return the final result) to the user “as soon as it is ready.” Ex.1006, 4.

350. Second, in limitation [1.4.2], it was shown that the interactive processor of the MultiMATLAB architecture has a user interface (*the user interface*) such as a PDE toolkit graphical interface and that a “user interacts directly with one MATLAB process, called the interactive process, and operates within that process’s MATLAB environment” using the user interface, as shown in the combined figure of Figures 1 and 2 of Menon below. Ex.1005, 3, 6.

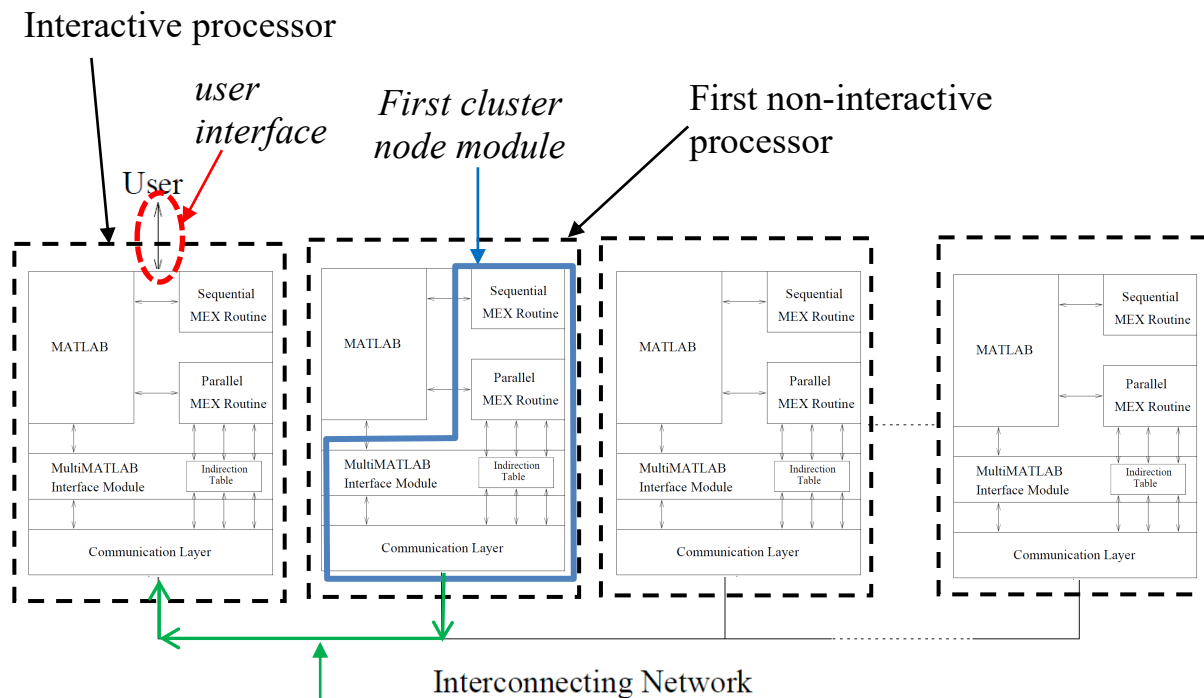


Figure 1: MultiMATLAB Architecture

Final result returned from *first cluster node module* on the first non-interactive processor to the *user interface* on the interactive processor

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

351. Third, Trefethen teaches that “multiprocessor computations” performed by multiple processors, for example, computations performed by MATLAB processes on the processors of the MultiMATLAB architecture, are plotted at the interactive processor, explaining that:

In many applications, the user will find it most convenient to compute on multiple processors but produce plots on the master process, after sending data as necessary. Equally often, however, it may be desirable to produce 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 other words, Trefethen describes either sending data from multiple processors to the master process for plotting thereon or producing “plots in a distributed fashion [at the multiple processors] that are then sent to the user’s screen.” Ex.1006, 7.

352. A POSITA would recognize and understand that the “user’s screen” is at the “master process” and that the “user’s screen” is a screen that includes the user interface such as a PDE toolkit graphical interface and to which “screen output[s]” are sent when sending outputs “to the master process.” Ex.1006, 5, 7. Accordingly,

Trefethen's MATLAB process ID=1 (i.e., the MATLAB process on the first non-interactive processor) returns the output of its calculations using Menon's MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*the first cluster node module*) for display on the user's screen (*to the user interface*) on the interactive processor, rendering obvious *wherein the first cluster node module is configured to return the final result to the user interface*.

17. Claim 15

- a. **[15.0]** *The computer cluster of claim 1, further comprising a communications network interface for the cluster node modules to communicate, with each other using a peer-to-peer architecture, results of execution of the matrix operations.*

353. Menon and RS6000 render obvious *further comprising a communications network interface for the cluster node modules to communicate, with each other using a peer-to-peer architecture, results of execution of the matrix operations* because Menon teaches MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*cluster node modules*) in the MATLAB processes on each processor of the MultiMATLAB architecture communicate results of matrix-related mathematical evaluations with each other over the MultiMATLAB architecture's interconnection network using SPMD point-to-point communication paradigm, in view of RS6000 teaching that the IBM SP2's

communications network includes a local area network (LAN), a wide area network (WAN), and a TCP/IP communication network.

354. First, in limitation [1.4.1], it was shown that the MEX routines, MultiMATLAB interface module, indirection table, and communication layer in a MATLAB process on a processor of the MultiMATLAB architecture teaches *a first cluster node module* of that MATLAB process.

355. Second, as shown in limitation [1.6.3], Menon discloses that each MATLAB process on a processor of the MultiMATLAB architecture has MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*cluster node module*), and Menon and Trefethen disclose that the MEX routines, implemented using standard MPI send, receive, and broadcast calls, enable SPMD point-to-point communication so that each MATLAB process can communicate, i.e., send, receive, and broadcast, mathematical evaluation results, commands (e.g., tasks), and data directly with each other over the MultiMATLAB architecture's interconnection network using their respective *cluster node module*, as depicted in the combination figure of Figures 1 and 2 of Menon below.

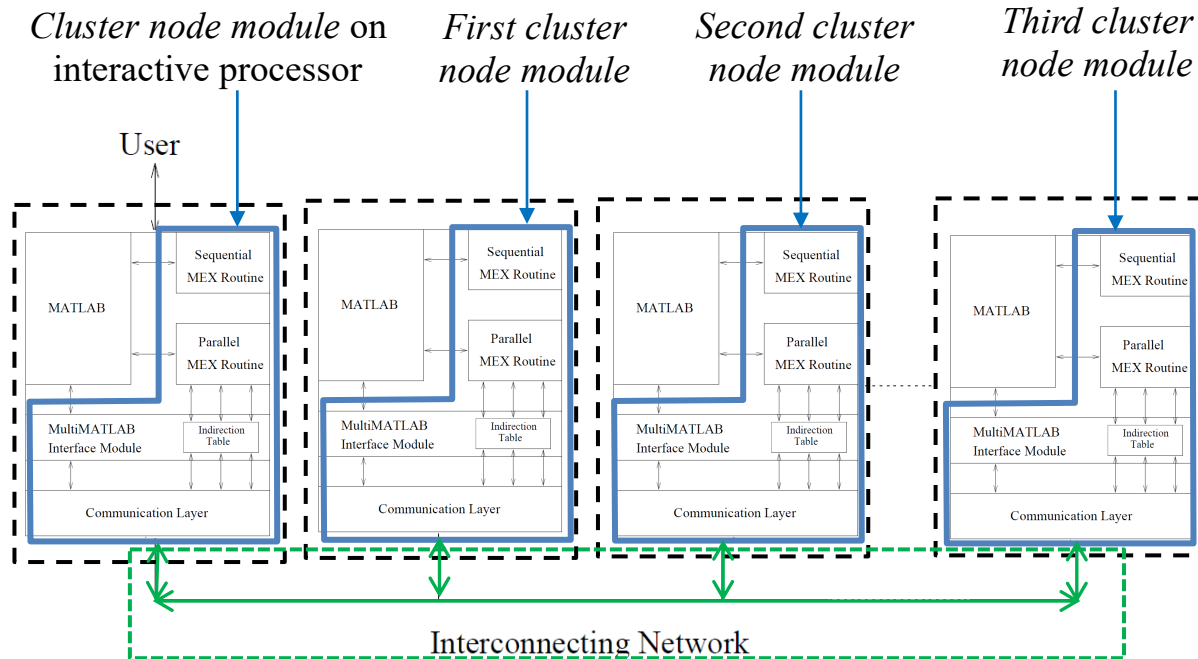


Figure 1: MultiMATLAB Architecture

Cluster node modules communicating to and from each other over an interconnection network

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

356. A POSITA would recognize and understand that the SPMD point-to-point communication paradigm where *cluster node modules* in MATLAB processes on each processor communicate mathematical evaluation results, commands (e.g., tasks), and data directly with each other over the MultiMATLAB architecture's interconnection network renders obvious *a peer-to-peer architecture*. Further, it was shown in limitations [1.3.2], [1.5.2], and [1.7.2] that the mathematical evaluation results include results of matrix-related mathematical evaluations. Accordingly, a POSITA would recognize and understand that the MEX routines, MultiMATLAB interface module, indirection table, and communication layer

(cluster node modules) in the MATLAB processes on each processor of the MultiMATLAB architecture communicate results of matrix-related mathematical evaluations with each other over the MultiMATLAB architecture's interconnection network using SPMD point-to-point communication paradigm, rendering obvious the cluster node modules [] communicate, with each other using a peer-to-peer architecture, results of execution of the matrix operations.

357. Third, in limitation [1.0], it was shown that Menon's MultiMATLAB architecture is implemented on an IBM SP2, which is "scalable from one to 128 processor nodes" each having a symmetric multiprocessor (SMP) with "four, eight, twelve, or sixteen 375 MHz 630FP 64-bit processors." Ex.1007, 2, 9. RS6000 teaches that "[t]he basic components of the RS/6000 SP system [i.e., IBM SP2]" include "[s]witches," which "provide a message-passing network that connects all processor nodes with a minimum of four paths between every pair of nodes. The SP Switch series can also be used to connect the SP system with optional external devices." Ex.1007, 2, 4; 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"). RS6000 further explains that "[t]he SP Switch is a high-speed communication network" where internode communication is supported by "[t]he industry standard Internet Protocol (IP) family of communication protocols (which include TCP/IP and UDP/IP)." Ex.1010, 215.

358. RS6000 also teaches that SMPs “use PCI [Peripheral Component Interconnect] bus architecture” with “adaptor slots” for Ethernet, wide area network (WAN), etc. Ex.1007, 9, 2 (“Peripheral Component Interconnect”), 12. Further, with respect to “[n]etwork communications” in the IBM SP2, “[a]ll SP systems require an SP Ethernet LAN [local area network] for system administration.” Ex.1007, 5, 2 (“local area network”). Ethernet is an example of “adapter types that can be used as part of an RS/6000 SP system” for “network connectivity.” Ex.1007, 5.

359. Accordingly, Menon’s MEX routines that enable each MATLAB process to communicate tasks and data directly with the other MATLAB processes over an interconnecting network implemented on an IBM SP2, in view of Trefethen’s teaching that MEX routines enable communicating results of matrix operations from any process to another process, further in view of RS6000 teaching that the IBM SP2 uses Ethernet and other communication adapters to communicate over a message-passing network, rendering obvious *further comprising a communications network interface*.

18. Claim 16

- a. [16.0] *The computer cluster of claim 15, wherein one or more of the cluster node modules are configured to:***

360. Limitation [16.0] is rendered obvious for the same reasons presented above for limitations [1.4.1], [1.6.1], and [1.8.1].

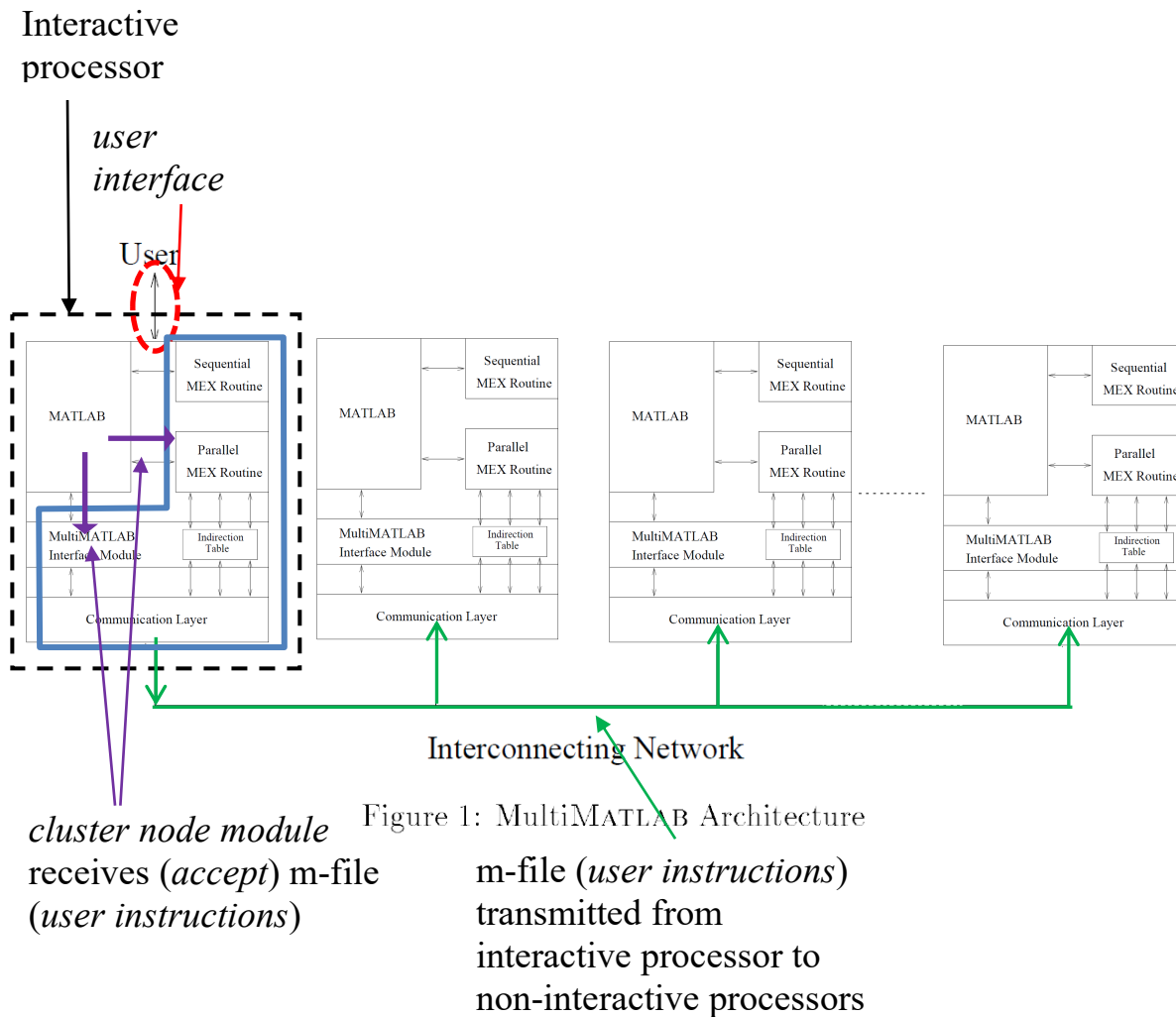
b. [16.1] *accept user instructions*;

361. Menon and Trefethen render obvious *accept user instructions* because Menon discloses running an Eval MEX routine on the interactive processor of the MultiMATLAB architecture to send MATLAB commands to MATLAB processes on the non-interactive processors of the MultiMATLAB architecture over the MATLAB processes' respective communication layer for execution by those MATLAB processes, in view of Trefethen teaching that the MATLAB commands are in the form of an m-file.

362. Menon discloses that the interactive MATLAB process allows the user to directly interact with the MEX routines, 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. “[The] communication layer [of each MATLAB process] runs over the parallel

platform's interconnection network." Ex.1005, 3. The "MEX routines" (such as Eval() "may be run directly from MATLAB." Ex.1005, 3.

363. Trefethen discloses that a user can provide instructions via an "m-file." Ex.1006, 5. 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. As an example, Trefethen discloses the user instruction "Eval('cycle')" on the interactive processor where "cycle" refers to the "filename" of the m-file "cycle.m." Ex.1006, 6.



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

- c. [16.2] *after accepting the user instructions, communicate at least some of the user instructions using the communications network interface for the cluster node modules to communicate with each other; and*

364. Trefethen renders obvious *after accepting the user instructions, communicate at least some of the user instructions using the communications network interface for the cluster node modules to communicate with each other* because Trefethen teaches that after the MEX routines in the cluster node module of

the interactive processor receives the instructions via the Eval MEX routine, it communicates the instructions to the cluster node modules of the non-interactive processors using the MultiMATLAB architecture's interconnection network.

365. First, in limitation [16.1], it was shown that the MEX routines (part of the *cluster node module*) of the interactive processor receive the user instructions using the Eval MEX routine (which could be in the form of an m-file).

366. Second, 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.

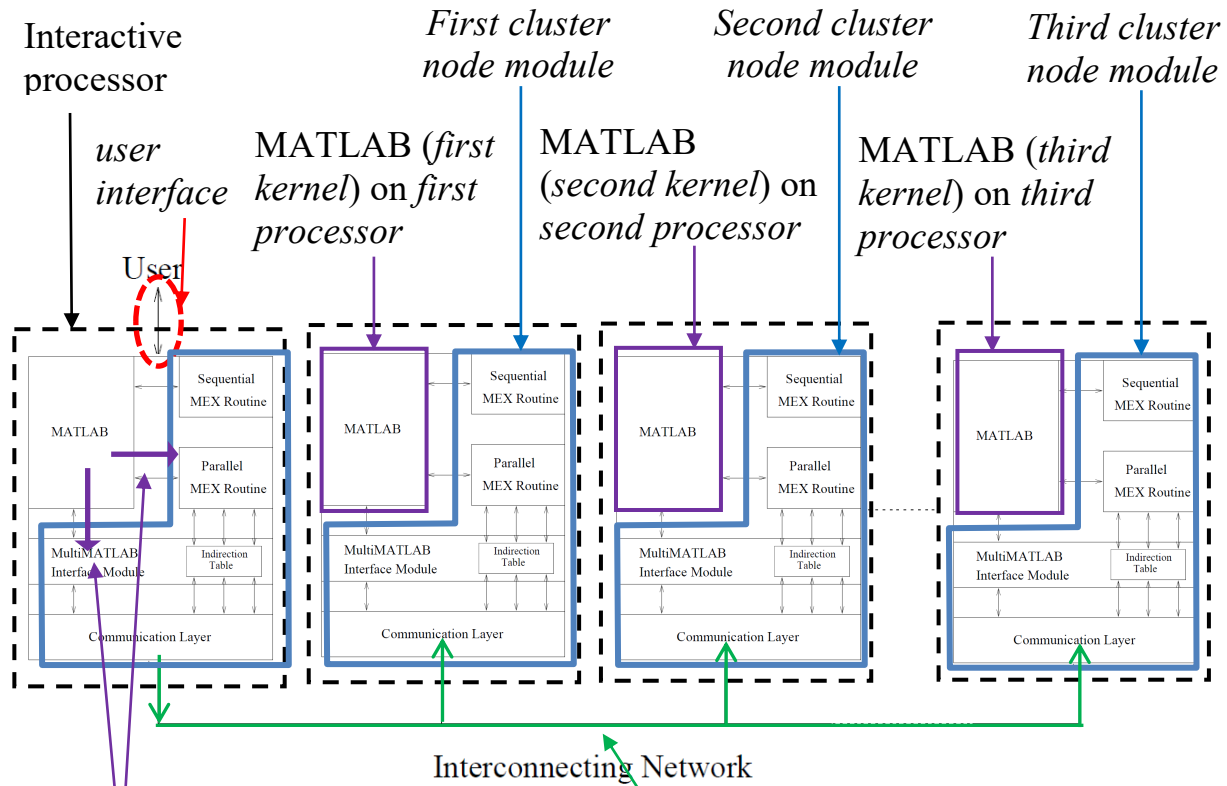


Figure 1: MultiMATLAB Architecture

cluster node module receives (accept) m-file (user instructions)

m-file (user instructions) transmitted from interactive processor to non-interactive processors

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

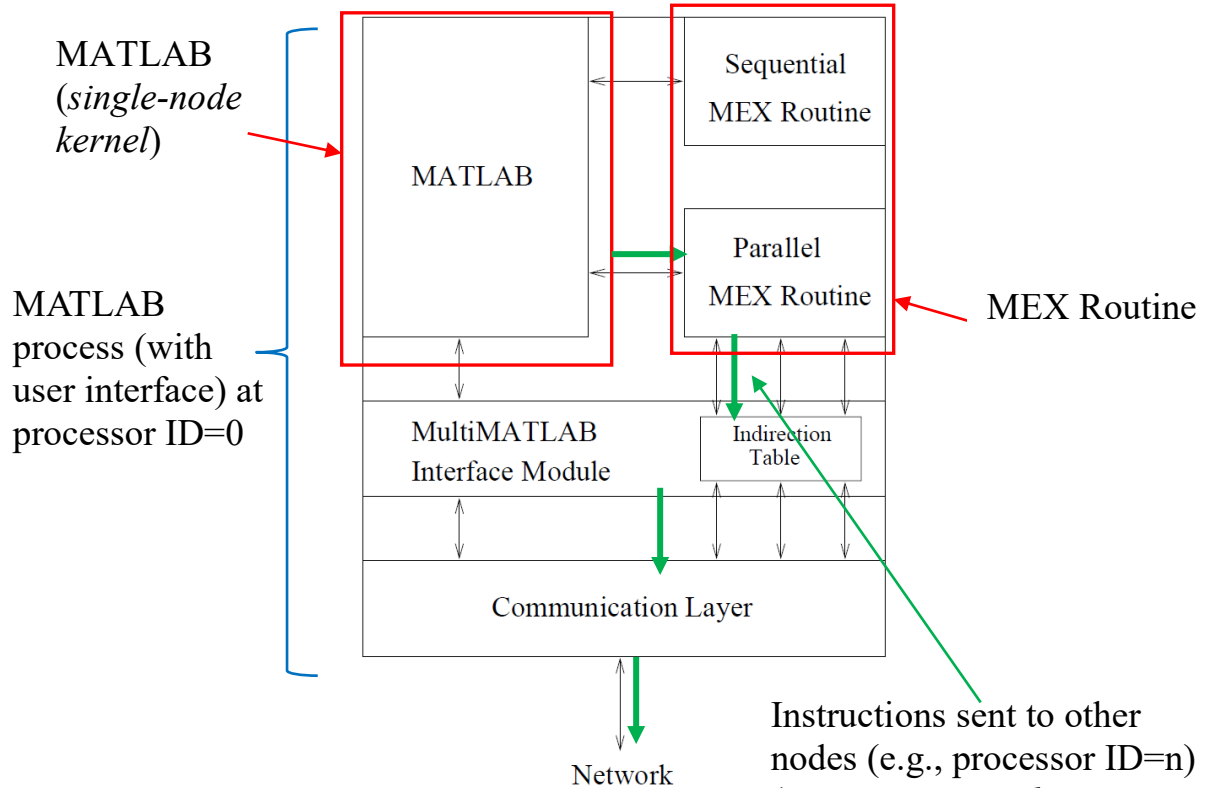


Figure 2: MATLAB Process Architecture
Instructions sent to other nodes (e.g., processor ID=n) (communicate at least some of the user instructions using the communication network interface)

Ex.1005, FIG. 2 (annotated)

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 the user instructions, communicate at least some*

of the user instructions using the communications network interface for the cluster node modules to communicate with each other.

- d. **[16.3] *after communicating at least some of the user instructions using the communications network interface, communicate at least some of the user instructions to one or more of the kernels.***

367. 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 processor (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 communications network interface, communicate at least some of the user instructions to one or more of the kernels.*

368. In limitation [16.2] 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.

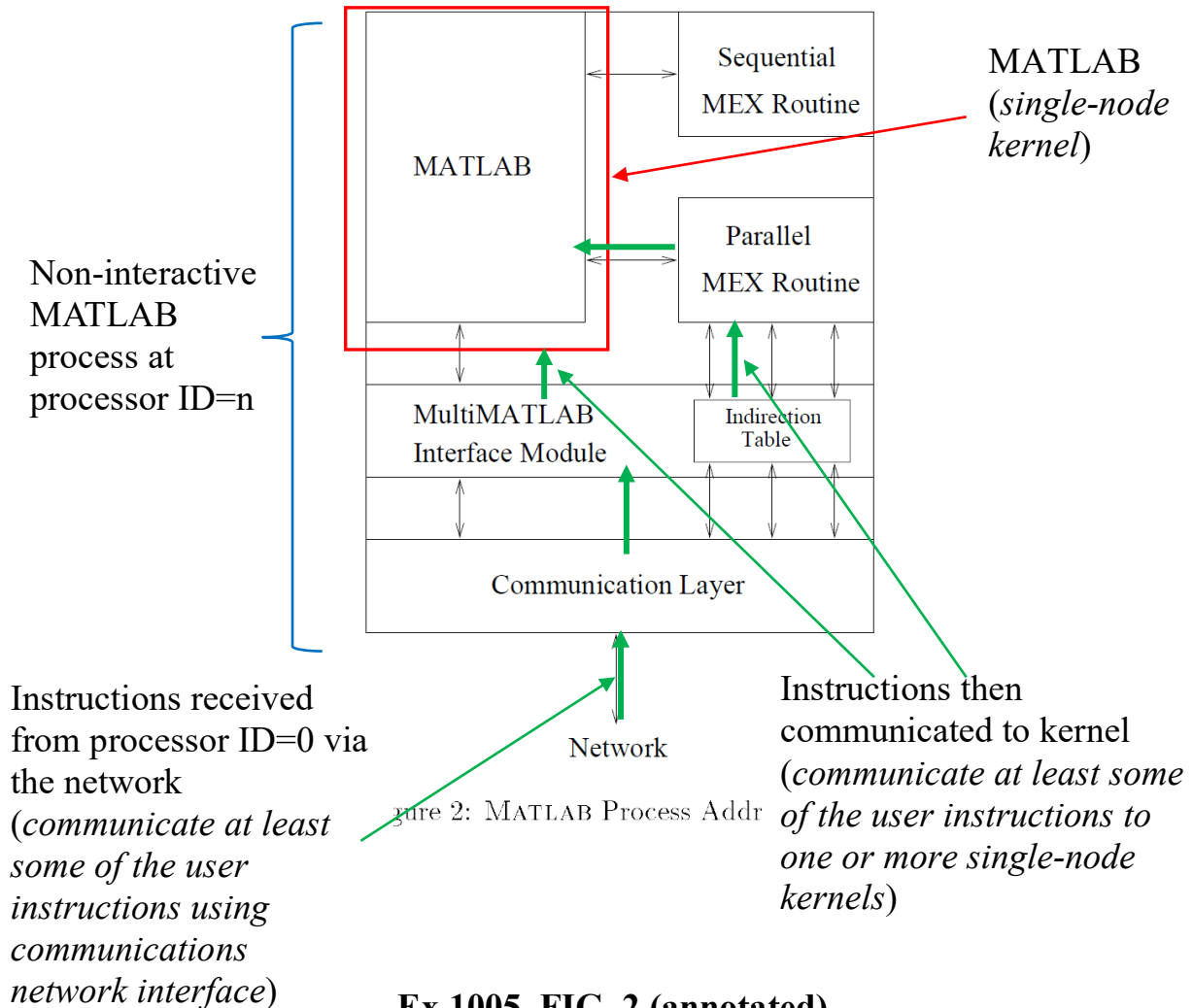
369. Trefethen provides example illustrations showing that commands sent to the other or non-interactive MATLAB processes corresponding to the IDs specified in the *user instructions* are processed by MATLAB at the respective MATLAB processes identified by the IDs. Ex.1006, 3-7. For example, as discussed 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)

370. 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 processor (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 communications network interface, communicate at least some of the user instructions to one or more of the kernels.*

19. Claim 17

- a. **[17.0] *The computer cluster of claim 15, wherein the communications network interface comprises at least one of: a local bus, a local area network (LAN), a wide area network (WAN), a storage area network (SAN), a TCP/IP interface over Ethernet, or a wireless network interface.***

371. RS6000 renders obvious *wherein the communications network interface comprises at least one of: a local bus, a local area network (LAN), a wide area network (WAN), a storage area network (SAN), a TCP/IP interface over Ethernet, or a wireless network interface* because RS6000 teaches that the MultiMATLAB architecture's interconnection network is or includes a bus, a local area network (LAN), a wide area network (WAN), a TCP/IP communication network.

372. In limitation [15.0], it was shown that the MultiMATLAB architecture's interconnection network is or includes a bus, a local area network (LAN), a wide area network (WAN), a TCP/IP communication network, etc., rendering obvious *wherein the communications network interface comprises at least one of: a local bus, a local area network (LAN), a wide area network (WAN), a*

storage area network (SAN), a TCP/IP interface over Ethernet, or a wireless network interface.

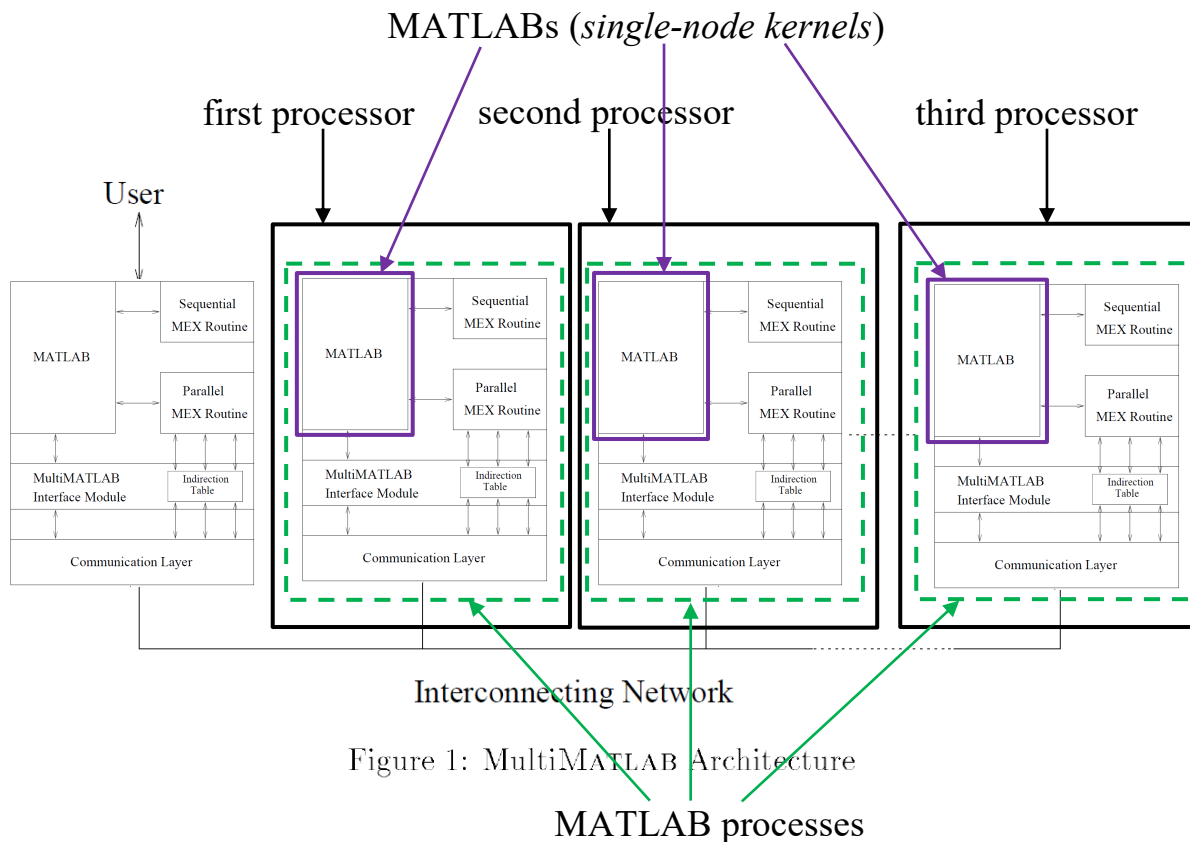
20. Claim 18

- a. [18.0] *The computer cluster of claim 1, wherein at least one of the first, second, or third kernels comprises a single-node kernel.***

373. Menon renders obvious *wherein at least one of the first, second, or third kernels comprises a single-node kernel* because Menon teaches that MATLAB in the MATLAB processes on each non-interactive processor of the MultiMATLAB architecture executes only on their respective non-interactive processor.

374. In limitation [1.3.1], [1.5.1], and [1.7.1], it was shown that MATLAB in the MATLAB processes on the first non-interactive processor, the second non-interactive processor, and the third non-interactive processor of the MultiMATLAB architecture teaches *a first kernel, a second kernel, and a third kernel*. As discussed therein, Menon teaches that “each processor ... individually runs a MATLAB process,” which includes MATLAB. Ex.1005, 3, FIG. 1 (showing each processor having a MATLAB process), FIG. 2 (showing a MATLAB process having MATLAB). A POSITA would recognize and understand that, as shown in the combination figure of Figures 1 and 2 of Menon below, MATLAB (*the first kernel*) in the MATLAB process on the first non-interactive processor executes only on the

first non-interactive processor, MATLAB (*the second kernel*) in the MATLAB process on the second non-interactive processor executes only on the second non-interactive processor, and MATLAB (*the third kernel*) in the MATLAB process on the third non-interactive processor executes only on the third non-interactive processor, rendering obvious *wherein at least one of the first, second, or third kernels comprises a single-node kernel*.



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

21. Claim 19

- a. [19.0] *The computer cluster of claim 1, wherein the cluster node modules communicate with one another to act as a cluster.*

375. Menon and Trefethen renders obvious *wherein the cluster node modules communicate with one another to act as a cluster* because Menon and Trefethen teach that the MEX routines, the MultiMATLAB interface module, the indirection table, and the communication layer of each MATLAB process in the MultiMATLAB architecture work in parallel and/or 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 and generate a final output together as if the processors were a single computer.

376. 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 MATLAB processes on the interactive and non-interactive processors of the MultiMATLAB architecture that execute commands in parallel as if they were a single computer (*act as a cluster*) when a user provides the instructions from the MATLAB process on the interactive processor.

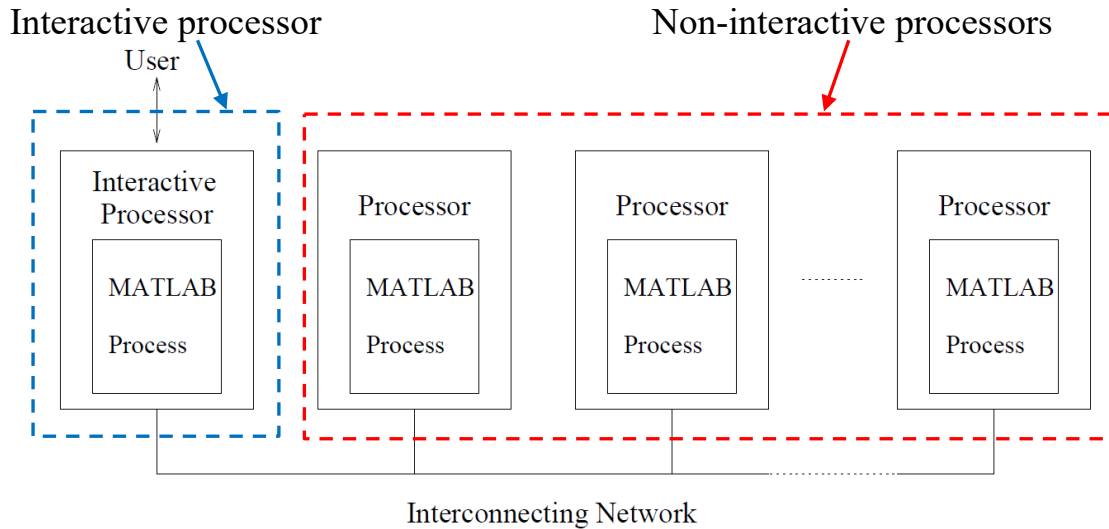


Figure 1: MultiMATLAB Architecture

Ex.1005, FIG. 1 (annotated)

377. 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 "a distributed memory message passing system" used "to execute parallel programs on ... a networked cluster of [IBM SP2] RS/6000 processors." Ex.1008, 1. That is, the

interactive and non-interactive processors of the MultiMATLAB architecture that have MATLAB processes thereon executing commands in parallel are a group of processors that perform mathematical computations as if they were a single computer (*act as a cluster*). Ex.1005, 6.

378. Trefethen provides several illustrative examples of a user interacting directly with a MATLAB process in an interactive processor to have MATLAB commands executed in parallel in the MATLAB environments of MATLAB processes on all processors of the MultiMATLAB architecture as if the processors were a single computer. 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 MATLAB process on the interactive processor, “the MATLAB command `sqrt(2)` is executed on all six processors” of the MultiMATLAB architecture and “the MultiMATLAB command ID” is executed “on each of the processors running.” Ex.1006, 3-4.

379. Menon provides examples of MATLAB processes on processors of the MultiMATLAB architecture performing computations in parallel while communicating mathematical results with each other. 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 communicating and performing computations in parallel together (*communicate with one another to act as a cluster*). Ex.1005, 11. Specifically, 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 while communicating with each other:

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 ρ and α 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 executing commands in parallel while communicating values

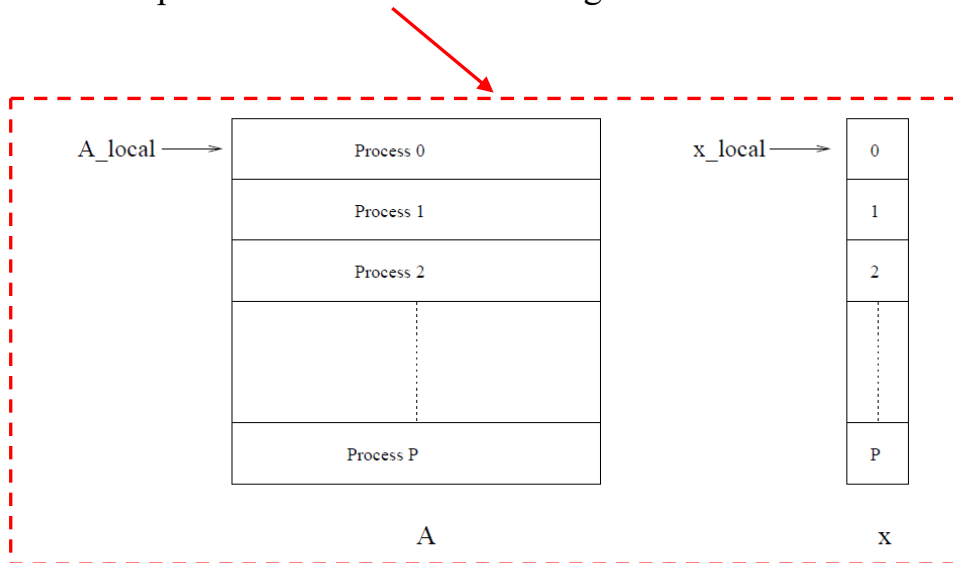
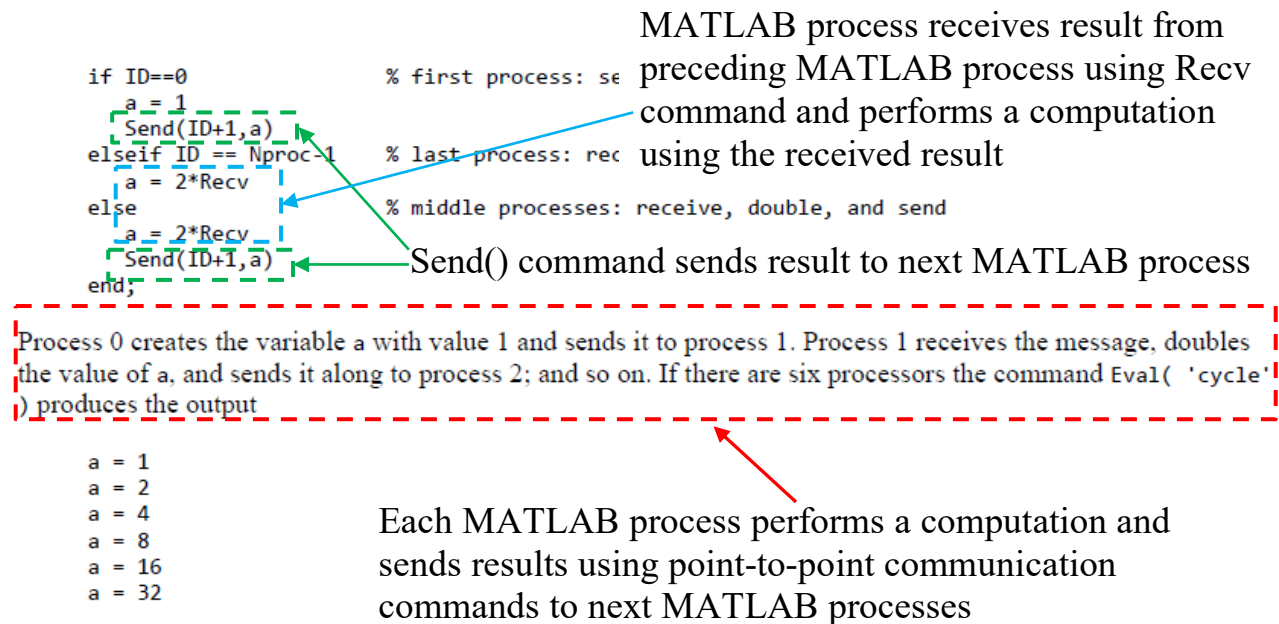


Figure 6: Matrix A and vector x distributed by row over P processors

Ex.1005, FIG. 6

380. Trefethen also provides an example of MATLAB processes on processors of the MultiMATLAB architecture executing MATLAB commands and performing mathematical computations while communicating variables with each other. Ex.1006, 6. Specifically, Trefethen provides the following example code in which MATLAB processes perform a mathematical computation and send to the next MATLAB process the result of the computation so that that next MATLAB process uses the received result to perform its own computation and send the new result to the next MATLAB process, and so on:



Ex.1006, 6 (annotated). In other words, in Trefethen's example code, MATLAB processes on processors of the MultiMATLAB architecture execute MATLAB commands and perform mathematical computations while communicating results

with each other (*communicate with one another*) to generate an output containing the results from all the MATLAB processes (*to act as a cluster*). Ex.1006, 6.

381. Further, as discussed in limitation [1.4.1], Menon discloses that each MATLAB process on a processor of the MultiMATLAB architecture has a set of components (MEX routines, a MultiMATLAB interface modules, an indirection table, and a communication layer) (*cluster node modules*) to communicate with other processors in the MultiMATLAB architecture. That is, as shown in limitation [15.0], MATLAB processes on each processor of the MultiMATLAB architecture use their respective MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*cluster node modules*) to communicate mathematical evaluation results, commands (e.g., tasks), and data with each other over the MultiMATLAB architecture's interconnection network using SPMD point-to-point communication paradigm (i.e., using a peer-to-peer architecture).

382. Accordingly, Menon and Trefethen teach that the MEX routines, the MultiMATLAB interface module, the indirection table, and the communication layer of each MATLAB process (*the 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 and/or together, where each MATLAB process performs its computation and communicates with other MATLAB processes (*communicate with one another*) for all the MATLAB processes on the

processors of the MultiMATLAB architecture to perform computations in parallel and/or together as if the processors were a single computer (*to act as a cluster*), rendering obvious *wherein the cluster node modules communicate with one another to act as a cluster*.

C. Ground 2: Claims 4 and 20 are obvious over Menon in view of Trefethen, RS6000, and MPIref further in view of POEref

383. The combination of Menon, Trefethen, RS6000, MPIref, and POEref renders obvious claims 4 and 20 as discussed below.

1. Combining POEref with Menon

a. POEref is Analogous Art

384. POEref is analogous art at least because it is directed to the same field of endeavor as the '582 patent. The '582 patent is directed to “the field of cluster computing.” Ex.1001, 1:15-16. Similarly, POEref is directed to “compil[ing], execut[ing], and analyz[ing] parallel programs.” Ex.1008, ix.

385. Furthermore, POEref is analogous art to the '582 patent because it is reasonably pertinent to a problem the inventors of the '582 patent attempted to solve. The '582 patent purports to provide a solution “for adding cluster computing functionality to a computer program.” Ex.1001, 1:16-17. The '582 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, 2:14-17. The '582 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, 1:15-18, 29-31, 2:13-14.

386. 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.

387. Accordingly, in my opinion, a POSITA would understand that POEref is a prior art that is analogous to the '582 patent.

b. Motivation to Combine POEref with Menon

388. 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.

389. 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.

390. 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.

391. 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, 14; Ex.1008, ix.

2. Claim 4

- a. [4.0] *The computer cluster of claim 1, further comprising a cluster configuration module that initializes at least one of the first cluster node module, the second cluster node module, or the third cluster node module.***

392. The '582 patent describes a cluster configuration module as “includ[ing] program code for initializing a plurality of cluster node modules to add cluster computing support to computer systems.” Ex.1001, 22:4-6. Likewise, Menon teaches that a user starts a MATLAB process on each processor of the MultiMATLAB architecture by using IBM’s POE, an execution environment that includes a program code for loading a program on each one of a number of processor nodes, as taught by POEref, and that each MATLAB process on the first non-interactive processor, the second non-interactive processor, and the third non-interactive processor includes MEX routines, a MultiMATLAB interface module, indirection table, and a communication layer (*a first cluster node module, a second cluster node module, and a third cluster node module, respectively*), rendering obvious *further comprising a cluster configuration module that initializes at least*

one of the first cluster node module, the second cluster node module, or the third cluster node module.

393. First, 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 on a "home node" and "a number of **remote nodes**" 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, 3, 28 (emphasis original).

394. In other words, when a user on an IBM SP2 that implements the MultiMATLAB architecture 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”) (the processor with which the user is interacting is called the interactive processor and the other processors (i.e., “remote nodes”) of the MultiMATLAB architecture are referred to as the non-interactive processors. Ex.1005, 3, FIG. 1). 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 first non-interactive processor (*first processor*), the second non-interactive processor (*second processor*), and the third non-interactive processor (*third processor*) of the MultiMATLAB architecture is illustrated schematically in FIG. 1 of Menon below.

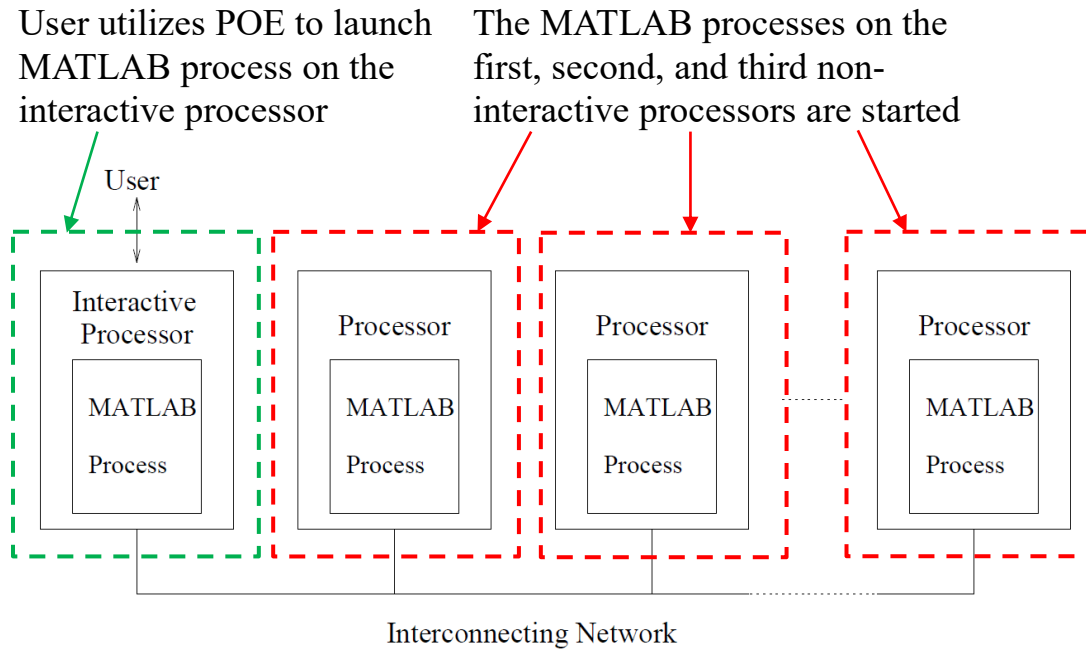


Figure 1: MultiMATLAB Architecture

Ex.1005, FIG. 1 (annotated)

395. Second, in limitation [1.4.1], it was shown that the MEX routines, MultiMATLAB interface module, indirection table, and communication layer of a MATLAB process on a processor constitute a set of components of that processor that handle communications with other processors and that they correspond to the *cluster node module* of that processor. That is, the first non-interactive processor (*first processor*), the second non-interactive processor (*second processor*), and the third non-interactive processor (*third processor*) of the MultiMATLAB architecture each have MEX routines, MultiMATLAB interface module, indirection table, and communication layer that correspond to the *first cluster node module*, *second*

POE includes a program code that “loads your program, and reproduces your local environment, on each processor node.” Ex.1008, 28.

397. Accordingly, Menon teaches that a user starts a MATLAB process on each processor of the MultiMATLAB architecture by using IBM’s POE, an execution environment that includes a program code for loading a program on each one of multiple processor nodes, as taught by POEref, and that each MATLAB process on the first non-interactive processor, the second non-interactive processor, and the third non-interactive processor includes MEX routines, a MultiMATLAB interface module, indirection table, and a communication layer (*a first cluster node module, a second cluster node module, and a third cluster node module, respectively*), rendering obvious *further comprising a cluster configuration module that initializes at least one of the first cluster node module, the second cluster node module, or the third cluster node module.*

3. Claim 20

- a. [20.0]** *The computer cluster of claim 1, wherein one or more of the cluster node modules are configured to receive a command to start a cluster initialization process for the computer cluster.*

398. Menon and POEref render obvious *wherein one or more of the cluster node modules are configured to receive a command to start a cluster initialization process for the computer cluster* because Menon and POEref teach that each of the first non-interactive processor, the second non-interactive processor, and the third

non-interactive processor of the MultiMATLAB architecture has a MATLAB process with a respective communication layer (*first cluster node module, second cluster node module, and third cluster node module, respectively*) that receives a start command to start that MATLAB process.

399. First, 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

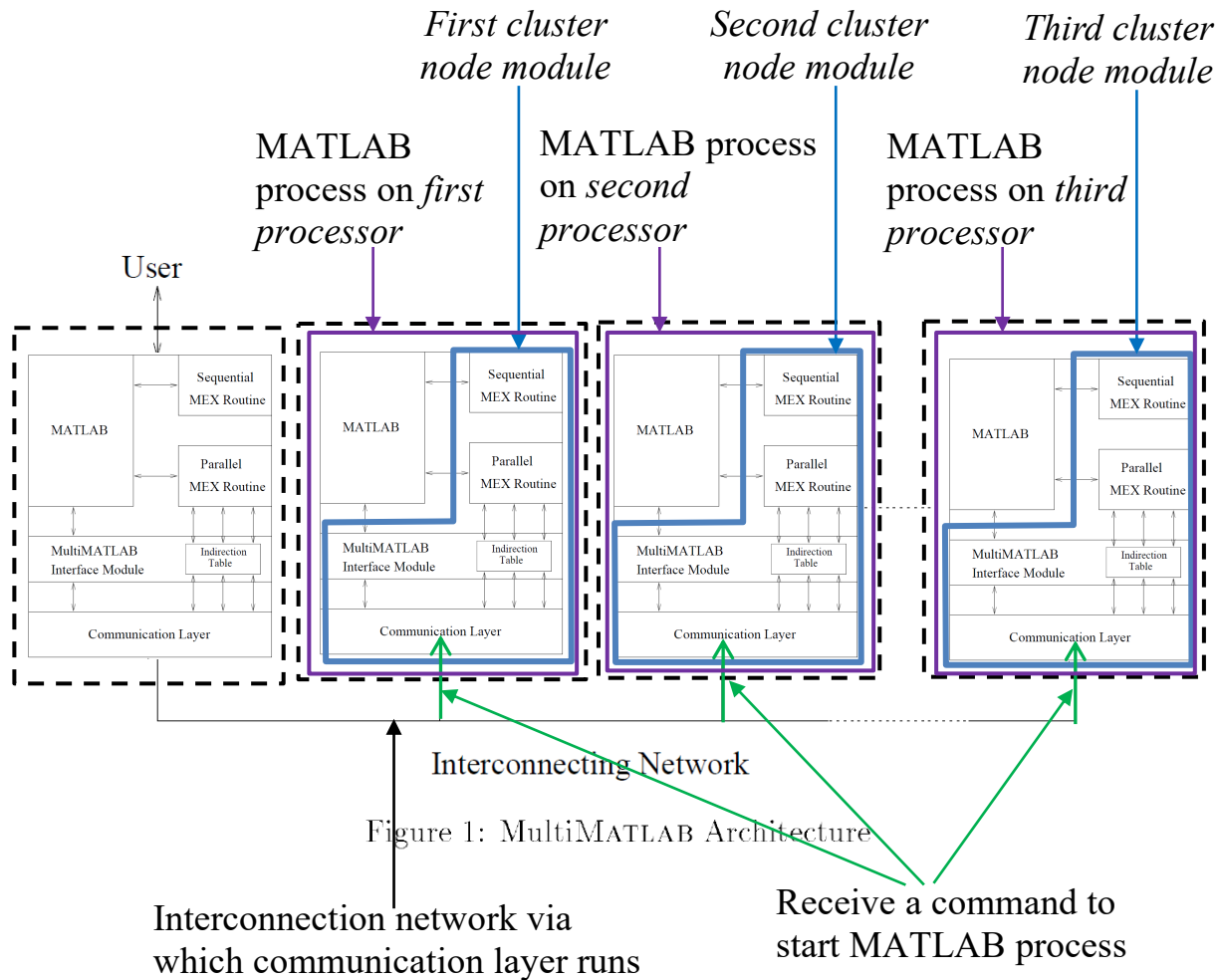
subsystem library implementation (IP or US) is
automatically loaded at this time.

Ex.1008, 28 (emphasis original).

400. 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.

401. Second, in limitation [1.4.1], it was shown that each processor of the MultiMATLAB architecture includes a MATLAB process that has a set of components for handling communications with other processors of the MultiMATLAB architecture, these components being MEX routines, MultiMATLAB interface module, indirection table, and communication layer

(*cluster node module*). 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. The “communication layer [] runs over the parallel platform's interconnection network.” Ex.1005, 3.



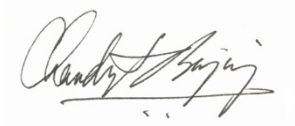
Ex.1005, FIGs. 1 and 2 (combined and annotated)

402. Accordingly, Menon and POEref teach using IBM's POE to start all the MATLAB processes on the processors of the MultiMATLAB architecture, where the MATLAB processes on each of the first non-interactive processor, the second non-interactive processor, and the third non-interactive processor receive the start command via their respective communication layer (part of the *first cluster node module*, part of the *second cluster node module*, and part of the *third cluster node module*, respectively). Thus, a POSITA would recognize and understand that a

communication layer (part of *one or more of the cluster node modules*) of each MATLAB process on the first non-interactive processor, the second non-interactive processor, and the third non-interactive processor receives a start command (*are configured to receive a command to start*) that causes the MATLAB process on that non-interactive processor of the MultiMATLAB architecture (*for the computer cluster*) to start (*a cluster initialization process*) and await further commands, rendering obvious *wherein one or more of the cluster node modules are configured to receive a command to start a cluster initialization process for the computer cluster*.

VI. DECLARATION

403. 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.

A handwritten signature in black ink, appearing to read "Chandrajit Bajaj", is written over a horizontal line.

Dated: April 29, 2025

Chandrajit L. Bajaj, Ph.D.