

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

INTEL CORPORATION,
ADVANCED MICRO DEVICES, INC.,
Petitioners

IPR2025-00914
U.S. Patent No. 12,021,679

**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.....	4
II.	QUALIFICATIONS.....	7
III.	RELEVANT LEGAL STANDARDS.....	13
IV.	FOCUSED ANALYSIS.....	15
A.	Level of Ordinary Skill in the Art	15
B.	Background	16
C.	The '679 Patent	27
1.	Summary of the '679 Patent	27
2.	Prosecution History.....	30
D.	Claim Construction	30
1.	“peer-to-peer architecture”	31
E.	Key Teachings of the Prior Art	31
1.	Kernel.....	31
2.	Cluster Node Module.....	33
3.	Peer-to-Peer Mathematical Computation.....	35
F.	Reasons to Combine.....	37
1.	Combining Trefethen with Menon	37
2.	Combining RS6000 with Menon	42
3.	Combining POEref with Menon	45
4.	Combining MPIref with the MultiMATLAB system of Menon, Trefethen, RS6000 and POEref.....	48
V.	DETAILED ANALYSIS	51
A.	PRIOR ART	51
1.	Menon	51
2.	Trefethen.....	60

3.	RS6000.....	64	
4.	POEref	66	
5.	MPIref.....	69	
B. Ground 1: Claims 1-19 are obvious over Menon in view of Trefethen, RS6000, and POEref			72
5.	Combining Trefethen with Menon	73	
6.	Combining RS6000 with Menon	77	
7.	Combining POEref with Menon	81	
8.	Claim 1	84	
9.	Claim 2.....	140	
10.	Claim 3.....	141	
11.	Claim 4.....	146	
12.	Claim 5.....	153	
13.	Claim 6.....	157	
14.	Claim 7.....	165	
15.	Claim 8.....	168	
16.	Claim 9.....	170	
17.	Claim 10.....	173	
18.	Claim 11.....	177	
19.	Claim 12.....	179	
20.	Claim 13.....	181	
21.	Claim 14.....	183	
22.	Claim 15.....	184	
23.	Claim 16.....	187	
24.	Claim 17.....	191	
25.	Claim 18.....	191	
26.	Claim 19.....	195	

VI. DECLARATION 198

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. 12,021,679 (“the ’679 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 19 (the “Challenged Claims”) of the ’679 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 ’679 patent, Ex.1001;

- (2) the prosecution history of the '679 patent, Ex.1002;
- (3) "MultiMATLAB: Integrating MATLAB with High-Performance Parallel Computing," Menon et al., SC '97: Proceedings of the 1997 ACM/IEEE Conference on Supercomputing 1997 ("Menon"), Ex.1005;
- (4) "MultiMATLAB: MATLAB on Multiple Processors," Trefethen et al., Cornell University 1996 ("Trefethen"), Ex.1006;
- (5) "RS/6000 SP: Planning Vol. 1, Hardware and Physical Environment," IBM 2001, ("RS6000"), Ex.1007;
- (6) "IBM Parallel Environment for AIX – Operation and Use, Volume 1, Using the Parallel Operating Environment," IBM 2001, ("POEref"), Ex.1008;
- (7) "Single program multiple data" in Dictionary of Algorithms and Data Structures, P. E. Black, December 2004, Ex.1009;
- (8) "The RS/6000 SP Inside Out" to Barrios et al., IBM 1999, Ex.1010;
- (9) "High Performance Cluster Computing: Programming and Applications," Rajkumar Buyya, Vol. 2, 1999, Ex.1011;
- (10) "Analysis of 100Mb/s Ethernet for the Whitney Commodity Computing Testbed," Fineberg et al., NAS Technical Report NAS-97-025 1997, Ex.1012;
- (11) "A Parallel Linear Algebra Server for Matlab-like Environments," G. Morrow et al., SC '98: Proceedings of the 1998 ACM/IEEE Conference on Supercomputing 1998, Ex.1013;
- (12) "Parallel MATLAB: Doing It Right," R. Choy et al., Proceedings of the IEEE, Vol. 93, No. 2, 2005, Ex.1014;
- (13) "Mathematica Parallel Computing Toolkit - Unleash the Power of Parallel Computing," R. Maeder, 2005, Ex.1015;

- (14) “Mastering MATLAB® 5 – A Comprehensive Tutorial and Reference,” D. Hanselman et al., 1998, Ex.1016;
- (15) “MPI: A Message-Passing Interface Standard,” Message Passing Interface Forum, May 5, 1994 (“MPIref”), Ex.1017;
- (16) “Modern Operating Systems,” A.S. Tanenbaum, 2001, Ex.1018;
- (17) Cornell Websites, Ex.1023;
- (18) “pyMPI–An introduction to parallel Python using MPI,” P. Miller 2002, Ex.1024;
- (19) Amended Joint Claim Construction Chart for *Advanced Cluster Systems, Inc. v. NVIDIA Corporation*, 1:19-cv-02032 (D. DE.), Ex.1025;
- (20) SC ‘97 Conference Materials, Ex.1026;
- (21) Declaration of Gordon MacPherson, IEEE, Ex.1027;
- (22) A copy of Menon with list of publications citing Menon, Ex.1028;
- (23) Affidavit of Mina Ching, Internet Archive, Ex.1029;
- (24) “Visual Supercomputing to be Demonstrated at SC97,” HPC wire, 1997, Ex.1030;
- (25) SC1997 flyer, 1997, Ex.1031;
- (26) IEEE Xplore’s “About IEEE Xplore” page, Ex.1032;
- (27) “As Cornell Theory Center winds up Microsoft pact, it seeks faculty advice on future, direction,” Cornell Chronicle, 2005; Ex.1033;
- (28) ACM’s description of “ACM Digital Library,” Ex.1034;
- (29) A copy of Trefethen with list of publications citing Trefethen, Ex.1035;
- (30) “A Beginner’s Guide to the IBM SP,” Stephanie Wolf et al., University of Tennessee, 2000, Ex.1036;
- (31) List of publications citing MPIref, Ex.1037;

- (32) IBM website's documentation library for RS6000, Ex.1038
- (33) IBM website's documentation library for POEref, Ex.1039;
- (34) ACS Preliminary Response, IPR2021-00019, Ex.1040;
- (35) Affidavit of Tanya Zeif, Internet Archive, Ex.1048; and
- (36) 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 CPUs,” *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 and multi-core 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, 34 peer-reviewed book chapters, and 157 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 198 invited speaker keynotes on GUIs. I am a Fellow of the American Association for the Advancement of Science, a Fellow of

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

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

III. RELEVANT LEGAL STANDARDS

18. I am not an attorney. In preparing and expressing my opinions and considering the subject matter of the '679 patent, I am relying on certain legal principles that counsel has explained to me.

19. I understand that prior art to the '679 patent includes patents and printed publications in the relevant art that predate the priority date of the '679 patent. For purposes of this Declaration, I am applying the earliest claimed priority date of June 13, 2006, as the priority date of the '679 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 '679 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 '679 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 '679 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." Ex.1001, 1:21-25. Computer clusters (and cluster computing techniques), however, were well known in the art prior to the earliest priority date of the '679 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 a message passing interface for the nodes to communicate in a peer-to-peer fashion, where the nodes have single-node programs executing in a single-program multiple-data (SPMD) cluster computing paradigm, and parallel algorithms running on the computer clusters can employ data pipeline structures for data flows between adjacent nodes. Ex.1011, 4-7 (cluster computers), 19-22 (SPMD cluster computing paradigm and data pipelining), 53-56 (peer-to-peer communication), and 56-60 and 64-68 (mechanisms for cluster nodes to communicate with each other).

31. Advances in "commodity" computer technology and "off-the-shelf" networking technology in the early 1990s, more than a decade before the '679 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.

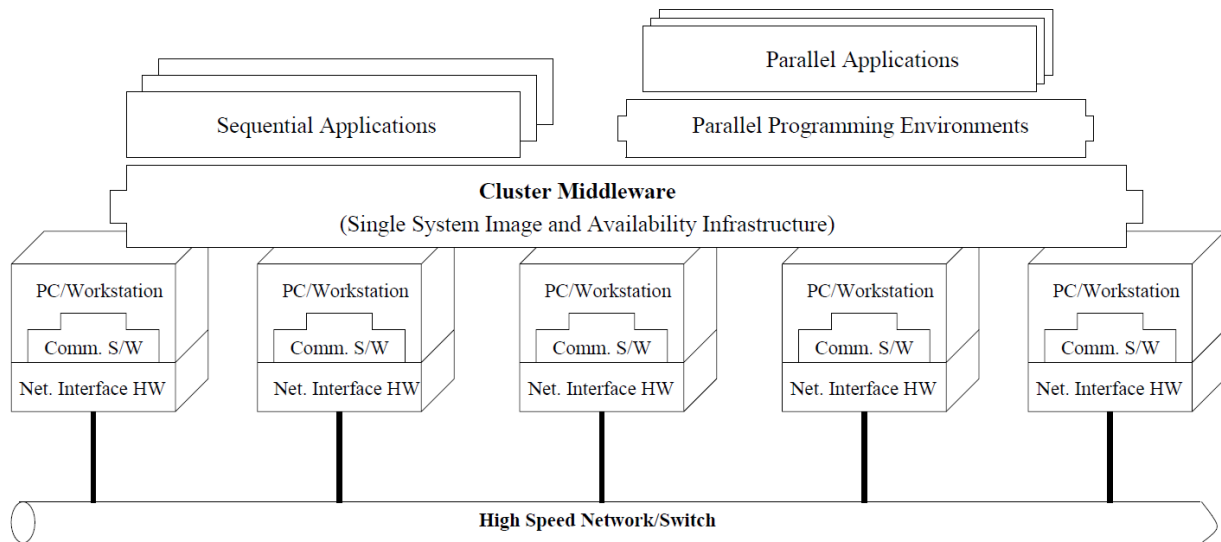


Figure 1.1 Cluster computer architecture.

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 '679 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 '679 patent identifies, and recites in its claims, 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 '679 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 '679 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 ’679 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.

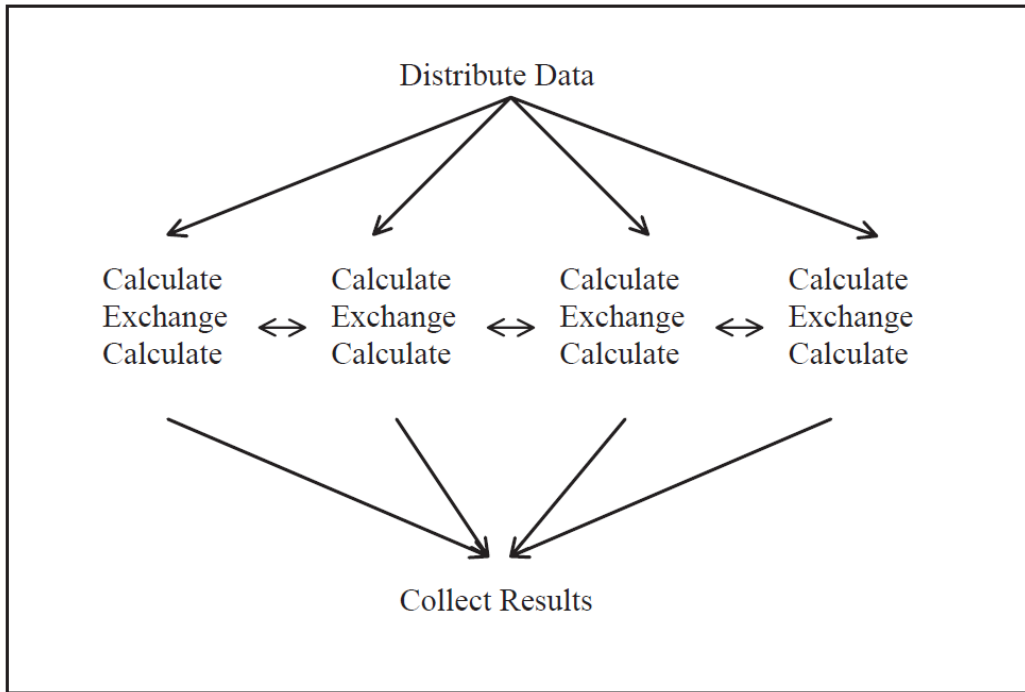


Figure 1.5 Basic structure of a SPMD program.

Ex.1011, FIG. 1.5

40. Sequential Processing from One Node to the Next

41. Also known in the art prior to the '679 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.

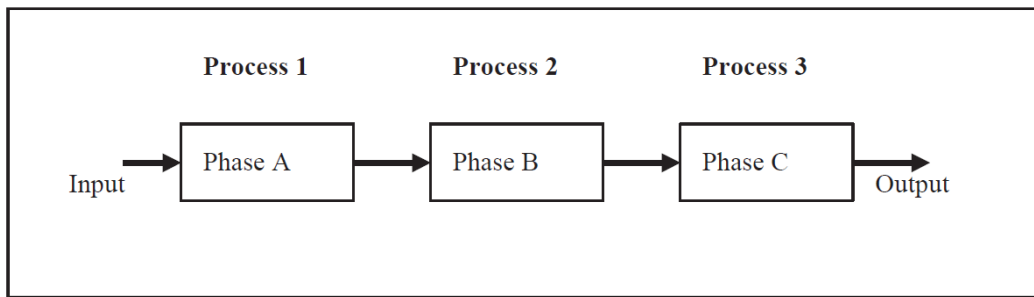


Figure 1.6 Data pipeline structure.

Ex.1011, FIG. 1.6

42. 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.

43. 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

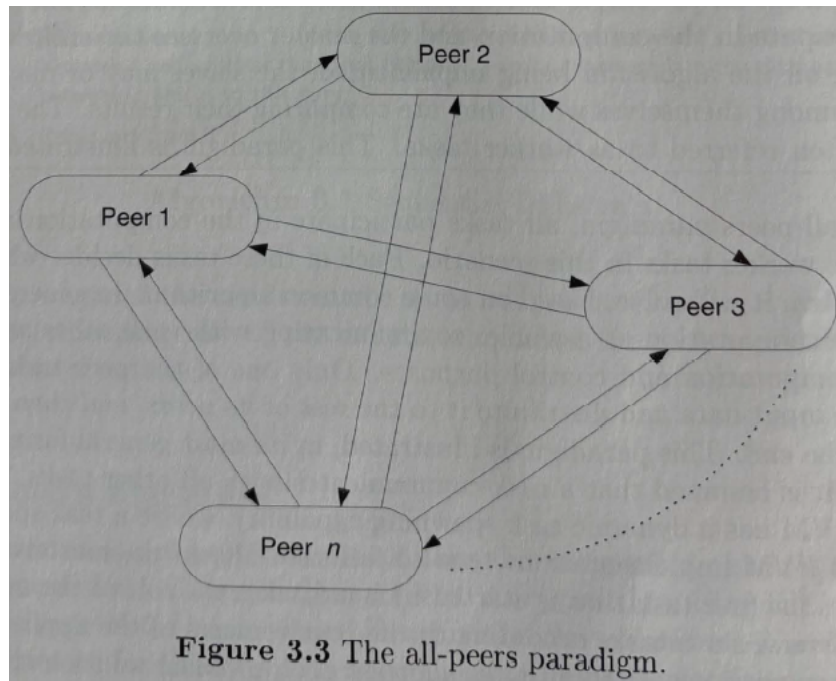
may involve “situations where it makes sense to mix data and task parallelism simultaneously.” Ex.1011, 23. For example, the SPMD cluster computing paradigm and the data pipeline structure for data flows, depicted above in Buyya’s FIG. 1.5 and FIG. 1.6, respectively, can be combined to implement an example 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.

44. 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

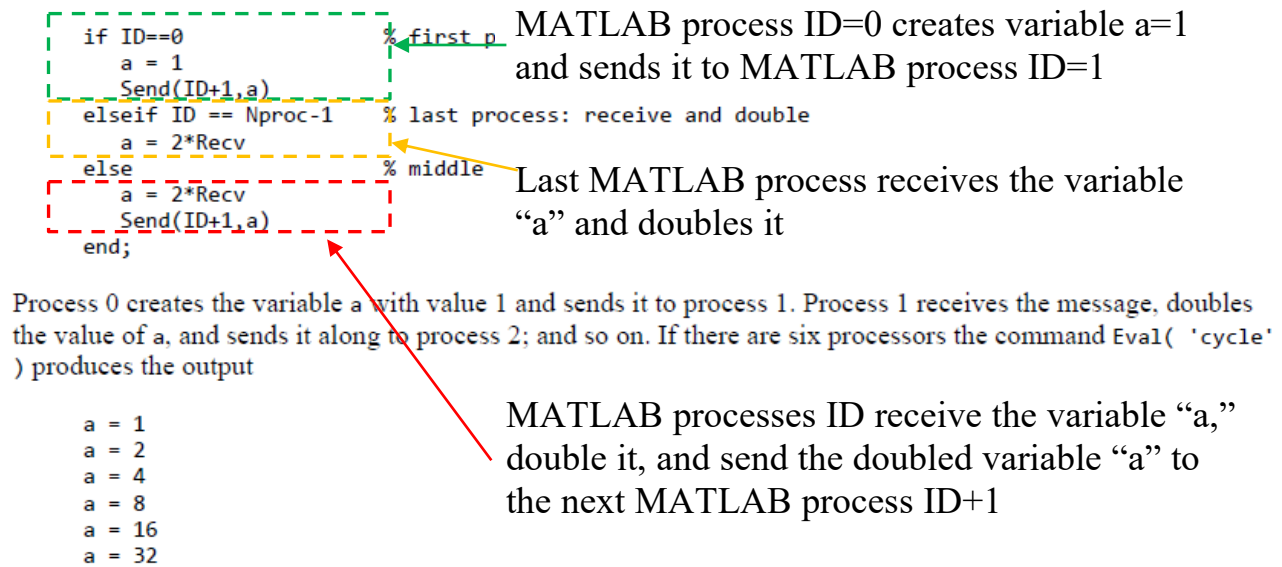
tasks may still read in the input data and distribute it to the rest of its peers, and then collect the results in the end.

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

45. 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).

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

47. Accordingly, it was well known, and a POSITA would have understood, before the priority date of the '679 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.

C. The '679 Patent

1. Summary of the '679 Patent

48. The '679 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 '679 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, 1:21-29. The purported novelty of the '679 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 '679 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, 4:19-28.

49. 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, 5:23-25, 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, 11:45-52. 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, 25:45-46, 24:62-65.

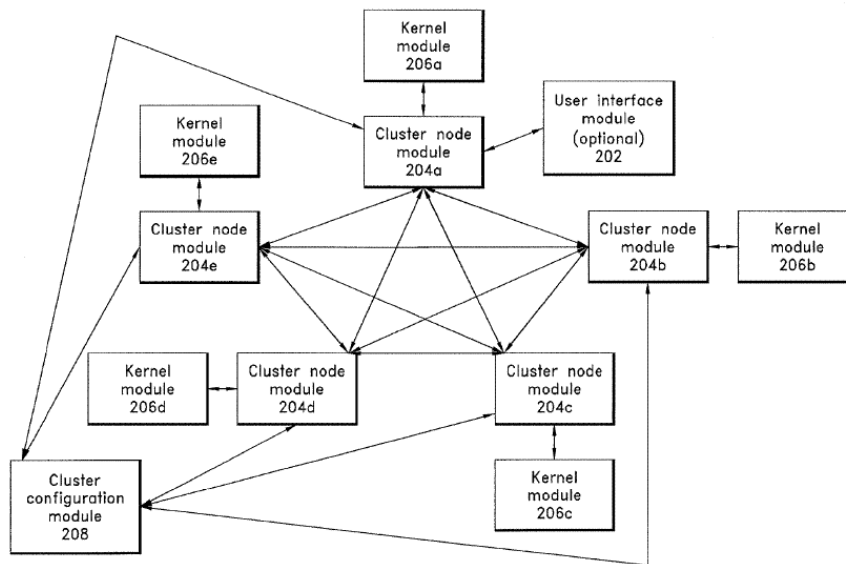


FIG. 2

Ex.1001, FIG. 2

50. Representative Claim 1 of the '679 patent is reproduced below:

1. A computer cluster comprising:
 - a plurality of nodes each comprising a hardware processor and configured to access a non-transitory computer-readable medium comprising program code that, when executed, is capable of causing the hardware processor to evaluate mathematical expressions comprising matrix operations; and
 - a communications network interface for the nodes to communicate results of mathematical expression evaluation with each other using a peer-to-peer architecture; wherein the plurality of nodes comprises:
 - a first node comprising a first hardware processor configured to access a first memory comprising first program code for a user interface and second program code configured to interpret user instructions, identify an array of data elements, and distribute calls to at least one of a plurality of other nodes for execution of matrix operations on the array of data elements; and
 - a second node comprising a second hardware processor, wherein the second node is configured to receive the calls from the first node, execute the matrix operations on the array of data elements, and communicate a result of the matrix operations to a third node;
 - wherein the third node comprises a third hardware processor, wherein the third node is configured to receive the result of the matrix operations from the second node, execute at least a mathematical expression evaluation using the received result, and communicate the result of the mathematical expression evaluation to the first node;
 - wherein the first node is configured to return the result of the mathematical expression evaluation to the user interface;
 - wherein one or more of the nodes are configured to:
 - accept, via the user interface or via a script, user instructions; and
 - after accepting the user instructions, communicate at least some of the user instructions using the communications network interface for the nodes to communicate with each other.

As I explain in detail below, there is nothing novel about the alleged invention disclosed and claimed in the '679 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 ’679 patent. Ex.1005, Abstract; Ex.1006, Abstract.

2. Prosecution History

51. The ’679 patent was filed on October 2, 2023, and issued on June 25, 2024. Ex.1002, 14, 2-4. The Examiner allowed the claims to issue merely noting that the “prior art of records fail to teach or suggest individually or in combination as set forth in independent claims 2 files on 03/19/2024. Claims 3-20 are also allowed for the same reasons.” Ex.1002, 28.

52. Notably, as I demonstrate below, there is nothing new about claim 1 (or any of the other claims) of the ’679 patent, as well as its combination with all the other limitations of the claim(s), which appear to be the reason for allowance, were known in the art and it would have been obvious to combine the prior art as claimed.

D. Claim Construction

53. It is my understanding that in order to properly consider whether the ’679 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”

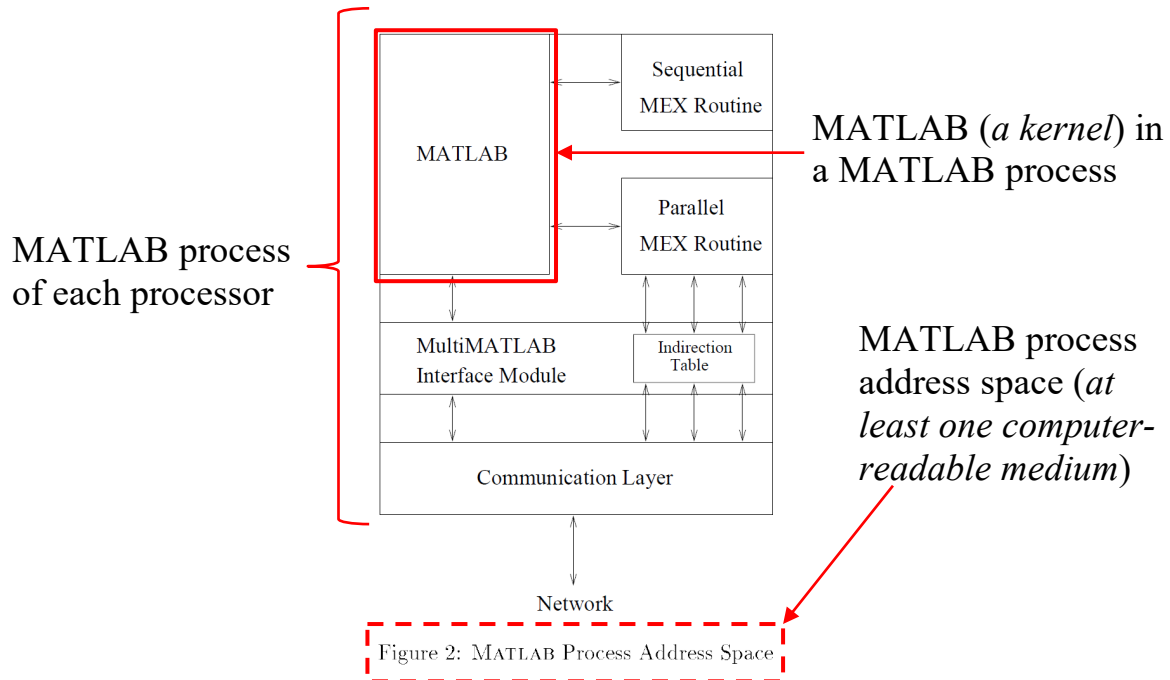
54. The limitation “peer-to-peer architecture” appears in independent claim 1.

55. 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, 1. I will apply this construction for the term when applying the prior art presented herein to evaluate the patentability of the claims.

E. Key Teachings of the Prior Art

1. Kernel

56. 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)

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

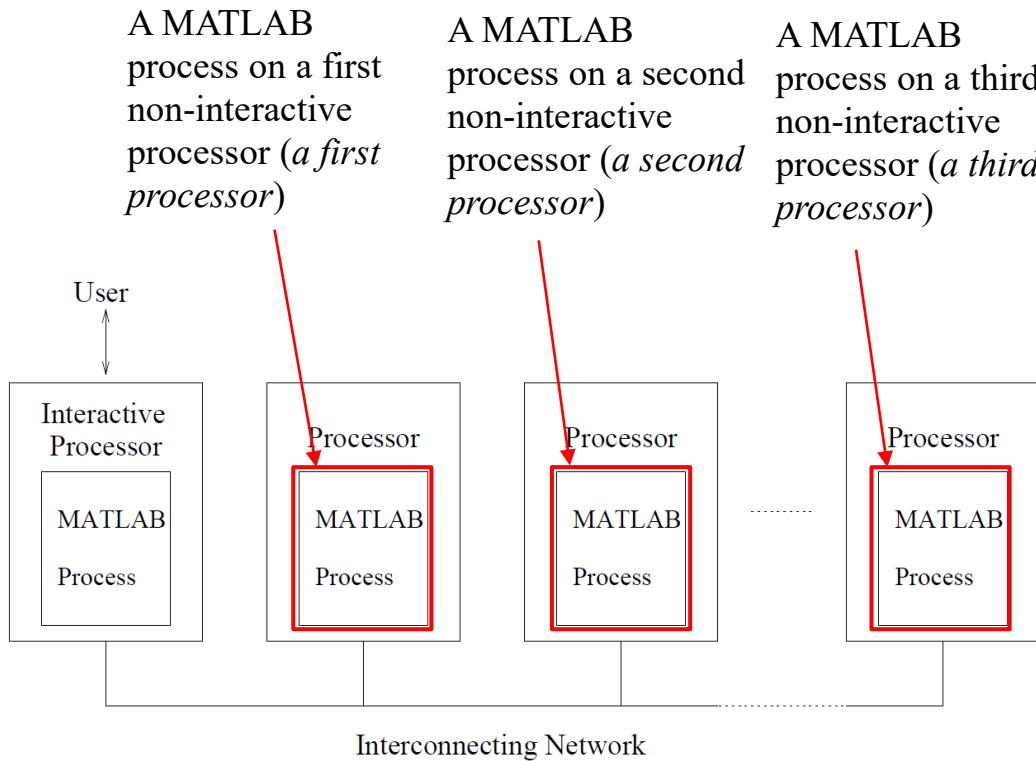


Figure 1: MultiMATLAB Architecture

Ex.1005, FIG. 1 (annotated)

58. 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

59. 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.

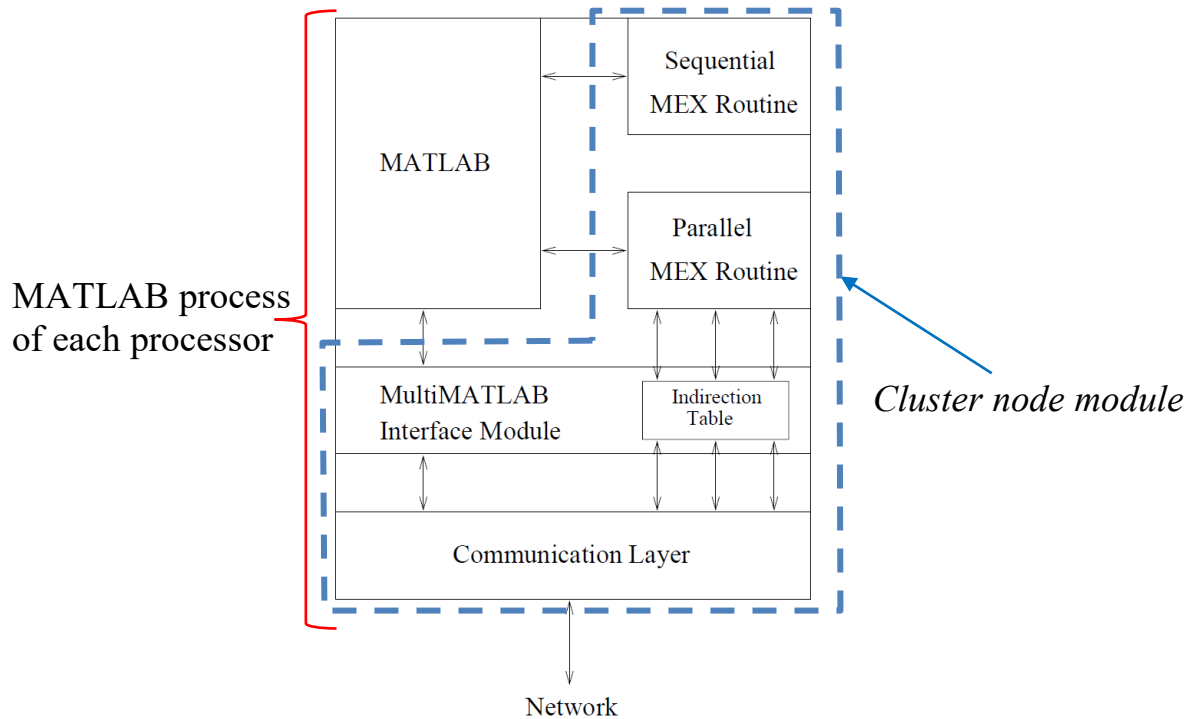
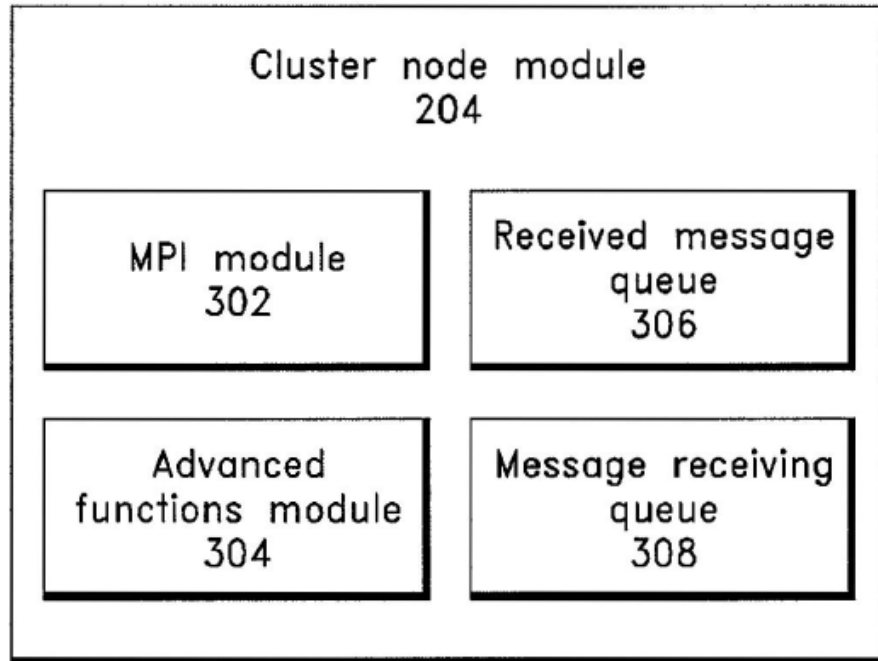


Figure 2: MATLAB Process Address Space

Ex.1005, FIG. 2 (annotated)

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



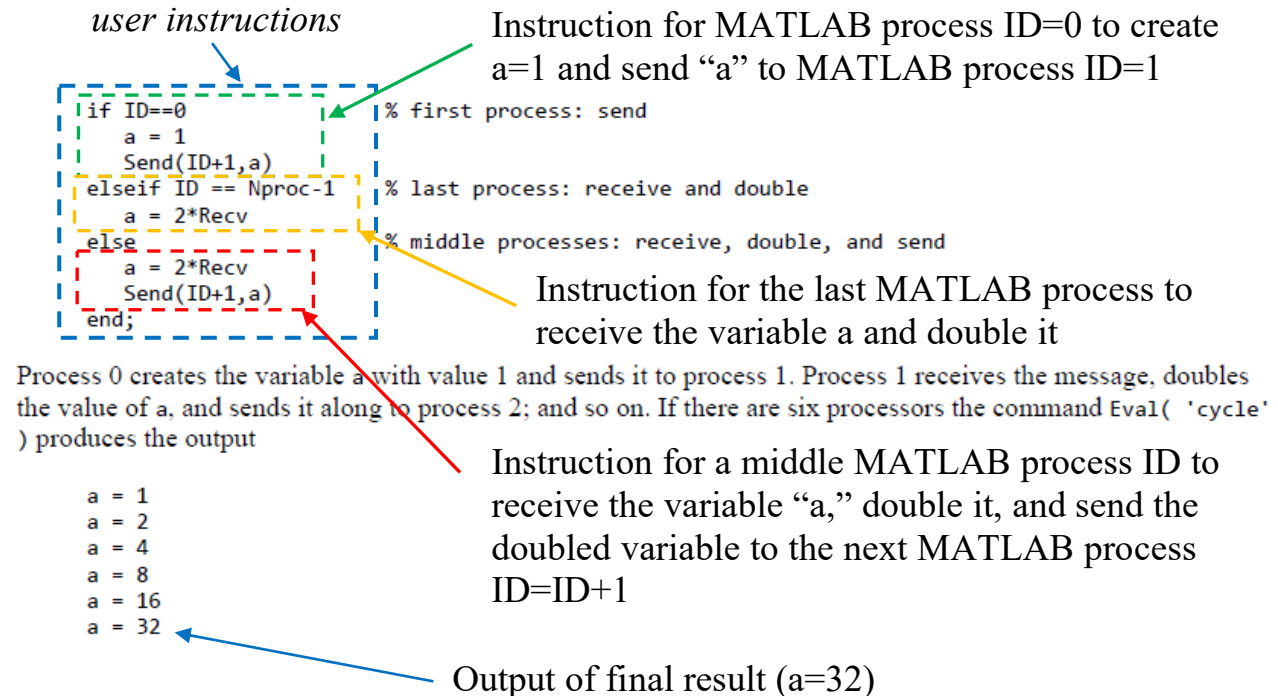
Ex.1001, FIG. 3

61. 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.

3. Peer-to-Peer Mathematical Computation

62. 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.

63. 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

64. Menon and Trefethen are each analogous art at least because Menon and Trefethen are each directed to the same field of endeavor as the '034 patent. The '034 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.

65. Furthermore, Menon and Trefethen are each analogous art to the '034 patent because each is reasonably pertinent to a problem the inventors of the '034 patent attempted to solve. The '034 patent addresses the problem of “adding cluster computing functionality to a computer program” such as Mathematica, MATLAB, etc. Ex.1001, 1:16-17, 4:19-25. As an example, the '034 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.

66. 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.

67. 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.

68. 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 '034 patent.

b. Motivation to Combine Menon with Trefethen

69. 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.

70. 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.

71. 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.

72. 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

73. RS6000 is analogous art at least because it is directed to the same field of endeavor as the '034 patent. The '034 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.

74. Furthermore, RS6000 is analogous art to the '034 patent because RS6000 is reasonably pertinent to a problem the inventors of the '034 patent attempted to solve. The '034 patent purports to provide a solution “for adding cluster computing functionality to a computer program.” Ex.1001, 1:16-17. The '034 patent describes “[c]omputer clusters includ[ing] ... processor cores (‘nodes’)

that intercommunicate so that the nodes can accomplish a task as though they were a single computer.” The ’034 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.

75. 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.

76. Accordingly, in my opinion, a POSITA would understand that RS6000 is a prior art that is analogous to the ’034 patent.

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

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

¹ As discussed in ¶76, IBM SP2 is the same as IBM RS/6000 SP.

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

78. 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.

79. 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.

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

3. Combining POEref with Menon

a. POEref is Analogous Art

81. POEref is analogous art at least because it is directed to the same field of endeavor as the '034 patent. The '034 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.

82. Furthermore, POEref is analogous art to the '034 patent because it is reasonably pertinent to a problem the inventors of the '034 patent attempted to solve. The '034 patent purports to provide a solution “for adding cluster computing functionality to a computer program.” Ex.1001, 1:16-17. The '034 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 '034 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, 1:29-31, 2:12-14.

83. 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.

84. Accordingly, in my opinion, a POSITA would understand that POEref is a prior art that is analogous to the '034 patent.

b. Motivation to Combine POEref with Menon

85. 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.

86. 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.

87. 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.

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

4. Combining MPIref with Menon

a. MPIref is Analogous Art

89. MPIref is analogous to the '034 patent at least because MPIref is directed to the same field of endeavor as the '034 patent. The '034 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.

90. Furthermore, MPIref is analogous to the '034 patent because MPIref is reasonably pertinent to a problem the inventors of the '034 patent attempted to solve. The '034 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.

91. Accordingly, in my opinion, a POSITA would understand that MPIref is a prior art that is analogous to the '034 patent.

b. Motivation to Combine MPIref with Menon

92. 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.

93. 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.

94. 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.

95. 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.

V. DETAILED ANALYSIS

A. PRIOR ART

96. Below is a detailed discussion of the prior art references I considered as part of my analysis of '679 patent claims 1-19:

- (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

97. Like the '679 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.

98. 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.

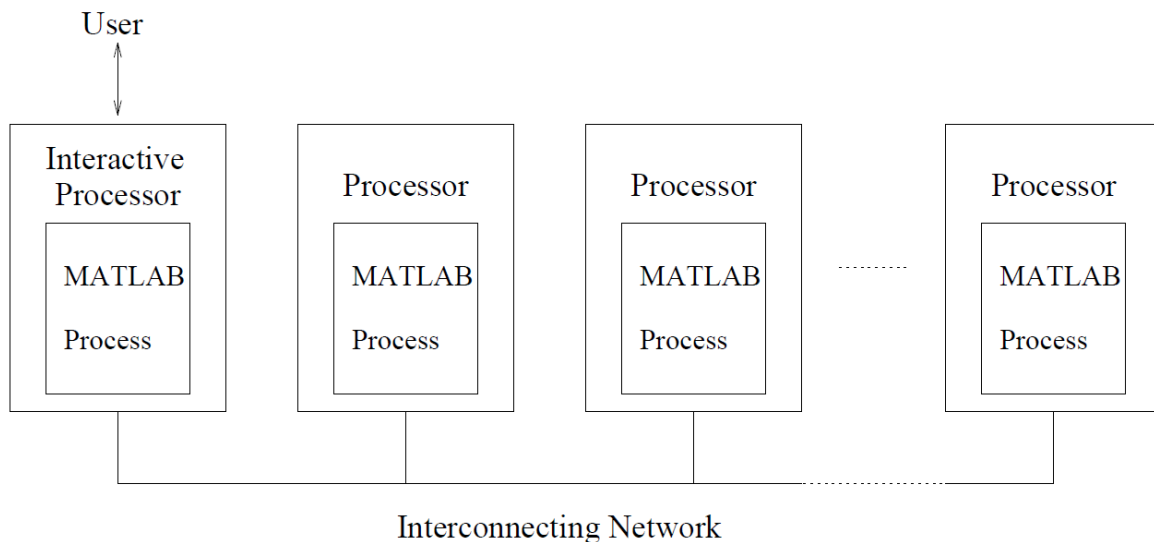


Figure 1: MultiMATLAB Architecture

Ex.1005, FIG. 1.

99. 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.

100. 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.

101. 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.

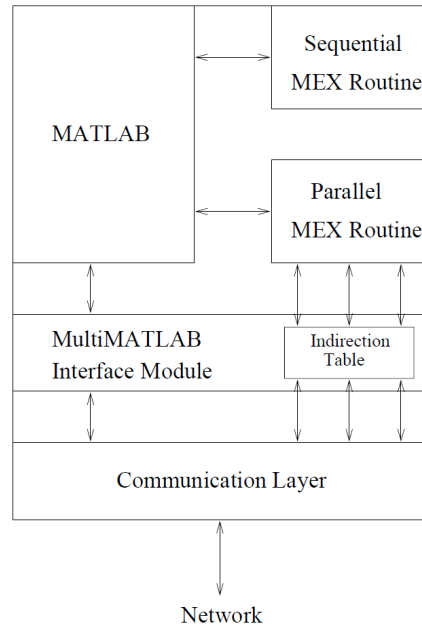


Figure 2: MATLAB Process Address Space

Ex.1005, FIG. 2

102. 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.

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

104. 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.

105. 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.

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)

106. 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

107. 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 '679 patent. Ex.1028, 4-9.

108. 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.

109. 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 '679 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 '679 patent. Ex.1028, 4-9.

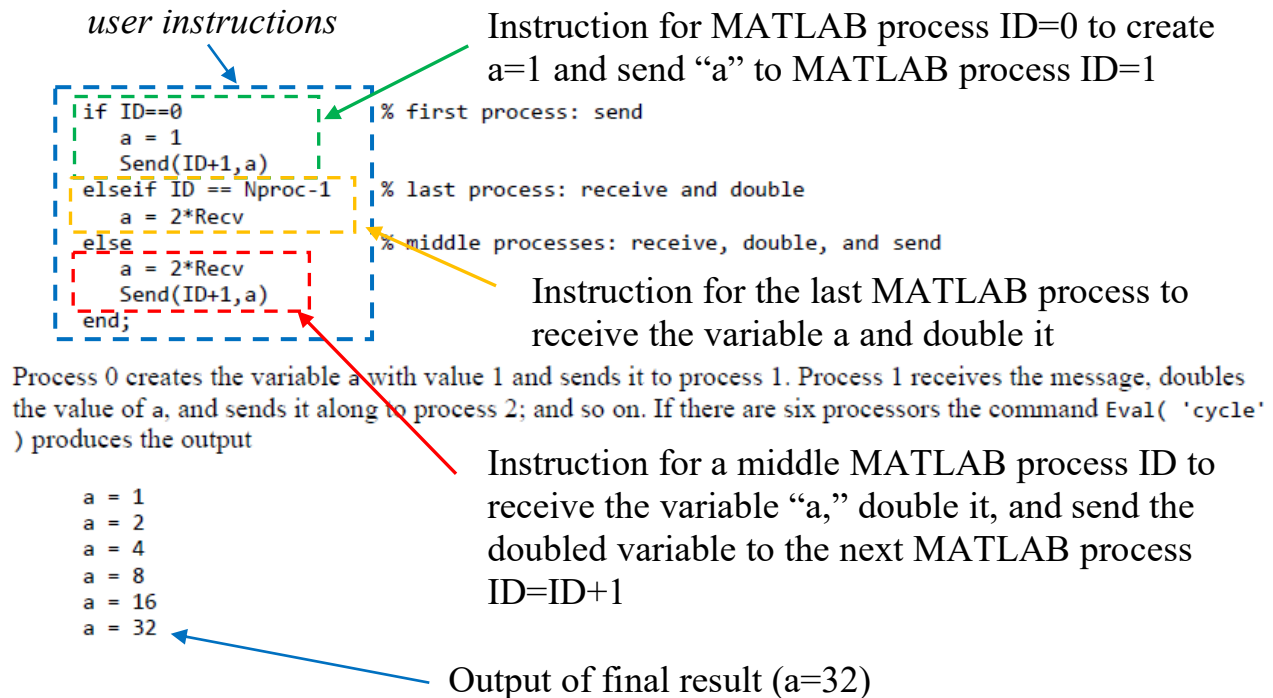
2. Trefethen

a. Summary of Trefethen

110. 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.

111. 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.

112. 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

113. 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 '679 patent, including by Menon. Ex.1035, 2-3.

114. 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 ’679 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

115. 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

116. 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.

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

4. POEref

a. Summary of POEref

118. 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.

119. 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

120. 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.

121. 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.

122. 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

123. 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.

124. 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).

125. 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

126. 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 '679 patent. Ex.1037, 2-16. Further, 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. 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.

127. 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 '679 patent. Ex.1037, 2-16.

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

128. The combination of Menon, Trefethen, RS6000, and POEref renders obvious claims 1-19 as discussed below.

1. Combining Trefethen with Menon

a. Menon and Trefethen Are Analogous Art

129. Menon and Trefethen are each analogous art at least because Menon and Trefethen are each directed to the same field of endeavor as the '679 patent. The '679 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.

130. Furthermore, Menon and Trefethen are each analogous art to the '679 patent because each is reasonably pertinent to a problem the inventors of the '679 patent attempted to solve. The '679 patent addresses the problem of “adding cluster computing functionality to a computer program” such as Mathematica, MATLAB, etc. Ex.1001, 1:16-17, 4:19-25. As an example, the '679 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.

131. 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.

132. 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.

133. 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 '679 patent.

b. Motivation to Combine Menon with Trefethen

134. 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.

135. 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.

136. 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.

137. 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

138. RS6000 is analogous art at least because it is directed to the same field of endeavor as the '679 patent. The '679 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.

139. Furthermore, RS6000 is analogous art to the ’679 patent because RS6000 is reasonably pertinent to a problem the inventors of the ’679 patent attempted to solve. The ’679 patent purports to provide a solution “for adding cluster computing functionality to a computer program.” Ex.1001, 1:16-17. The ’679 patent describes “[c]omputer clusters includ[ing] ... processor cores (‘nodes’) that intercommunicate so that the nodes can accomplish a task as though they were a single computer.” The ’679 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.

140. 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.

141. Accordingly, in my opinion, a POSITA would understand that RS6000 is a prior art that is analogous to the ’679 patent.

b. Motivation to Combine RS6000 with Menon

142. 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.

143. 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.

² As discussed in ¶76, IBM SP2 is the same as IBM RS/6000 SP.

144. 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.

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

that the MultiMATLAB system can be an effective platform for parallel computing.” Ex.1005, 14.

3. Combining POEref with Menon

a. POEref is Analogous Art

146. POEref is analogous art at least because it is directed to the same field of endeavor as the ’679 patent. The ’679 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.

147. Furthermore, POEref is analogous art to the ’679 patent because it is reasonably pertinent to a problem the inventors of the ’679 patent attempted to solve. The ’679 patent purports to provide a solution “for adding cluster computing functionality to a computer program.” Ex.1001, 1:16-17. The ’679 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 ’679 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, 1:29-31, 2:12-14.

148. 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.

149. Accordingly, in my opinion, a POSITA would understand that POEref is a prior art that is analogous to the '679 patent.

b. Motivation to Combine POEref with Menon

150. 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.

151. 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.

152. 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.

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

4. Claim 1

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

154. Menon in combination with RS6000 renders obvious the preamble.

The '679 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, 1:21-25. Likewise, Menon discloses a MultiMATLAB cluster with a series of MATLAB processor nodes, implemented on an IBM SP2 that uses multiple processor nodes, switches, network connectivity, and disk drives, for executing software, to work in parallel to calculate mathematical formulas, and RS6000 describes the IBM SP2 as a networked cluster computing system for executing software in parallel, rendering obvious *a computer cluster*.

155. 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:

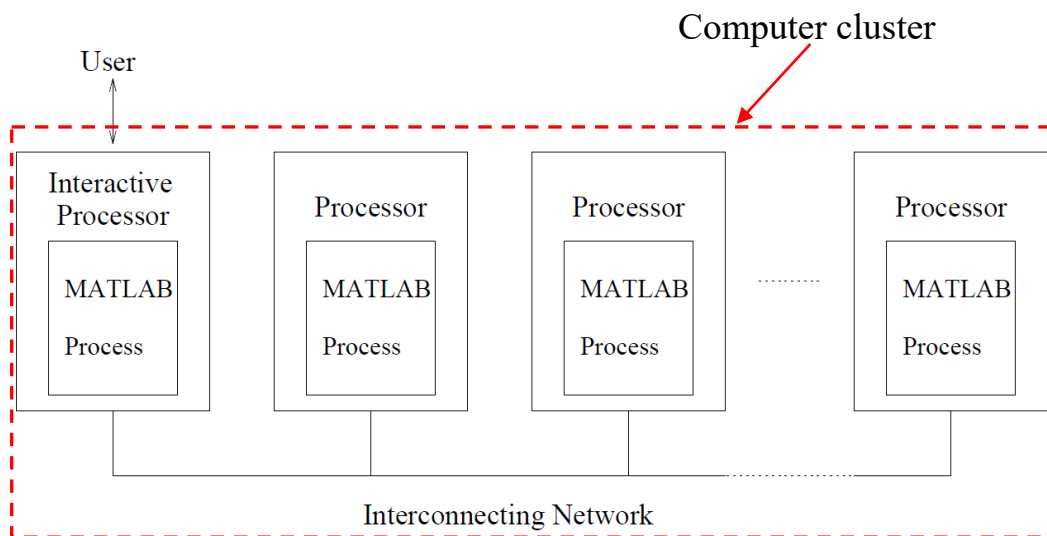


Figure 1: MultiMATLAB Architecture

Ex.1005, FIG. 1 (annotated)

156. 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.

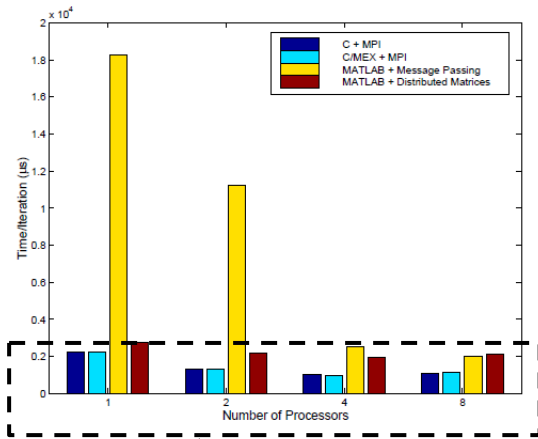


Figure 7: Parallel Conjugate Gradients on the IBM SP-2: 256×256 matrix

8 processors

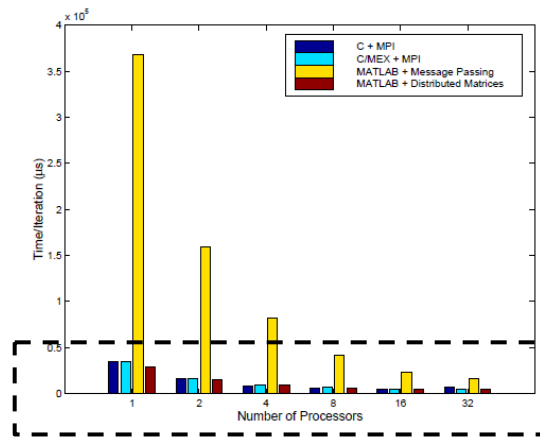


Figure 8: Parallel Conjugate Gradients on the IBM SP-2: 1024×1024 matrix

32 processors

Ex.1005, FIGs. 7 and 8

157. 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.

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

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

view of RS6000's teaching of the parallel computing system with multiple processor nodes would result in Menon's MATLAB processes each running on a processor node of the IBM SP2, as exemplified below:

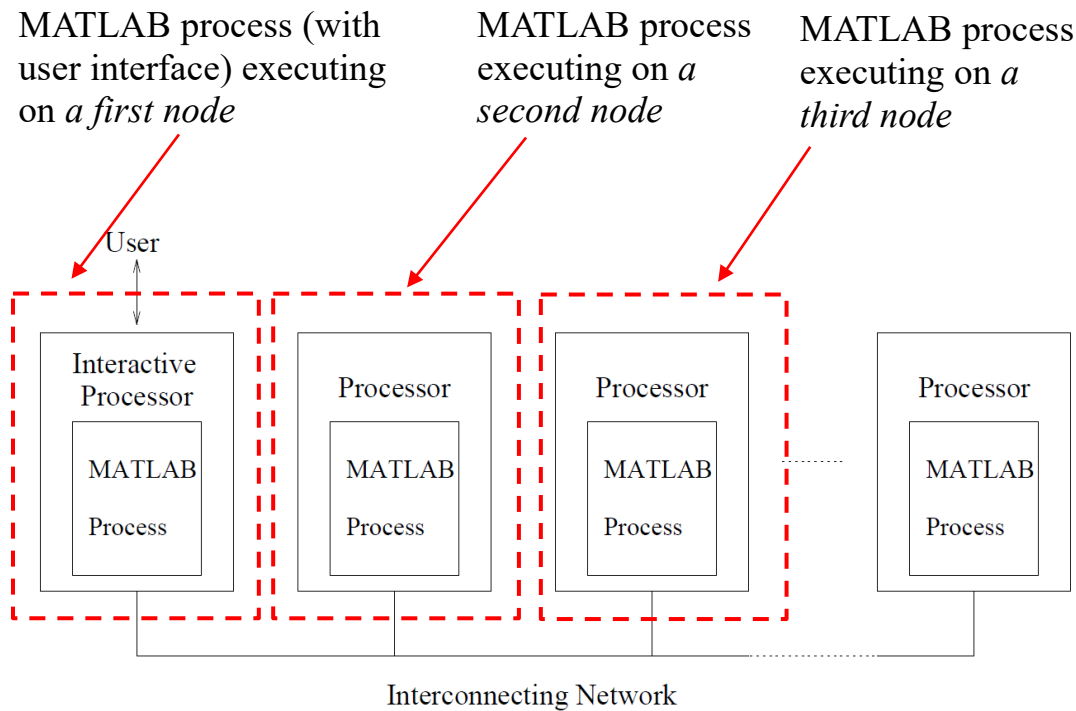


Figure 1: MultiMATLAB Architecture

Ex.1005, FIG. 1 (annotated).

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

b. [1.1.1] a plurality of nodes each comprising a hardware processor and

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

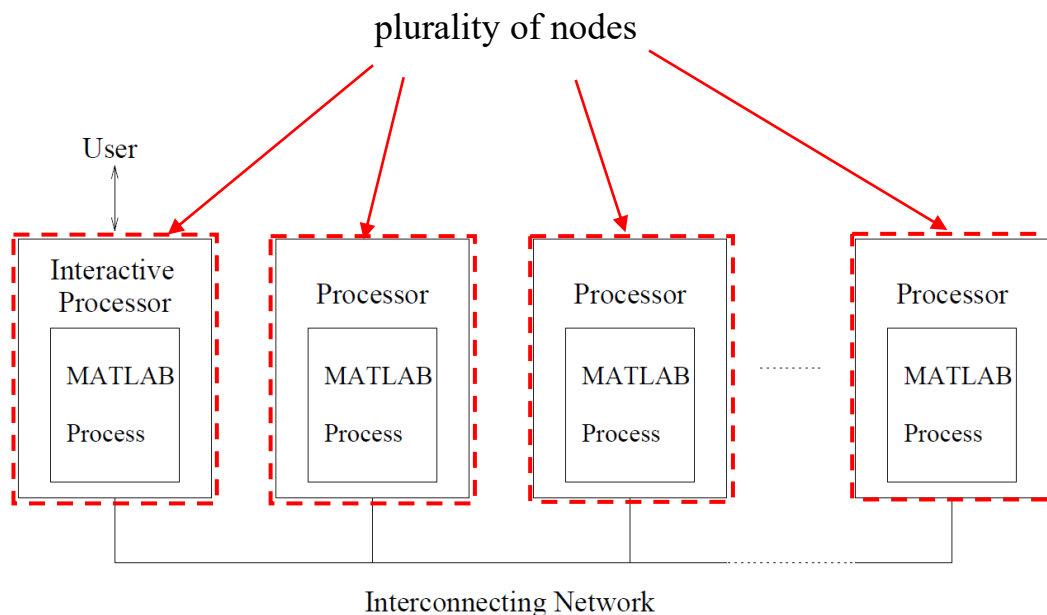


Figure 1: MultiMATLAB Architecture

Ex.1005, FIG. 1 (annotated)

162. Menon describes an “implementation of MultiMATLAB” architecture “for the IBM SP2” that comprises “32 processors of the IBM SP2.” Ex.1005, 5, 13.

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

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

- c. ***[1.1.2] [a plurality of nodes each...] configured to access a non-transitory computer-readable medium comprising program code that, when executed, is capable of causing the hardware processor to evaluate mathematical expressions comprising matrix operations; and***

164. Menon in view of Trefethen and RS6000 renders obvious this element. Specifically, Menon discloses each processor accesses the MATLAB process

address space (*computer-readable medium*) having MATLAB (*program code*) for executing single matrix-vector multiplication (*evaluate mathematical expressions comprising matrix operations*), Trefethen teaches each processor performing mathematical evaluations such as matrix operations, doubling a variable, singular value decomposition, fast Fourier transform, and polynomial zero-finding (*evaluate mathematical expressions comprising matrix operations*), and RS6000 discloses that each processor node of the IBM SP2 has hard drives and up to 64 GB memory for storing program code accessible by the processor nodes, the combination of Menon and RS6000 this rendering obvious a plurality of nodes each *configured to access a non-transitory computer-readable medium comprising program code that, when executed, is capable of causing the hardware processor to evaluate mathematical expressions comprising matrix operations*.

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

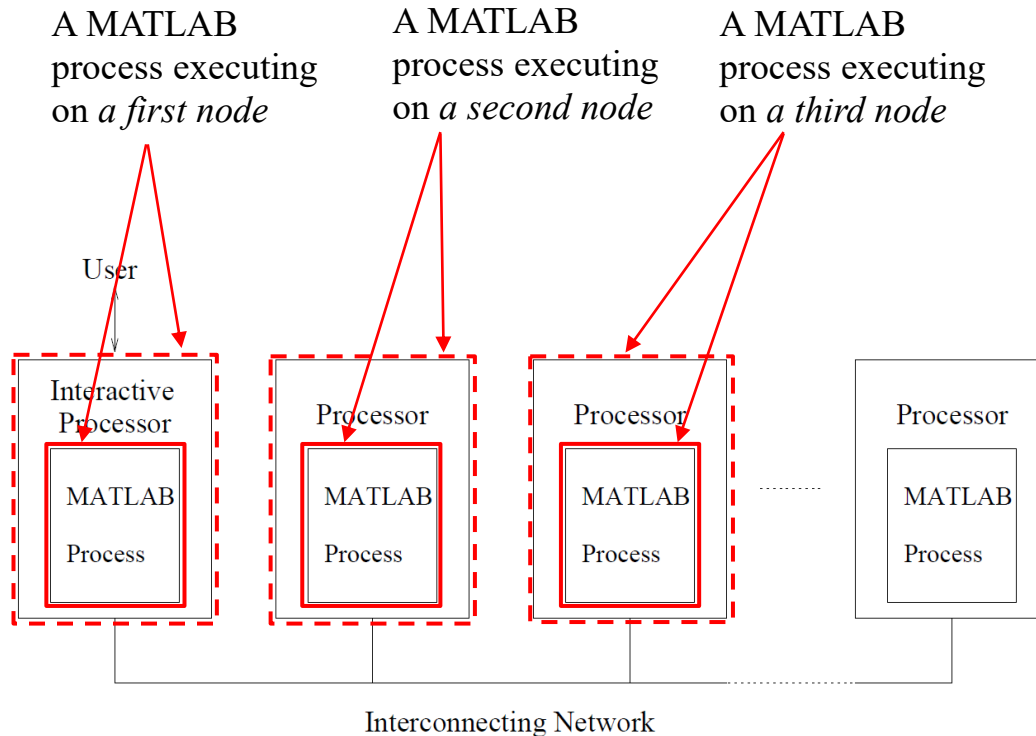


Figure 1: MultiMATLAB Architecture

Ex.1005, FIG. 1 (annotated).

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

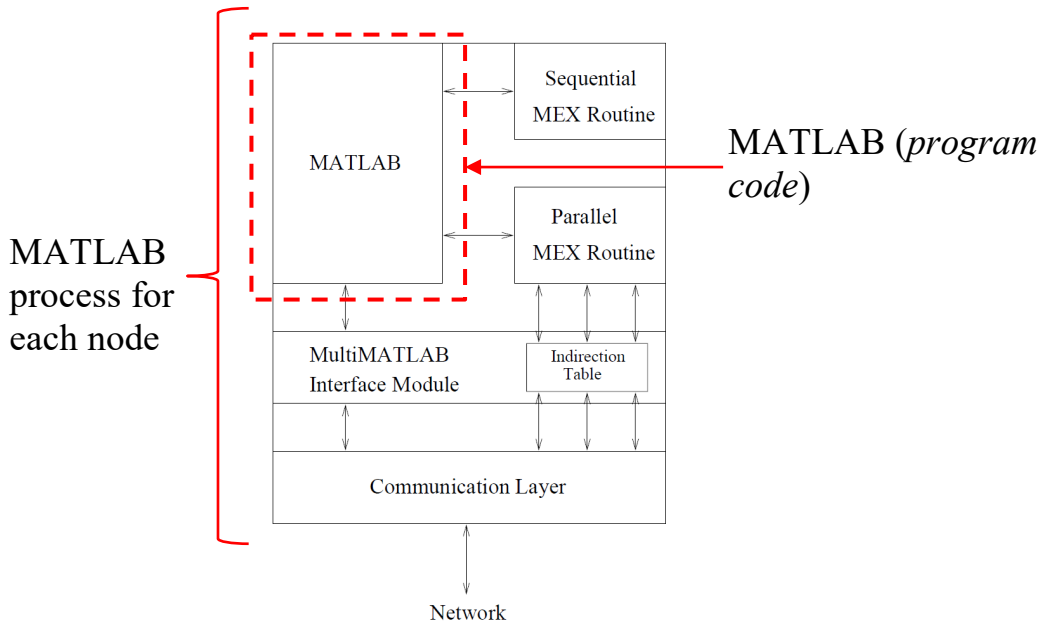


Figure 2: MATLAB Process Address Space

Ex.1005, FIG. 2 (annotated)

167. Because MATLAB is “MATLAB software,” a POSITA would recognize MATLAB is a *program code* that is executed by a processor. Ex.1006, 2; Ex.1005, 3. Accordingly, Menon’s disclosure of MATLAB that is executed by a processor teaches a *program code* that is *executed*.

168. Second, Menon and Trefethen teach MATLAB is capable of causing a processor to evaluate mathematical expressions, as illustrated by the “MultiMATLAB implementations of the conjugate gradients algorithm on the IBM SP2” discussed in Menon and mathematical evaluations such as arithmetic operations (e.g., doubling a value, calculating square root of numbers, etc.), singular value decomposition, fast Fourier transform, polynomial zerofinding, and matrix

operations (e.g., calculations of eigenvalues and condition number of Hilbert matrices, etc.) discussed in Trefethen. Ex.1005, 11-13; Ex.1006, 2-6.

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

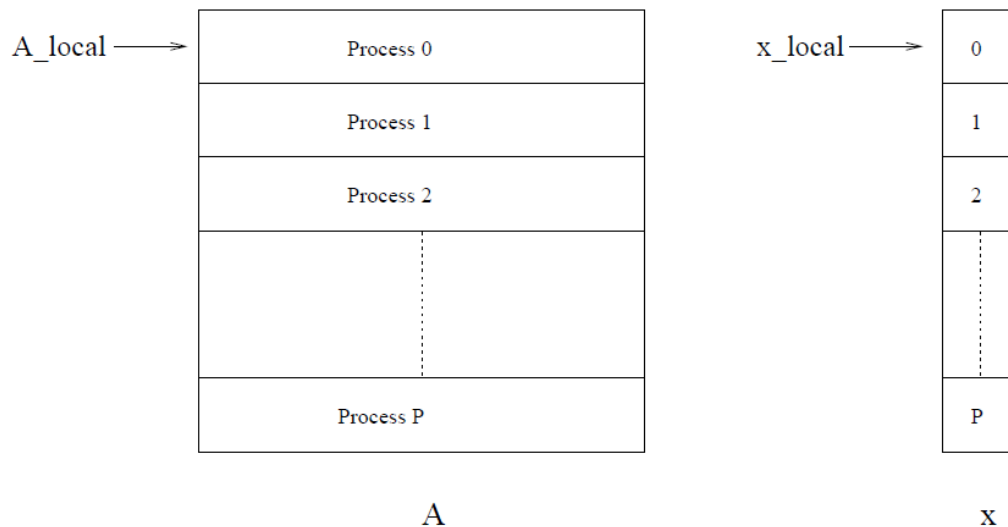


Figure 6: Matrix A and vector x distributed by row over P processors

Ex.1005, FIG. 6

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

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

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

code to *evaluate mathematical expressions*

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

a. Sequential MATLAB

```
r_local = b_local;  
rho = Sum(r_local'*r_local);  
k = 0;  
while((k < kmax))  
    k = k+1;  
    if (k == 1)  
        p_local = r_local;  
    else  
        beta = rho/oldrho;  
        p_local = r_local + beta*p_local;  
    end  
  
p = Gather(p_local);  
w_local = A_local * p;  
  
alpha = rho/(Sum(p_local'*w_local));  
x_local = x_local + alpha * p_local;  
r_local = r_local - alpha * w_local;  
oldrho = rho;  
rho = Sum(r_local'*r_local);  
end
```

b. Message Passing MATLAB

Figure 4: Conjugate Gradients

Ex.1005, FIG. 4 (annotated)

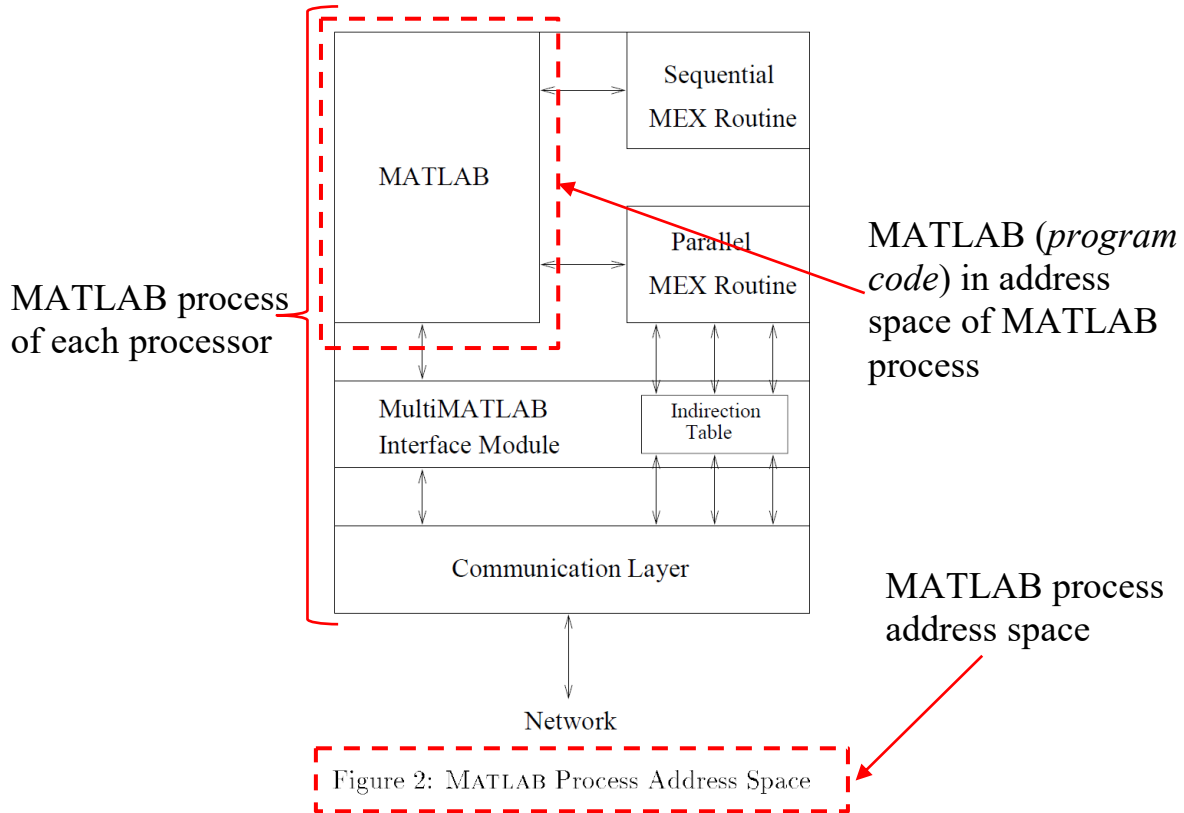
171. Further, Trefethen discloses “[a] system ... that enables one to run ... MATLAB conveniently on multiple processors” to perform *evaluat[i]ons of mathematical expressions* such as doubling a value, calculating square root of numbers, singular value decomposition, fast Fourier transform, polynomial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

172. Accordingly, Menon and Trefethen teach that MATLAB (*program code*) of a MATLAB process cause *a hardware processor* that includes the MATLAB process to *evaluate mathematical expressions comprising matrix operations*.

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

174. As discussed previously, a POSITA would understand an “address space” (as described in Menon) to be a “a list of memory locations from some minimum (usually 0) to some maximum, which the process can read and write. The address space contains the executable program, the program’s data, and its stack.” Ex.1018, 67-68. Because Menon’s address space refers to a computer’s memory, a POSITA would recognize that the MATLAB process and MATLAB are in a *computer-readable medium*, and that MATLAB is accessed from a *computer-readable medium*, i.e., a hardware processor accesses MATLAB from a *computer-readable medium*. FIG. 2 of Menon below shows MATLAB as part of the MATLAB process address space. The *computer-readable medium* of the processors of the IBM SP2, on which the conjugate gradients algorithm is implemented, can be in the form of internal and external hard disk drives, or memory cards, as taught by RS6000. Ex.1007, 6, 10 (disclosing IBM SP2 processors have internal and external

hard disk drives), 10 (disclosing IBM SP2 processors have “one to four memory cards” that support a “maximum of 64 GB”).



Ex.1005, FIG. 2 (annotated)

175. Accordingly, Menon’s disclosure of each processor accessing the MATLAB process address space (*computer-readable medium*) having MATLAB (*computer program code*) for executing single matrix-vector multiplication (*evaluate mathematical expressions comprising matrix operations*) in view of Trefethen teaching each processor performing mathematical evaluations such as matrix operations, doubling a variable, singular value decomposition, fast Fourier transform, and polynomial zerofinding (*evaluate mathematical expressions*

comprising matrix operations) and RS6000 teaching that each processor node of the IBM SP2 has hard drives and up to 64 GB memory for storing program code accessible by that processor node, renders obvious a plurality of nodes each *configured to access a non-transitory computer-readable medium comprising program code that, when executed, is capable of causing the hardware processor to evaluate mathematical expressions comprising matrix operations.*

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

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

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

178. The underlying communication layer of the MultiMATLAB architecture “runs over the parallel platform’s interconnection network” and “provide[s] [a MATLAB process on a processor] with the ability to communicate with other processes.” Ex.1005, 3. The ’679 patent states that a network interface “can include one or more of a LAN, a WAN, an intranet, a wireless network, and/or the Internet.” Ex.1001, 7:1-4. Menon explains that MultiMATLAB is intended to

run on “any network of workstations supporting Matlab” and that the MPICH implementation of MultiMATLAB “run[s] over a network of workstations connected by TCP/IP.” Ex.1005, 2, 5. A POSITA would have known that each of a LAN, WAN, intranet, wireless network, or the Internet communicate using TCP/IP.³ Table 1 of Menon below shows MultiMATLAB commands implemented as MEX Routines. Further, FIG. 2 and FIG. 1 of Menon below show the MEX Routines of a MATLAB process and the interconnection network of the MultiMATLAB architecture linking the processors thereof, respectively.

³ As a further example, Trefethen explains that its MultiMATLAB implementation was run on machines “connected to Cornell’s Computer Science Department network.” Ex.1006, 2. A POSITA would have known that such an “internal” University network is an example of an intranet.

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)

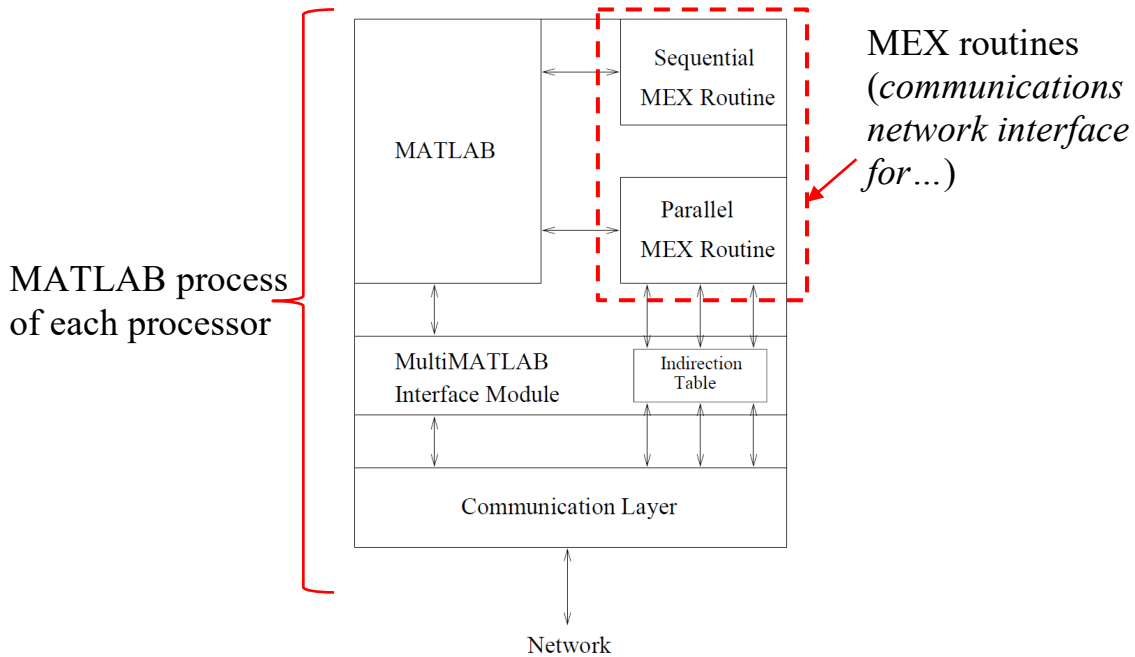


Figure 2: MATLAB Process Address Space

Ex.1005, FIG. 2 (annotated)

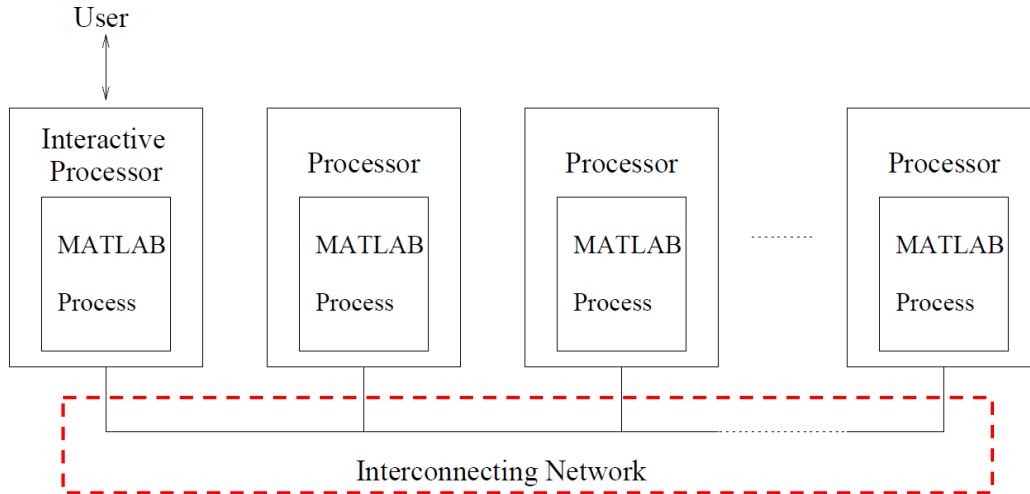


Figure 1: MultiMATLAB Architecture

Ex.1005, FIG. 1 (annotated)

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

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

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

The command

Eval(‘ID^ID’)

might produce

ans = 1

ans = 1

ans = 256

ans = 3125

ans = 27

ans = 4.

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

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

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

might return

ans = 1.5514e+04

ans = 4.7661e+05

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

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

4.7661e+05,” “the condition number[] of the Hilbert [matrix] of dimension[] 5.”

Ex.1006, 4.

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

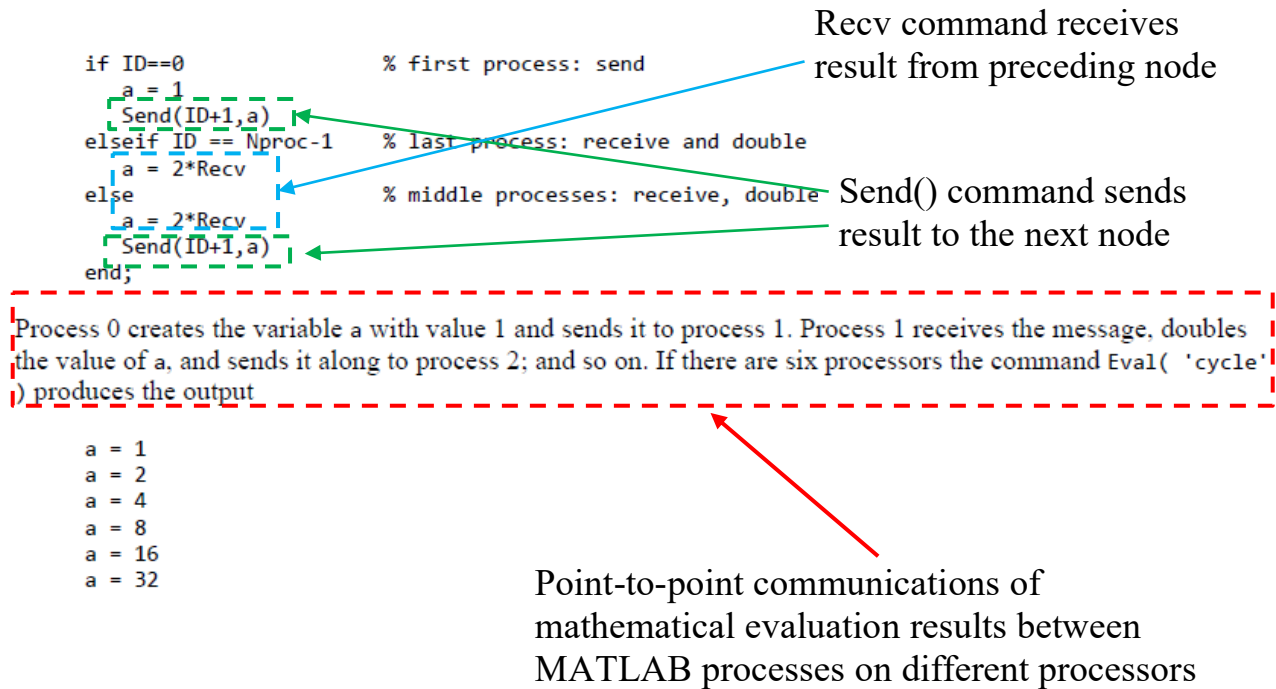
MultiMATLAB commands	
Starting and stopping MultiMATLAB (MPICH only)	
Start(n)	Initializes n remote MATLAB processes.
Quit	Closes remote MATLAB processes.
Running commands on multiple processors	
Nproc	Returns number of MATLAB processes.
ID	Returns ID of calling MATLAB process.
Eval(pid,'command')	Evaluates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

Table 1: Selected commands in the MultiMATLAB system

Commands for point-to-point communication

Ex.1005, Table 1 (annotated)

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



Ex.1006, 6 (annotated).

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

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

The command

`Eval( 'Sum(1,[1 ID Nproc])' )`

executed on six processes will return the vector

`[6 15 36]`

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

Ex.1006, 7.

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

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

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

f. *[1.3.1] a first node comprising a first hardware processor configured to access a first memory comprising first program code for a user interface and*

187. Menon in view of RS6000 render obvious this element. Specifically, Menon and RS6000 describe a node having (i) a symmetric multiprocessor that

accesses *program code* stored in *memory* to be executed by a *processor*, and (ii) the MATLAB code stored in the address space of, and accessed by, the processor, includes a toolkit with a graphical interface to allow users to provide equations and directly interact with the interactive processor node (*user interface*), thereby rendering obvious *a first node comprising a first hardware processor configured to access a first memory comprising first program code for a user interface*.

188. First, it was shown in limitation [1.1.1] that the combination of Menon and RS6000 renders obvious *a node comprising a hardware processor*. Further, it was shown in limitation [1.1.2] that the combination of Menon and RS6000 renders obvious *a node configured to access Menon's address space*, which refers to a computer's memory, comprising *program code*, where the memory can be in the form of memory cards having up to 64 GB memory for storing program code accessible by that processor.

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

the interactive MATLAB.” Ex.1005, 6 (emphasis original). “The user interacts directly with one MATLAB process, called the interactive process, and operates within that process’s MATLAB environment.” Ex.1005, 3.

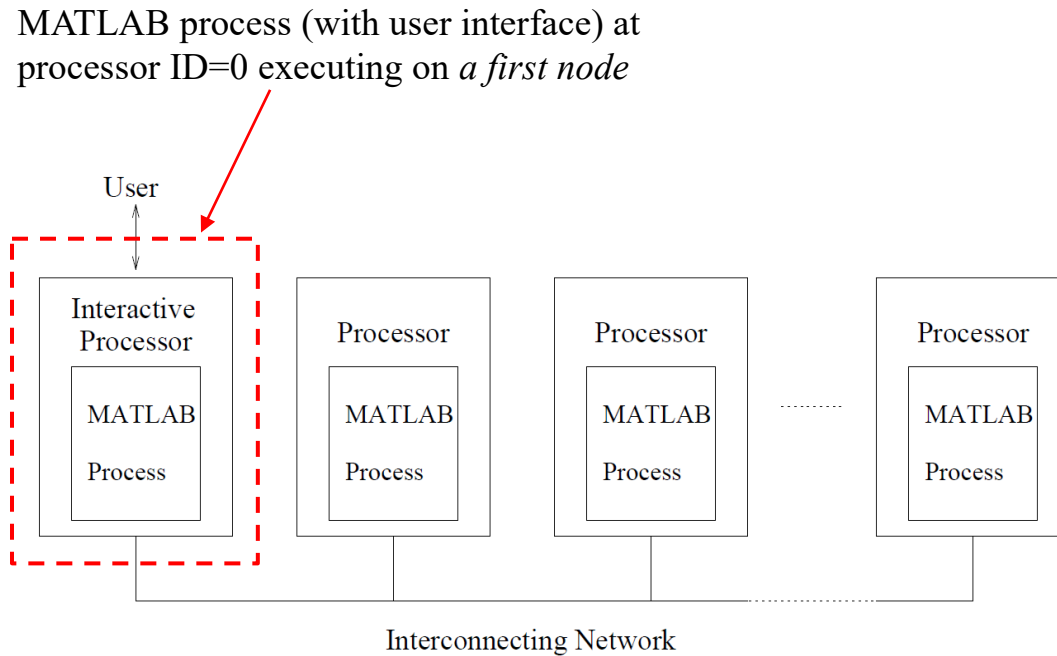


Figure 1: MultiMATLAB Architecture

Ex.1005, FIG. 1 (annotated).

190. Menon discloses a “toolkit” that provides “an intuitive graphical interface” to allow users to specify equations. Ex.1005, 6. This toolkit is “MATLAB’s Partial Differential Equation Toolkit,” in which “users may interactively create and solve rather general PDE systems using the finite element method. The PDE toolkit provides an intuitive graphical interface allowing users to specify geometries, differential equations, and boundary conditions.” Ex.1005, 6. In

other words, MATLAB's partial differential equation (PDE) toolkit has a graphical interface that allow users to directly interact with MATLAB.

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

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

- g.** ***[1.3.2] [a first node comprising...] second program code configured to interpret user instructions, identify an array of data elements, and distribute calls to at least one of a plurality of other nodes for execution of matrix operations on the array of data elements; and***

193. Menon in view of Trefethen render obvious this element. Specifically, Menon's teaching that MATLAB (*second program code*) receives the Eval command including MATLAB commands and data including processor ID, implements the conjugate gradients algorithm on MATLAB data structures

(identify an array of data elements), and causes calls to be sent to (distribute calls to) the MATLAB processes on other processors (at least one of a plurality of other nodes) in accordance with the IDs specified by the user instructions, in view of Trefethen's teaching that MATLAB interprets Eval commands (interpret user instructions) in order to assign different actions to different processes for those actions to be performed by those processes (for execution), renders obvious the first node comprising second program code configured to interpret user instructions, identify an array of data elements, and distribute calls to at least one of a plurality of other nodes for execution of matrix operations on the array of data elements.

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

IDs.” Ex.1005, 4. Further, the user interface is configured so that the MEX routines “may be run directly from [the] MATLAB” environment. Ex.1005, Abstract, 3. Thus, on the interactive MATLAB process, MATLAB (*second program code*) receives *user instructions* including MATLAB commands from a user and sends the commands to the other MATLAB processes for evaluation.

195. Menon also discloses that MATLAB data structures can be “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, 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. In this way, MATLAB *identif[ies] an array of data elements* and, as described in limitation [1.1.2], MATLAB *execut[es] ... matrix operations* on the *array of data elements*.

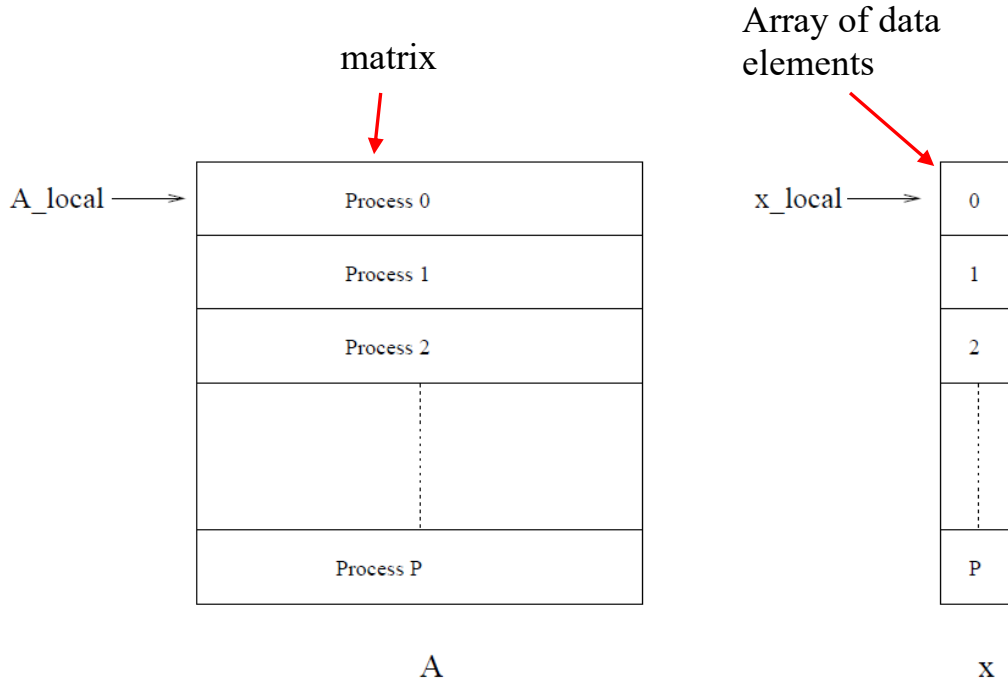


Figure 6: Matrix A and vector x distributed by row over P processors

Ex.1005, FIG. 6 (annotated)

196. Trefethen provides example illustrations where *user instructions* are interpreted in order to send different actions for different MATLAB processes.

Ex.1006, 3-7. For example, Trefethen discloses MATLAB (*second program code*) receiving `Eval( [4 5] , 'cond(hilb(ID))' )`, which includes the data "ID," from the user. Ex.1006, 4. The Eval command (*user instruction*) instructs MATLAB (*second program code*) to "select a subset of the processes by passing two arguments to the Eval command, the first being a vector of process IDs." Ex.1006, 4. In accordance with that instruction, the calls "cond(hilb(ID))" are distributed to the MATLAB processes on processor ID = 4 and ID = 5 for each to perform that call, i.e.,

processor ID=4 calculates `cond(hilb(ID=4))` and processor ID=5 calculates `cond(hilb(ID=5))`, where the results are:

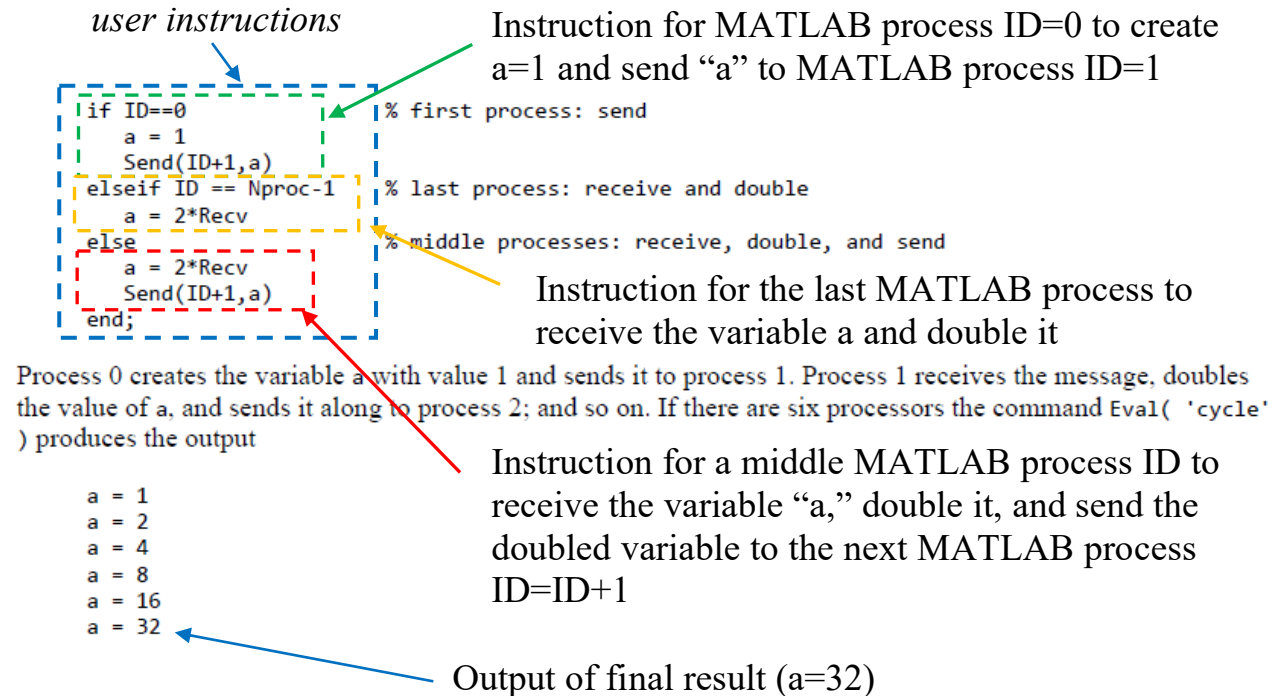
“ans = 1.5514e+04,
ans = 4.7661e+05

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

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

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

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



Ex.1006, 6 (annotated).

199. Thus, Trefethen discloses that MATLAB (*second program code*) receives and interprets the command Eval (‘cycle’) (*configured to interpret user instructions and*) to send respective MATLAB commands to the MATLAB processes on the processors (*distribute calls to at least one of a plurality of other nodes*) so that those MATLAB processes perform the MATLAB commands (*for execution*). Accordingly, Menon’s teaching that MATLAB is configured to interpret command and then distributes calls (*e.g., using the Eval MEX routine*) to the

MATLAB processes on other nodes in accordance with the IDs specified by the user instructions in order to perform mathematical calculations using matrices and an array of data elements, in view of Trefethen's teaching that MATLAB interprets Eval commands in order to assign different actions to different processes for those actions to be performed by those processes, renders obvious a first node comprising *second program code configured to interpret user instructions, identify an array of data elements, and distribute calls to at least one of a plurality of other nodes for execution of matrix operations on the array of data elements.*

h. [1.4.1] a second node comprising a second hardware processor,

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

201. In limitation [1.1.1], it was shown that Menon discloses a parallel processing system, the MultiMATLAB architecture, executing a series of MATLAB processes on 32 processors, implemented on the IBM SP2, and RS6000 discloses that the IBM SP2 is a parallel processing system having up to 128 or more processor nodes, where each processor node has a symmetric processor having up

to sixteen processors. Further, in limitation [1.1.2], it was shown that each processor node of the IBM SP2 has hard drives and up to 64 GB memory for storing program code accessible by the processor nodes. A processor node of the MultiMATLAB architecture other than the interactive processor (*first node*) of FIG. 1 discussed in limitation [1.3.1], such as processor ID = 1, teaches *a second node comprising a second hardware processor*. FIG. 1 below shows an example *second node comprising a second hardware processor*:

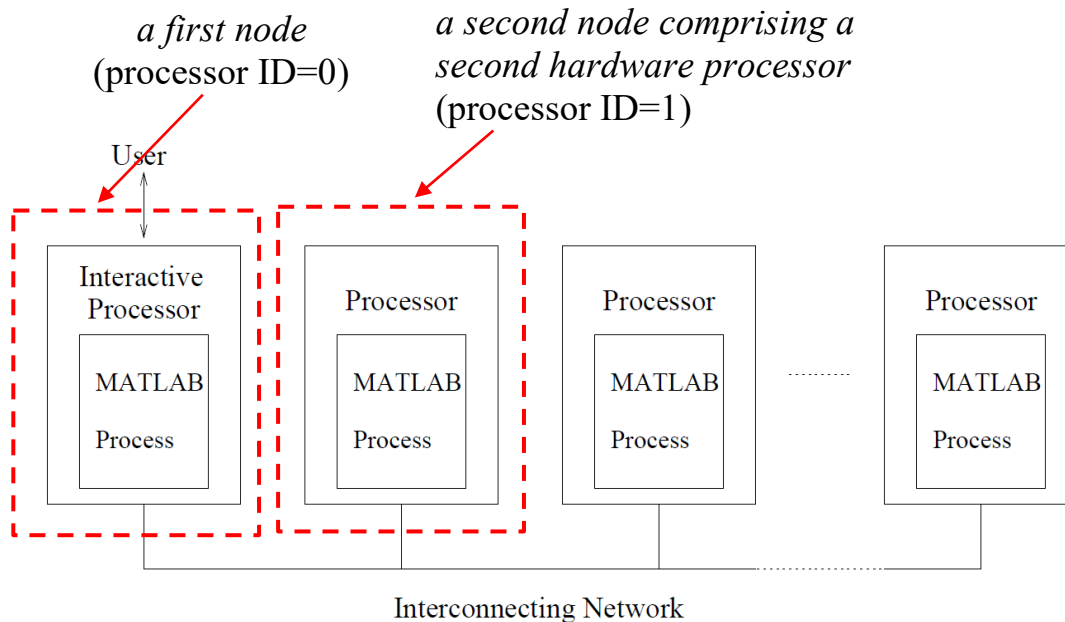


Figure 1: MultiMATLAB Architecture

Ex.1005, FIG. 1 (annotated).

202. RS6000 also teaches that each of the nodes in the IBM SP2 has “a Symmetric MultiProcessor (SMP)” (“*hardware processor*”). Ex.1007, 2. Thus, each node has *hardware processor*.

203. Accordingly, Menon discloses executing a series of parallel processes on 32 processors of the IBM SP2 system and RS6000 teaches that each IBM SP2 node has a symmetric multiprocessor, rendering obvious *a second node comprising a second hardware processor.*

- i. ***[1.4.2] wherein the second node is configured to receive the calls from the first node, execute the matrix operations on the array of data elements, and communicate a result of the matrix operations to a third node;***

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

205. In limitation [1.2], it was shown that Menon teaches MEX routines to enable SPMD point-to-point communication so that the MATLAB processes can communicate tasks and data directly with each other. Ex.1005, 9. The point-to-point communication is established by IBM's POE, where "[w]hen you invoke **poe**, the Partition Manager allocates processor nodes for each task and initializes the local

environment. It then loads your program, and reproduces your local environment, on each processor node. ... If you are using the dynamic message passing interface, the appropriate communication subsystem library implementation (IP or US) is automatically loaded at this time.” Ex.1008, 28 (emphasis original). “For each of p MATLAB processes, POE assigns an identification number between 0 and $p - 1$. The process numbered 0 is considered the interactive MATLAB. All other processes wait for commands from the interactive MATLAB process.” Ex.1005, 6 (emphasis original).

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

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

ans = 0

ans = 1

ans = 5

ans = 2

ans = 3

ans = 4

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

207. In limitation [1.3.2], it was shown that Menon teaches that the nodes communicate with each other to perform mathematical calculations using matrix multiplication and an array of data elements and Trefethen discloses an example user instruction, Eval ('cycle'), where "the m-file cycle.m," instructs: (i) the interactive MATLAB process to create a variable "a" with a value and send the variable to the next MATLAB process, and (ii) the next MATLAB process to receive the variable "a" from the interactive MATLAB process, double the value of the received variable "a," and send the result to the next MATLAB process.

Ex.1006, 6. That is, Trefethen teaches that processor ID =1 (*the second node*) that is next to the interactive processor (i.e., processor ID = 0) receives a command (*is*

configured to receive calls) from the processor ID = 0 (*from the first node*) of the interactive MATLAB that the user is interacting with to receive and perform the conjugate gradients algorithm on MATLAB data structures, such as “matrices, matrix sections,” “sparse matrices, cell arrays, and structures” (*execute the matrix operations on the array of data elements*), and send the result of the conjugate gradients algorithm (*and communicate a result of the matrix operations*) to the next MATLAB process at the next processor, processor ID = 2 (*to a third node*).

208. Accordingly, Menon teaching MEX routines having code that implement standard variants of MPI calls to enable SPMD point-to-point communication so that the MATLAB processes can communicate tasks and data directly with each other, where the mathematical expressions use “matrices, matrix sections,” “sparse matrices, cell arrays, and structures,” in view of Trefethen teaching example instructions that uses point to message passing to send calls from a first node that are received by the second node, where the second node sends the results of the mathematical expressions to a third node, renders obvious *wherein the second node is configured to receive the calls from the first node, execute the matrix operations on the array of data elements, and communicate a result of the matrix operations to a third node.*

j. [1.5.1] wherein the third node comprises a third hardware processor,

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

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

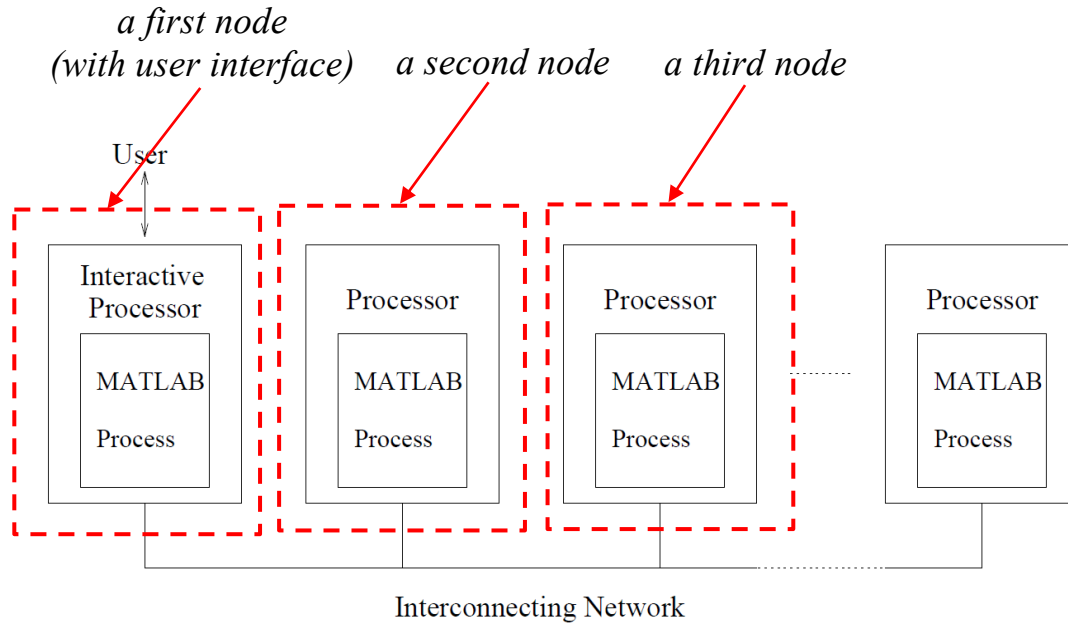


Figure 1: MultiMATLAB Architecture

Ex.1005, FIG. 1 (annotated).

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

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

second node, execute at least a mathematical expression evaluation using the received result, and communicate the result of the mathematical expression evaluation to the first node.

212. In limitation [1.1.2], it was shown that Menon teaches *evaluat[i]ons of mathematical expressions*, such as the conjugate gradient algorithm and performing single matrix-vector multiplication: “[t]he conjugate gradients algorithm is iterative method to solve a linear system of the form $Ax = b$, where A is a symmetric positive definite matrix and x and b are vectors, that is *matrix operations*.

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

214. Trefethen also teaches that the MATLAB process at processor ID = 2, receives a variable from processor ID = 1 and uses its processor to double the received variable, as shown below:

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

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

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

```
a = 1
a = 2
a = 4 ← Output of third node (a=4)
a = 8
a = 16
a = 32 ← Output of final result (a=32)
```

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

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

ans = 0

ans = 1

ans = 5

ans = 2

ans = 3

ans = 4

The ordering of these numbers is arbitrary, since the processors are not synchronized and **output is sent to the master process as soon as it is ready.**"

Ex.1006, 4. In other words, as soon as a MATLAB process at a processor completes its computation, it uses its processor to send the result of the computation to the interactive processor. A POSITA would recognize that the "master process" described by Trefethen would be the interactive process of Menon, because the interactive process is the MATLAB process in the MultiMATLAB architecture with which the user interacts.⁴ Ex.1005, 3. As a specific example, as soon as the MATLAB process at processor ID=2 completes the operation of doubling the variable that it has received from processor ID=1, the MATLAB process uses that

⁴ In the context of SPMD point-to-point communication, a "master process" is a "pseudo-master [that] is arbitrarily chosen to be the task [that] all the peers know a priori who to send the results back to." Ex.1011, 64. That is, here, the "master" process is the process with which the user is directly interacting in a peer-to-peer paradigm, and not a "master" in a master/slave paradigm.

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

216. Further, Menon “describe[s] different MultiMATLAB implementations of the conjugate gradients algorithm on the IBM SP2,” where the algorithm is an “iterative method to solve a linear system of the form $Ax = b$, where A is a symmetric positive definite matrix and x and b are vectors. The computational core of this method is a single matrix-vector multiplication at each iteration.”

Ex.1005, 11. The MultiMATLAB architecture implementation of the conjugate gradients algorithm is shown below in FIG. 4 of Menon below:

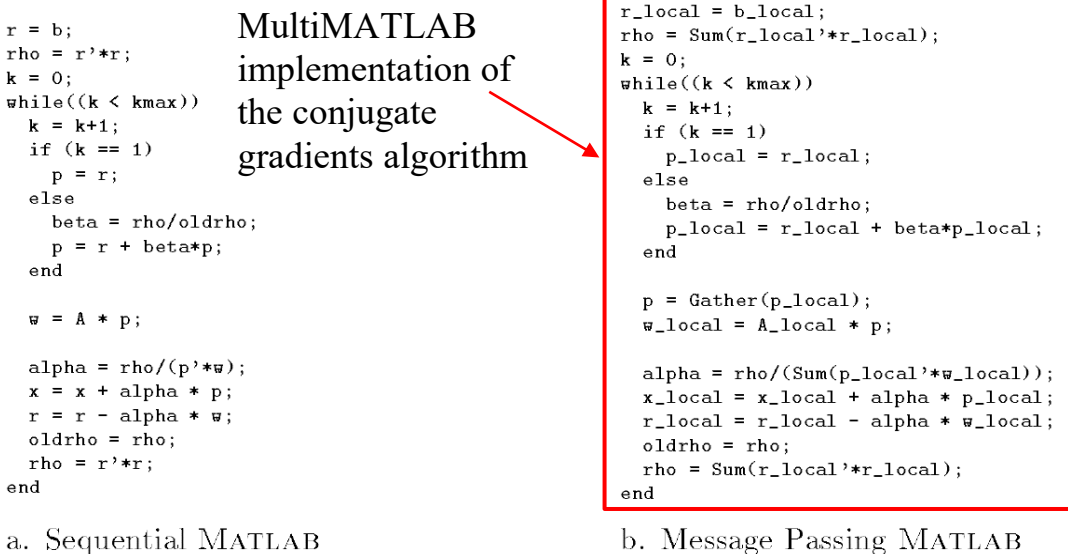


Figure 4: Conjugate Gradients

Ex.1005, FIG. 4 (annotated)

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

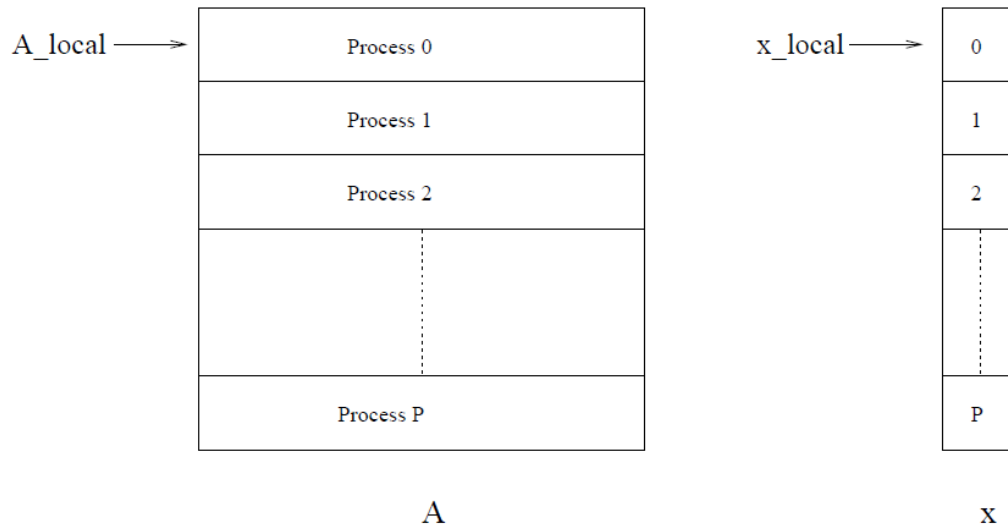


Figure 6: Matrix A and vector x distributed by row over P processors

Ex.1005, FIG. 6

Ex.1005, 11.

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

219. Accordingly, Menon teaching performing mathematical expressions, such as the conjugate gradients algorithm on MATLAB data structures (“matrices, matrix sections,” “sparse matrices, cell arrays, and structures,” Ex.1005, 9) and performing matrix operations (“[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) in view of Trefethen teaching passing mathematical results from a second MATLAB node that are received by a third MATLAB node, which perform a calculation, and that, when the output is ready, sends the output to the interactive MATLAB process (*first node*), renders obvious *wherein the third node is configured to receive the result of the matrix operations from the second node, execute at least a mathematical expression evaluation using the received result, and communicate the result of the mathematical expression evaluation to the first node.*

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

220. Menon in view of Trefethen render obvious this element. Specifically, Menon teaches an interactive MATLAB process at an interactive processor that allows a user to directly interact with, and operate within, the MATLAB process environment, and Trefethen teaches that the progress of computations on several

processors of the MultiMATLAB architecture are displayed on the user's screen as soon as the results are ready, thus rendering obvious *wherein the first node is configured to return the result of the mathematical expression evaluation to the user interface.*

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

222. In limitation [1.5.2], it was shown that as soon as the MATLAB process at processor ID=2 completes the operation of doubling the variable that it has received from processor ID=1, it uses its processor ID=2 (*third node*) to send

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

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

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

n. [1.7.1] accept, via the user interface or via a script, user instructions; and

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

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

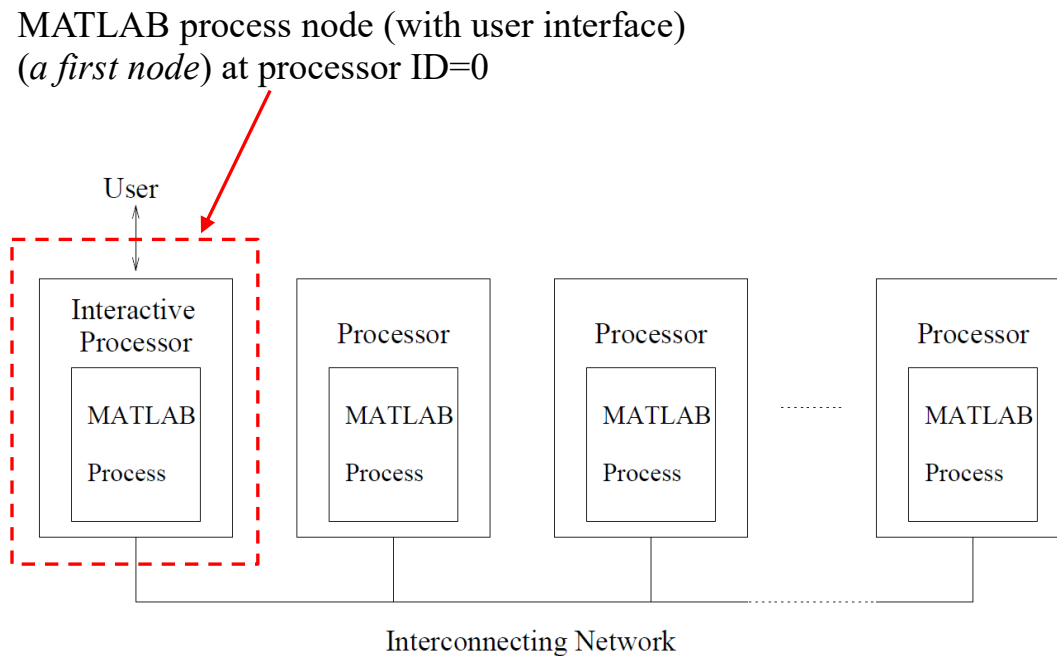


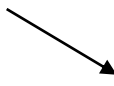
Figure 1: MultiMATLAB Architecture

Ex.1005, FIG. 1 (annotated)

226. Menon explains that users can provide the graphical interface “geometries, differential equations, and boundary conditions.” Ex.1005, 6. This shows that *user instructions* can be provided *via the user interface*. For example, users can provide mathematical formulas to calculate a conjugate gradient, as shown below in FIG. 4 of Menon:

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

User instructions to
perform mathematical
expression evaluation



```
r_local = b_local;  
rho = Sum(r_local'*r_local);  
k = 0;  
while((k < kmax))  
    k = k+1;  
    if (k == 1)  
        p_local = r_local;  
    else  
        beta = rho/oldrho;  
        p_local = r_local + beta*p_local;  
    end  
  
p = Gather(p_local);  
w_local = A_local * p;  
  
alpha = rho/(Sum(p_local'*w_local));  
x_local = x_local + alpha * p_local;  
r_local = r_local - alpha * w_local;  
oldrho = rho;  
rho = Sum(r_local'*r_local);  
end
```

a. Sequential MATLAB

b. Message Passing MATLAB

Figure 4: Conjugate Gradients

Ex.1005, FIG. 4 (annotated)

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

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

- o.** *[1.7.2] after accepting the user instructions, communicate at least some of the user instructions using the communications network interface for the nodes to communicate with each other.*

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

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

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

231. In limitation [1.2], it was shown that Menon teaches MEX routines exchanged over a network so that the MATLAB processes can communicate tasks and data directly with each other, rendering obvious *a communications network interface for the nodes to communicate with each other.*

232. Further, Menon describes how the interactive processor (e.g., processor ID=0) sends commands to the other MATLAB processes, stating that the Eval routine “is implemented as a MEX routine which accepts a vector of MATLAB process IDs and a MATLAB command. This routine sends the commands over the underlying communication layer to the processes corresponding to the specified IDs.” Ex.1005, 4. The “communication layer [] runs over the parallel platform’s interconnection network.” Ex.1005, 3. FIG. 1 and FIG. 2 of Menon shown below illustrate the interactive processor sending commands to the MATLAB processes on the other processors of the MultiMATLAB architecture over the communication layer. As discussed in [1.2], Trefethen teaches that the user instructions are accepted from the user and then communicated to the other nodes. For example,

the user instruction “Eval([4 5] , ‘cond(hilb(ID))’)” causes the MATLAB processes on processor ID=4 and ID=5 each perform the task “cond(hilb(ID))”.

Ex.1006, 4.

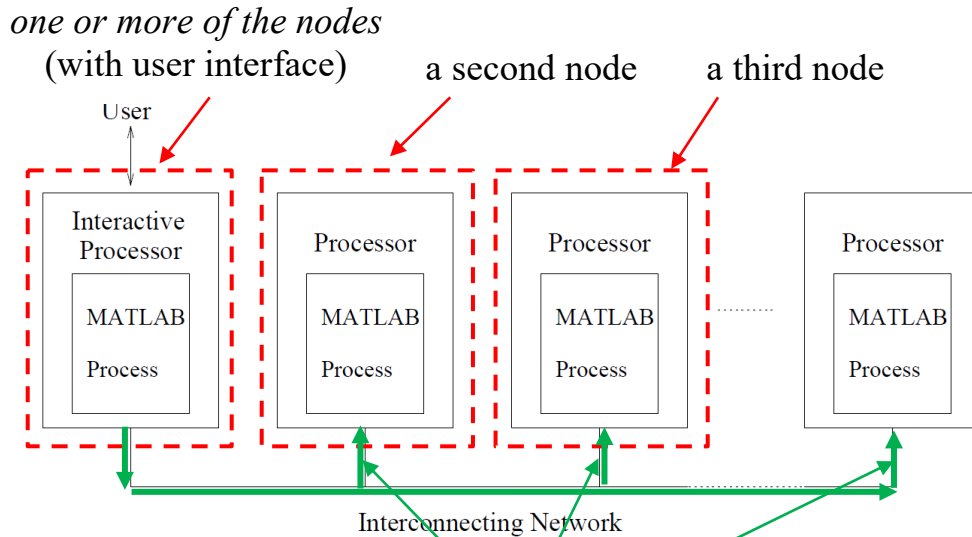


Figure 1: MultiMATLAB Architecture

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

Ex.1005, FIG. 1 (annotated)

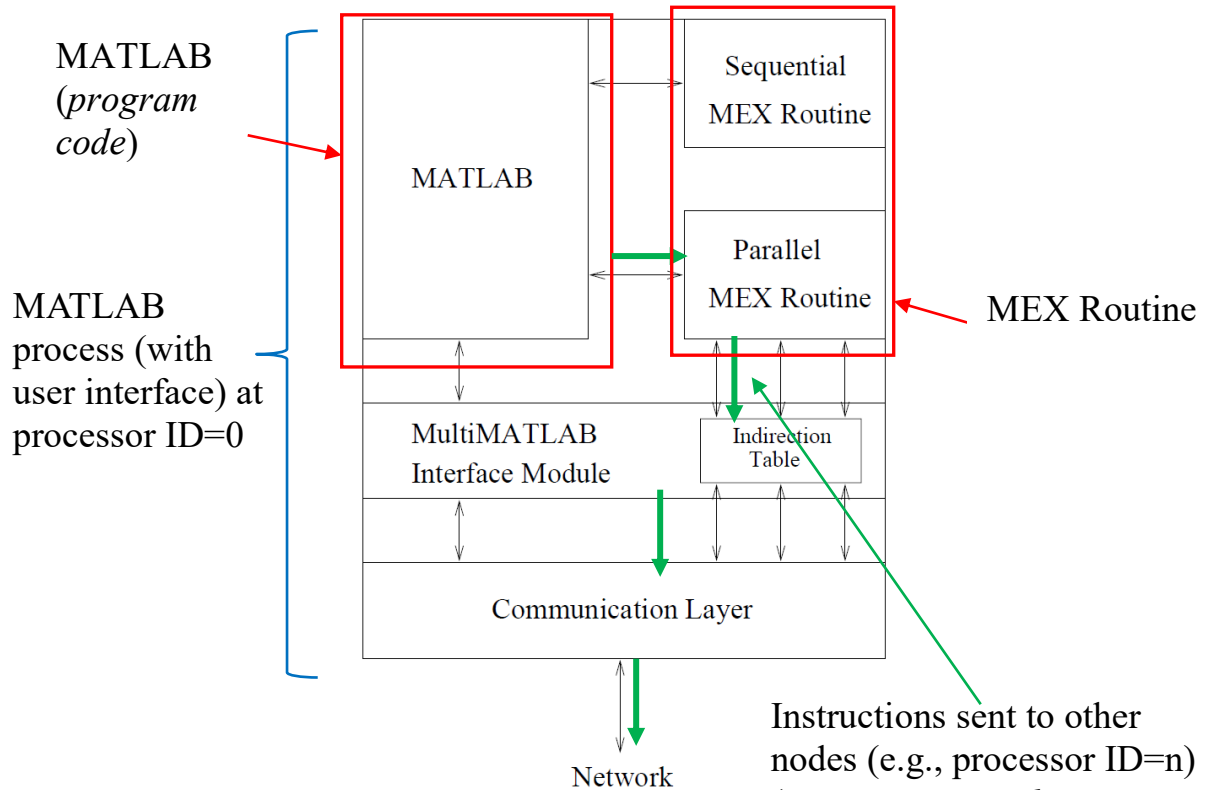


Figure 2: MATLAB Process Architecture
Instructions sent to other nodes (e.g., processor ID=n) (communicate at least some of the user instructions using the communications network interface)

Ex.1005, FIG. 2 (annotated)

233. 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 so that the MATLAB processes can communicate tasks and data directly with each other over a network, thereby rendering obvious *after accepting the user*

instructions, communicate at least some of the user instructions using the communications network interface for the nodes to communicate with each other.

5. Claim 2

- a. [2.0] *The computer cluster of claim 1, 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.***

234. Menon in view of Trefethen render 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 they teach MultiMATLAB running on “any network” of Matlab workstations.

235. Menon discloses that MultiMATLAB is intended to run on “any network of workstations supporting Matlab” and that the MPICH implementation of MultiMATLAB “run[s] over a network of workstations connected by TCP/IP.” Ex.1005, 2, 5. Trefethen also teaches that its MultiMATLAB implementation was run on machines “connected to Cornell’s Computer Science Department network.” Ex.1006, 2. A POSITA would have recognized the Cornell network as an example of a local area network (LAN).

6. Claim 3

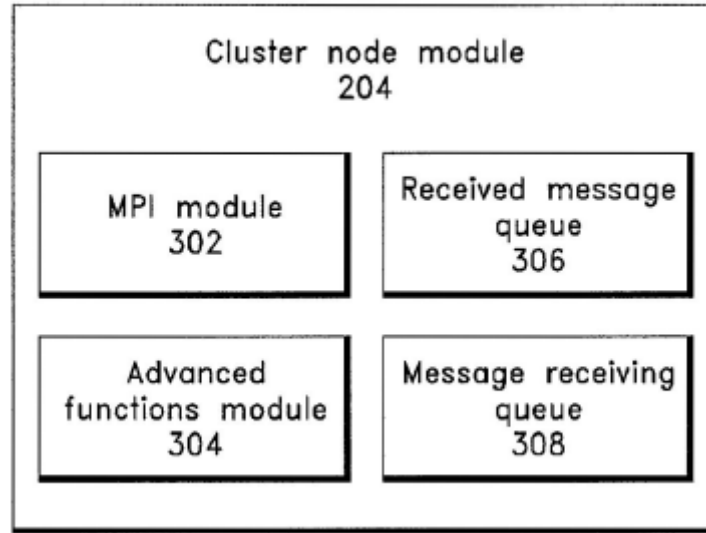
- a. ***[3.0] The computer cluster of claim 1, wherein each of the nodes comprises one or more cluster node modules.***

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

237. The '679 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, 11:45-57. Fig. 3 depicts a single cluster node module comprising multiple sub-components:



Ex.1001, FIG. 3

238. FIG. 2 of the '679 patent depicting the cluster node modules 204a-e is shown below.

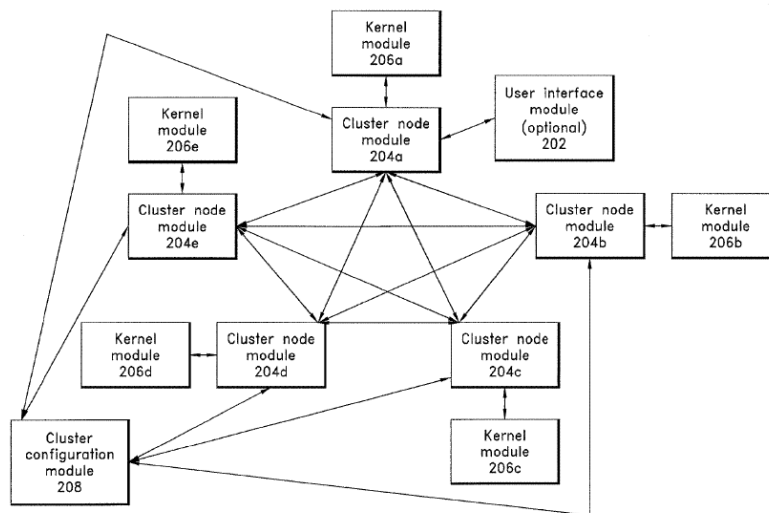


FIG. 2

Ex.1001, FIG. 2

239. Similarly, Menon discloses that each processor of “the MultiMATLAB architecture ... individually runs a MATLAB process” that includes MEX Routines, a set of components to provide interconnection with other modules, a “MultiMATLAB interface module,” a “key component in [the MultiMATLAB] architecture,” along with an “indirection table” and “communication layer.” Ex.1005, 3, 5. The “MEX routines” provide “[a]ll parallel functionality in the MultiMATLAB system.” Ex.1005, 3. “The interface module, shown in Figure 2, is responsible for initializing an underlying communication layer, such as MPI ..., or any other package available on the platform, and exposing it to the rest of the system.” Ex.1005, 3. A POSITA would recognize that the MEX Routines, interface module, indirection table, and communication layer correspond to a *cluster node module* of the ’679 patent because the ’679 patent describes its cluster node module as a grouping of subcomponents, as shown in its Fig. 3 depicted above.

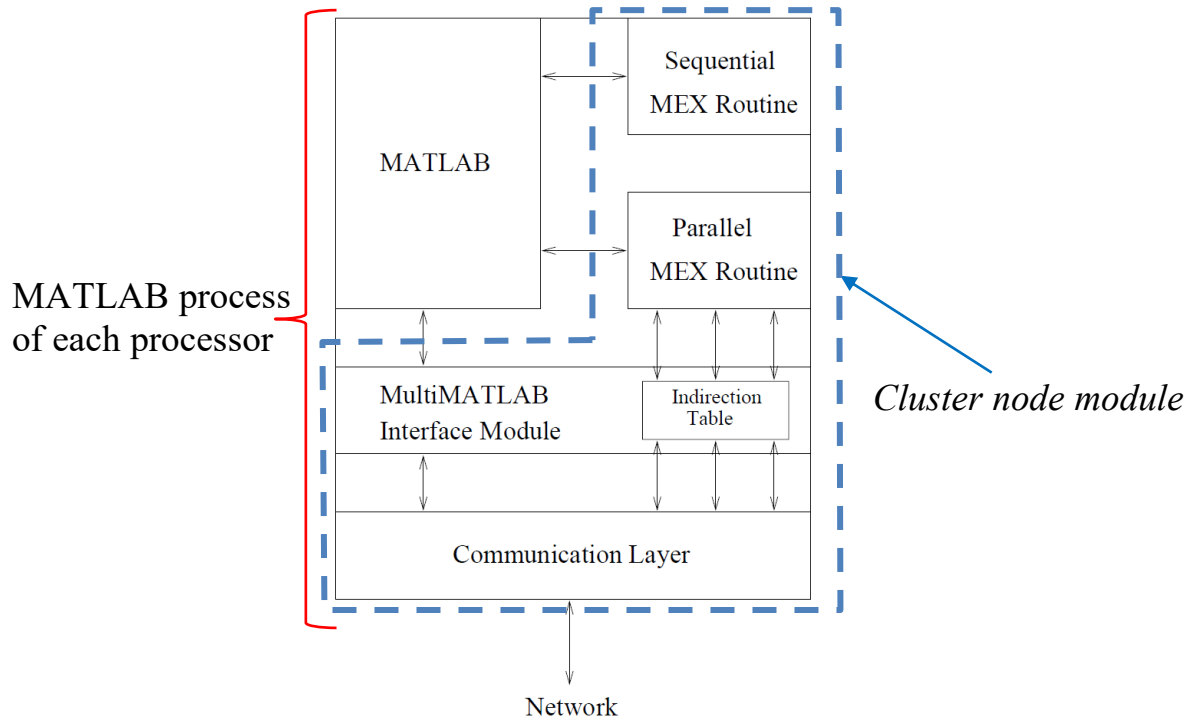


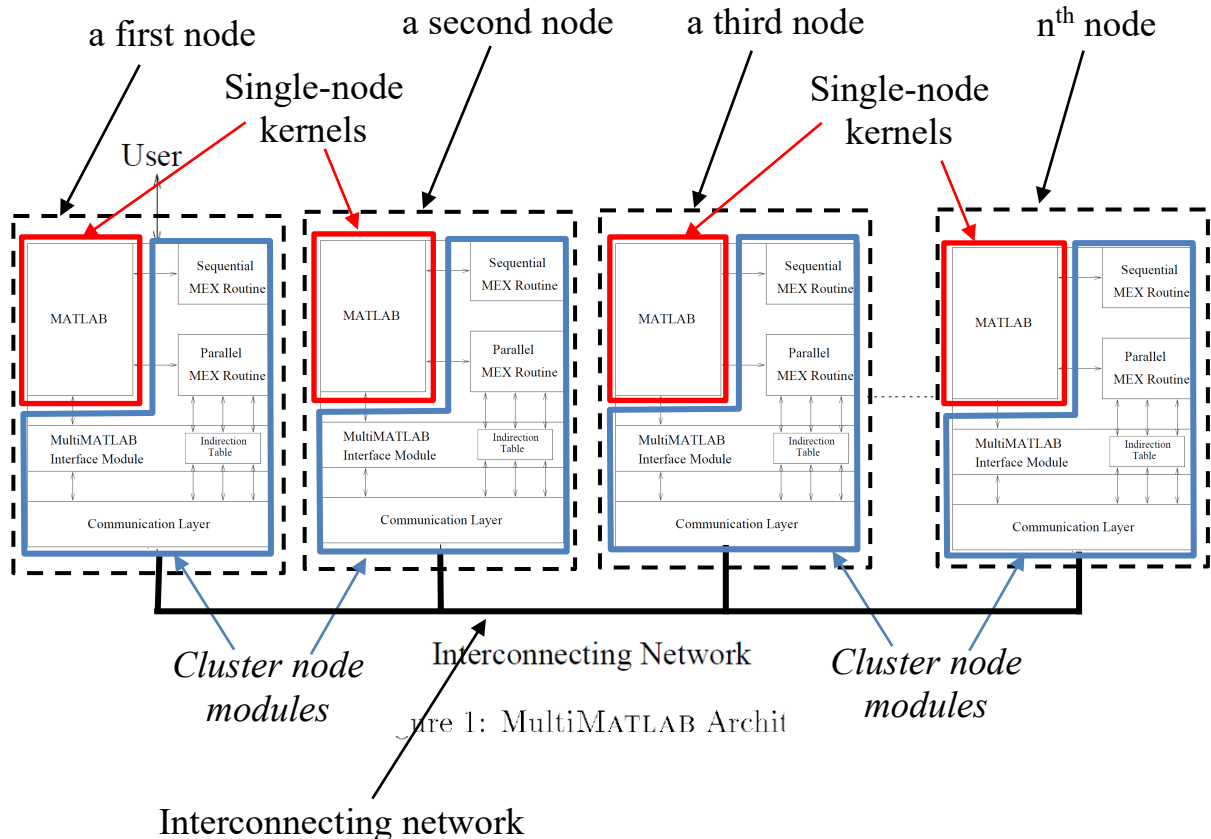
Figure 2: MATLAB Process Address Space

Ex.1005, FIG. 2 (annotated)

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

interconnection network with the communication layer in the other nodes. Ex.1005,

3.



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

241. Accordingly, Menon describes each processor of the MultiMATLAB architecture having a MATLAB process, which in turn has MEX Routines, a MultiMATLAB interface module, indirection table, and a communication layer (*cluster node module*), rendering obvious *wherein each of the nodes comprises one or more cluster node modules*.

7. Claim 4

- a. ***[4.0] The computer cluster of claim 1, wherein each single-node kernel is stored in a non-transitory computer-readable medium and configured to accept and execute a request.***

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

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

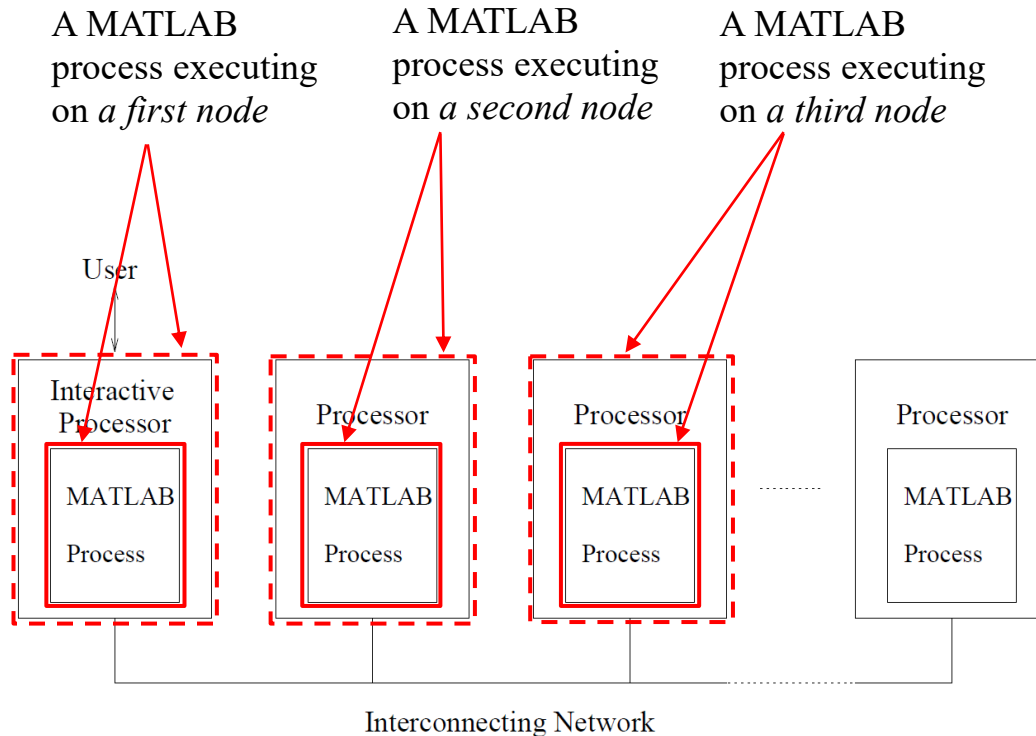


Figure 1: MultiMATLAB Architecture

Ex.1005, FIG. 1 (annotated).

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

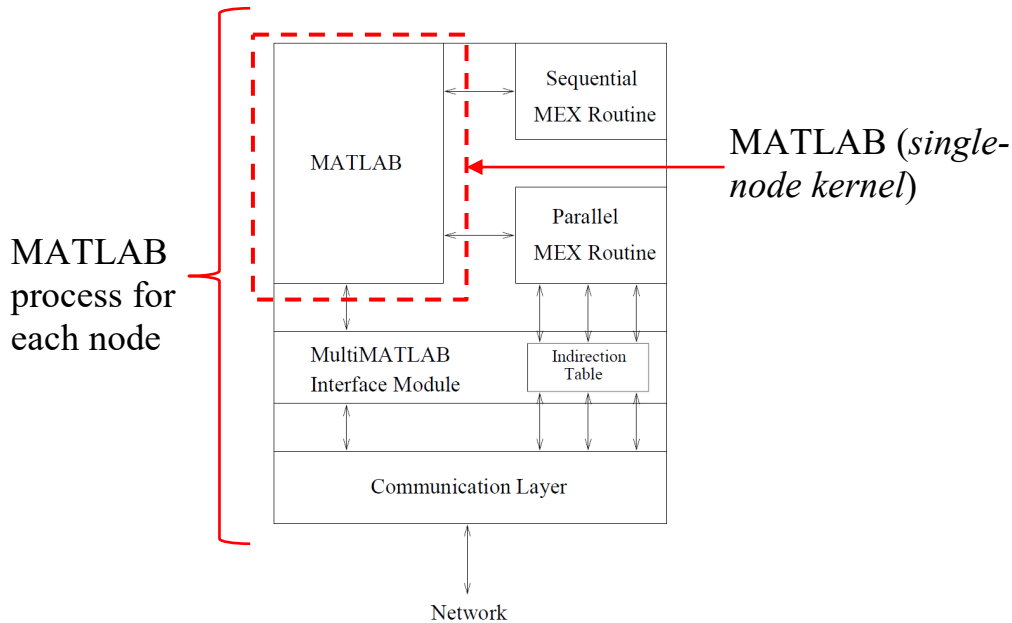


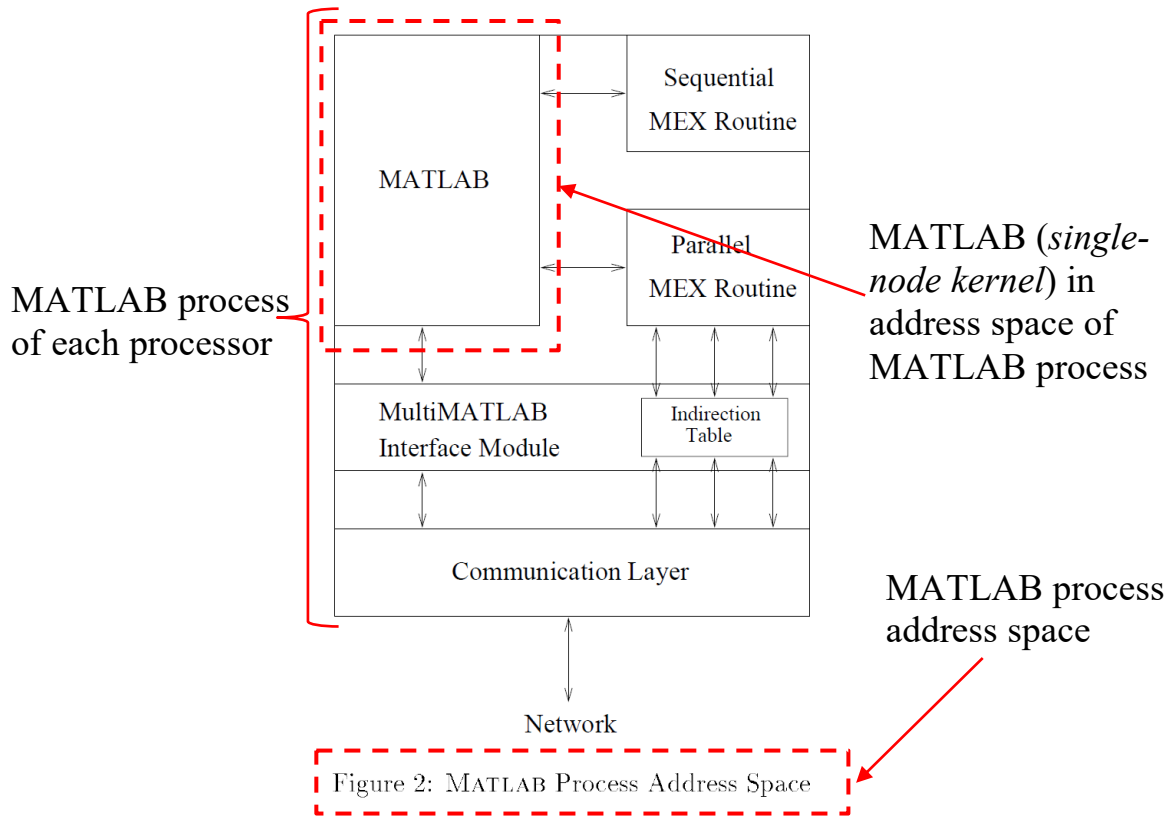
Figure 2: MATLAB Process Address Space

Ex.1005, FIG. 2 (annotated)

245. Because MATLAB is “MATLAB software,” a POSITA would recognize MATLAB is executed by a processor. Ex.1006, 2; Ex.1005, 3. Accordingly, Menon’s disclosure of MATLAB that is executed by a processor teaches *a single-node kernel*.

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

247. A POSITA would understand an “address space” (as described in Menon) to be a “a list of memory locations from some minimum (usually 0) to some maximum, which the process can read and write. The address space contains the executable program, the program’s data, and its stack.” Ex.1018, 67-68. Because Menon’s address space refers to a computer’s memory, a POSITA would recognize that the MATLAB process and MATLAB are in a *computer-readable medium*, and that MATLAB is accessed from a *computer-readable medium*. FIG. 2 of Menon below shows MATLAB as part of the MATLAB process address space. The *computer-readable medium* of the processors of the IBM SP2, on which the conjugate gradients algorithm is implemented, can be in the form of internal and external hard disk drives, or memory cards, as taught by RS6000. Ex.1007, 6, 10 (disclosing IBM SP2 processors have internal and external hard disk drives), 10 (disclosing IBM SP2 processors have “one to four memory cards” that support a “maximum of 64 GB”).



Ex.1005, FIG. 2 (annotated)

248. Third, Menon discloses that MATLAB of a MATLAB process on each processor awaits for MATLAB commands and executes routines based on those MATLAB commands when received. Ex.1005, 3-4. Specifically, Menon teaches that to have user instructions executed on all the MATLAB processes of the MultiMATLAB architecture, a “user interacts directly with one MATLAB process, called the interactive process, and operates within that process’s MATLAB environment.” Ex.1005, 3. For example, the user may use the “Eval routine for parallel evaluation,” which is a “routine [that] allows users to execute commands on the other processes.” Ex.1005, 4. “Other MATLAB processes await commands

from the interactive process. These other processes are used either by explicitly sending MATLAB commands to them or by executing routines that, in turn, use them.” Ex.1005, 3.

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

250. Trefethen provides an example illustration where a user “connected to a node of the IBM SP2, running MATLAB” is operating in a MultiMATLAB

architecture where “6 MultiMATLAB processes [are] running.” Ex.1006, 3. If the user types the command Eval('ID'), the result is

ans = 0

ans = 1

ans = 5

ans = 2

ans = 3

ans = 4,

indicating that “the MultiMATLAB command ID” is executed on each MATLAB process. Ex.1006, 3. That is the case because “Eval routine is implemented as a MEX Routine which accepts a vector of MATLAB process IDs and a MATLAB command. This routine sends the commands over the underlying communication layer to the processes corresponding to the specified IDs.” Ex.1005, 4. “[E]ach command passed to Eval was executed on all MATLAB processes” unless a “subset of the processes [is selected] by passing two arguments to the Eval command, the first being a vector of process IDs.” Ex.1006, 4. In other words, the interactive as well as all the non-interactive MATLAB processes of “the 6 MultiMATLAB processes” receive and execute commands thereon, generating the above result. Ex.1006, 3.

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

8. Claim 5

- a.** *[5.0] The computer cluster of claim 4, wherein each of the nodes comprises one or more cluster node modules, wherein each of the cluster node modules comprises instructions stored in a non-transitory computer-readable medium, and wherein the instructions, when executed by the hardware processor, cause the cluster node module to communicate with the single-node kernel and with one or more other cluster node modules.*

252. Menon discloses *wherein each of the nodes comprises one or more cluster node modules, wherein each of the cluster node modules comprises instructions stored in a non-transitory computer-readable medium, and wherein the instructions, when executed by the hardware processor, cause the cluster node module to communicate with the single-node kernel and with one or more other cluster node modules* because it teaches a set of components (MEX routines, a MultiMATLAB interface module, an indirection table, and a communication layer of a MATLAB process) are computer software that, when executed by the

processor of the MATLAB process, communicates with MATLAB and the communications components of the other nodes.

253. First, in limitation [3.0], it was shown that each processor of the MultiMATLAB architecture has a MATLAB process, which in turn has a set of components for handling communications with other processors (MEX routines, a MultiMATLAB interface module, an indirection table, and a communication layer (*cluster node module*)).

254. Second, in limitation [1.1.2], it was shown that the MATLAB process address space on each processor is in a computer-readable medium in the form of internal or external hard disk drives, or memory cards (e.g., RAMs) that can be accessed by the processor of the MATLAB process. A POSITA would recognize that the MATLAB process, which includes the MEX routines, the MultiMATLAB interface module, the indirection table, and the communication layer (*cluster node module*), is software (*computer instructions*) stored in memory address space on either a hard drive or in RAM for execution by the processor. Ex.1005, 4 (Menon disclosing that “[i]t is important to note that the interface module exists in the corresponding MATLAB process’s address space.”); FIG. 2 (showing the communication layer for handling communications with other processors in the MATLAB process address space). FIG. 2 of Menon below shows the MEX

routines, the MultiMATLAB interface module, the indirection table, and the communication layer (*cluster node module*) in the MATLAB process address space.

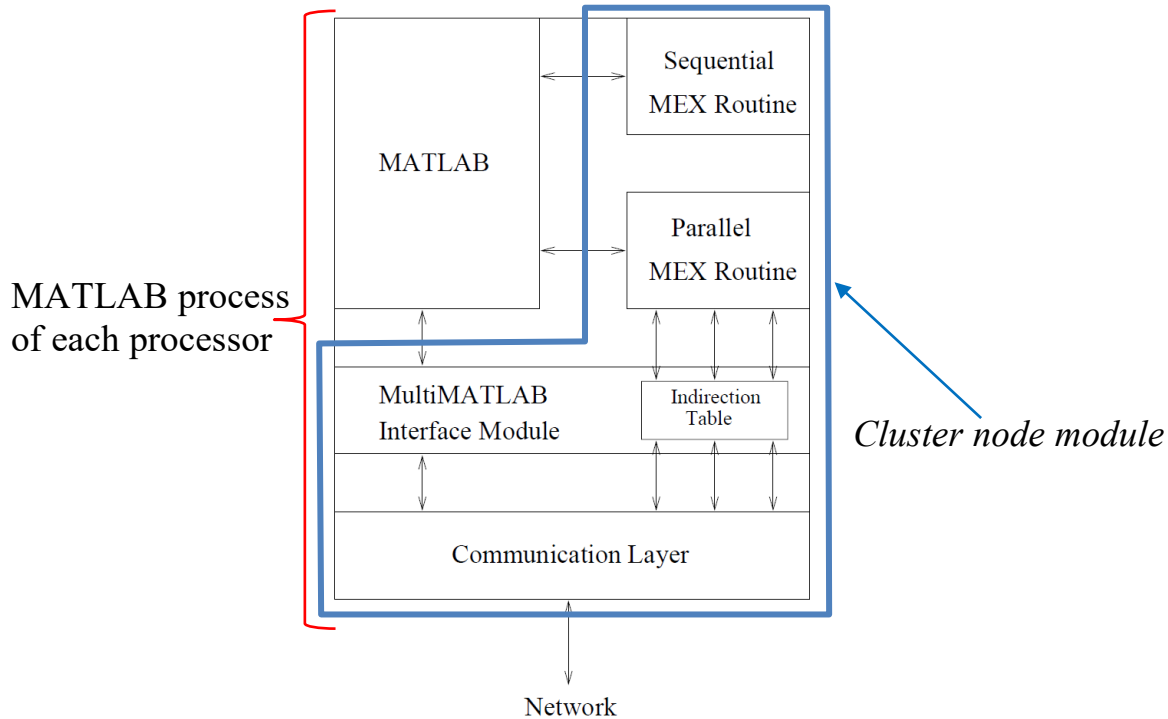


Figure 2: MATLAB Process Address Space

Ex.1005, FIG. 2 (annotated)

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

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

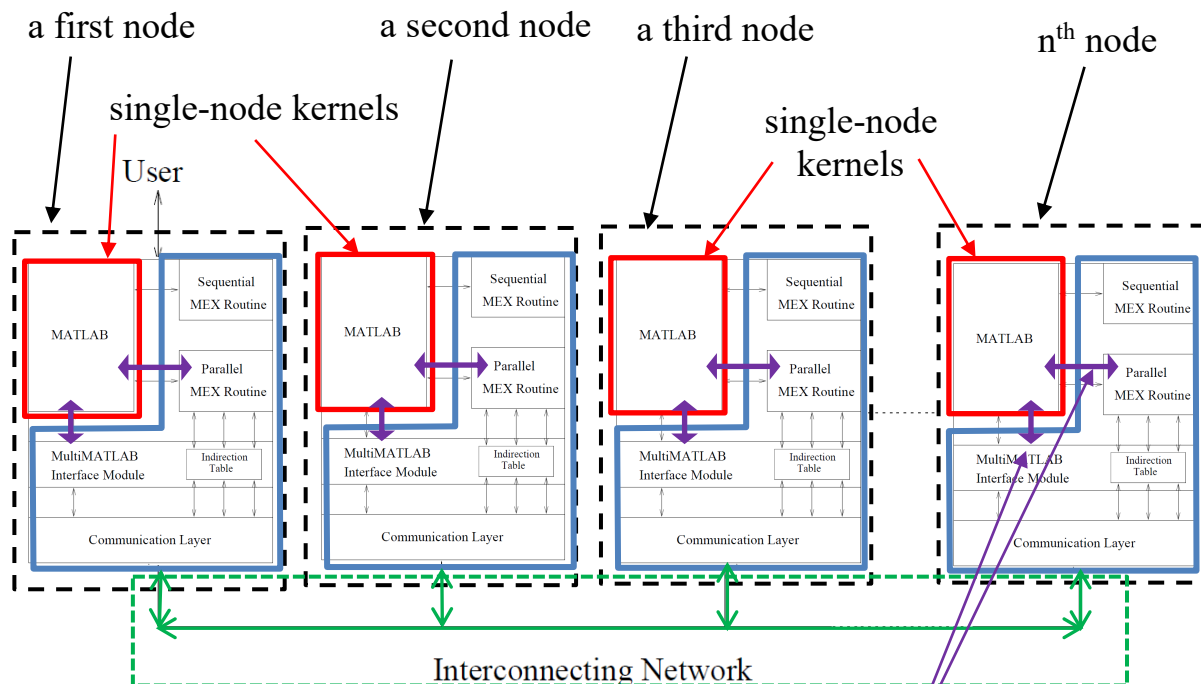


Figure 1: MultiMATLAB Architecture

*Cluster node modules
communicating to and
from each other*

*Single-node kernel
communicating to and
from cluster node module*

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

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

9. Claim 6

- a. ***[6.0] The computer cluster of claim 5, wherein the one or more cluster node modules act as a cluster.***

257. Menon, Trefethen, and POEref renders obvious *wherein the one or more cluster node modules act as a cluster* because Menon describes that the MEX routines, the MultiMATLAB interface module, the indirection table, and the communication layer of each MATLAB process work together to enable a user to solve a mathematical problem as if they were a single computer, POEref describes how POE executes parallel programs on a networked cluster, and Trefethen

describes how the user can issue a single command to cause all the processors to run at the same time.

258. Menon discloses that “[i]n the MultiMATLAB architecture ... each processor in a parallel platform individually runs a MATLAB process.” Ex.1005, 3. An “important goal of the MultiMATLAB system is to allow users to integrate high performance parallel routines.” Ex.1005, 6. A user can run commands on all the MATLAB processes by “interact[ing] directly with one MATLAB process, called the interactive process, and operat[ing] within that process’s MATLAB environment,” and then by “explicitly sending MATLAB commands to [the other MATLAB processes] or by executing routines that, in turn, use them” as the “[o]ther MATLAB processes await commands from the interactive process.” Ex.1005, 3. For example, the user can utilize:

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

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

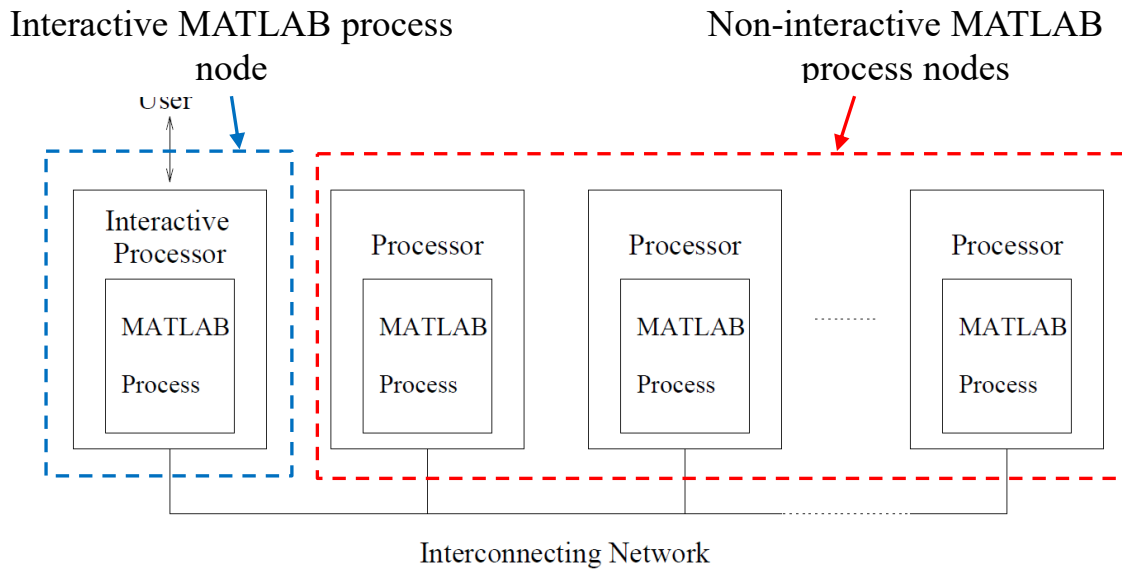


Figure 1: MultiMATLAB Architecture

Ex.1005, FIG. 1 (annotated)

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

MPI-F implementation of MultiMATLAB [that] was design specifically for the IBM SP2. ... In this implementation, all MATLAB processes are started via POE, IBM’s Parallel Operating Environment. For each of

p MATLAB processes, POE assigns an identification number between 0 and $p-1$. The process numbered 0 is considered the interactive MATLAB. All other processes wait for commands from the interactive MATLAB process.

Ex.1005, 6 (emphasis original). IBM's Parallel Environment is used "to execute parallel programs on ... a networked cluster of [IBM SP2] RS/6000 processors." POEref, 1. That is, the interactive and non-interactive MATLAB processes of the MultiMATLAB architecture executing commands in parallel are a group of processors communicating with each other to accomplish a mathematical computation as if they were a single computer (*act as a cluster*). Ex.1005, 6; POEref, 1.

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

```
Eval( 'sqrt(2)' )
```

then the MATLAB command sqrt(2) is executed on all six processors. The result is six repetitions of 1.4142. ...

On the other hand the command

```
Eval( 'ID' )
```

calls the MultiMATLAB command ID on each of the processors running. This command returns the number of the current process, an integer from 0 to Nproc-1.

Running it on all nodes might give the result

```
ans = 0
```

```
ans = 1
```

```
ans = 5
```

```
ans = 2
```

```
ans = 3
```

```
ans = 4.”
```

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

261. Further, as discussed in limitation [5.0], Menon discloses each MATLAB process on a node has a set of components (MEX routines, a

MultiMATLAB interface modules, an indirection table, and a communication layer) (*the one or more cluster node modules*) to communicate with other nodes in the MultiMATLAB architecture. The communications can be or include mathematical results. Ex.1005, 10-13. For example, Menon describes “MultiMATLAB implementations of the conjugate gradients algorithm on the IBM SP2” so that all the MATLAB processes are working in parallel together (*act as a cluster*). Ex.1005, 11.

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

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

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

Multiple MATLAB processes executing commands in parallel while communicating values (*act as a cluster*)

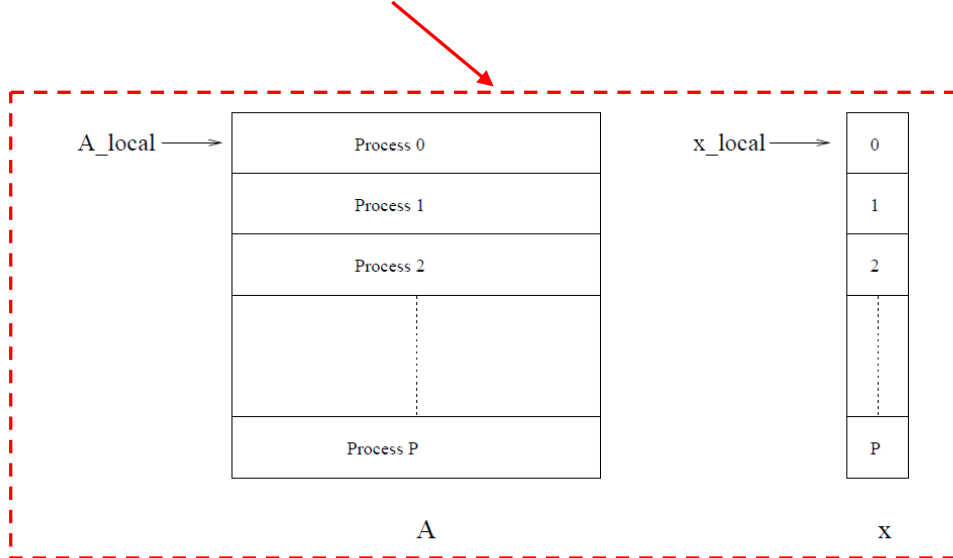


Figure 6: Matrix A and vector x distributed by row over P processors

Ex.1005, FIG. 6

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

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

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

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

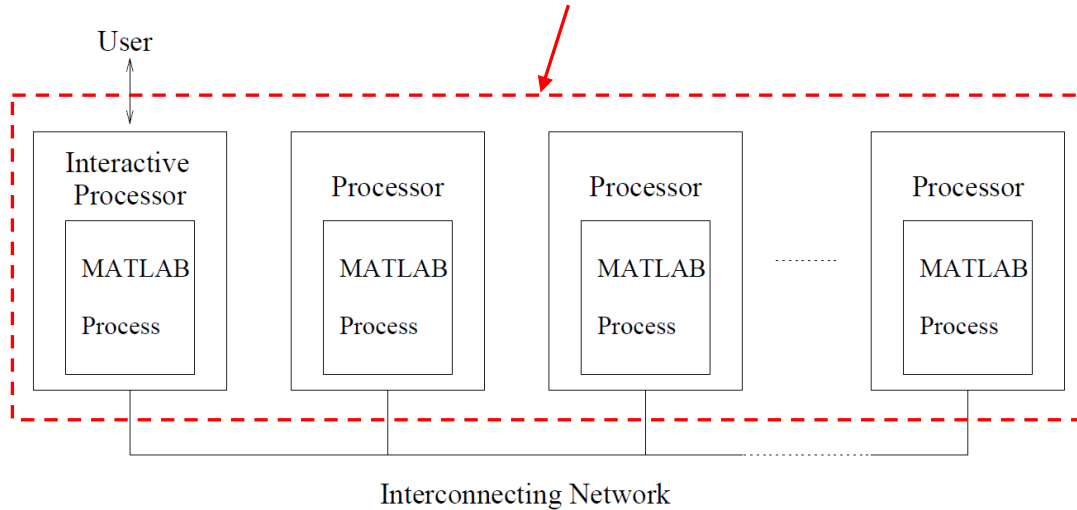


Figure 1: MultiMATLAB Architecture

Ex.1005, FIG. 1 (annotated)

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

10.Claim 7

- a. ***[7.0] The computer cluster of claim 5, wherein the one or more cluster node modules communicate with one another to act as a cluster.***

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

267. As discussed in limitation [6.0], Menon describes that the MEX routines, the MultiMATLAB interface module, the indirection table, and the communication layer of each MATLAB process (*the one or more cluster node modules*) work together to enable a user to solve a mathematical problem as if they were a single computer and Trefethen describes how the user can issue a single command to cause all the processors of the MultiMATLAB architecture to execute

at the same time. It was also shown in limitation [6.0] that the MultiMATLAB architecture can be implemented in IBM SP2 as “a networked cluster of [IBM SP2] RS/6000 processors,” in which case all the MATLAB processes execute commands in parallel are a group of processors communicating with each other to accomplish a mathematical computation as if they were a single computer. Ex.1005, 6; POEref, 1.

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

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

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

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

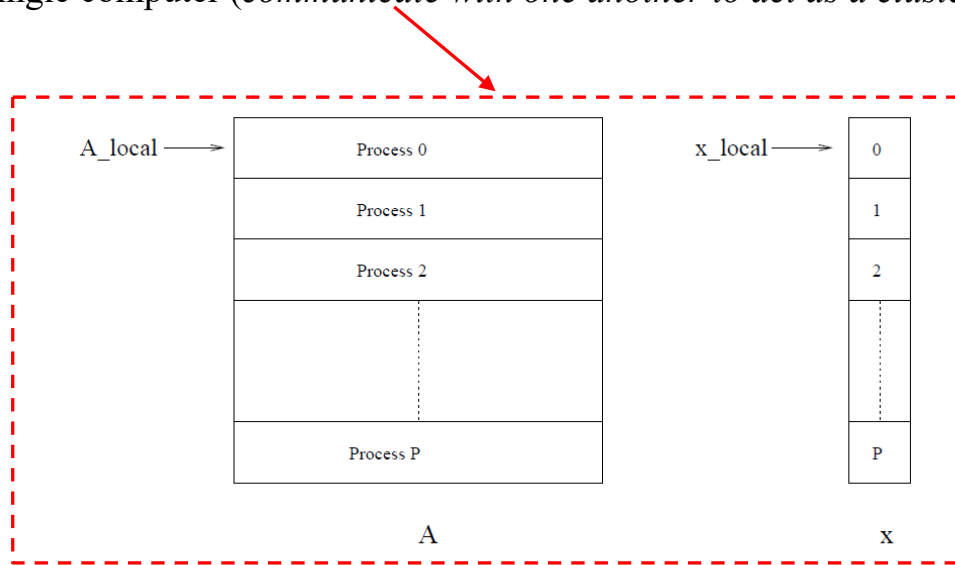


Figure 6: Matrix A and vector x distributed by row over P processors

Ex.1005, FIG. 6

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

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

11.Claim 8

- a. [8.0] *The computer cluster of claim 1, wherein the computer cluster includes the user interface.***

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

271. As discussed in limitation [6.0], Menon describes a MultiMATLAB architecture (*computer cluster*), shown below in FIG. 1 of Menon, in which all the MATLAB processes are running in parallel to solve a mathematical problem.

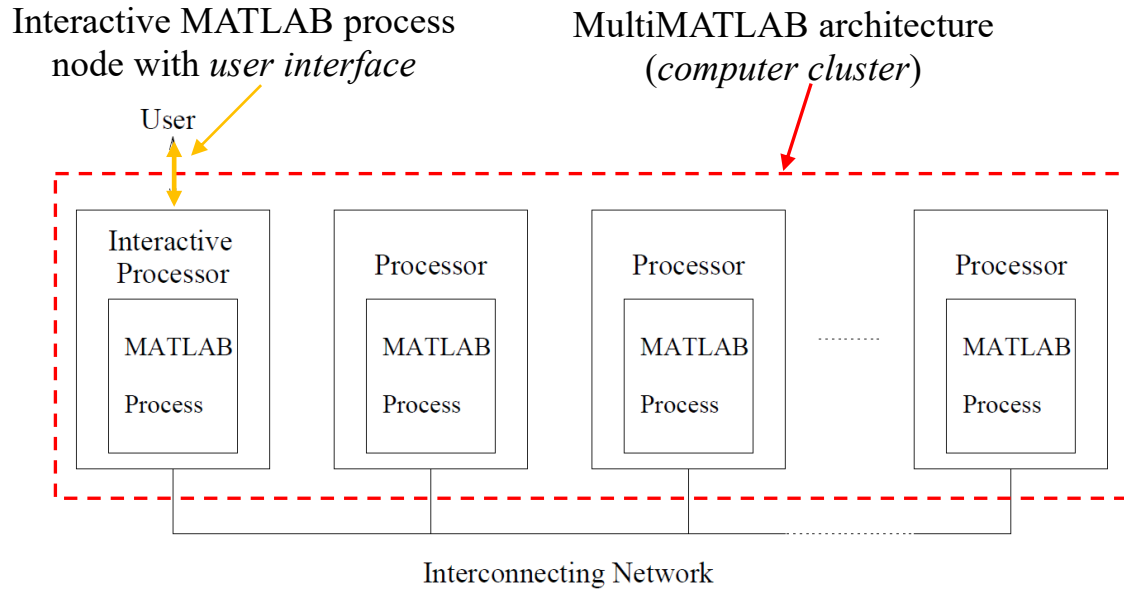


Figure 1: MultiMATLAB Architecture

Ex.1005, FIG. 1 (annotated)

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

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

12.Claim 9

- a. ***[9.0] The computer cluster of claim 5, wherein each cluster node module accepts instructions from the user interface and interprets one or more of the instructions.***

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

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

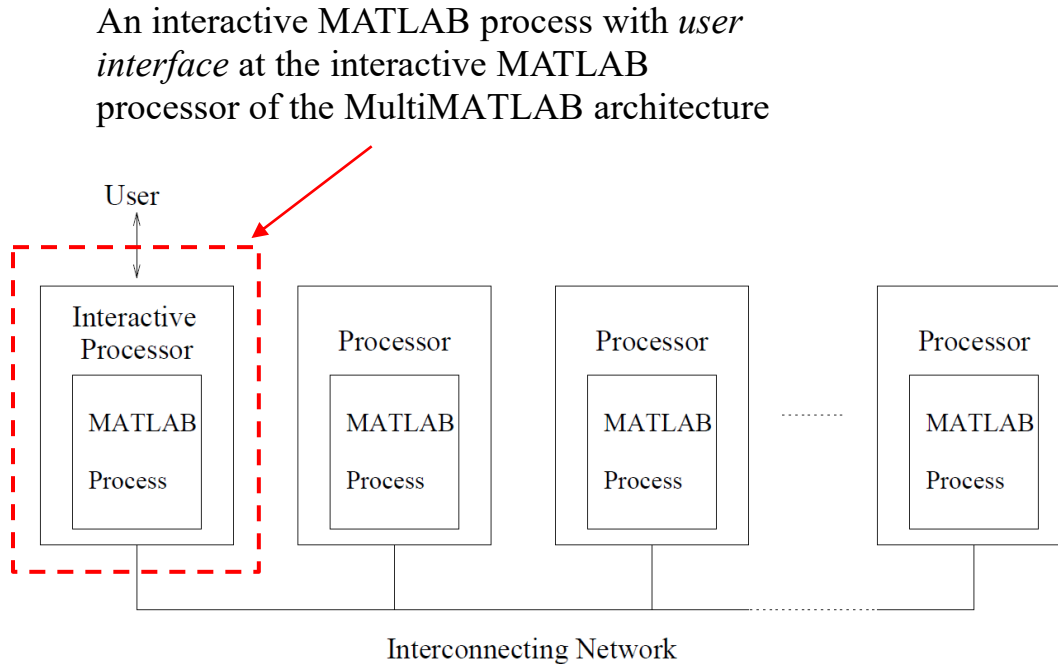


Figure 1: MultiMATLAB Architecture

Ex.1005, FIG. 1 (annotated).

276. Second, as discussed in [5.0], Menon discloses that each MATLAB process of the MultiMATLAB architecture has MEX routines, a MultiMATLAB interface module, an indirection table, and a communication layer (*a cluster node module*) for communicating with other nodes. **Third**, as discussed in limitation [1.7.2], Menon describes that (i) the interactive MATLAB process receives the user instructions and (ii) those instructions are evaluated in order to send commands to the other MATLAB processes. A combination figure of FIG. 1 and FIG. 2 of Menon below schematically shows the *cluster node modules* of each MATLAB process and communication of user instructions from the interactive MATLAB process to all the other (non-interactive) MATLAB processes.

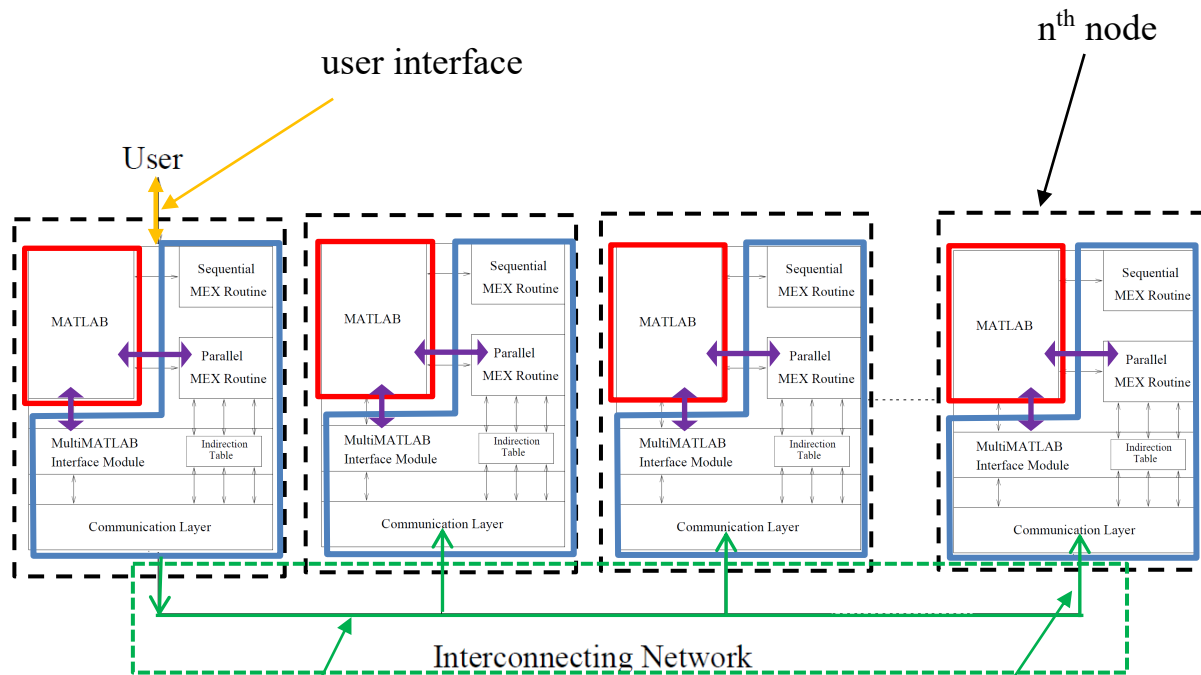


Figure 1: MultiMATLAB Architecture

Instructions transmitted to other nodes using the communication layer

Instructions received from interactive processor via the network

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

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

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

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

Ex.1005, 6.

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

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

13.Claim 10

- a. ***[10.0] The computer cluster of claim1, wherein the program code, when executed, is further capable of causing the hardware processor to execute a cluster***

***initialization process configured to establish
communication among two or more of the nodes.***

280. Menon in view of POEref renders obvious this element. Specifically, Menon discloses an IBM SP2 processor from which a user, by utilizing IBM's POE, launches an interactive MATLAB process on that IBM SP2 processor and a non-interactive MATLAB process on each of the other IBM SP2 processors, where the non-interactive MATLAB processes wait for further commands from the interactive MATLAB process. Further, POEref discloses that POE initializes a local environment, i.e., a MATLAB process, on a processor node and reproduces that local environment on each of the remote processor nodes. The combination of Menon and POEref thus renders obvious *wherein the program code, when executed, is further capable of causing the hardware processor to execute a cluster initialization process configured to establish communication among two or more of the nodes.*

281. 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 (*program code*) with which the user interacts (program code that is *executed*). Ex.1005, 3. The user invokes the initialization process using the “poe” command:

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

Ex.1008, 28.

282. In other words, when a user on an IBM SP2 invokes **poe**, a local environment, i.e., a MATLAB process (*program code*), is initiated on the IBM SP2 processor (*hardware 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”). Invoking **poe** also

automatically loads the appropriate communication subsystem on each remote node (*initialization process configured to establish communication among two or more of the nodes*). Specifically, “[o]n non-interactive processes, a separate top-level MEX Routine is immediately run upon initialization of the system. This MEX Routine runs in a loop in which it waits for MATLAB commands to arrive from the interactive process and executes them in its own MATLAB environment.” Ex.1005, 4. The initialization of MATLAB processes on the IBM SP2 are illustrated in FIG. 1 of Menon below.

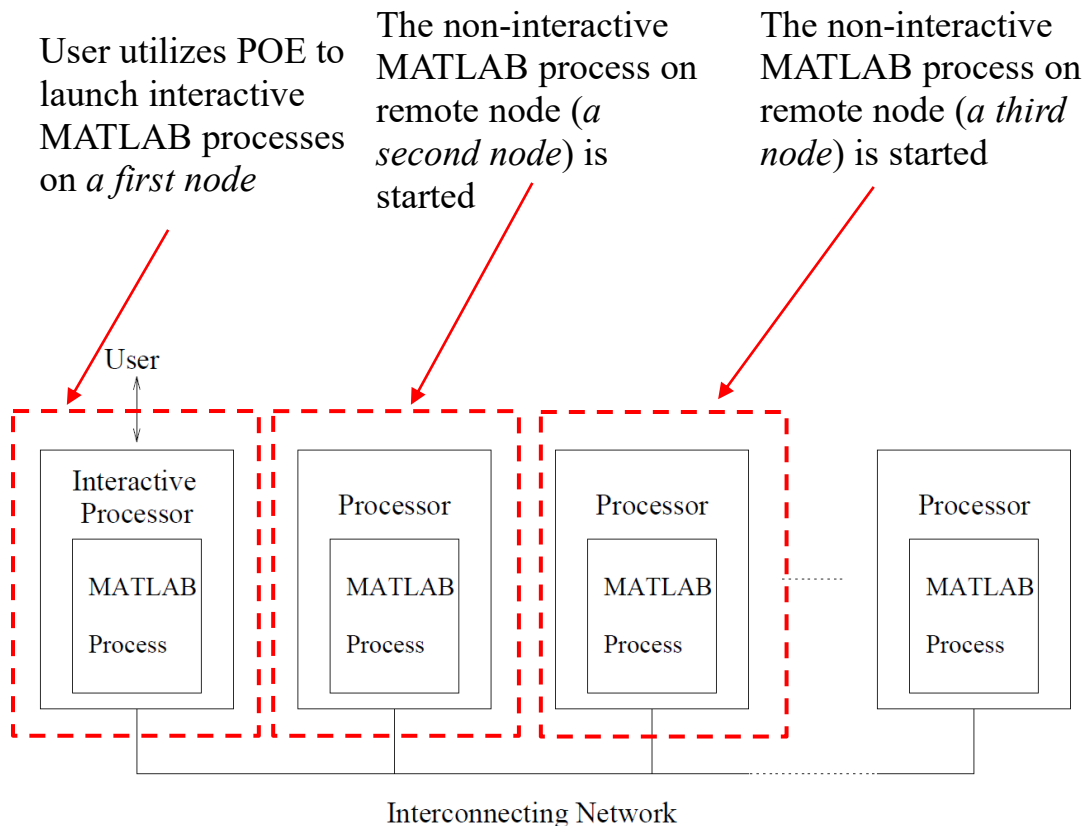


Figure 1: MultiMATLAB Architecture

Ex.1005, FIG. 1 (annotated)

283. Accordingly, Menon discloses an IBM SP2 processor from which a user, utilizing IBM's POE, launches an interactive MATLAB process on that IBM SP2 processor and a non-interactive MATLAB process on each of the other IBM SP2 processors (where the non-interactive MATLAB processes wait for further commands from the interactive MATLAB process), and POEref discloses that POE initializes a local environment, i.e., a MATLAB process, on the processor node the user is using and reproduces that local environment on each of the remote processor nodes, including automatically loading the appropriate communication subsystem on each remote node, the combination of Menon and POEref thus rendering obvious *wherein the program code, when executed, is further capable of causing the hardware processor to execute a cluster initialization process configured to establish communication among two or more of the nodes.*

14.Claim 11

- a. [11.0] *The computer cluster of claim 1, wherein the cluster initialization process comprises configuring access by one or more of the nodes to a computer-readable medium comprising program code for a single-node kernel.***

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

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

285. First, in limitation [4.0], it was shown that the MATLAB process address space having MATLAB teaches that *each single-node kernel is stored in a non-transitory computer-readable medium*.

286. Second, in limitation [10.0], it was shown that Menon discloses a user utilizing IBM's POE on an IBM SP2 system, in which a MultiMATLAB architecture is implemented, to launch a MATLAB process (which includes MATLAB) on each IBM SP2 processor node of the MultiMATLAB architecture, teaching *the cluster initialization process*.

287. Third, when a user starts a job with POE, the LoadLeveler of POE "allocates [] nodes to the job." Ex.1008, 54. Accordingly, a POSITA would understand that, when a job (i.e., the *single-node kernel*) is allocated to a node when

the job is being started (i.e., during the initialization process), the node is configured to access the computer-readable medium having the program code for the job.

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

15.Claim 12

- a. ***[12.0] The computer cluster of claim 11, wherein the cluster initialization process comprises establishing message-passing support among the nodes in the cluster.***

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

290. In limitation [10.0], it was shown that Menon discloses a user utilizing IBM's POE on an IBM SP2 system, "a modern high performance distributed memory multiprocessor" in which a MultiMATLAB architecture is implemented, to launch a MATLAB process on each IBM SP2 processor node, teaching *the cluster initialization process*. Ex.1005, 5. This implementation of MultiMATLAB architecture "is based upon MPI-F [6], IBM's proprietary version of MPI specifically optimized for the SP2." Ex.1005, 5.

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

292. Accordingly, Menon describes using POE to launch a MATLAB process on each IBM SP2 processor node of the MultiMATLAB architecture, in view of POEref describing that POE ensures that the MPI communication API is initialized properly so that each node can communicate, renders obvious *wherein*

the cluster initialization process comprises establishing message-passing support among the nodes in the cluster.

16.Claim 13

- a. ***[13.0] The computer cluster of claim 11, wherein the cluster initialization process comprises launching cluster node modules by the cluster.***

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

294. In limitation [10.0], it was shown that Menon discloses a user utilizing IBM's POE on an IBM SP2 system, in which a MultiMATLAB architecture is implemented, to launch a MATLAB process on each IBM SP2 processor node of the MultiMATLAB architecture, teaching *the cluster initialization process*. In limitation [4.0], it was shown that each MATLAB process includes MEX Routines, a MultiMATLAB interface module, indirection table, and a communication layer (*a cluster node module*), as shown in FIG. 2 of Menon below.

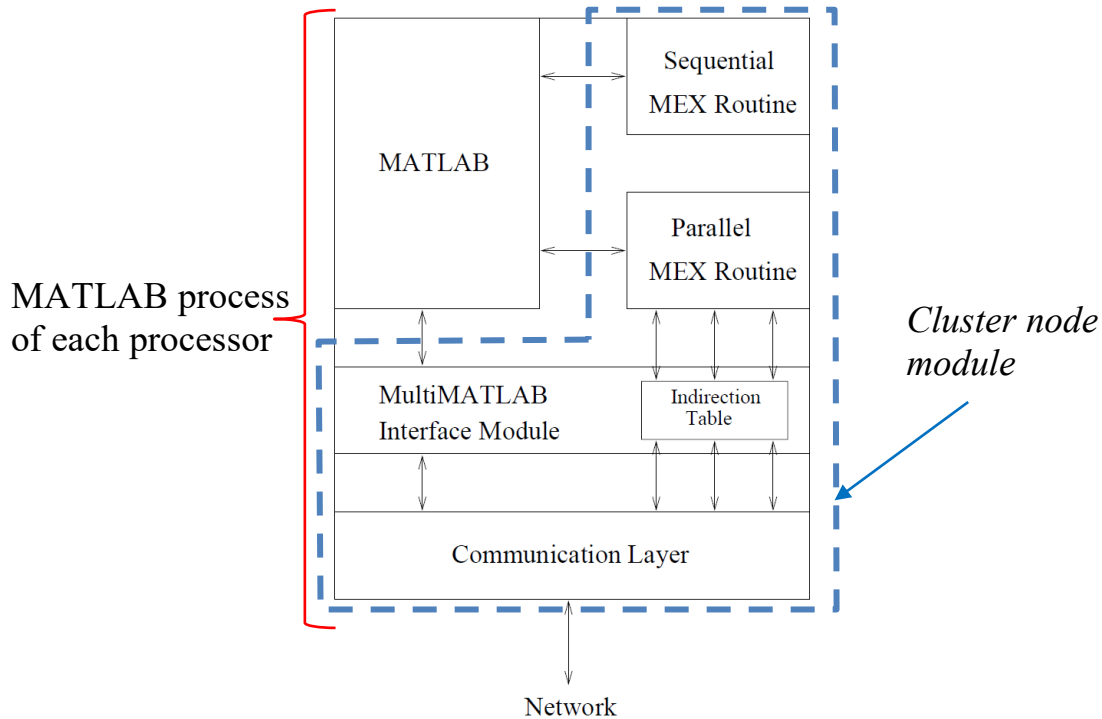


Figure 2: MATLAB Process Address Space

Ex.1005, FIG. 2 (annotated)

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

17.Claim 14

- a. ***[14.0] The computer cluster of claim 13, wherein the cluster initialization process comprises assigning a processor identification number to each of the cluster node modules.***

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

297. In limitation [10.0], it was shown that Menon discloses a user utilizing IBM's POE on an IBM SP2 system, in which a MultiMATLAB architecture is implemented, to launch a MATLAB process on each IBM SP2 processor node, teaching *the cluster initialization process*. Specifically, Menon explains that “[f]or each of p MATLAB processes, POE assigns an identification number between 0 and $p-1$. The process numbered 0 is considered the interactive MATLAB.”
Ex.1005, 6.

298. In limitation [3.0], it was shown that each MATLAB process includes MEX Routines, a MultiMATLAB interface module, indirection table, and a communication layer (*a cluster node module*), as shown in FIG. 2 of Menon below.

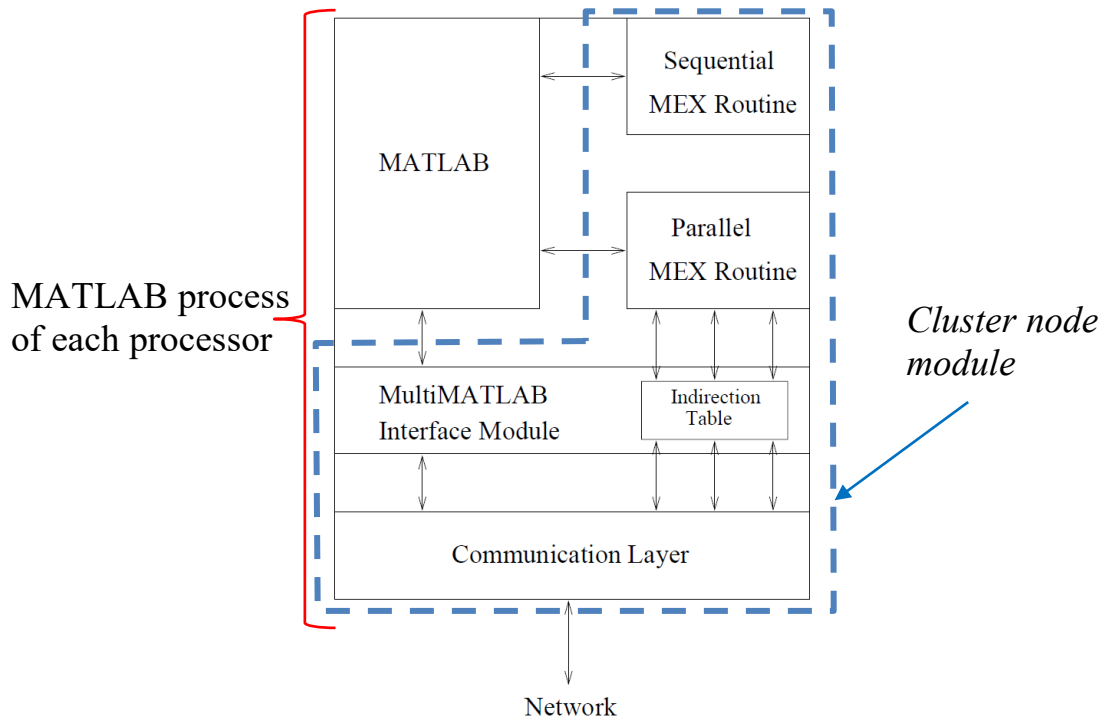


Figure 2: MATLAB Process Address Space

Ex.1005, FIG. 2 (annotated)

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

18.Claim 15

- a.** *[15.0] The computer cluster of claim 10, wherein the cluster initialization process comprises:*

300. Limitation [15.0] is disclosed or rendered obvious for the same reasons presented above for limitation [10.0].

**b. [15.1] launching cluster node modules by the cluster;
and**

301. Limitation [15.1] is disclosed or rendered obvious for the same reasons presented above for limitation [13.0].

**c. [15.2] after launching the cluster node modules,
configuring access by one or more of the nodes to a non-
transitory computer-readable medium comprising
program code for a single-node kernel.**

302. As discussed in limitation [13.0], Menon teaches *launching cluster node modules*.

303. Further, as discussed in limitation [11.0], Menon teaches using POE to initialize an IBM SP2 system in which a MultiMATLAB architecture is implemented (*the cluster initialization process*) by launching a MATLAB process having a MATLAB process address space (that includes MATLAB (*single-node kernel*)) on each IBM SP2 processor node and POEref discloses that when a user starts a job with POE, the LoadLeveler of POE “allocates [] nodes to the job.” Ex.1008, 54. Accordingly, a POSITA would understand that, when a job (i.e., a *single-node kernel*) is allocated to a node when the job is being started (i.e., during the initialization process), the node is configured to access the computer-readable medium having the program code for the job, rendering obvious *configuring access by one or more of the nodes to a non-transitory computer-readable medium comprising program code for a single-node kernel*.

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

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

19.Claim 16

- a. ***[16.0] The computer cluster of claim 10, wherein the cluster initialization process comprises:***

306. Limitation [16.0] is disclosed or rendered obvious for the same reasons presented above for limitation [10.0].

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

307. Limitation [16.1] is disclosed or rendered obvious for the same reasons presented above for limitation [13.0].

- c. ***[16.2] after launching the cluster node modules, establishing communication among two or more of the nodes; and***

308. As discussed in limitation [13.0], Menon teaches *the cluster initialization process comprises launching cluster node modules.*

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

310. In limitation [10.0], it was shown that Menon discloses a user utilizing IBM's POE on an IBM SP2 system, in which a MultiMATLAB architecture is implemented, to launch an interactive MATLAB process on that IBM SP2 processor node and the non-interactive MATLAB processes of each of the other

IBM SP2 processor nodes, and that POEref discloses the use of the “poe” command of POE to initialize a MATLAB process on each of the IBM SP2 processor nodes of the MultiMATLAB architecture. FIG. 1 of Menon below illustrates the initialization of MATLAB processes on the IBM SP2 MultiMATLAB architecture.

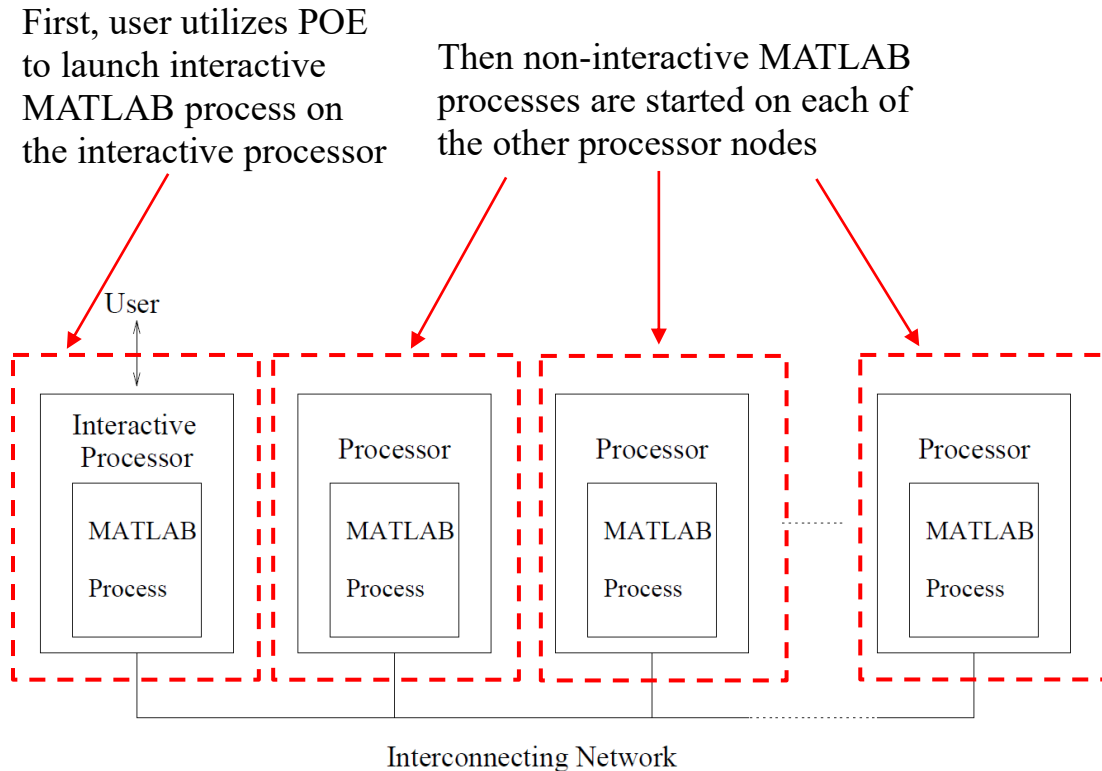


Figure 1: MultiMATLAB Architecture

Ex.1005, FIG. 1 (annotated)

311. POEref further discloses that during the initialization of MATLAB processes, POE ensures that the communication API is initialized properly, explaining that “POE now provides an option to enable you to specify whether your program will use MPI, LAPI [low-level application programming interface], or both. Using this option, POE ensures that each API initializes properly and informs

LoadLeveler which APIs are used so each node is set up completely.” Ex.1008, 3.

LoadLeveler is used “to allocate nodes” and LAPI is used to “run [a program] as a number of parallel tasks.” Ex.1008, 3 (LoadLeveler), 1-2 (LAPI). Further, POEref describes that the Partition Manager, which the command “poe” of POE starts to “allocate[] the nodes of your partition and initialize[] the local environment,” “connects standard I/O to each remote node so the parallel tasks can communicate with the home node.” Ex.1008, 3.

312. Menon teaches the use of POE on the interactive MATLAB process to start MATLAB processes on each processor node of the MultiMATLAB architecture (*the cluster initialization process*) and POEref teaches the “poe” command for POE that ensures that the communication API is initialized properly and that each processor node can communicate (*establishing communication among two or more of the nodes*), renders obvious *establishing communication among two or more of the nodes*.

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

step can occur, such as **before** or **after**, it is a simple and obvious design choice for a POSITA to make.

- d. ***[16.3] after establishing communication among two or more of the nodes, assigning a processor identification number to each of the cluster node modules.***

314. As discussed in limitation [16.2], Menon in view of POEref teaches that *the cluster initialization process comprises ... establishing communication among two or more of the nodes*. Specifically, POEref teaches the “poe” command for POE that ensures that the communication API is initialized properly and that each processor node can communicate (*establishing communication among two or more of the nodes*)

315. Further, as discussed in limitation [14.0], Menon teaches that *the cluster initialization process comprises assigning a processor identification number to each of the cluster node modules*. Specifically, Menon explains that “[f]or each of p MATLAB processes, POE assigns an identification number between 0 and $p-1$. The process numbered 0 is considered the interactive MATLAB.” Ex.1005, 6.

316. A POSITA would have found it obvious to *assign[] a processor identification number to each of the cluster node modules **after** establishing communication among two or more of the nodes* because POEref teaches that the user enters the “poe” command to establish communication among and with the

nodes. Ex. 1008, 30. After the poe command, then the nodes are identified by the node id, as shown below:

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

```
% poe
0:host1_name> master [options]
1:host2_name> workers [options]
2:host3_name> workers [options]
3:host4_name> workers [options]
4:host5_name> workers [options]
Partition loaded...
```

Ex. 1008, 31.

20.Claim 17

- a. ***[17.0] The computer cluster of claim 1, wherein one or more of the nodes are configured to communicate at least some of the user instructions.***

317. Limitation [17.0] is disclosed or rendered obvious for the same reasons presented above for limitations [1.7.2].

21.Claim 18

- a. ***[18.0] The computer cluster of claim 17, wherein one or more of the nodes are configured to communicate at least some of the user instructions to one or more single-node kernels.***

318. Menon in view of Trefethen render obvious this element. Specifically, Menon and Trefethen describe that the commands are communicated using the MEX routines from the interactive processor (ID=0), and once received by each

non-interactive node (processor ID=n), are then communicated to MATLAB (single-node kernel) for processing, thereby rendering obvious *wherein one or more of the nodes are configured to communicate at least some of the user instructions to one or more single-node kernels.*

319. In limitations [1.7.2] and [4.0] 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 so that the MATLAB processes can communicate tasks and data directly with each other over a network, rendering obvious *communicating at least some of the user instructions using the communications network interface.* 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.

320. 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.

22.Claim 19

- a. ***[19.0] The computer cluster of claim 1, wherein one or more of the nodes are configured to accept user instructions via one or more of the nodes.***

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

323. As discussed in limitation [18.0], Menon teaches an interactive processor that communicates the *user instructions* to the other processors, which are received at the communication layer and communicated to the respective MATLAB of the processors.

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

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

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

325. FIG. 1 of Menon below illustrates the interactive MATLAB process at the interactive processor transmitting commands that are based on *user instructions* to the non-interactive MATLAB processes on the other processors of the MultiMATLAB architecture, which then execute the commands. Limitation [18.0] shows example illustrations from Trefethen showing that commands sent to the other or non-interactive MATLAB processes corresponding to IDs specified in the *user instructions* are processed by MATLAB at the respective MATLAB processes identified by the IDs (and as such, are accepted by the processors of the non-interactive MATLAB processes).

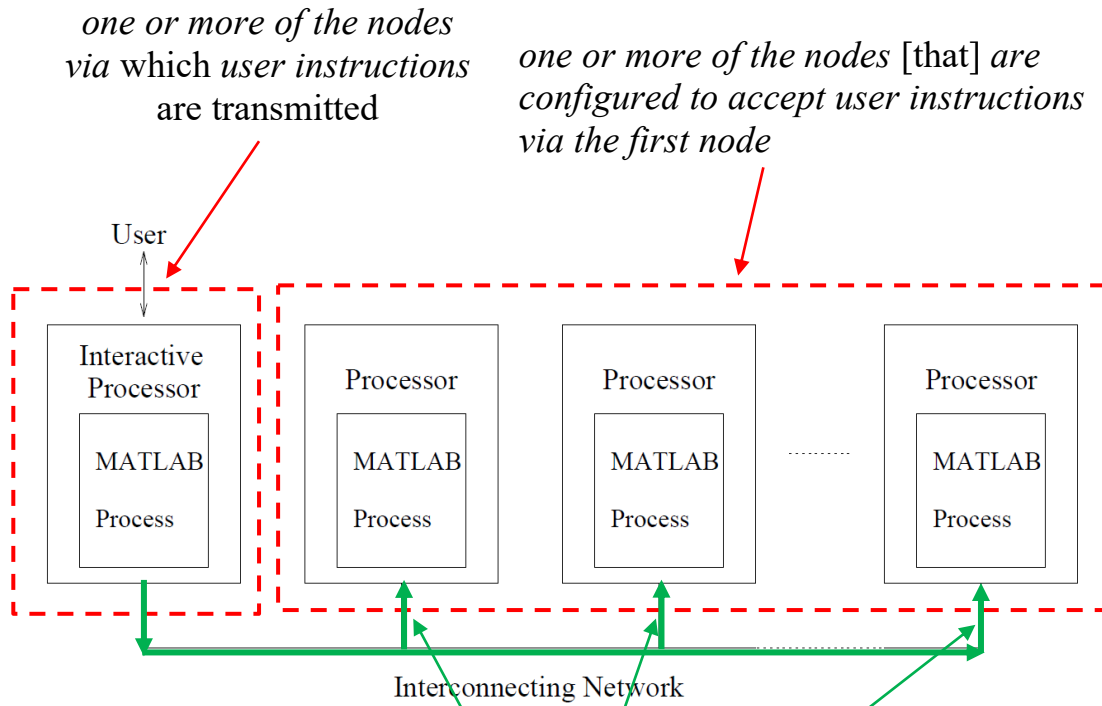


Figure 1: MultiMATLAB Architecture

User instructions are sent from the interactive MATLAB process (one or more of the nodes) to the other processors (one or more of the nodes) which accept the user instructions

Ex.1005, FIG. 1 (annotated)

VI. DECLARATION

326. I declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true, and that these statements were made with knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under section 1001 of Title 18 of the United States Code.



Dated: April 29, 2025

Chandrajit L. Bajaj, Ph.D.