

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

INTEL CORPORATION,
ADVANCED MICRO DEVICES, INC.
Petitioners,

v.

ADVANCED CLUSTER SYSTEMS, INC.,
Patent Owner.

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

**PETITION FOR *INTER PARTES* REVIEW
UNDER 35 U.S.C. § 312 AND 37 C.F.R. § 42.104**

TABLE OF CONTENTS

Petitioner’s Exhibit List	4
I. Introduction.....	8
II. Grounds for standing	9
III. Note.....	9
IV. The Petition’s Reliance on Expert Testimony.....	9
A. Technological Background	9
B. Focused Expert Testimony.....	10
C. Detailed Expert Testimony.....	10
V. The ’582 patent.....	10
A. Technological Background	10
B. Overview	11
VI. Priority date of the ’582 patent.....	12
VII. Prosecution History of the ’582 patent.....	12
VIII. Level of ordinary skill in the art.....	12
IX. Claim construction.....	13
A. “peer-to-peer architecture”.....	13
X. Relief requested and the reasons for the requested relief.....	13
XI. Identification of how the claims are unpatentable.....	14
A. Challenged Claims	14
B. Statutory grounds for challenges.....	14

C.	Ground 1: Claims 1-3 and 5-19 are obvious over Menon in view of Trefethen, RS6000, and MPIref.....	16
1.	Menon	17
2.	Trefethen	20
3.	RS6000.....	21
4.	MPIref	22
5.	Combining Trefethen with Menon	23
6.	Combining RS6000 with Menon	25
7.	Combining MPIref with Menon	27
8.	Challenges.....	28
D.	Ground 2: Claims 4 and 20 are obvious over Menon in view of Trefethen, RS6000, and MPIref further in view of POEref.....	110
1.	POEref.....	110
2.	Combining POEref with Menon.....	111
XII.	Sotera Stipulation.....	115
XIII.	Conclusion	115
XIV.	Mandatory notices	116
A.	Real party-in-interest.....	116
B.	Related matters	116
C.	Lead and back-up counsel and service information.....	117
	Certificate of Word Count	119
	Certificate of Service	120

PETITIONER’S EXHIBIT LIST

Ex.1001	U.S. Patent No. 11,811,582
Ex.1002	Prosecution History of U.S. Patent No. 11,811,582
Ex.1003	Declaration of Dr. Chandrajit L. Bajaj under 37 C.F.R. § 1.68
Ex.1004	<i>Curriculum Vitae</i> of Dr. Chandrajit L. Bajaj
Ex.1005	“MultiMATLAB: Integrating MATLAB with High-Performance Parallel Computing,” Menon et al., SC '97: Proceedings of the 1997 ACM/IEEE Conference on Supercomputing 1997
Ex.1006	“MultiMATLAB: MATLAB on Multiple Processors,” Trefethen et al., 1996 Computer Science Technical Report, Cornell University 1996
Ex.1007	“RS/6000 SP: Planning Vol. 1, Hardware and Physical Environment,” IBM 2001
Ex.1008	“Operation and Use, Volume 1, Using the Parallel Operating Environment,” IBM 2001
Ex.1009	“Single program multiple data” in Dictionary of Algorithms and Data Structures, P. E. Black, (accessible at https://www.nist.gov/dads/HTML/singleprogrm.html) December 2004
Ex.1010	The RS/6000 SP Inside Out” to Barrios et al., IBM 1999
Ex.1011	“High Performance Cluster Computing: Programming and Applications,” Rajkumar Buyya, Vol. 2, 1999
Ex.1012	“Analysis of 100Mb/s Ethernet for the Whitney Commodity Computing Testbed,” Fineberg et al., NAS Technical Report NAS-97-025 1997
Ex.1013	“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.1014	“Parallel MATLAB: Doing It Right,” R. Choy et al., Proceedings of

	the IEEE, Vol. 93, No. 2, 2005
Ex.1015	“Mathematica Parallel Computing Toolkit - Unleash the Power of Parallel Computing,” R. Maeder, 2005
Ex.1016	“Mastering MATLAB® 5 – A Comprehensive Tutorial and Reference,” D. Hanselman et al., 1998
Ex.1017	“MPI: A Message-Passing Interface Standard,” Message Passing Interface Forum, 1994
Ex.1018	“Modern Operating Systems,” A.S. Tanenbaum, 2001
Ex.1019	September 26, 2024 Complaint against Intel in Western District of Texas
Ex.1020	Order Granting Joint Motion for Entry of Amended Scheduling Order for <i>Advanced Cluster Systems, Inc. v. Intel Corporation</i> , 7:24-CV-00245 (W.D.TX.)
Ex.1021	Federal Court Statistics
Ex.1022	Advanced Cluster Systems, Inc.’s Preliminary Infringement Contentions, January 22, 2025, <i>Advanced Cluster Systems, Inc. v. Intel Corporation</i> , 7:24-cv-00245 (WDTX)
Ex.1023	Cornell Websites
Ex.1024	“pyMPI–An introduction to parallel Python using MPI,” P. Miller 2002
Ex.1025	Amended Joint Claim Construction Chart for <i>Advanced Cluster Systems, Inc. v. NVIDIA Corporation</i> , 1:19-cv-02032 (DDE)
Ex.1026	SC97: High Performance Networking and Computing Conference Website
Ex.1027	Declaration of IEEE
Ex.1028	ACM Digital Library - Citations to Menon
Ex.1029	Affidavit of Archive.org

Ex.1030	HPC Wire Article about the SC97 High Performance Computing Conference
Ex.1031	Flyer about the SC97 High Performance Computing Conference
Ex.1032	IEEE Xplore Website
Ex.1033	Article about Cornell Theory Center
Ex.1034	ACM Digital Library
Ex.1035	ACM Digital Library - Citations to Trefethen
Ex.1036	A Beginner's Guide to the IBM SP
Ex.1037	ACM Digital Library - Citations to MPIref
Ex.1038	IBM Website for RS6000
Ex.1039	IBM Website for POE
Ex.1040	ACS Preliminary Response, IPR2021-00019
Ex.1041	Decision Denying Institution, IPR2021-00019
Ex.1042	Sotera Stipulation, <i>Advanced Cluster Systems, Inc. v. Intel Corporation</i> , 7:24-cv-00245 (WDTX)
Ex.1043	September 26, 2024 Complaint against AMD in Western District of Texas
Ex.1044	Scheduling Order for <i>Advanced Cluster Systems, Inc. v. Advanced Micro Devices, Inc.</i> , 7:24-CV-00244 (W.D.TX.)
Ex.1045	Advanced Cluster Systems, Inc.'s Preliminary Infringement Contentions, February 24, 2025, <i>Advanced Cluster Systems, Inc. v. Advanced Micro Devices, Inc.</i> , 7:24-cv-00244 (WDTX)
Ex.1046	Sotera Stipulation, <i>Advanced Cluster Systems, Inc. v. Advanced Micro Devices, Inc.</i> , 7:24-cv-00244 (WDTX)
Ex.1047	Declaration of International Business Machines Corp.

Ex.1048	Affidavit of Tanya Zeif, Internet Archive
Ex.1049	MPIref at Argonne National Laboratory

I. INTRODUCTION

U.S. Patent No. 11,811,582 (the “’582 patent,” Ex.1001) contains long-winded claims that obfuscate the well-known concepts in those claims. The ’582 patent’s purported invention takes a known mathematical tool (Mathematica) that runs on a single computer and enables it to run in parallel on multiple computers. A Cornell research team, however, made this same solution years earlier by enabling a known mathematical tool (MATLAB), which ran on a single computer, to run in parallel on multiple computers.

The Cornell team’s implementation of MATLAB in parallel computers was named MultiMATLAB, and the principal researchers published a pair of papers titled “MultiMATLAB: Integrating MATLAB with High-Performance Parallel Computing” (“Menon”) and “MultiMATLAB: MATLAB on Multiple Processors” (“Trefethen”) more than five years before the earliest priority date of the ’582 patent.

Anything purportedly novel about the ’582 patent was published in Menon and Trefethen. The ’582 patent describes cluster computing in a peer-to-peer architecture—the Cornell team used cluster computing in a peer-to-peer architecture. The ’582 patent recites a module of code for communication among the nodes in the computer cluster—the Cornell team used a module of code for communication among the nodes. The ’582 patent recites a sequence starting at

node 1, then node 2, then node 3—the Cornell team provided example software code that starts at node 1, then node 2, then node 3. Menon and Trefethen teach all purportedly novel features of the ’582 patent.

Accordingly, pursuant to 35 U.S.C. §§ 311, 314(a), Intel Corporation and Advanced Micro Devices, Inc. (“Petitioners”) respectfully request that the Board find claims 1-20 of the ’582 patent unpatentable.

II. GROUNDS FOR STANDING

Petitioners certify the ’582 patent is eligible for IPR and that Petitioners are not barred or estopped from requesting an IPR. 37 C.F.R. § 42.104(a).

III. NOTE

Petitioners cite exhibits’ original page numbers, unless noted otherwise.

Emphasis in quoted material has been added. Claim terms are presented in *italics*.

IV. THE PETITION’S RELIANCE ON EXPERT TESTIMONY

A. Technological Background

To the extent the Board needs discussion of the relevant background technology, Dr. Bajaj’s declaration describes the background technology predating the ’621 patent, including (i) cluster computing, (ii) parallelizing mathematical processing, (iii) peer-to-peer architecture, and (iv) sequential processing. *See* Ex.1003, ¶¶28-52.

B. Focused Expert Testimony

Dr. Bajaj's declaration provides focused expert testimony on the following topics: (i) the level of skill in the relevant art (Ex.1003, ¶¶24-27); (ii) claim construction (¶¶58-60); (iii) reasons to combine for the different prior art combinations (¶¶77-109); and (iv) key teachings of the prior art, including:

- Kernel (¶¶61-63);
- Cluster node module (¶¶64-70);
- User commands received by the first cluster node module without first passing through the first kernel (¶¶71-74); and
- Peer-to-peer mathematical computation (¶¶75-76).

C. Detailed Expert Testimony

To the extent the Board is interested in more detailed analysis, Dr. Bajaj's declaration also includes a detailed claim-by-claim analysis of the prior art. Ex.1003, ¶¶110-402.

V. THE '582 PATENT

A. Technological Background

Dr. Bajaj's declaration describes the background technology predating the '582 patent, including (i) cluster computing, (ii) parallelizing mathematical processing, (iii) peer-to-peer architecture, and (iv) sequential processing. *See* Ex.1003, ¶¶28-52.

B. Overview

The '582 patent describes computer clusters as “two or more computers, microprocessors, and/or processor cores (‘nodes’) that intercommunicate so that the nodes can accomplish a task as though they were a single computer.” Ex.1001, 1:22-27.

The purported novelty of the '582 patent is to take a computer program, such as “Mathematica software,” as well as “Maple®” or “MATLAB®,” and then “adding cluster computing functionality” to it. Ex.1001, 1:16-17, 4:21-26. Figure 2 shows “kernel modules 206 a-e” (e.g. Mathematica) each “in communication with a single cluster node module 204 a-e.” Ex.1001, 5:21-40.

The cluster node modules use Message-Passing Interfaces (“MPIs”) to “distribute tasks among the kernel modules 206 a-e.” Ex.1001, 6:19-23. Tasks include “performing calculations” from the “simple (for example, ‘1+1’), [to] entire subroutines and sequences of code (such as, for example, Mathematica code).” Ex.1001, 25:14-18, 24:34-37.

There is nothing novel, however, about cluster node modules using MPI to parallelize an existing mathematical software program on multiple computers. For example, the Cornell team’s MultiMATLAB provides an “interface module” using “the MPI communication standard” to enable MATLAB to perform “easily parallelizable numerical computations on multiple processors.” Ex.1005, Abstract;

Ex.1006, Abstract.

VI. PRIORITY DATE OF THE '582 PATENT

The '582 patent's earliest possible priority date is June 13, 2006. It is unnecessary to determine whether it is entitled to this date in this proceeding as all relied-upon prior art predates June 13, 2006. Petitioners do not waive any right to dispute the '582 patent's priority date in this or another forum.

VII. PROSECUTION HISTORY OF THE '582 PATENT

The applicant filed a terminal disclaimer over U.S. Patent No. 11,563,621. Ex.1002, 33, 202. The Examiner allowed the claims without rejection and provided no analysis with respect to any non-patent literature. *See generally*, Ex.1002.

VIII. LEVEL OF ORDINARY SKILL IN THE ART

A person of ordinary skill in the art ("POSITA") in June 2006 would have been someone knowledgeable of 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. Ex.1003, ¶24-27.

IX. CLAIM CONSTRUCTION

In IPR proceedings, claim terms receive their “ordinary and customary meaning[s]” from the perspective of those of ordinary skill in the art. 37 C.F.R. § 42.100(b). For purposes of this proceeding, Petitioners submit that no claim term requires express construction other than the following terms. *Nidec Motor Corp. v. Zhongshan Broad Ocean Motor Co.*, 868 F.3d 1013, 1017 (Fed. Cir. 2017).

A. “peer-to-peer architecture”

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

In a previous IPR proceeding for a parent patent (IPR2021-00019), Patent Owner construed “peer-to-peer architecture” as requiring “an architecture in which each node can communicate tasks and data with other nodes without the tasks and data being required to go through a central server or master node.” Ex.1040, 8. For purposes of this proceeding, Petitioners and its expert used this construction.

X. RELIEF REQUESTED AND THE REASONS FOR THE REQUESTED RELIEF

Petitioners ask that the Board institute an IPR trial and cancel the challenged claims in view of the analysis below.

XI. IDENTIFICATION OF HOW THE CLAIMS ARE UNPATENTABLE

A. Challenged Claims

Petitioners challenge claims 1-20.

B. Statutory grounds for challenges

Grounds	Claims	Basis (pre-AIA)
#1	1-3, 5-19	35 U.S.C. §103 over Menon in view of Trefethen, RS6000, and MPIref
#2	4, 20	35 U.S.C. §103 over Menon in view of Trefethen, RS6000, and MPIref further in view of POEref

“MultiMATLAB: Integrating MATLAB with High-Performance Parallel Computing” (“Menon”) is a paper authored by Vijay Menon and Anne Trefethen of the Cornell research team. It qualifies as a printed publication because it was published and distributed at the Proceedings of the 1997 ACM/IEEE Conference on Supercomputing 1997. Ex.1005; Ex.1026; Ex.1027. Menon was also published and distributed on Cornell’s website at least as early as 2001. Ex.1023; Ex.1029. Menon was reasonably located by POSITAs because it had been cited 17 times in papers prior to the priority date of the ’582 patent. Ex.1028, 4-9, Ex.1029, 1-3; *see Blue Calypso, LLC. v. Groupon, Inc.*, 815 F.3d 1331, 1350 (Fed. Cir. 2016) (public accessibility can be shown via an express citation); *see also* Ex.1003, ¶¶123-125.

“MultiMATLAB: MATLAB on Multiple Processors” (“Trefethen”) is a

second paper authored by Vijay Menon, Anne Trefethen, and other members of the Cornell research team. It qualifies as a printed publication because it was published and distributed on the webpage of one of its authors at least as early as 2001. Ex.1023; Ex.1029. Trefethen was also published on a Cornell University website no later than May 10, 1996. Ex.1023; Ex.1029. Trefethen was reasonably located by POSITAs because it had been cited 10 times in papers prior to the priority date of the '582 patent. Ex.1035, 2-3; *see Blue Calypso, LLC. v. Groupon, Inc.*, 815 F.3d 1331, 1350 (Fed. Cir. 2016) (public accessibility can be shown via an express citation); *see also* Ex.1003, ¶¶129-130.

“RS/6000 SP: Planning Vol. 1, Hardware and Physical Environment” (“RS6000”) is a product manual by IBM in support of the IBM RS/6000 SP computing solution. Ex.1007. RS6000 qualifies as a printed publication because it was catalogued and indexed in IBM’s online document library at least as early as 2002 on IBM’s website. Ex.1007; Ex.1029; Ex.1038; *see also* Ex.1003, ¶¶132-133.

“Operation and Use, Volume 1, Using the Parallel Operating Environment” (“POEref”) is a product manual published by IBM in support for the Parallel Operating Environment running on an RS/6000 system. POEref qualifies as a printed publication because it was catalogued and indexed in IBM’s online document library at least as early as 2001. Ex.1039; Ex.1029. POEref was distributed with the IBM SP2 systems and published at least as early as 2001.

Ex.1008, 157; Ex.1036, 14; Ex.1047; *see also* Ex.1003, ¶¶136-138.

“MPI: A Message-Passing Interface Standard” (“MPIref”) is the message passing standard written by the Message Passing Interface Forum. MPIref qualifies as a printed publication because it was distributed to POSITAs using FTP mail servers by the University of Tennessee and Oak Ridge National Library in 1994. Ex.1017, cover page; Ex.1029; Ex.1003, ¶142. It was published on MPI Forum’s website no later than July 3, 1998. Ex.1029, 1-3. MPIref was reasonably located by POSITAs because it had been cited 74 times in papers with online publications dates before the earliest priority date of the ’582 patent. Ex.1037, 2-16; *see also* Ex.1003, ¶¶142-144.

Further, each prior art reference is (i) more than 20 years old; (ii) is regular on its face with no signs of obvious alterations; and (iii) was found in a place where it could be expected to be found. Accordingly, each document is *prima facie* authenticated as an ancient document. *See* Fed. R. Evid. 901(8).

Petitioners cite additional documents to evidence a POSITA’s knowledge and understanding of the art. *See, e.g., Yeda Research v. Mylan Pharm. Inc.*, 906 F.3d 1031, 1041-42 (Fed. Cir. 2018).

C. Ground 1: Claims 1-3 and 5-19 are obvious over Menon in view of

Trefethen, RS6000, and MPIref

1. Menon

Menon discloses “MultiMATLAB,” a platform that enables MATLAB to operate in parallel on multiple processor nodes. Ex.1005, Abstract, Title.

a) Menon describes cluster computing

Menon discloses MultiMATLAB as “a general extension” of MATLAB Ex.1005, Abstract. MultiMATLAB comprises several processors, each configured to run a MATLAB process that communicates “with other processes through a communication layer.” Ex.1005, 3, FIG. 1. The communication layer uses “MPI” to communicate “over the parallel platform’s interconnection network.” Ex.1005, 3, 5.

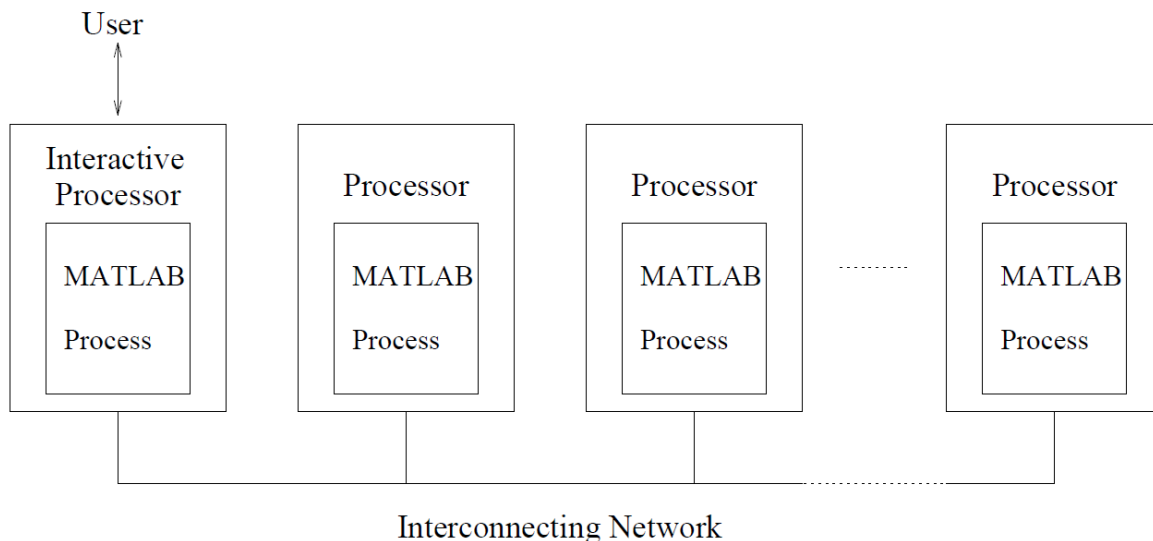


Figure 1: MultiMATLAB Architecture

Ex.1005, FIG. 1.

A user “interacts directly with ... the interactive process,” and the “other processes wait for commands from the interactive MATLAB process.” Ex.1005, 3, 6.

b) Menon describes parallelizing mathematical calculations

Menon describes using MultiMATLAB on a “parallel platform,” such as the “the IBM SP2” to calculate mathematical expressions in parallel. Ex.1005, 2, 5, 11.

c) Menon describes peer-to-peer architecture

Menon explains that MultiMATLAB works with different parallel programming paradigms. Ex.1005, 8. Notably, MultiMATLAB enables peer-to-peer communication through “SPMD/Message passing,” which provides “point to point” communication. Ex.1005, 9-10.

To provide “point to point” communication, Menon describes “message passing routines” that achieve “finer grain communication ... between any processes.” Ex.1005, 9, 8; Table 1¹.

¹ 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. Ex.1003, ¶121.

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

Table 1: Selected commands in the MultiMATLAB system

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

Ex.1005, Table 1 (annotated); Ex.1003, ¶70

Menon discloses that MultiMATLAB uses a set of components to provide direct communication with other modules: “MEX Routines,” “interface module,” “indirection table” and “communication layer.” Ex.1005, 3, 5.

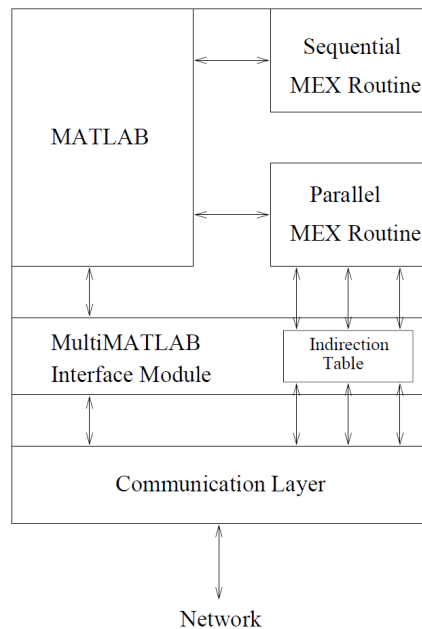


Figure 2: MATLAB Process Address Space

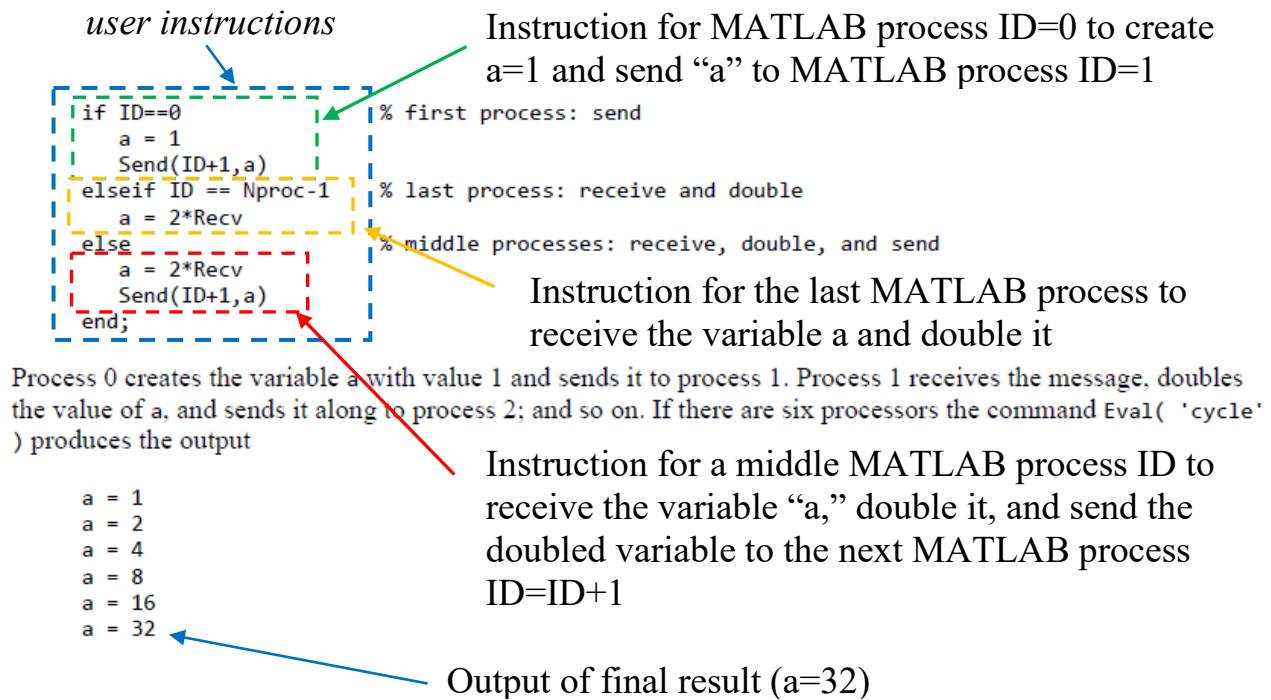
Ex.1005, FIG. 2.

2. Trefethen

Trefethen describes how MultiMATLAB runs MATLAB on multiple IBM SP2 nodes. Ex.1006, 3-9.

a) Trefethen describes sequential processing from one node to the next

Trefethen describes “point-to-point communication is accomplished by send and receive commands.” Ex.1006, 5, 6. Trefethen demonstrates that the Send and Recv commands allow direct communication among nodes. Ex.1006, 6:



Ex.1006, 6 (annotated); Ex.1003, ¶127.

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; Ex.1003, ¶127.

3. RS6000

"RS/6000 SP: Planning Vol. 1, Hardware and Physical Environment"

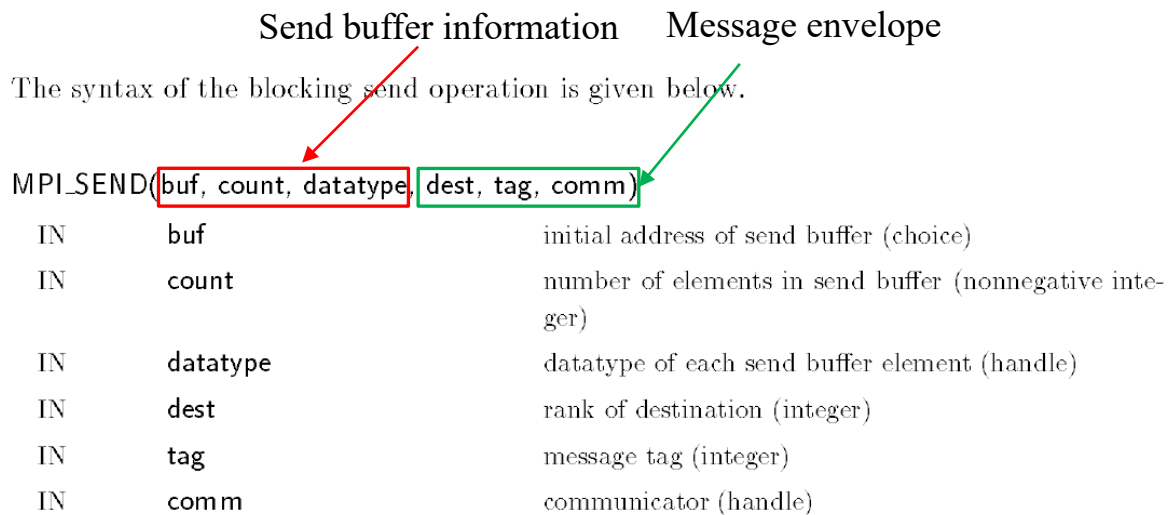
("RS6000," Ex.1007) is part of an IBM installation guide for the IBM SP2 system

used in Menon. Ex.1010, 6; Ex.1007, 1-2; Ex.1003, ¶131.

4. MPIref

“MPI: A Message-Passing Interface Standard” (“MPIref,” Ex.1017) is a published report from the Message Passing Interface Forum “contain[ing] all the technical features.” Ex.1017, ii. MPI allows the MATLAB processes of MultiMATLAB to communicate with each other. Ex.1005, 3.

MPIref discloses that MPI includes “point-to-point communication,” and that the “basic point-to-point communication operations are **send** and **receive**.” Ex.1017, iii, 15. The **send** operation MPI_SEND “specifies a **send buffer**.” Ex.1017, 15-16. MPIref also discloses a “destination” field to specify the “message destination.” Ex.1017, 18. (emphases original).



Ex.1017, 16 (annotated); Ex.1003, ¶140.

MPIref discloses that processes receive a message “with the **receive**

operation MPI_RECV,” where “the message data is stored into the **receive buffer**.” Ex.1017, 16 (emphases original). MPIref also discloses a “source” field. Ex.1017, 19.

	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). Ex.1003, ¶141.

5. Combining Trefethen with Menon

a) *Analogous art*

Menon and Trefethen are analogous prior art because they pertain to the same field of endeavor as the '582 patent, namely, the “the field of cluster computing” and are reasonably pertinent to the problem of the '582 patent, namely, executing a single program on multiple nodes. Ex.1005, 2; Ex.1006, Abstract; Ex.1001, 1:14-15; Ex.1003, ¶¶146-150.

b) Motivation to combine

Menon suggests the combination by expressly citing to Trefethen and explaining that Menon builds on the teachings of Trefethen. Ex.1005, 1-2.

Additionally, Trefethen describes “how the [MultiMATLAB] system is used” using code examples that perform parallel mathematical calculations using MATLAB, Ex.1006, 2, while Menon describes “tak[ing] advantage of high performance distributed memory processors” to enable MATLAB to perform mathematical calculations in parallel. Ex.1005, Abstract, 16. Accordingly, Menon and Trefethen together represent a combination of prior art elements according to known methods (executing MATLAB code on a multiprocessor system) to yield the predictable result discussed in both papers: the MultiMATLAB system executing MATLAB in parallel to perform mathematical calculations. Ex.1005, Abstract; Ex.1006, 2-7.

A POSITA seeking to understand MultiMATLAB’s capabilities would have consulted the Cornell website dedicated to MultiMATLAB, which contained links to both papers. Ex.1023. Co-author Vijay Menon likewise included links to both papers within his Cornell web page. Ex.1023. Further, the authors “Vijay Menon” and “Anne E. Trefethen” are common across both papers. *See* Ex.1005, 1 and Ex.1006, 1. Accordingly, any POSITA seeking to understand MultiMATLAB would have encountered and considered both Menon and Trefethen.

c) Predictable and reasonable expectation of success

The combination of Menon and Trefethen to parallelize MATLAB would have been predictable because researchers had a strong desire to parallelize MATLAB. Ex.1006, 11 (“Many people must have thought about parallelizing MATLAB”). The goal of parallelizing MATLAB was “inevitable.” Ex.1006, 12. The combination of Menon and Trefethen achieved this goal: the MultiMATLAB system executes MATLAB in parallel to perform mathematical calculations. Ex.1005, 2 (“In this paper, we ... achieve ... MATLAB code with high-performance parallel routines”).

Accordingly, a POSITA would have been motivated and found it predictable to combine Trefethen with Menon. *See also*, Ex.1003, ¶¶151-154.

6. Combining RS6000 with Menon

a) Analogous art

RS6000 is analogous prior art because it pertains to the same field of endeavor as the '582 patent, namely, the “the field of cluster computing” and is reasonably pertinent to the problem of the '582 patent, namely, executing a single program on multiple nodes. Ex.1007, 1; Ex.1001, 1:14-15; Ex.1003, ¶¶155-158.

b) Motivation to combine

A POSITA would have combined the teachings of RS6000 with Menon because Menon suggests the combination by teaching that MultiMATLAB uses the

“IBM SP2, a modern high performance distributed memory multiprocessor.”

Ex.1005, 5. RS6000 is provides “an overview of the IBM RS/6000 SP.”²

Ex.1007, 1.

A POSITA seeking to understand MultiMATLAB would be motivated to seek the documentation corresponding to the systems for which “MultiMATLAB was design[ed] specifically.” Ex.1005, 6. Additionally, the IBM SP2 was considered “a state-of-the-art parallel computing system” designed to execute “parallel applications.” Ex.1007, 1.

Menon and RS6000 together represent a combination of prior art elements (Menon describes the MultiMATLAB architecture on an “IBM SP2” and RS6000 provides details of that system) according to known methods (as taught by Menon) to yield the predictable result of MultiMATLAB implemented on an IBM SP2 having the physical attributes disclosed by RS6000. Ex.1005, Abstract, 6; Ex.1007, 1, 11.

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 MultiMATLAB on an IBM SP2

² “IBM SP2” is the same as “IBM RS/6000 SP.” Ex.1010, 6 (“the SP2 was renamed to simply the SP”).

and that MultiMATLAB “was design[ed] specifically for the IBM SP2.” Ex.1005, 6, 14 (“These results demonstrate that the MultiMATLAB system can be an effective platform for parallel computing”). *See*, Ex.1003, ¶¶159-162.

7. Combining MPIref with Menon

a) Analogous art

MPIref is analogous prior art because it pertains to the same field of endeavor as both the '582 patent and Menon, namely, the “the field of cluster computing” and is reasonably pertinent to the problem of the '582 patent, namely, executing a single program on multiple nodes. Ex.1001, 1:14-15; Ex.1003, ¶¶163-165.

b) Motivation to combine

Menon suggests the combination by expressly citing to “the MPI communication standard.” Ex.1005, 5. MPIref *is* the MPI communication standard referenced by Menon. Ex.1017, 1 (“MPI: A Message-Passing Interface Standard”).

Menon discloses using MPI as the “underlying communication substrate.” Ex.1005, 3, 5. Further, Menon indicates that a POSITA would look to use the MPI standard because of MPI’s “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.

The combination MPIref with Menon would have been obvious because Menon and MPIref together represent a combination of known prior art elements (Menon describing an implementation of MultiMATLAB using MPI as its communication layer and MPIref describing “all the technical features” of MPI) to yield the predictable result of an implementation of MultiMATLAB utilizing the features of MPI disclosed by MPIref. Ex.1005, 3-10; Ex.1017, ii.

The results 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 MultiMATLAB, and MPIref provided a “practical, portable, efficient and flexible standard for message passing.” Ex.1005, 3; Ex.1017, 2. Ex.1003, ¶¶166-170.

8. Challenges

Claim 1

[1.0] *A computer cluster comprising:*

Menon discloses a “MultiMATLAB architecture” providing “high-performance parallel routines.” Ex.1005, 2. Each processor of MultiMATLAB’s cluster “runs a MATLAB process” and “communicate[s] with other processes through a communication layer.” Ex.1005, 3.

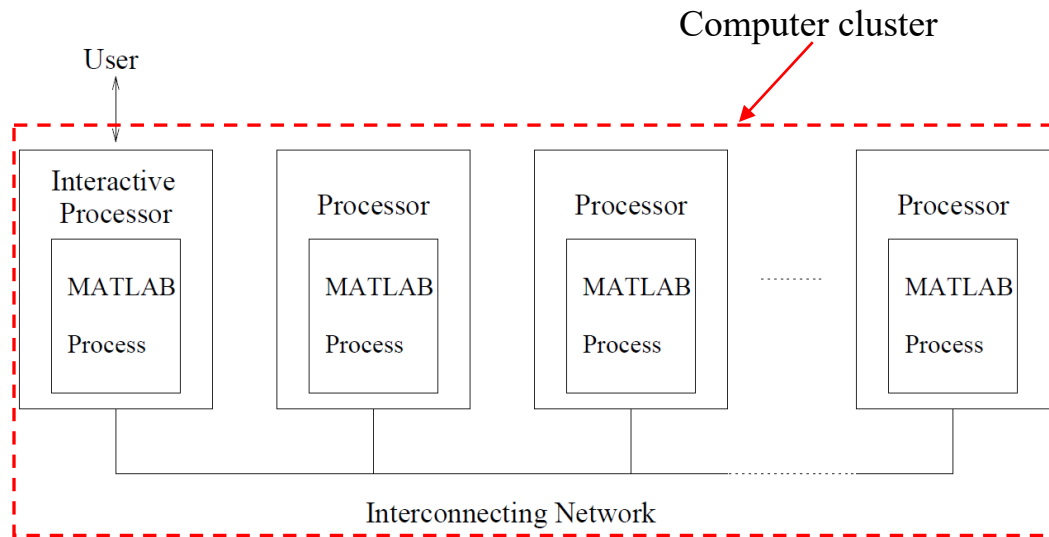


Figure 1: MultiMATLAB Architecture

Ex.1005, FIG. 1 (annotated); Ex.1003, ¶172

Menon discloses that MultiMATLAB was designed “specifically for the IBM SP2.” Ex.1005, 5, 6. The IBM SP2 system “execute[s] parallel programs on ... a **networked cluster** of RS/6000 processors.” Ex.1008, 1.

Menon also describes using a plurality of processor nodes to perform mathematical calculations. Ex.1005, 13, FIGs. 7, 8 (using “8 processors” and “32 processors,” respectively).

Accordingly, Menon’s cluster with multiple MATLAB processes implemented on the IBM SP2 to execute parallel mathematical computations, in view of RS6000’s description that the IBM SP2 is a networked cluster computing system for executing parallel programs, renders obvious this limitation. Ex.1003, ¶¶171-177.

[1.1.1] *a first processor;*

Menon’s cluster comprises multiple processors, where each processor “runs a MATLAB process” and “communicate[s] with other processes through a communication layer.” Ex.1005, 3. As shown in FIG. 1, Menon’s cluster includes a plurality of processors, including a first non-interactive processor (*first processor*).

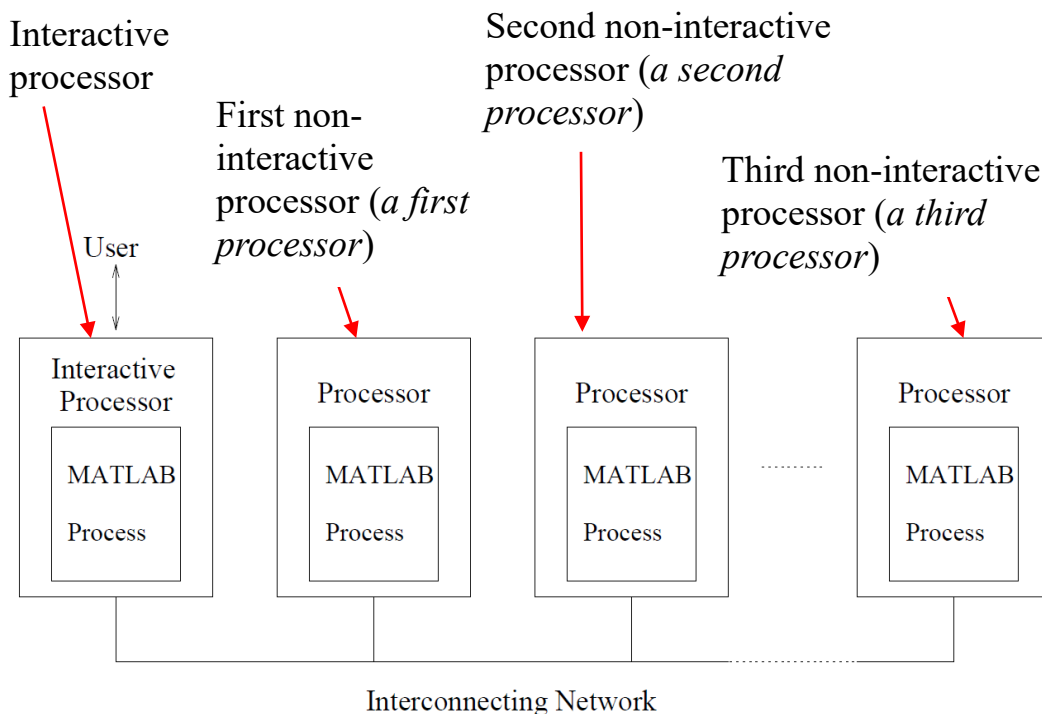


Figure 1: MultiMATLAB Architecture

Ex.1005, FIG. 1 (annotated); Ex.1003, ¶180

RS6000 describes that each IBM SP2 “processor node[]” has a “minimum of four [] processors.” Ex.1007, 10. A POSITA would recognize and understand that Menon’s first non-interactive processor is one of the four processors on the IBM

SP2, rendering obvious *a first processor*. Ex.1005, 3 (“each processor in a parallel platform individually runs a MATLAB process”).

Thus, Menon discloses a cluster operating on the IBM SP2 and RS6000 teaches that the IBM SP2 has a first processor, rendering obvious this limitation. Ex.1003, ¶¶178-181.

[1.1.2] *a second processor*;

As shown in [1.1.1], Menon’s MultiMATLAB architecture has a plurality of processors, including a second non-interactive processor (*second processor*).

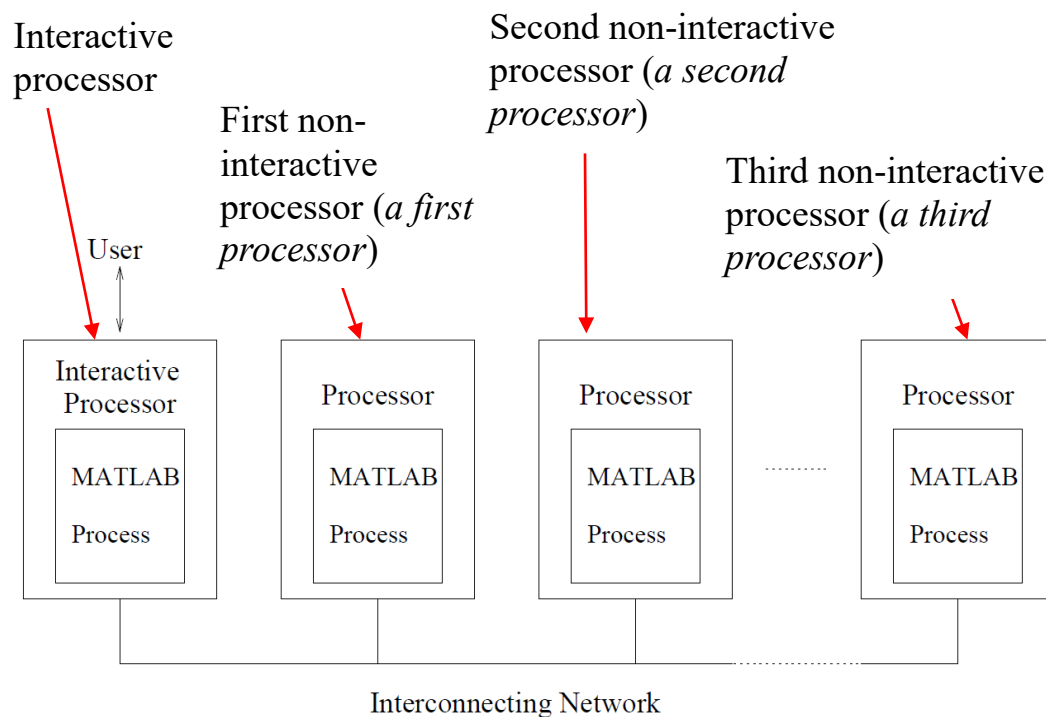


Figure 1: MultiMATLAB Architecture

Ex.1005, FIG. 1 (annotated); Ex.1003, ¶180

Accordingly, a POSITA would recognize and understand that Menon's second non-interactive processor is one of the four processors on the IBM SP2, rendering obvious *a second processor*. Ex.1005, 3 ("each processor in a parallel platform individually runs a MATLAB process"). Ex.1003, ¶¶182-183.

[1.1.3] *a third processor*;

As shown in [1.1.1], Menon's MultiMATLAB architecture has a plurality of processors, including a third non-interactive processor (*third processor*).

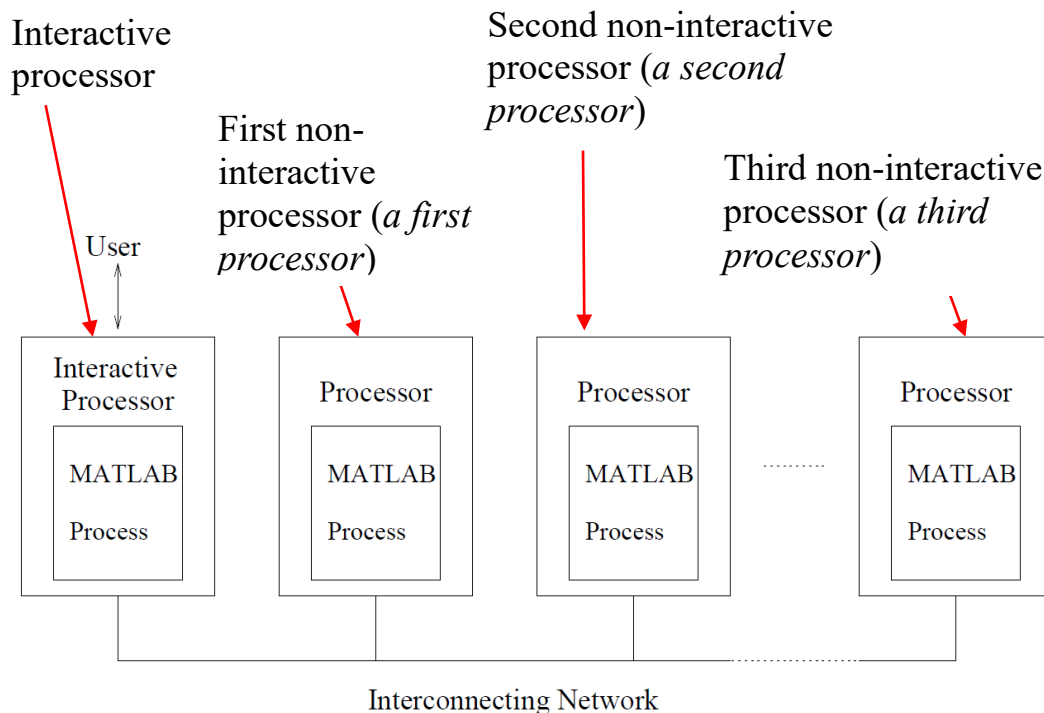


Figure 1: MultiMATLAB Architecture

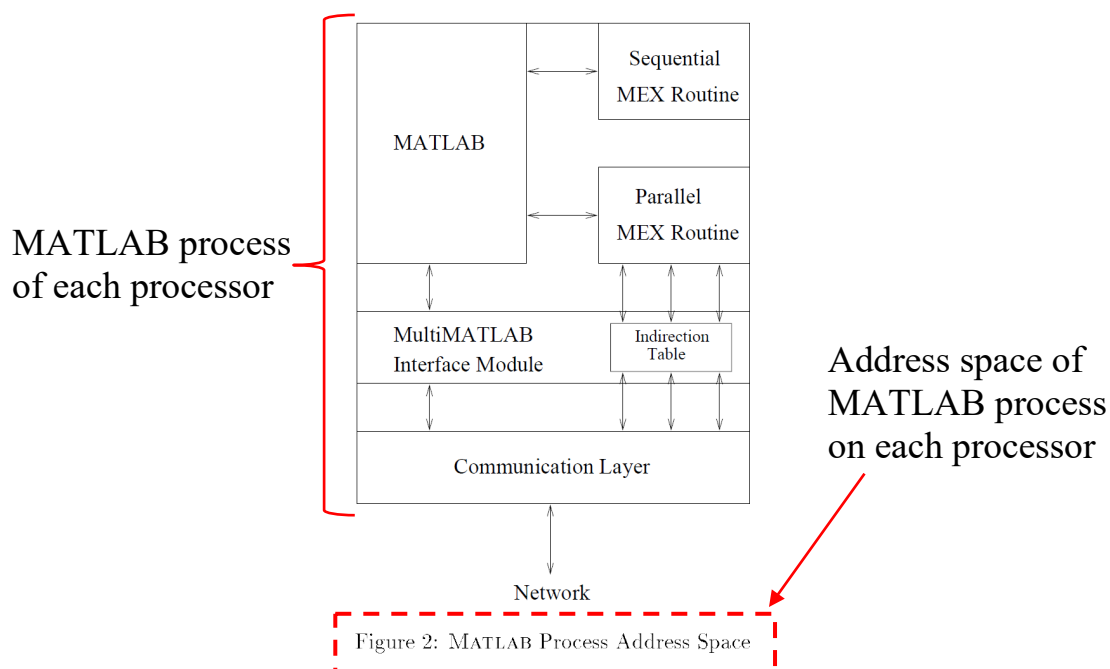
Ex.1005, FIG. 1 (annotated); Ex.1003, ¶180

A POSITA would recognize and understand that Menon's third non-interactive processor is one of the four processors on the IBM SP2, rendering

obvious *a third processor*. Ex.1005, 3 (“each processor in a parallel platform individually runs a MATLAB process”). Ex.1003, ¶¶184-185.

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

Menon describes that MATLAB process has MATLAB in an “address space.” Ex.1005, FIGS. 1, 2, 5.



Ex.1005, FIG. 2 (annotated); Ex. 1003, ¶188

A POSITA would understand Menon’s “address space” to be “memory locations ... contain[ing] the executable program.” Ex.1018, 67-68. POSITA would then recognize that MATLAB is in, and accessed by, for example, *the first processor* from, a *computer-readable medium*. The *computer-readable medium* of

the IBM SP2 processor nodes can be disk drives or memory cards, as taught by RS6000. Ex.1007, 6, 10.

Each MATLAB process “does the computation required for its local data.” Ex.1005, 11-13. Thus, MATLAB causes the hardware processor of the IBM SP2 node to perform computation (*in communication with at least one of the first processor, the second processor, or the third processor*).

Accordingly, each MATLAB process address space (*a computer-readable medium*) that the processor accesses (*in communication with at least one of the first processor, the second processor, or the third processor*) in view of RS6000 teaching that the IBM SP2 has disk drives and memory cards for storing program code accessible by the node’s hardware processors, renders obvious this limitation. Ex.1003, ¶¶186-190.

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

Menon explains that “[i]n the MultiMATLAB architecture, illustrated in Figure 1, each processor in a parallel platform individually runs a MATLAB process.” Ex.1005, 3. As discussed in [1.2], each MATLAB process is in an address space in memory (*computer-readable medium*). FIG. 1 of Menon below shows a MATLAB process running on each processor:

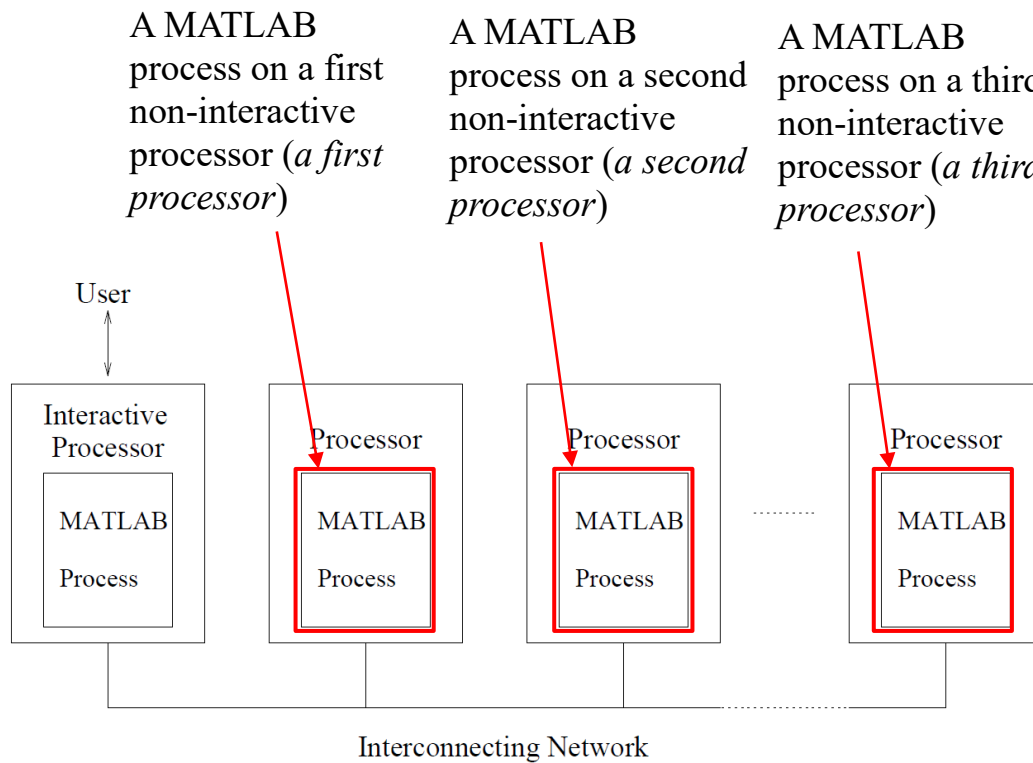
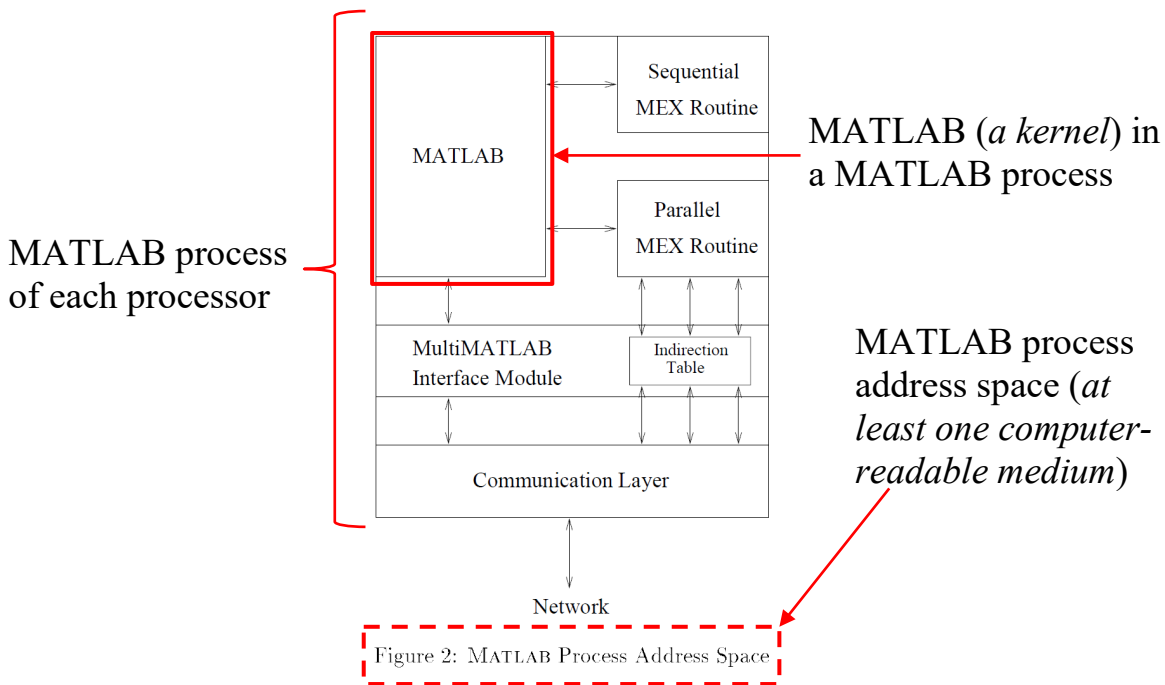


Figure 1: MultiMATLAB Architecture

Ex.1005, FIG. 1 (annotated); Ex.1003, ¶192

Menon explains that each MATLAB process of MultiMATLAB 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).



Ex.1005, FIG. 2 (annotated); Ex.1003, ¶193

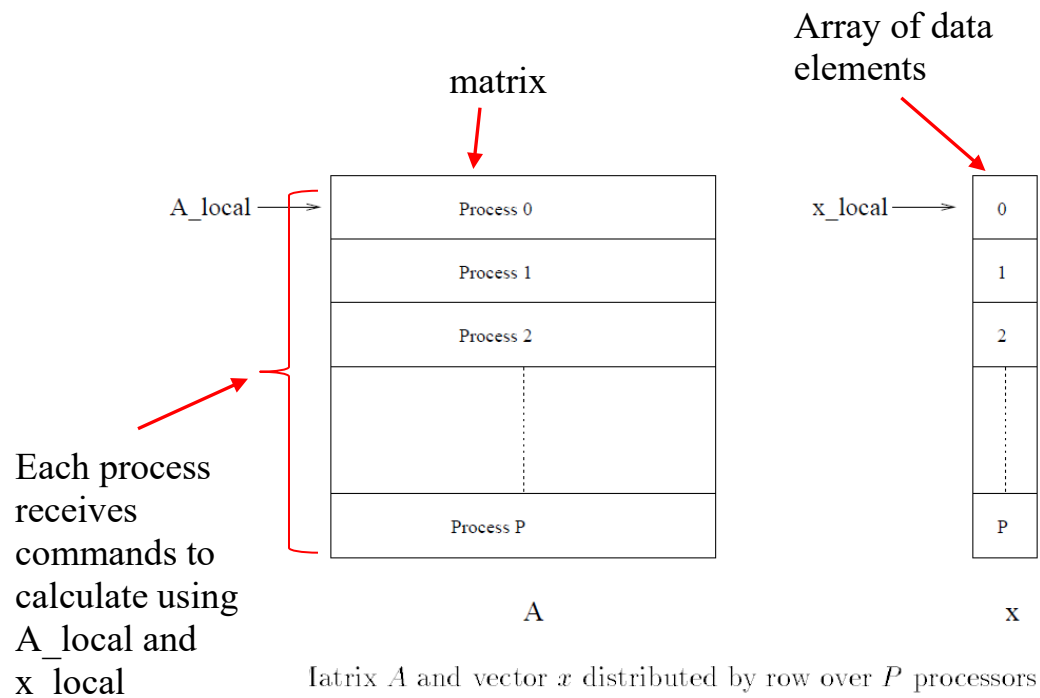
MATLAB is a “scientific computing environment” that provides users with “an extensive library of high quality numerical routines.” Ex.1005, 1. For example, MATLAB “provides a high-level matrix based language that permits users to express computations in an exceptionally concise manner via matrix or vector formulations.” Ex.1005, 1. The ’582 patent identifies MATLAB as an example of a *kernel*. Ex.1001, 1:16-17, 4:21-26.

Accordingly, Menon discloses the first non-interactive processor having MATLAB (*first kernel*) in an address space (*computer-readable medium*), rendering obvious this limitation. Ex.1003, ¶¶191-195.

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

As shown in [1.3.1], Menon describes MATLAB on the first non-interactive processor (*first kernel*).

Menon describes MATLAB as performing “high-level MATLAB operations” such as “matrix multiplications [and] solve for eigenvalues,” and provides example “MultiMATLAB implementations of the conjugate gradients algorithm” that “consists of a **matrix-vector multiplication** at every iteration.” Ex.1005, 10. The “parallel MATLAB implementation of conjugate gradients” includes the step of “matrix A and the different vectors [] each [being] distributed by row over all processes” and “communication is required for two different operations.” Ex.1005, 11. FIG. 6 of Menon shown below illustrates this step where “[m]atrix A and vector x [are] distributed by row over P processors” when the MultiMATLAB implementation includes “ p MATLAB processes.” Ex.1005, FIG. 6, 6 (emphasis original).



Ex.1005, FIG. 6 (annotated); Ex.1003, ¶201

Trefethen describes several examples of a MATLAB receiving commands that are translated into calls for execution. For example, Trefethen describes a user typing in commands or entering scripts (such as “Eval(‘cycle’)”) into the interactive processor “for executing commands on one or more processors.”

Ex.1006, 3, 6. The user commands of m-file cycle.m are translated into the mathematical computations and calls executed by each MATLAB process, including the MATLAB process (i.e., MATLAB process ID=1) on the first non-interactive processor. Ex.1006, 4. For example, MATLAB process ID=1 receives

from MATLAB process ID=0 the variable “a” having the value “1,” the command (“a=2*Recv”) is translated into multiplication that is executed, and then the command (“Send(ID+1,a)”) is translated into a call that sends the doubled variable “a=2” to MATLAB process ID+1, i.e., MATLAB process 2. Ex.1006, 6. In turn, MATLAB process 2 receives the variable a, translates the command into multiplication, and then translates the command into a call that sends the value of the variable “a” to the next MATLAB process, MATLAB process ID=3. Ex.1006, 4.

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

Accordingly, Menon teaching MATLAB in the MATLAB process of the first non-interactive processor translates user commands into matrix-related mathematical evaluations of matrices and an array of data elements, in view of

Trefethen teaching user commands and script files for performing matrix-related mathematical evaluations of matrices, renders obvious this limitation. Ex.1003, ¶¶196-204.

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

As discussed in [1.2], the MATLAB process resides in an address space located in memory. Menon further discloses that the MATLAB process address space located in memory (*computer-readable medium*) includes a set of components to provide interconnection with other processes: “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.

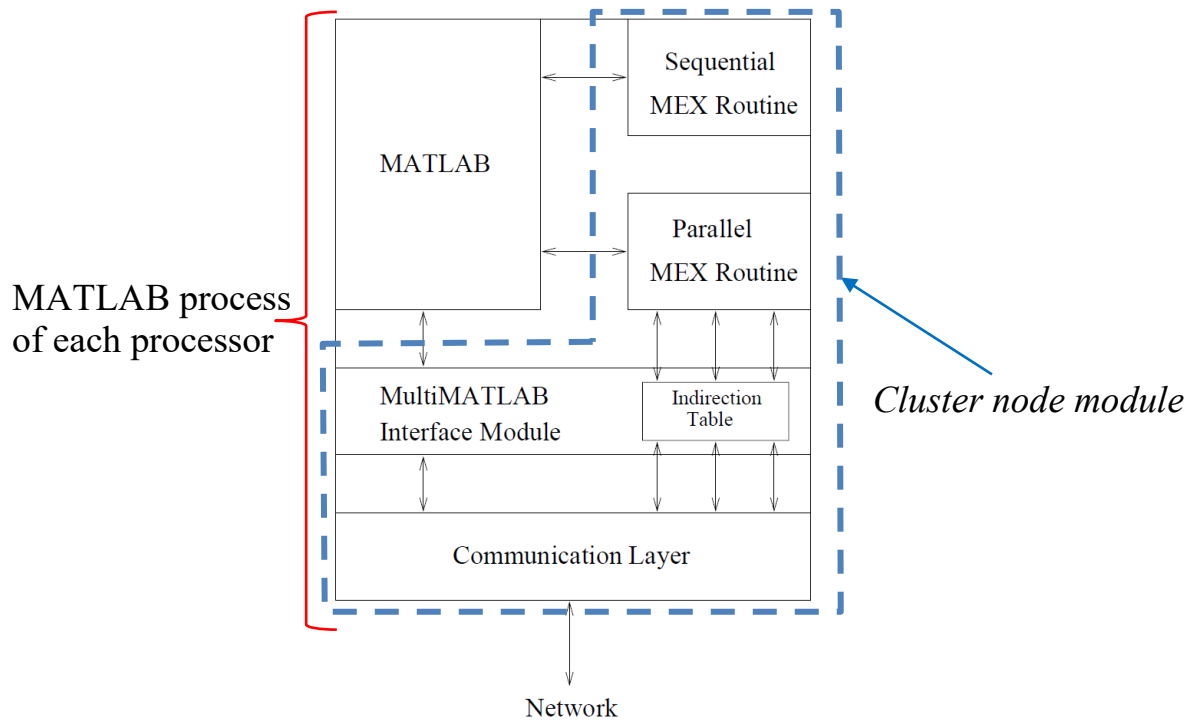
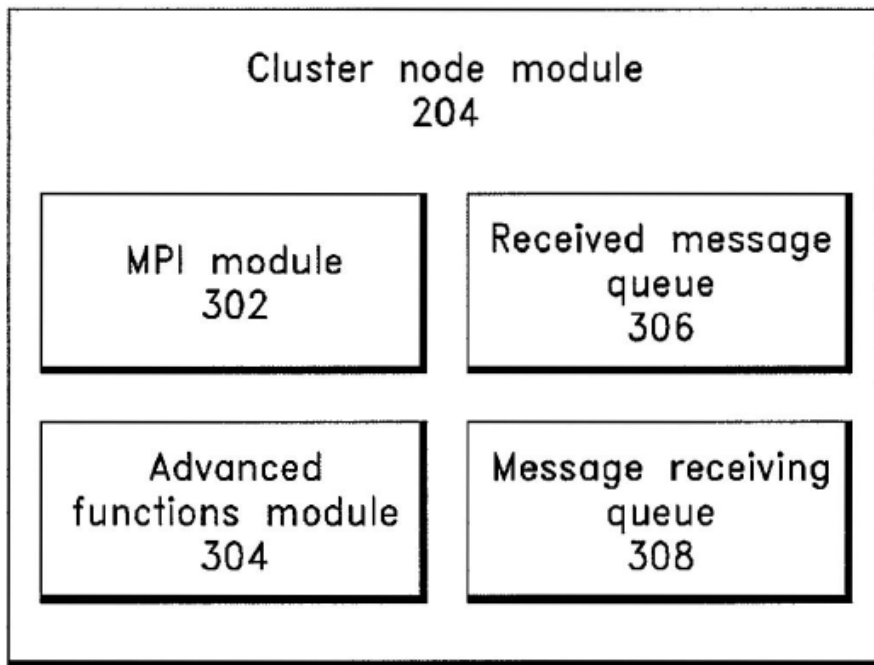


Figure 2: MATLAB Process Address Space

Ex.1005, FIG. 2 (annotated); Ex.1003, ¶209

Correspondingly, the '582 patent depicts its cluster node module as a set of components to provide communication among the nodes:



Ex.1001, FIG. 3

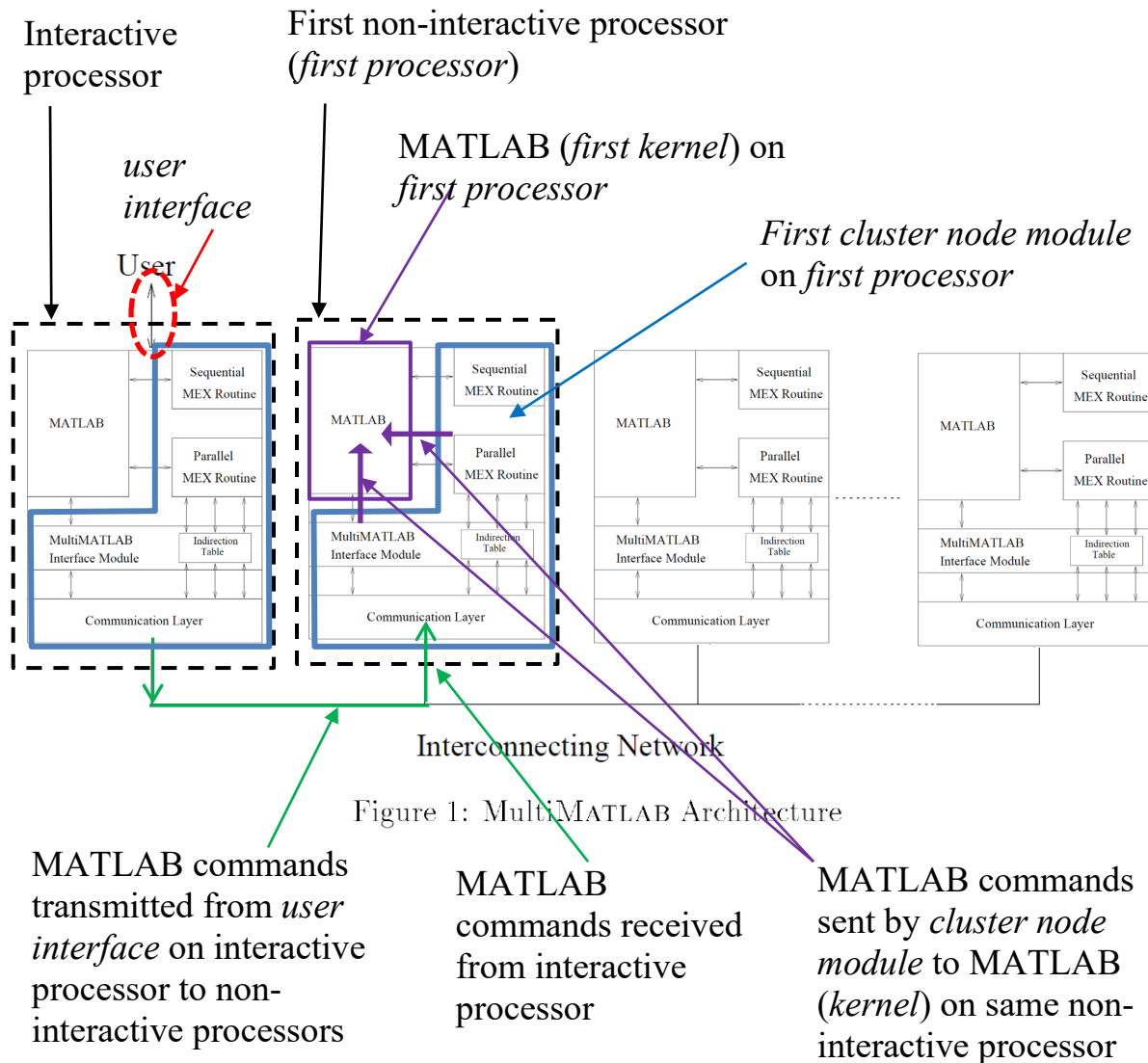
Accordingly, Menon describes each MATLAB process having a set of components to provide interconnection with other modules (MEX routines, MultiMATLAB interface module, indirection table, and communication layer) (*cluster node module*) residing in memory, rendering obvious this limitation.

Ex.1003, ¶¶205-210.

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

As shown in [1.4.1], Menon teaches a set of components to provide interconnection with other modules: “MEX Routines,” “MultiMATLAB interface module,” “indirection table” and “communication layer” (collectively, the *cluster node module*).

Menon describes that the “user interacts directly with one MATLAB process, called the interactive process, and operates within that process’s MATLAB environment” to have user instructions executed in parallel by the MultiMATLAB architecture. Ex.1005, 3. “On non-interactive processes, a separate top-level MEX routine ... runs in a loop in which it waits for MATLAB commands to arrive from the interactive process and executes them in its own MATLAB environment.” Ex.1005, 3-4.



Ex.1005, FIGs. 1 and 2 (combined and annotated); Ex.1003, ¶217

Trefethen teaches that the user enters commands that are sent directly to the other processes. For example, “the user now types Eval(‘sqrt(2)’),” which causes “the MATLAB command sqrt(2) [to be] executed on all six processors.” Ex.1006, 3. In another example, the user can select a subset of processes by providing the process IDs to perform the command, such as “Eval([4 5] , ‘cond(hilb(ID))’),”

which causes the MATLAB command ‘cond(hilb(ID))’ to be performed by processes ID=4 and ID=5.

Accordingly, the sequence taught by Menon and Trefethen is: (i) the user enters a command via the user interface on the interactive processor, (ii) that command is provided to the MEX routine of the cluster node module on the interactive processor, (iii) the cluster node module on the interactive processor communicates the command to the cluster node modules of the other processors, (iv) the cluster node module of the first processor receives the command via the interconnecting network, and (v) the cluster node module of the first processor communicates the command to the first kernel. Ex.1003, ¶¶211-219.

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

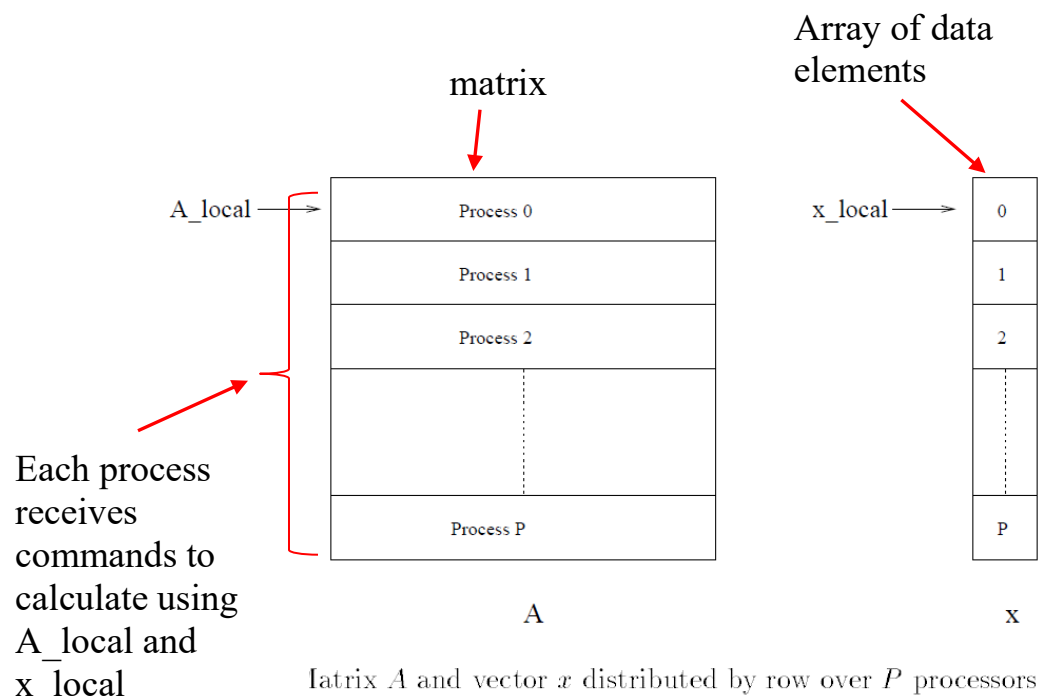
As shown in [1.3.1], MATLAB of a MATLAB process on a processor teaches *a kernel*.

Further, in limitation [1.2], the MATLAB process of each processor of Menon’s MultiMATLAB architecture is in a MATLAB process address space (*computer-readable medium*). Accordingly, Menon teaches MATLAB (*a second kernel*) in the MATLAB process on the second non-interactive processor (*second processor*), where the MATLAB process is contained in (*residing in*) a MATLAB process address space in memory (*the at least one computer-readable medium*). Ex.1003, ¶¶220-222.

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

First, as shown in [1.1.2], the second non-interactive processor teaches *the second processor*. Further, as shown in [1.5.1], MATLAB in the MATLAB process on the second non-interactive processor teaches *said second kernel*.

Second, as shown in [1.3.2], Trefethen and Menon teach that the user instructions are translated into calls that cause MATLAB on each of the non-interactive processors to execute matrix operations on an array of data elements.



Ex.1005, FIG. 6 (annotated); Ex.1003, ¶201

Trefethen also provides an example algorithm (the “cycle.m” m-file) where mathematical results are returned from one processor to the next:

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

← user command

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

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

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

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

Ex.1006, 6 (annotated); Ex.1003, ¶225.

Accordingly, Menon teaching MATLAB in the MATLAB process of the second non-interactive processor, in view of Trefethen teaching an m-file where that MATLAB in the MATLAB process on the second non-interactive processor receives the variable “a=2,” doubles it to “a=4,” outputs the result “a=4,” and then sends it to the next MATLAB process, in further view of Menon and Trefethen teaching that each MATLAB process performs matrix-related mathematical evaluations on the array of data elements, renders obvious this limitation. Ex.1003, ¶¶223-228.

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

As shown in [1.4.1] the MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*cluster node module*) of each MATLAB process of MultiMATLAB are in the MATLAB process address space in memory. Accordingly, the MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*a second cluster node module*) of the MATLAB process on the second non-interactive processor are in (*residing in*) the MATLAB process address space in memory (*the at least one computer-readable medium*) of that MATLAB process, rendering obvious this limitation. Ex.1003, ¶¶229-230.

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

As shown in [1.6.1], the MEX routines, MultiMATLAB interface module, indirection table, and communication layer of the second non-interactive processor (*second processor*) teach *said second cluster node module*. Further, as shown in [1.5.1], MATLAB in the MATLAB process on the second non-interactive processor (*second processor*) teaches *the second kernel*.

As shown in [1.4.2], the interactive processor sends the user commands to the cluster node module of each non-interactive processor, which provides the commands to the MATLAB (*kernel*) for execution by MATLAB. Accordingly, the

MEX routines (part of *said second cluster node module*) in the MATLAB process of the second non-interactive processor receive MATLAB commands from the interactive processor and send the MATLAB commands (*configured to send commands*) to the MATLAB (*to the second kernel*) in that MATLAB process for execution by that MATLAB, as shown in the combined figure of Figures 1 and 2 of Menon:

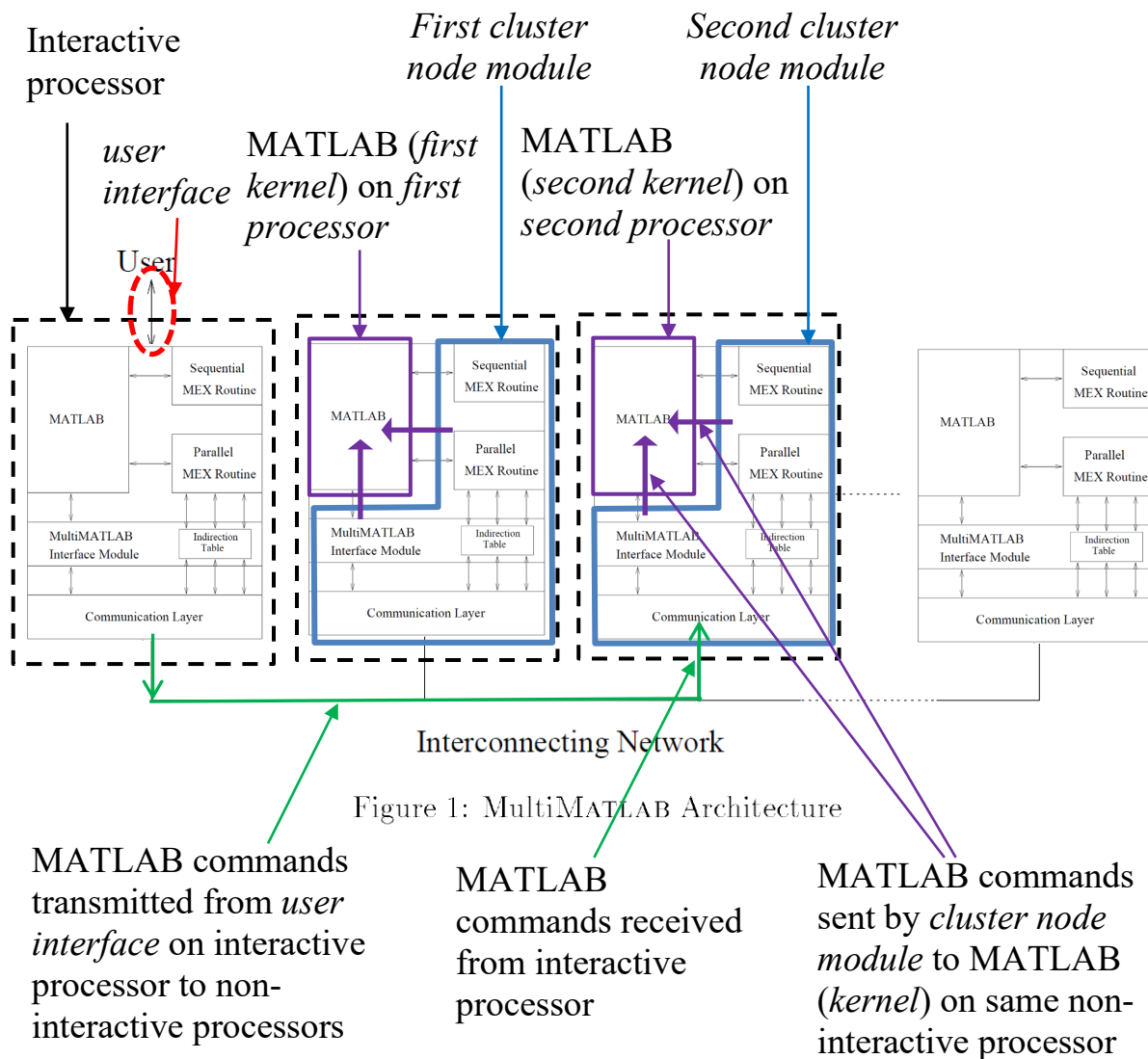


Figure 1: MultiMATLAB Architecture

Ex.1005, FIGs. 1 and 2 (combined and annotated); Ex.1003, ¶233

Ex.1003, ¶¶231-233.

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

First, as shown in [1.4.1] and [1.6.1], Menon describes the MEX routines, MultiMATLAB interface module, indirection table, and communication layer of the first non-interactive processor (*first processor*) and the second non-interactive

processor (*second processor*) teach the *first cluster node module* and said *second cluster node module*.

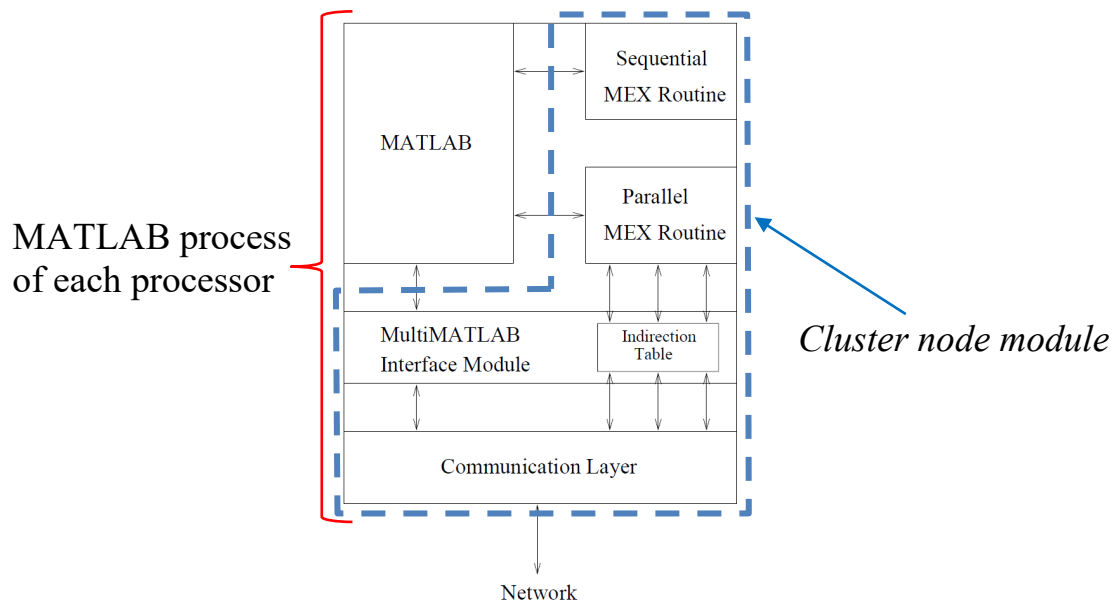


Figure 2: MATLAB Process Address Space

Ex.1005, FIG. 2 (annotated); Ex.1003, ¶236

Second, the cluster node module of each MATLAB process handles communications with other processes. Specifically, 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. And, it is the “MEX routines” of the cluster node module that “access the underlying communication layer and, hence, the network, through the MultiMATLAB interface module.” Ex.1005, 4.

MultiMATLAB’s interconnection network used by the cluster node module is shown in Figure 1.

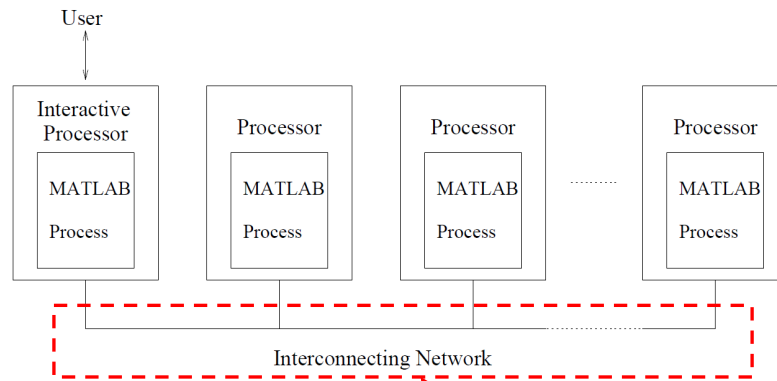


Figure 1: MultiMATLAB Architecture

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

Ex.1005, FIG. 1 (annotated); Ex.1003, ¶236

Table 1 of Menon below shows example point-to-point commands that enable communication between the cluster node modules, such as the “Eval(pid,‘command’)” command that “[e]valuates a command on one or more MATLAB process[es],” the “Send(pid,data)” command that “send[s] data from one process to another,” and the “Recv(pid)” command that “receive[s] data sent

from another process.” Ex.1005, 8, Table 1.

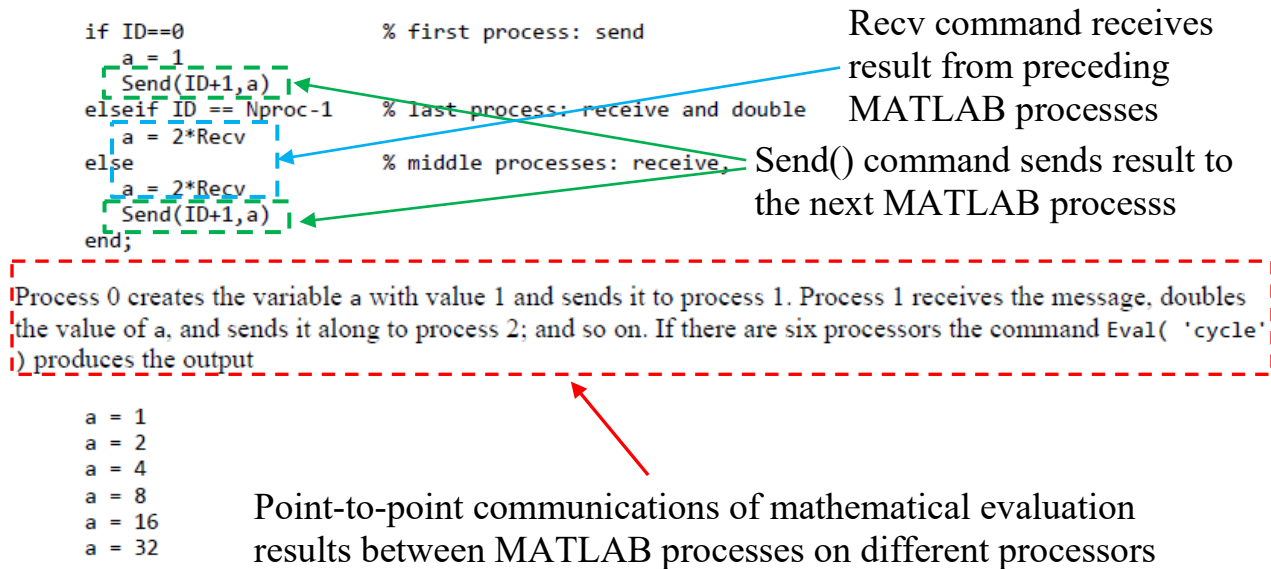
MultiMATLAB commands	
Starting and stopping MultiMATLAB (MPICH only)	
Start(n)	Initializes n remote MATLAB processes.
Quit	Closes remote MATLAB processes.
Running commands on multiple processors	
Nproc	Returns number of MATLAB processes.
ID	Returns ID of calling MATLAB process.
Eval(pid,'command')	Evaluates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

Table 1: Selected commands in the MultiMATLAB system

Commands for point to point communication

Ex.1005, Table 1 (annotated); Ex.1003, ¶239

Trefethen provides multiple examples where MATLAB processes use the MEX routines Send() and Recv() to communicate results of mathematical expression evaluation to other MATLAB processes on other processors. Ex.1006, 6; Ex.1005, 9. In one example, Trefethen shows how a MATLAB process communicates a result of a mathematical evaluation to a next MATLAB process, which in turn communicates the next mathematical result to the next MATLAB process, and so on, using the MEX routines Send() and Recv():



Ex.1006, 6 (annotated); Ex.1003, ¶240. In particular, the example code shows MATLAB process ID=2 using its MEX routines (part of *said second cluster node module*) to receive (*configured to communicate with*) a result that MATLAB process ID=1 sent using its own MEX routines (part of *the first cluster node module*).

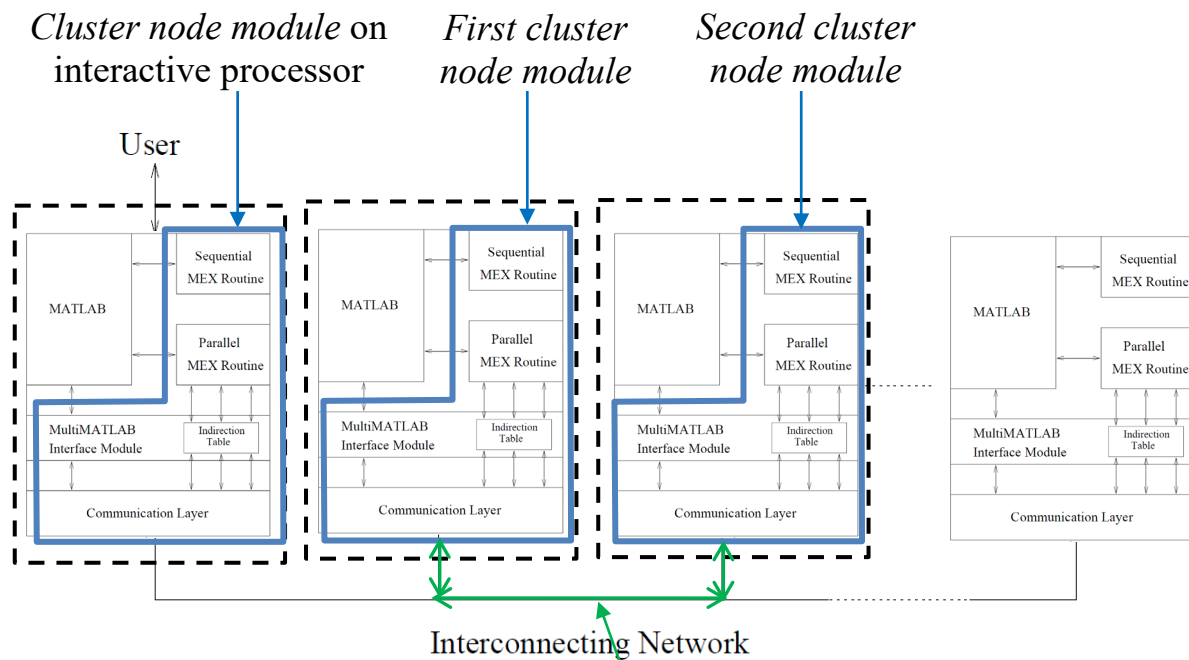


Figure 1: MultiMATLAB Architecture

Cluster node modules communicating to and from each other

Ex.1005, FIGs. 1 and 2 (combined and annotated); Ex.1003, ¶243

Accordingly, Menon and Trefethen teach that the MATLAB processes on the second non-interactive processor and the first non-interactive processor use their respective MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*said second cluster node module and the first cluster node module*) to communicate commands (e.g., tasks) and data directly with each other, rendering obvious this limitation. Ex.1003, ¶234-243.

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

As shown in [1.3.1], MATLAB of a MATLAB process teaches *a kernel*. Accordingly, MATLAB of the MATLAB process on the third non-interactive processor teaches *a third kernel*. Further, as shown in limitation [1.2], the MATLAB process of each processor of MultiMATLAB is in a MATLAB process address space (*computer-readable medium*). Accordingly, Menon teaches a MATLAB (*a third kernel*) in the MATLAB process on the third non-interactive processor (*third processor*), where the MATLAB process is contained in (*residing in*) a MATLAB process address space (*the at least one computer-readable medium*). Ex.1003, ¶¶244-246.

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

First, as shown in [1.1.3], the third non-interactive processor teaches *the third processor*. Further, as shown in [1.7.1], MATLAB in the MATLAB process on the third non-interactive processor teaches *said third kernel*.

Second, as shown in [1.5.2], in the “cycle.m” example previously discussed, MATLAB process ID=3 on the third non-interactive processor (*third kernel*) receives the result “a=4” (*the first result*) from MATLAB process ID=2. Further, as discussed in [1.3.2], Trefethen teaches that MATLAB in MATLAB process

ID=3 doubles the received variable (“a=4”) to “a=8” (*a second result*) and sends it to the next MATLAB process, i.e., MATLAB process ID=4, as shown below:

```

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

```

user command

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

```

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

```

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

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

Ex.1006, 6 (annotated); Ex.1003, ¶249.

Third, MATLAB (the *kernel*) communicates with other processes using its *cluster node module*; that is, for example, MATLAB process ID=3 on the third non-interactive processor sends the result “a=8” (*a second result*) to the next MATLAB process ID=4 via its *third cluster node module* (for the result to arrive at the next MATLAB process ID=4). A POSITA would recognize and understand that for *the third cluster node module* in MATLAB process ID=3 to send the result “a=8” (*a second result*) to the next MATLAB process, *the third cluster node module* receives the result “a=8” from MATLAB (*third kernel*). In other words, a

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

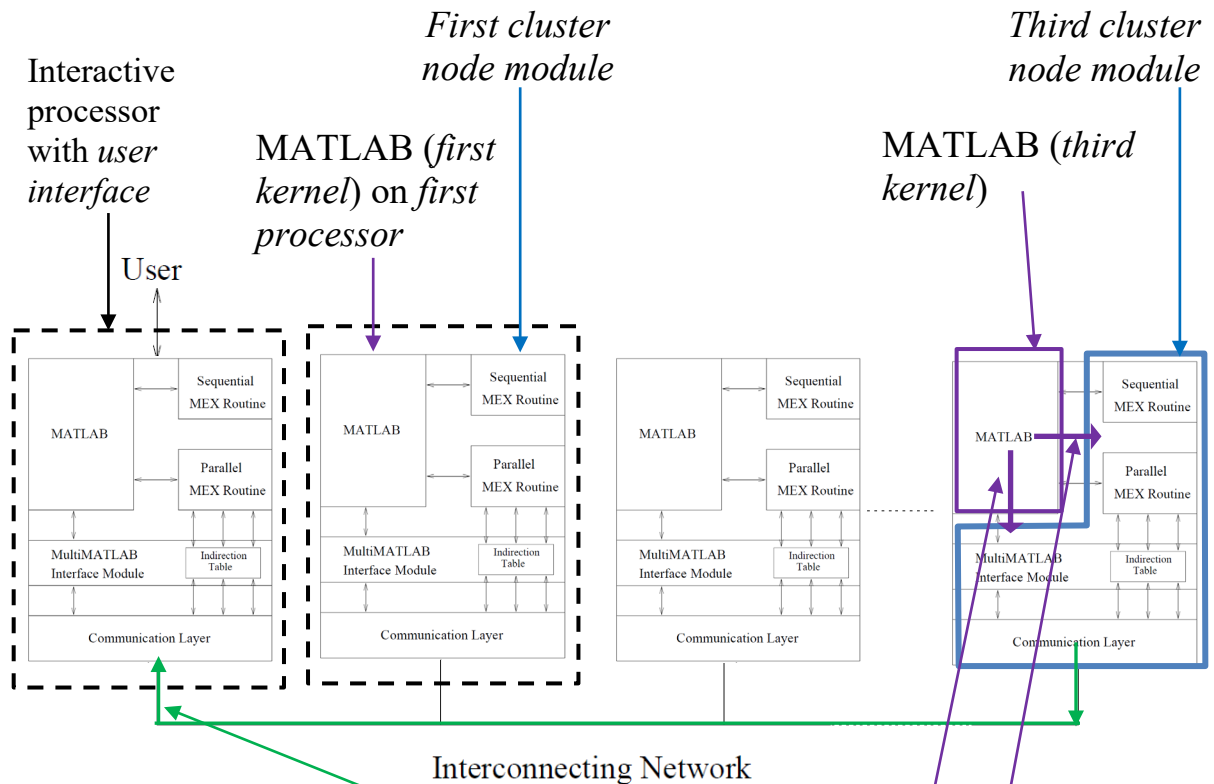


Figure 1. MultiMATLAB Architecture

MATLAB results
received from third
non-interactive
processor

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

Ex.1005, FIGs. 1 and 2 (combined and annotated); Ex.1003, ¶250

Fourth, as shown in [1.3.2], Menon and Trefethen describe MultiMATLAB performing mathematical evaluations of matrices such as multiplying matrices, computing condition numbers of Hilbert matrices, computing eigenvalues of matrices and for MATLAB in each MATLAB process to execute matrix-related mathematical evaluations of matrices when receiving and executing the m-file cycle.m. For example, Menon’s “parallel MATLAB implementation of conjugate gradients,” performs “matrix-vector multiplication” in “iteration” and “communication is required for two different operations.” Ex.1005, 11.

Accordingly, a POSITA would recognize and find it obvious for the result (*first result*) from MATLAB process ID=2 to be used by MATLAB (*said third kernel*) in the MATLAB process ID=3 to perform a next iteration of matrix-related mathematical evaluations (*execute additional matrix operations*) to generate a result (*second result*). Ex.1003, ¶252.

Accordingly, Menon teaching MATLAB in the MATLAB process on the third non-interactive processor, in view of Trefethen teaching an m-file where that MATLAB of the MATLAB process on the third non-interactive processor receives tasks and data, performs mathematical computation and provides the result to the *cluster node module* in the MATLAB process on the third non-interactive processor for sending to another processor, in further view of Menon and Trefethen

teaching that each MATLAB process performs mathematical evaluations of matrices, renders obvious this limitation. Ex.1003, ¶¶247-253.

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

As shown in [1.4.1], the MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*cluster node module*) of each MATLAB process on a processor of MultiMATLAB are in the MATLAB process address space. Accordingly, the MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*a third cluster node module*) of the MATLAB process on the third non-interactive processor are in (*residing in*) the MATLAB process address space (*the at least one computer-readable medium*) of that MATLAB process, rendering obvious this limitation. Ex.1003, ¶¶254-255.

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

As shown in [1.8.1], the MEX routines, MultiMATLAB interface module, indirection table, and communication layer of the third non-interactive processor (*third processor*) teach *said third cluster node module*. Further, as shown in [1.7.1], MATLAB in the MATLAB process on the third non-interactive processor (*third processor*) teaches *the third kernel*.

As shown in [1.4.2], the interactive processor sends the user commands to the cluster node module of each non-interactive processor, which provides

MATLAB commands to its respective MATLAB (*kernel*) for execution.

Accordingly, the MEX routines (part of *said third cluster node module*) in the MATLAB process of the third non-interactive processor receive MATLAB commands from the interactive processor and send the MATLAB commands (*configured to send commands*) to the MATLAB (*to the third kernel*) in that MATLAB process for execution by that MATLAB, as shown in the combined figure of Figures 1 and 2 of Menon:

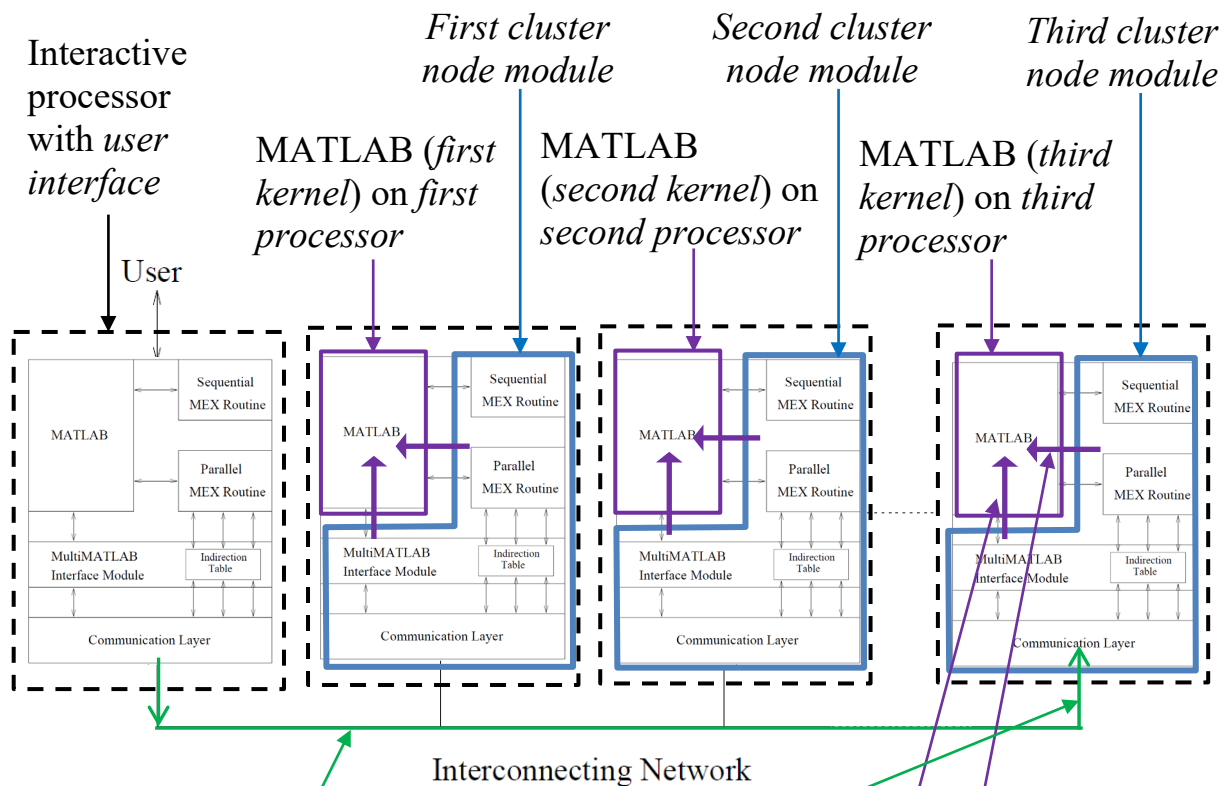


Figure 1: MultiMATLAB Architecture

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

MATLAB commands received from interactive processor

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

Ex.1005, FIGs. 1 and 2 (combined and annotated); Ex.1003 ¶258

Ex.1003, ¶¶256-258.

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

As discussed in [1.6.3], each MATLAB process has a cluster node module for communicating with the other cluster node modules. Further, each MATLAB

process is configured to use the Eval(), Send(), and Recv() commands to send data from one process to another.

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); Ex.1003, ¶239

As further discussed in [1.6.3], Trefethen provides multiple examples where MATLAB processes use the MEX routines Send() and Recv() to communicate results of mathematical expression evaluation to other MATLAB processes on other processors. Ex.1006, 6; Ex.1005, 8-9, Table 1.

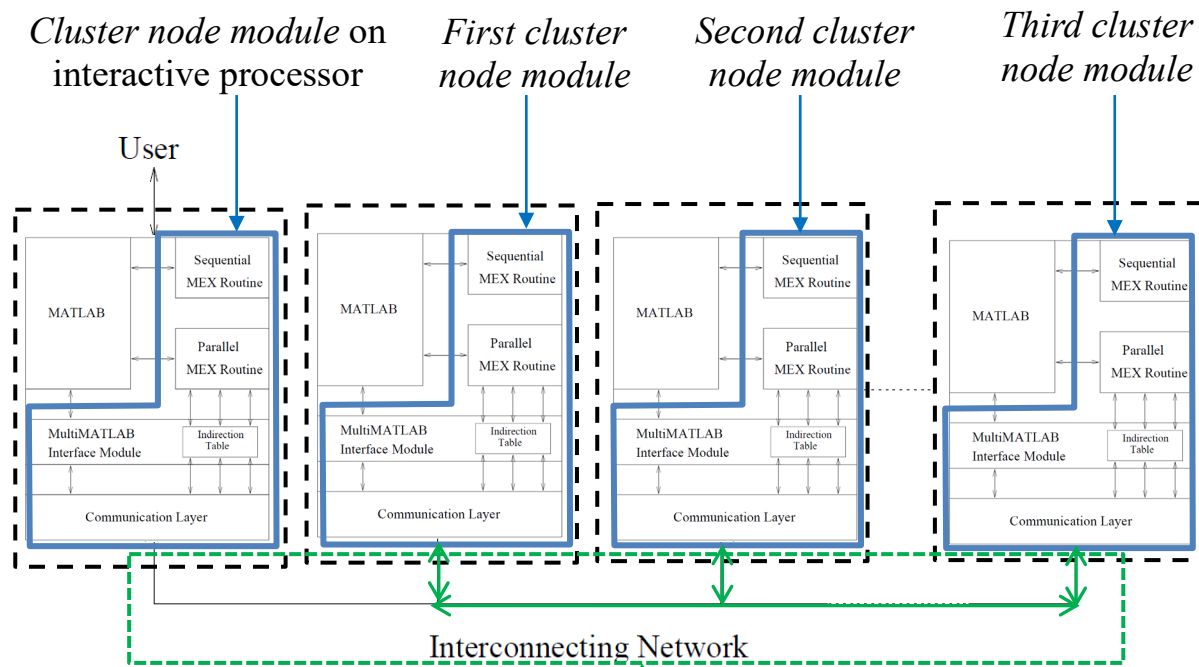


Figure 1: MultiMATLAB Architecture

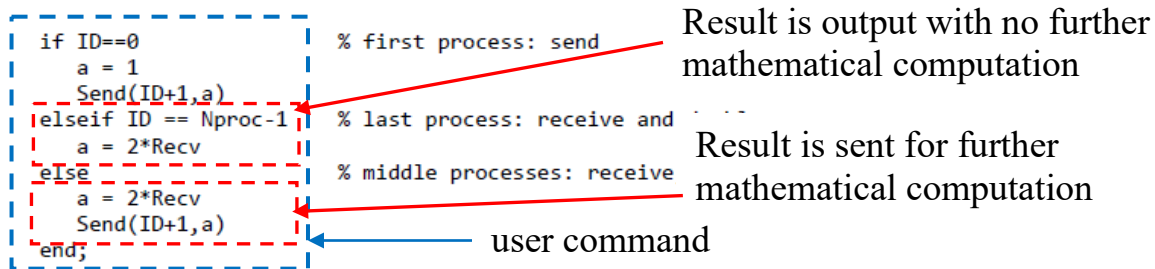
Cluster node modules communicating to and from each other

Ex.1005, FIGs. 1 and 2 (combined and annotated); Ex.1003, ¶260

Accordingly, Menon and Trefethen teach that the MATLAB processes on the third non-interactive processor, the second non-interactive processor, and the first non-interactive processor use their respective MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*said third cluster node module, the second cluster node module and the first cluster node module*) to communicate commands (e.g., tasks) and data with each other, rendering obvious this limitation. Ex.1003, ¶¶259-261.

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

First, as discussed in limitation [1.7.2], Trefethen teaches that, when executing “the m-file cycle.m,” the final MATLAB process is programmed to output the results to the screen of the interactive process.



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

```

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

```

Annotations:

- Result from MATLAB process ID=1** points to the second line `a = 2`.
- Result (a second result) from MATLAB process ID=3** points to the fourth line `a = 8`.

Ex.1006, 6 (annotated); Ex.1003, ¶263.

Trefethen’s “cycle.m” example code, however, is a simple algorithm exemplifying that MultiMATLAB can (i) pass results from one process to another process and (ii) the result is used to perform subsequent mathematical computations. MultiMATLAB, though, is intended for running a “program for six different boundary conditions, or a dozen different parameter choices, or a hundred different initial guesses.” Ex.1006, 3. Menon describes using MultiMATLAB “as a linear system solver,” calculating “discrete Fourier transform,” using “using 64 x

64 real matrix,” “matrix multiplications,” “solve for eigenvalues,” and calculating the “conjugate gradient algorithm.” Ex.1005, 7, 10-11.

To use MultiMATLAB in the complicated mathematical algorithms contemplated by Menon and Trefethen, Trefethen describes that “any of the MATLAB processes” can perform “general point-to-point communication” using “send and receive commands.” Ex.1006, 5. The Send(), Eval(), and Recv() MEX routines (built on MPI message passing) send results directly from one MATLAB process to another process. Ex.1005, 3; Ex.1006, 5-6. For example, the Send(), Eval(), and Recv() MEX routines can send a result to any other process by simply specifying the process ID. Ex.1006, 3, 5-6.

The algorithms contemplated by Menon and Trefethen that would send a result to a specific process were well-known in the art. For example, POSITAs knew that MPI message passing (such as used in Trefethen) was used to transmit results to a “chosen” process (other than the master) that the other processes (“peers”) know to send the results to. Ex.1011, 64 (a “pseudo-master [that] is arbitrarily chosen to be the task [that] all the peers know a priori who to send the results back to”).

To achieve the mathematical computations contemplated by Menon and Trefethen, a POSITA would use the MEX routine commands to send a result calculated by third kernel to the first cluster node module for subsequent

processing by the first kernel. *See* Ex.1011, 64. For example, in the conjugate gradient algorithm described in Menon, the kernels are each processing, in parallel, and awaiting results from the other processes to iteratively calculate further results, which includes the third process using the results from the second process and the first process using the results from the third process. Ex.1005, 11-14. As an additional example, the “cycle.m” algorithm of Trefethen would, as a final calculation by process ID=1, add 1 (or any other computation needed for a more complex mathematical algorithm) to a result received from the third process, for a final result of 9. Ex.1003, ¶267.

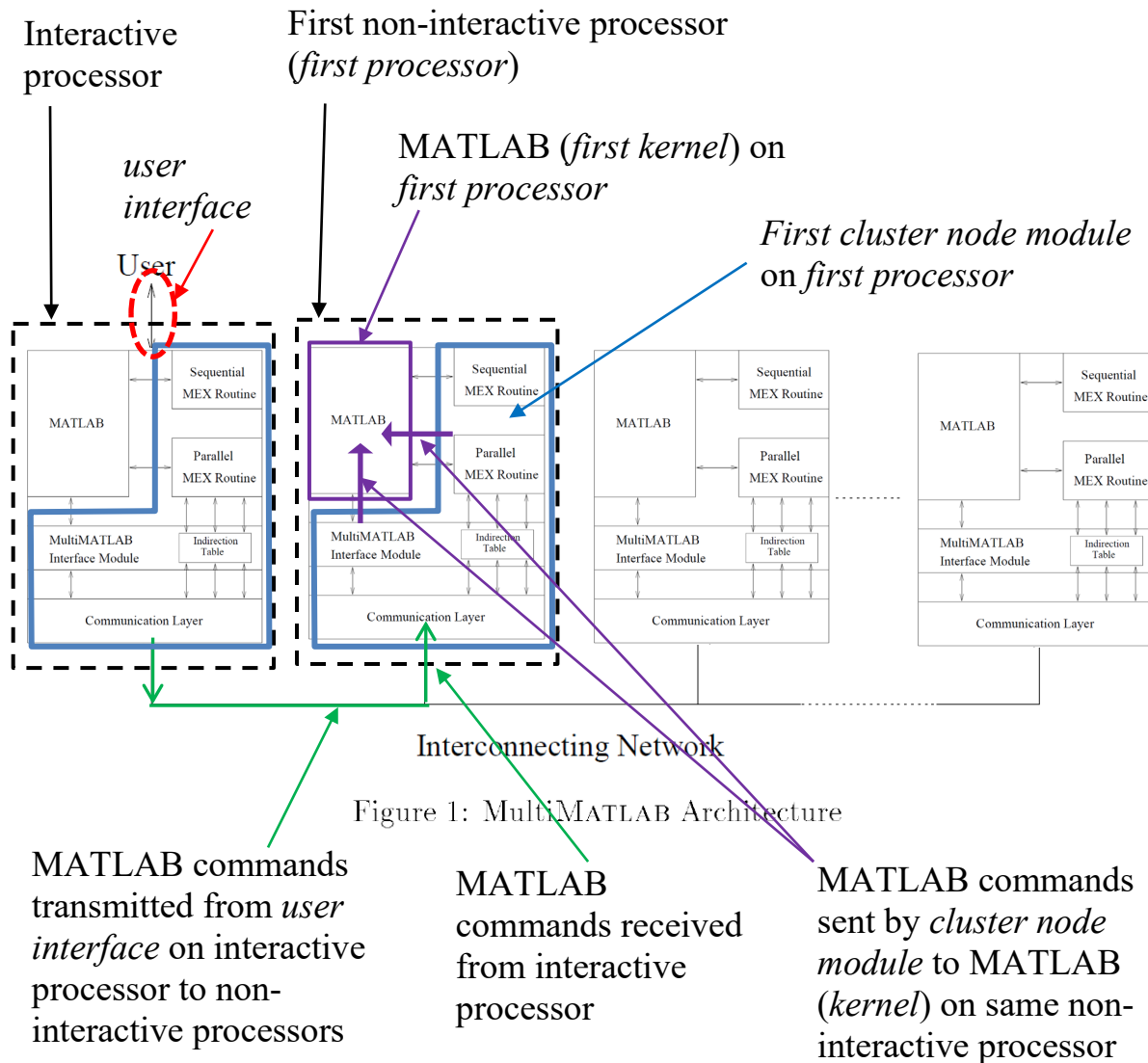
Further, as shown in [1.4.2], the interactive processor has a user interface for providing intermediary and final results of the mathematical calculations to the user.

Accordingly, because Menon and Trefethen teach that each MATLAB process (e.g., the first non-interactive process) is configured to receive a result from any other MATLAB process via its cluster node module, and to perform subsequent processing and returning the final result to the interactive processor, Menon and Trefethen render obvious this limitation. Ex.1003, ¶¶262-270.

Claim 2

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

As shown in [1.4.2], each non-interactive processor receives MATLAB commands via its cluster node module. Specifically, as illustrated in the combination figure of Figures 1 and 2 of Menon, the MEX routines of *the first cluster node module* on the first processor receive MATLAB commands from a user interface on the interactive processor:



Ex.1005, FIGs. 1 and 2 (combined and annotated); Ex.1003, ¶272

The *first cluster node module* on the first processor is logically positioned between: (i) the user interface on the interactive processor and (ii) the MATLAB kernel on the first non-interactive processor (*the first kernel*), rendering obvious this limitation. Ex.1003, ¶¶271-272.

Claim 3

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

Limitation [3.0] is rendered obvious for the reasons presented for [1.0].

Ex.1003, ¶273.

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

As shown in [1.1.1] and [1.1.2], each of the first non-interactive processor (*the first processor*) and the second non-interactive processor (*the second processor*) of MultiMATLAB has a respective MATLAB process:

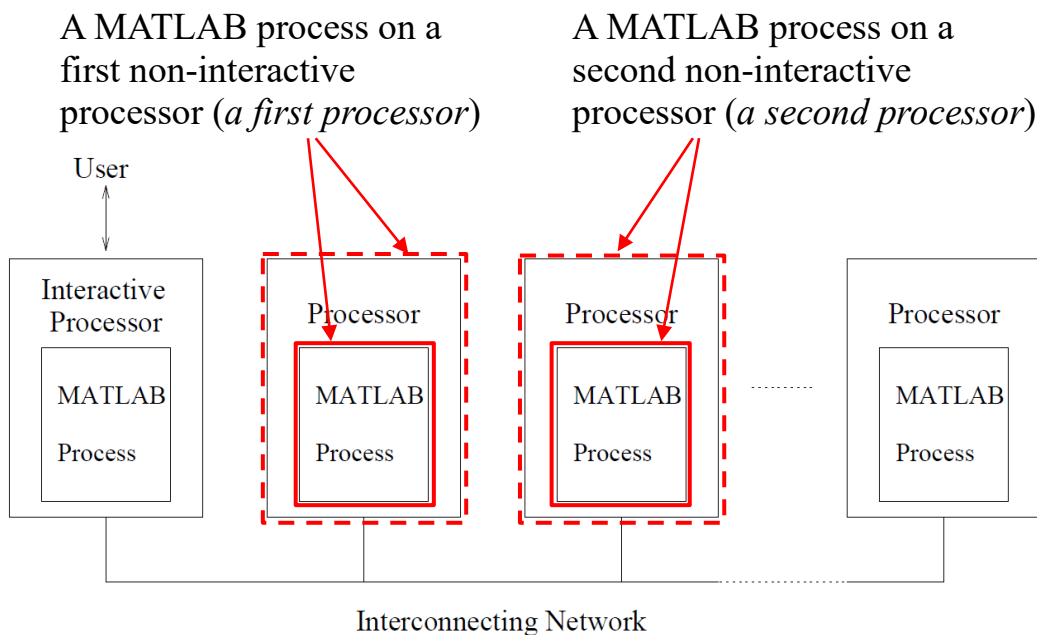


Figure 1: MultiMATLAB Architecture

Ex.1005, FIG. 1 (annotated); Ex.1003, ¶275

As shown in [1.1.1] and [1.1.2], Menon's first non-interactive processor and

the second non-interactive processor are implemented on an IBM SP2 in view of RS6000 teaching the IBM SP2 has four processors. Ex.1007, 9.

As discussed in [1.2], the MATLAB process address space (*a computer-readable medium*) of the IBM SP2 can be disk drives or memory cards. RS6000 also discloses that the processors can be attached to “hard disk drives” and “[m]emory.” Ex.1007, 11, 10.

Accordingly, Menon’s teaching that each MATLAB process is in an address space in memory (*a computer-readable medium*) on an IBM SP2 in view of RS6000’s teaching that the IBM SP2’s processors are each attached to disk drives and memory, renders obvious this limitation. Ex.1003, ¶¶274-278.

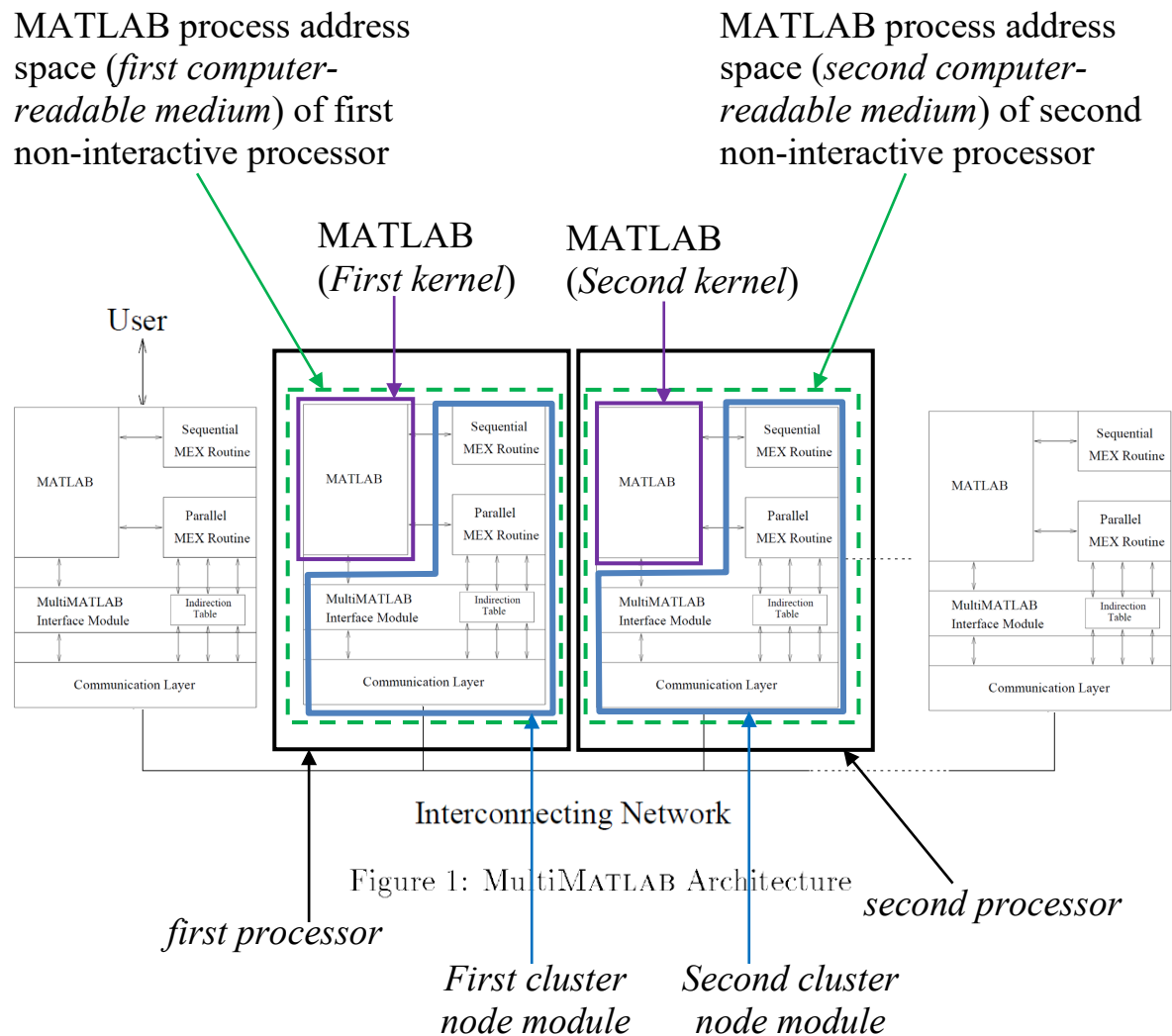
[3.2] wherein the first kernel and the first cluster node module reside in the first computer-readable medium, and wherein the second kernel and the second cluster node module reside in the second computer-readable medium.

As shown in [1.3.1], Menon teaches MATLAB (*a first kernel*) in the MATLAB process on the first non-interactive processor, where the MATLAB process is contained in (*residing in*) a MATLAB process address space (*first computer-readable medium*).

As shown in [1.4.1], the MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*a first cluster node module*) of the MATLAB process of the first non-interactive processor are in (*residing in*) the MATLAB process address space (*first computer-readable medium*).

As shown in [1.5.1], Menon teaches MATLAB (*a second kernel*) in the MATLAB process on the second non-interactive processor, where the MATLAB process is contained in (*residing in*) a MATLAB process address space (*second computer-readable medium*).

As shown in [1.6.1], the MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*a second cluster node module*) of the MATLAB process of the first non-interactive processor are in (*residing in*) the MATLAB process address space (*second computer-readable medium*).



Ex.1005, FIGs. 1 and 2 (combined and annotated); Ex.1003, ¶284

Ex.1003, ¶¶279-284.

Claim 5

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

As shown in [1.6.3], each MATLAB process in the MultiMATLAB

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

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

Table 1: Selected commands in the MultiMATLAB system

Commands for point to point communication

Ex.1005, Table 1 (annotated); Ex.1003, ¶286

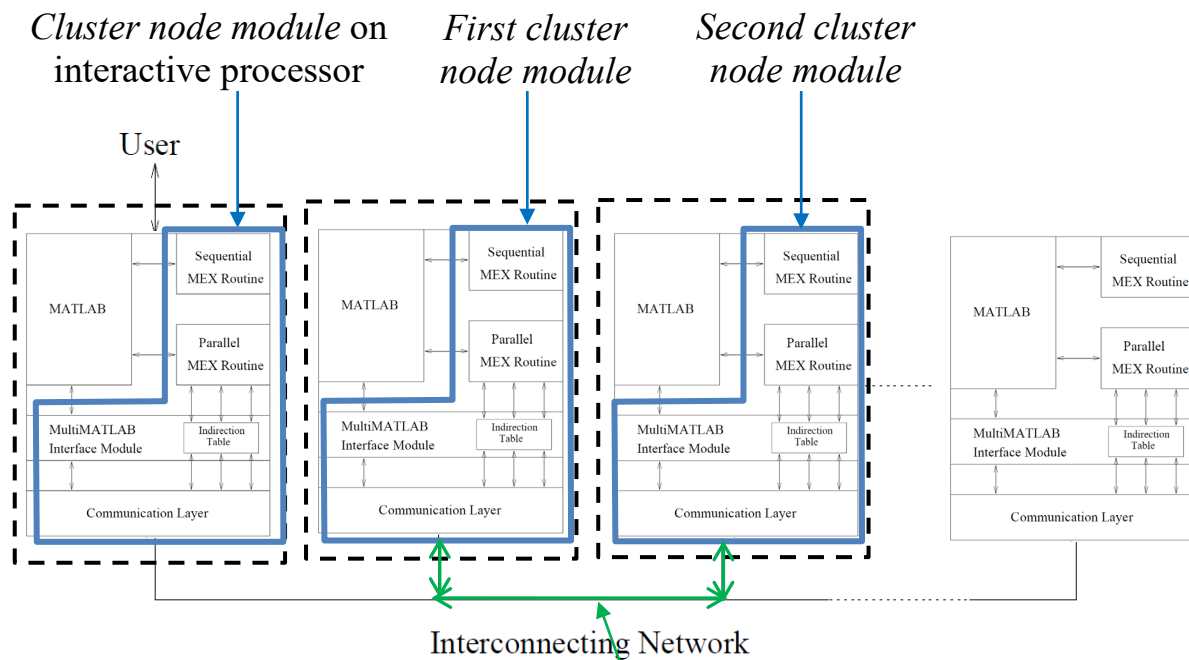


Figure 1: MultiMATLAB Architecture

Cluster node modules communicating to and from each other

Ex.1005, FIGs. 1 and 2 (combined and annotated); Ex.1003, ¶243

Menon describes how each parallel MEX routine (such as Send and Recv) is mapped to “MPI calls” while “maintaining the syntax of MPI.” Ex.1005, 6.

Trefethen further notes that “MultiMLATLAB is currently built upon the standard send and receive variants” and that it uses “MPI sends and receives.” Ex.1005, 10.

Trefethen describes an example where tasks are distributed using Send and Recv (which are built upon MPI sends and receives) without requiring any user commands. In this example code, the user initiates the computation by entering the instruction “Eval(‘cycle.m’).” During the process, tasks and data are

communicated among all of the kernels using Send() and Recv(), without requiring any user command.

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

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

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

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

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

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

SPMD point to point communication among processors of the MultiMATLAB architecture

user instructions/commands

Ex.1006, 6 (annotated); Ex.1003, ¶287.

Accordingly, Menon and Trefethen disclose that the second cluster node module of the MATLAB process on the second non-interactive processor directly communicates with the first cluster node module of the MATLAB process on the first non-interactive processor, where the MEX routines (part of the *second cluster node module*) in the MATLAB process on the second non-interactive processor use commands such as Send() and Recv() that implement standard variants of MPI

calls, which send MATLAB commands (*configured to distribute tasks*) to MATLAB (*among at least one of the first, second, or third kernels*) in the MATLAB process on the second non-interactive processor without any input or command from the user (*without requiring user commands*), rendering obvious this limitation. Ex.1003, ¶¶285-293.

Claim 6

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

As shown in [1.6.3], each MATLAB process in MultiMATLAB communicates tasks and data directly with each other using their respective MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*cluster node module*). For example, as shown in [1.8.3], the MATLAB processes on the third non-interactive processor, the second non-interactive processor, and the first non-interactive processor use their respective MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*said third cluster node module, the first cluster node module and the second cluster node module*) to communicate commands (e.g., tasks) and data with each other. Menon teaches that “[e]ach process is [] provided with the ability to communicate with other processes through a communication layer that runs over the parallel platform’s interconnection network.” Ex.1005, 3.

As also discussed in [1.6.3], Menon's MEX routines "expose some functionality of the underlying communication layer to" MATLAB. Ex.1005, 8, 3. The communication layer is MPI. Ex.1005, 3.

Menon discloses that "MATLAB communication routines" work with "data structures," such as sparse matrices, cell arrays, and structures and that these are "communicated data structures." Ex.1005, 9. 250. Menon also teaches that the MEX routines, such as Recv(), allocate memory for storing received data, stating "MATLAB communication routines do not expose any memory management issues to users. Routines do any allocation necessary for received data. Routines block until any data to be sent can safely be overwritten and any data to be received has arrived." Ex.1005, 9-10. Trefethen teaches that the "allocation necessary for received data" disclosed by Menon is "memory allocation for received matrices." Ex.1005, 10; Ex.1006, 10.

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

communication layer, using the MPI calls, communicates over the interconnection network with the communication layer in the other processor. Ex.1005, 3.

address of memory allocated
for storing received data

type of the received data

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

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

Ex.1017, 19 (annotated)(emphasis original); Ex.1003, ¶304.

MPIref also describes that the MPI_RECV call “en_queues” (i.e., stores in a queue) the information received by the MPI_RECV call. Ex.1017, 164.

Accordingly, Menon and Trefethen’s combined teaching of a memory allocation of a MEX routine for received data for “MATLAB data structures such as sparse matrices, cell arrays, and structures” that are communicated renders obvious this limitation.

Alternatively, Menon and Trefethen disclose that each MATLAB process in the MultiMATLAB architecture communicates tasks and data directly with any other MATLAB process using its MEX routines, MultiMATLAB interface

module, indirection table, and communication layer (cluster node module), where MEX routines implement standard variants of MPI calls to enable SPMD point-to-point communication so that the MATLAB processes communicate data directly with each other, where the data have MATLAB datatypes such as sparse matrices, cell arrays, and structures, and that the MEX routines store the received data in a memory allocated for storing data having datatype of the host language (i.e., data having the MATLAB datatypes) in view of MPIref disclosing a message buffer and queue for storing messages, rendering obvious this limitation. Ex.1003, ¶¶294-307.

Claim 7

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

As shown in [6.0], the MEX routines of each MATLAB process store the received data in a memory allocated for storing data having MATLAB datatypes such as matrices, arrays, structures. In particular, the MPI call MPI_RECV, to which the MEX routine Recv that “receive[s] data sent from another process” is mapped, stores received data having MATLAB datatypes such as matrices, arrays, structures in a receive buffer that consists of a storage containing count consecutive elements of the type specified by datatype, starting at address buf:

receive buffer (memory allocated for
storing received data)

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

Ex.1005, 9; Ex.1017, 19 (annotated)(emphasis original); Ex.1003, ¶309.

MPIref also describes that the MPI_RECV call “en_queues” (i.e., stores in a queue) the information received by the MPI_RECV call. Ex.1017, 164.

Accordingly, Menon describes allocating memory for storing received data having MATLAB datatypes such as matrices, arrays, structures, using MPI and MPIref describes the MPI_RECV function that includes a receive buffer that stores received data having matrix datatype, array datatype, structure datatype, and enqueues the message received by the MPI_RECV call, rendering obvious this limitation. Ex.1003, ¶¶308-311.

Claim 8

[8.0] The computer cluster of claim 7, wherein the first cluster node module further comprises a message receiving queue, in which data specifying a location to which the first cluster node module can expect to receive an expression and an

identifier of a processor from which the expression is expected to be sent are stored.

As shown in [6.0], the MEX routines of each MATLAB process store the received data in a memory allocated for storing data having MATLAB datatypes such as matrices, arrays, structures. In particular, the MPI call MPI_RECV, to which the MEX routine Recv that “receive[s] data sent from another process” is mapped, stores received data having MATLAB datatypes such as matrices, arrays, structures in a receive buffer that consists of a storage containing count consecutive elements of the type specified by datatype, starting at address buf:

receive buffer (memory allocated for
storing received data)



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

Ex.1005, 9; Ex.1017, 19 (annotated)(emphasis original); Ex.1003, ¶313.

MPIref provides example code showing the MPI_RECV call in operation. Ex.1017, 163-164. In a snippet of this code, the MPI_RECV call is executed and

the data obtained from the call is stored in variables named “client_tag,”

“client_source,” and “client_rank_in_new_world”:

```
int Do_server(server_comm)
MPI_Comm server_comm;
{
    void init_queue();
    int enqueue(), de_queue(); /* keep triplets of integers
                                for later matching (fns not shown) */

    MPI_Comm comm;
    MPI_Status status;
    int client_tag, client_source;
    int client_rank_in_new_world, pairs_rank_in_new_world;
    int buffer[10], count = 1;
```

```
void *queue;
init_queue(&queue);
```

```
for (;;)
{
```

```
    MPI_Recv(buffer, count, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG,
             server_comm, &status); /* accept from any client */
```

```
    /* determine client: */
```

```
    client_tag = status.MPI_TAG;
    client_source = status.MPI_SOURCE;
    client_rank_in_new_world = buffer[0];
```

MPI_RECV call in the
communication layer, i.e., MPI
(*cluster node module*)



Source of received
data



Contents of information from MPI_RECV



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



Ex.1007, 163 (annotated); Ex.1003, ¶1314.

Subsequently, the code “en_queues” (i.e., stores in a queue) the information received by the MPI_RECV call (the information in the variables named “client_tag,” “client_source,” and “client_rank_in_new_world”):

```
en_queue(queue, client_tag, client_source,  
         client_rank_in_new_world);
```

Ex.1017, 164; Ex.1003, ¶315. The variable “client_source” contains the source of the received data because “client_source = status.MPI_SOURCE” and “status.MPI_SOURCE ... contain[s] the source ... of the received message.”

Ex.1017, 164, 20. The variable “client_rank_in_new_world” contains the “initial address of receive buffer” because “client_rank_in_new_world = buffer[0]” and the receive buffer “start[s] at address buf” which is the “initial address of receive buffer.” Ex.1017, 164, 19 (emphasis original).

Accordingly, Menon discloses MEX routines (such as Recv()) (part of the *cluster node module*) that implement standard variants of MPI calls to enable SPMD point-to-point communication between any MATLAB processes (and as such including the MATLAB process on the first non-interactive processor), and MPIref discloses that the MPI call MPI_RECV, found in the communication layer (i.e., MPI) (part of the *first cluster node module*) of the MATLAB process on the first non-interactive processor and to which the MEX routine Recv() is mapped, stores in a queue (*a message receiving queue, in which data ... are stored*) the

information in the variables received by RECV_MPI and named “client_source,” which contains information about the source of the received data, (*an identifier of a processor from which the expression is expected to be sent*) and “client_rank_in_new_world,” which contains information about the initial address of receive buffer for storing the received data (*data specifying a location to which the first cluster node module can expect to receive an expression*), rendering obvious this limitation. Ex.1003, ¶¶312-316.

Claim 9

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

As shown in [7.0], the cluster node module of each MATLAB process allocates memory for storing received data having MATLAB datatypes such as matrices, arrays, structures, includes a receive buffer that stores received data having matrix datatype, array datatype, structure datatype, rendering obvious that each cluster node module has a data structure with a received message queue.

As shown in [1.6.3], Menon, Trefethen and MPIref, disclose that each MATLAB process in MultiMATLAB communicates tasks and data with any other MATLAB process using its MEX routines, MultiMATLAB interface module, indirection table, and communication layer. For example, the MEX routines, MultiMATLAB interface module, indirection table, and communication layer

(*second cluster node module*) of the MATLAB process on the second non-interactive processor communicates tasks and data with the MEX routines, MultiMATLAB interface module, indirection table, and communication layer of the MATLAB processes on the first non-interactive processor and the third non-interactive processor (*first cluster node module* and *third cluster node module*, respectively), where the MEX routines (part of the *cluster node module*) implement standard variants of MPI calls to enable SPMD point-to-point communication between the MATLAB processes.

Accordingly, Menon's MEX routines of the second cluster node module in the MATLAB process on the second non-interactive processor store the data received from the *first cluster node module* and the *third cluster node module* in a receive buffer that stores received data having matrix datatype, array datatype, structure datatype, etc., rendering obvious this limitation. Ex.1003, ¶¶317-320.

Claim 10

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

Limitation [10.0] is rendered obvious for the reasons presented for [1.0]. Ex.1003, ¶321.

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

As shown in [1.4.1], the MEX routines, MultiMATLAB interface module, indirection table, and communication layer correspond to the *cluster node module* of the '582 patent, as shown below.

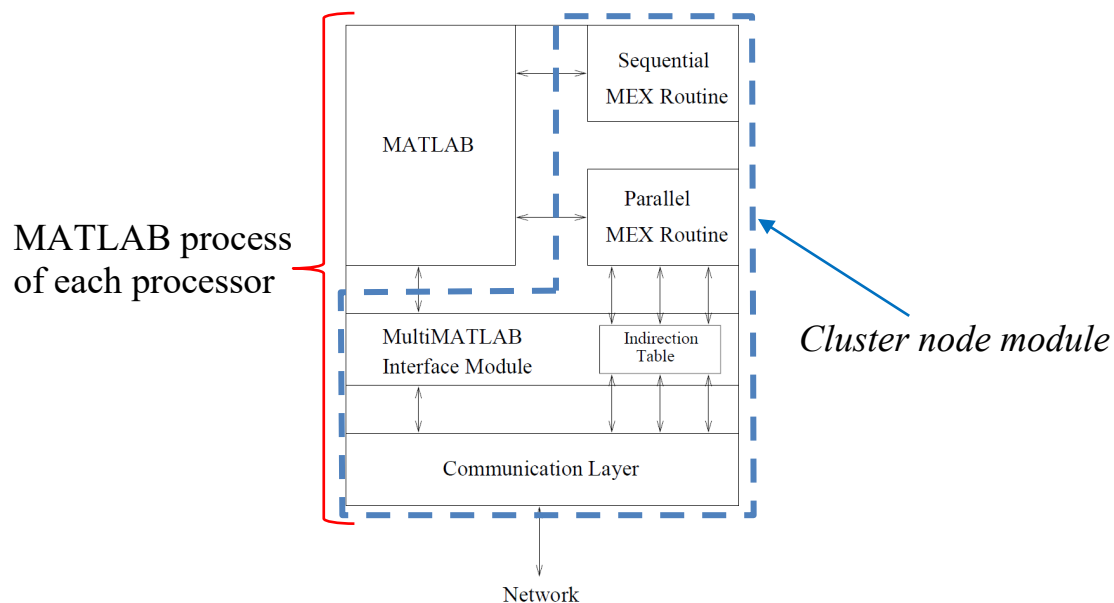
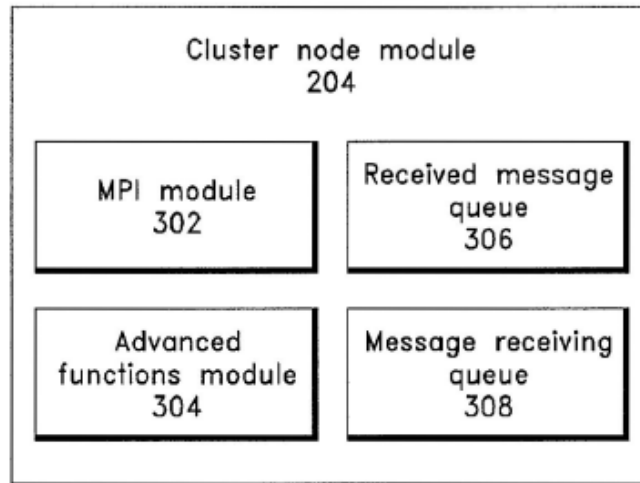


Figure 2: MATLAB Process Address Space

Ex.1005, FIG. 2 (annotated); Ex.1003, ¶325

Correspondingly, Fig. 3 of the '582 patent depicts a single cluster node module as a set of components:



Ex.1001, FIG. 3

The '582 patent describes the advanced functions module 304 as including “a custom set of directives or functions” or “program code that provides a toolkit of functions inconvenient or impractical to do with MPI instructions and calls implemented by the MPI module 302.” Ex.1001, 16:51-52, 16:44-47. Examples of the advanced functions module 304 calls include “functions providing for basic parallelization such as, for example, routines that would perform the same operations on many data elements or inputs, stored on many nodes,” “functions providing for linear algebra operations such as ... matrix and vector multiplication,” “program code for implementing large-scale parallel fast Fourier transforms (‘FFTs’),” “parallel disk input and output calls,” and “[a]utomatic [l]oad [b]alancing.” Ex.1001, 17:5-9, 18:12-16, 19:18-21, 20:17-19, 21:1.

Similarly, Menon explains that the MEX routines in the MATLAB processes are responsible for all the parallel functionalities of the MultiMATLAB architecture. Ex.1005, 3. Specifically, the parallel MEX routines “would distribute data from the interactive MATLAB process among all processes, perform the parallel computation, and collect results back to the interactive process.” Ex.1005, 10. The “[p]arallel MEX routines may be run ... via the Eval command,” which “[e]valuates a [MATLAB] command on one or more MATLAB processes.” Ex.1005, 6, Table 1. Menon explains that “[p]arallel MEX routines may be run in a SPMD fashion on any subset of MATLAB processes via the Eval command.” Ex.1005, 6. Specifically, “MEX routines are granted access to MPI routines” and are mapped to MPI calls using the indirection table. Ex.1005, 6.

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

Table 1: Selected commands in the MultiMATLAB system Eval routine for parallel evaluation

Ex.1005, Table 1 (annotated); Ex.1003, ¶326

Menon also discloses that “parallel MEX routines would perform matrix multiplications, solve for eigenvalues, compute Fourier transforms, and so on.”

Ex.1005, 10. Menon teaches that “parallel MEX routines” enable “parallelism” in applications “such as a linear system solver or a discrete Fourier transform.”

Ex.1005, 7. Trefethen provides several examples where the MEX routine Eval is used to perform parallel linear algebra operations. Ex.1006, 4-10. For example, Trefethen describes the use of Eval to compute “the condition numbers of the Hilbert matrices” and matrix “eigenvalue computation” in parallel, as well as “comput[ation of] vectors obtained by reducing data over the copies of a vector or matrix located on all processors.” Ex.1006, 4, 5, 7.

Accordingly, Menon's cluster node module (having parallel MEX routines) provides basic parallelization of operations on the MultiMATLAB architecture, where these operations include advanced functions operations such as linear algebra operations (e.g., matrix and vector operations), Fourier transform computations, render obvious this limitation. Ex.1003, ¶¶322-329.

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

As shown in limitation [10.1], the parallel MEX routines render obvious an advanced functions module.

As also discussed in [10.1], the MEX routines are responsible for all the parallel functionalities of MultiMATLAB. Ex.1005, 3. The parallel MEX routines “distribute data from the interactive MATLAB process among all processes, perform the parallel computation, and collect results back to the interactive process.” Ex.1005, 10. Menon discloses that the “[p]arallel MEX routines may be run ... via the Eval command,” which is a “routine for parallel evaluation” and “[e]valuates a [MATLAB] command on one or more MATLAB processes.” Ex.1005, 6, 4, Table 1.

Trefethen provides several examples where the MEX routine Eval is used to perform parallel evaluations. Ex.1006, 4-10. For example, “Eval(‘ID’)” runs on

“all nodes.” Ex.1006, 3-4. “Eval(‘sqrt(2)’)” causes “the MATLAB command sqrt(2) [to be] executed on all six processors.” Ex.1006, 3.

Accordingly, the parallel MEX routines (*the advanced functions module*) include the command Eval (*comprises a call*) that causes parallel evaluation of commands and mathematical expressions (*that evaluates an expression ... in parallel*) by the MATLAB processes on the processors of the MultiMATLAB architecture (*across the computer cluster*), rendering obvious this limitation.

Ex.1003, ¶¶330-334.

Claim 11

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

As shown in limitation [10.1], the parallel MEX routines render obvious an advanced functions module.

The parallel MEX routines “distribute data from the interactive MATLAB process among all processes, perform the parallel computation, and collect results back to the interactive process.” Ex.1005, 10.

Menon discloses that the “**parallel MEX routines would perform** matrix multiplications, solve for eigenvalues, **compute Fourier transforms**, and so on.” Ex.1005, 10.

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

Ex.1003, ¶335-338.

Claim 12

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

As shown in limitation [10.1], the parallel MEX routines render obvious an advanced functions module.

The parallel MEX routines “distribute data from the interactive MATLAB process among all processes, perform the parallel computation, and collect results back to the interactive process.” Ex.1005, 10.

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

... *in parallel*) matrix multiplications (*matrix operations*) by the MATLAB processes of MultiMATLAB (*across the computer cluster*). Ex.1003, ¶¶339-345.

Claim 13

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

As shown in [1.9], it is obvious that the MATLAB process on the first non-interactive processor receives a result (*the second result*) from the MATLAB process on the third non-interactive processor and performs further computations to achieve a final result (e.g., performing the conjugate gradient algorithm, eigenvalues or linear system solvers). Accordingly, the output of the first non-interactive processors further computations (*the final result*) includes the result from the third non-interactive processor (*at least a portion of the second result*). Ex.1003, ¶¶346-347.

Claim 14

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

As shown in [1.9], each MATLAB process (including the MATLAB process on the first non-interactive processor) is configured to use its MEX routines, MultiMATLAB interface module, indirection table, and communication layer (*first cluster node module*) to return an output (*is configured to return the final result*) to the user “as soon as it is ready.” Ex.1006, 4.

As shown in [1.4.2], a “user interacts directly with one MATLAB process, called the interactive process, and operates within that process’s MATLAB environment” using a user interface, as shown in the combined figure of Figures 1 and 2 of Menon below. Ex.1005, 3, 6.

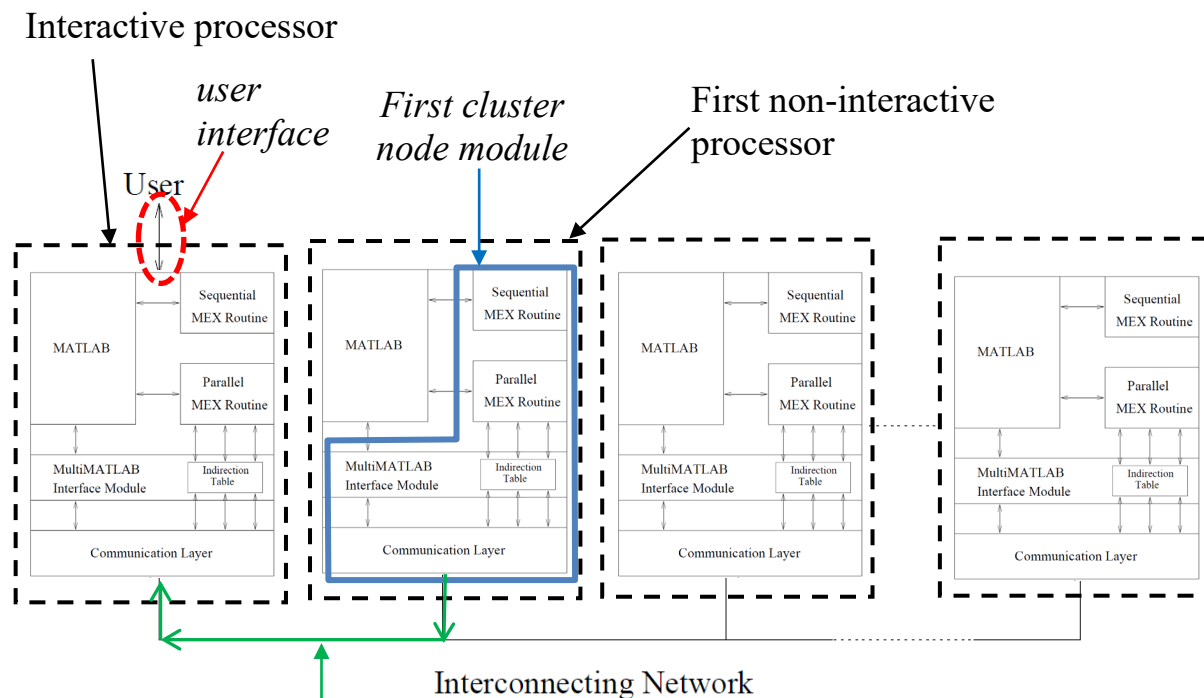


Figure 1: MultiMATLAB Architecture

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

Ex.1005, FIGs. 1 and 2 (combined and annotated); Ex.1003, ¶350

Trefethen also describes sending data from multiple processors to produce “plots in a distributed fashion that are then sent to the user’s screen.” Ex.1006, 7.

Accordingly, Trefethen's MATLAB process ID=1 (i.e., the MATLAB process on the first non-interactive processor) returns the output of its calculations using Menon's MEX routines, MultiMATLAB interface module, indirection table, and communication layer (the *first cluster node module*) for display on the user's screen (to the user interface) on the interactive processor, rendering obvious this limitation. Ex.1003, ¶¶348-352.

Claim 15

[15.0] *The computer cluster of claim 1, further comprising a communications network interface for the cluster node modules to communicate, with each other using a peer-to-peer architecture, results of execution of the matrix operations.*³

Menon discloses that “[a]ll parallel functionality in the MultiMATLAB system is provided through MEX routines.” Ex.1005, 3. The MEX routines send “commands over the underlying communication layer to the processes

³ In addition to the arguments presented here, “*communications network interface*” lacks patentable weight because it does not confer any structure on the claimed “*communications network interface*.” See *In re Miller*, No. 2021-1599, 2022 WL 594417, at *3 (Fed. Cir. Feb. 28, 2022); *Edgewell Pers. Care Brands, LLC v. Munchkin, Inc.*, 998 F.3d 917, 920–21 (Fed. Cir. 2021) (“An apparatus claim is generally to be construed according to what the apparatus is, not what the apparatus does.”).

corresponding to the specified IDs.” Ex.1005, 4. The underlying communication layer of MultiMATLAB “runs over the parallel platform’s interconnection network.” Ex.1005, 3. Menon describes MultiMatlab running on an “IBM SP2” and Figure 1 shows the interconnection network linking the processor nodes. Ex.1005, 5.

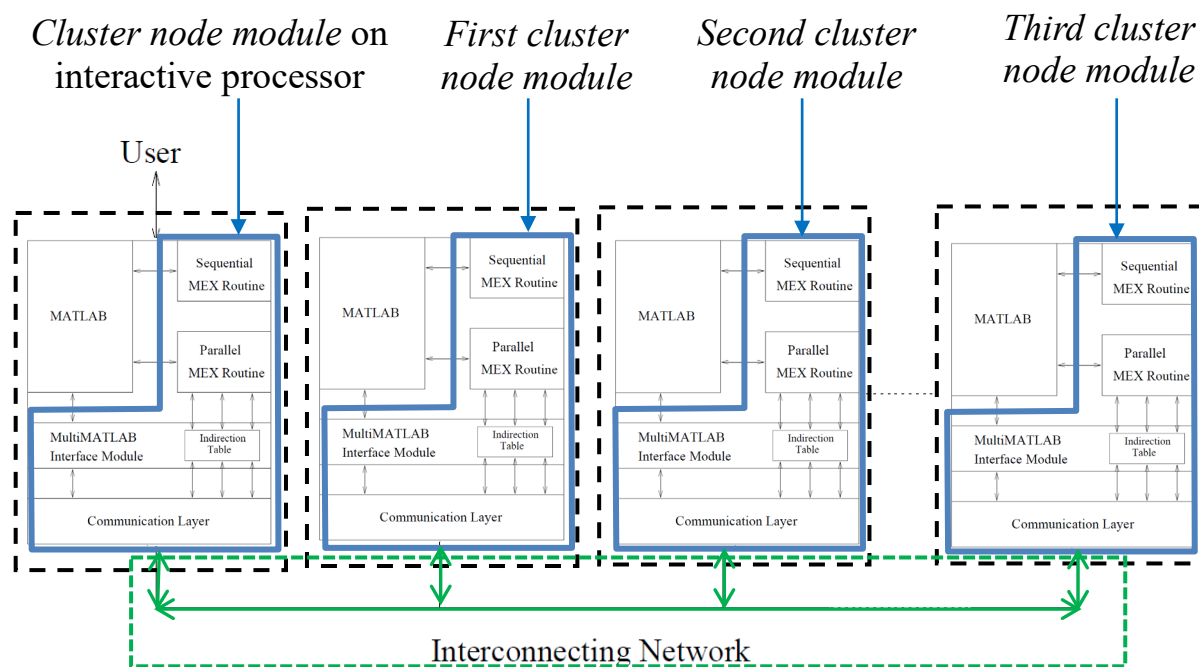


Figure 1: MultiMATLAB Architecture

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

Ex.1005, FIGs. 1 and 2 (combined and annotated); Ex.1003, ¶355

Trefethen discloses that MultiMATLAB’s commands 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 “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.

Trefethen also discloses sending tasks and data to other processor nodes and provides the example “Eval([4 5] , ‘cond(hilb(ID))’)”. Ex.1006, 6. MATLAB on each of node ID=4 and ID=5 then perform the task “cond(hilb(ID))” using the data ID. Ex.1006, 4.

Third, Menon’s MEX routines use a “point to point ... communication” paradigm. Ex.1005, 9. Unlike a “master/slave paradigm,” Menon’s SPMD/Message Passing paradigm uses “point to point ... communication” to provide a “finer degree of communication.” Ex.1005, 9. Table 1 of Menon lists point-to-point commands, including “Send(pid,data),” which “send[s] data from one process to another,” and “Recv(pid),” which “receive[s] data sent from another process.” Ex.1005, Table 1.

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

Table 1: Selected commands in the MultiMATLAB system

Commands for point to point communication

Ex.1005, Table 1 (annotated); Ex.1003, ¶239

Menon's 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*.

As discussed in [1.3.2], Menon and Trefethen describe MultiMATLAB performing mathematical evaluations of matrices such as multiplying matrices, computing condition numbers of Hilbert matrices, computing eigenvalues of matrices, etc., and for MATLAB in each MATLAB process to execute matrix-

related mathematical evaluations of matrices when receiving and executing the m-file cycle.m.

Trefethen discloses that MultiMATLAB commands can deliver a mathematical result to a designated process. For example, the instruction “Eval (‘Sum(1, [1 ID Nproc])’)” returns the result to process 1. “If the first argument [‘1,’] is omitted, the result is returned (broadcast) to all processes.” Ex.1006, 7.

RS6000 describes the IBM SP2’s network for connecting the nodes. For example, each node has “five [] adapter slots” that can include “Ethernet,” “Token Ring,” “ATM,” “Wide Area Network (WAN)” and more. Ex.1007, 2.

Additionally, each node has “integrated ethernet with BNC and RJ45 ports.” Ex.1007, 10. Further, the IBM SP2 includes a “switch” to “provide a message-passing network that connects all processor nodes with a minimum of four paths between every pair of nodes.” Ex.1007, 4.

Accordingly, Menon’s MEX routines that enable each processor node to communicate tasks and data directly with the other nodes over an interconnecting network implemented on an IBM SP2, in view of Trefethen’s teaching that MEX routines enable communicating results of matrix operations from any process to another process, further in view of RS6000 teaching that each IBM SP2 node uses Ethernet and other communication adapters to communicate over a message-passing network, renders obvious this limitation. Ex.1003, ¶¶353-359.

Claim 16

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

Limitation [16.0] is rendered obvious for the reasons presented for [1.4.1], [1.6.1], and [1.8.1]. Ex.1003, ¶360.

[16.1] *accept user instructions;*

As shown in [1.4.2], Menon discloses MultiMATLAB's interactive processor has a MATLAB process that provides a user interface to allow users to input equations.

Menon explains that users can provide “geometries, differential equations, and boundary conditions.” Ex.1005, 6. Further, as discussed in [1.3.2], a user can also provide *user instructions* via an “m-file,” which can contain “dozens of high-level commands.” Ex.1006, 2.

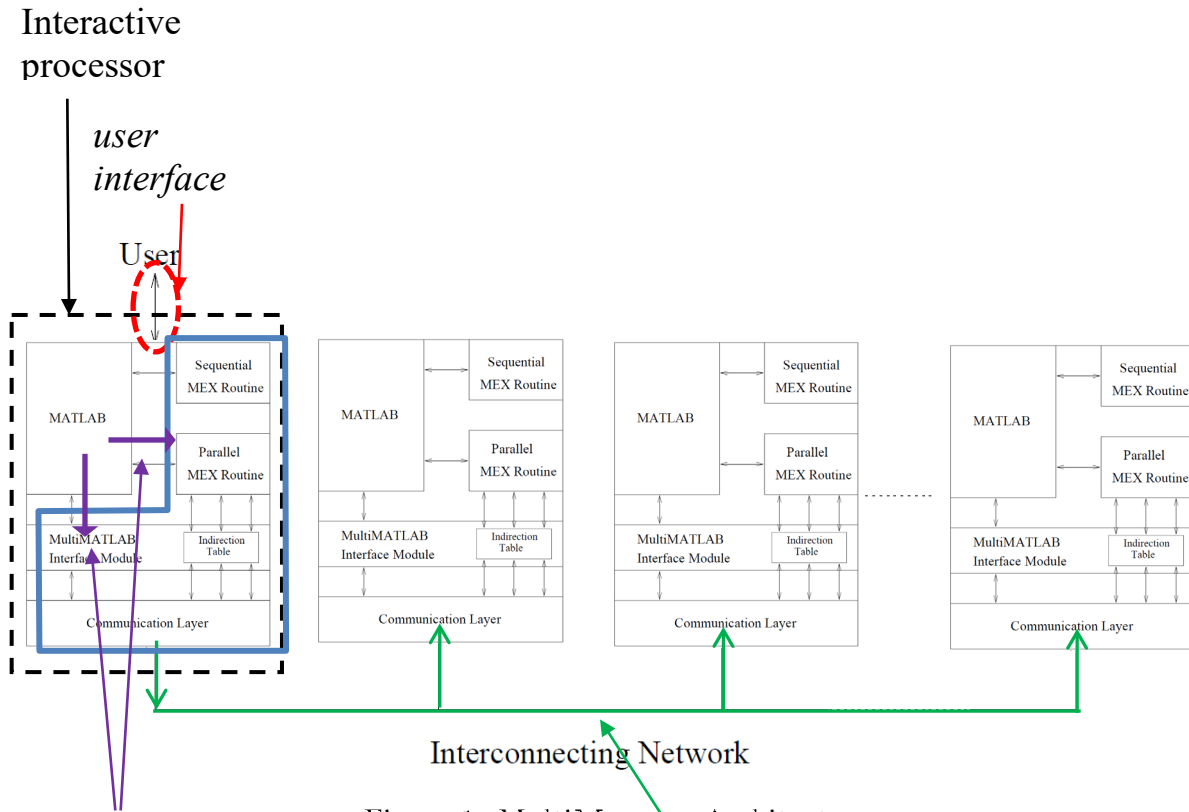


Figure 1: MultiMATLAB Architecture

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

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

Ex.1005, FIGs. 1 and 2 (combined and annotated); Ex.1003, ¶363

Accordingly, Menon's disclosure where the interactive process includes a cluster node module 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 this limitation. Ex.1003, ¶¶361-363.

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

As discussed in [16.1], *user instructions* can be provided via m-files that specify different actions for different MATLAB processes.

As discussed in [15.0], Menon and RS6000 render obvious a communications network interface for the nodes to communicate with each other.

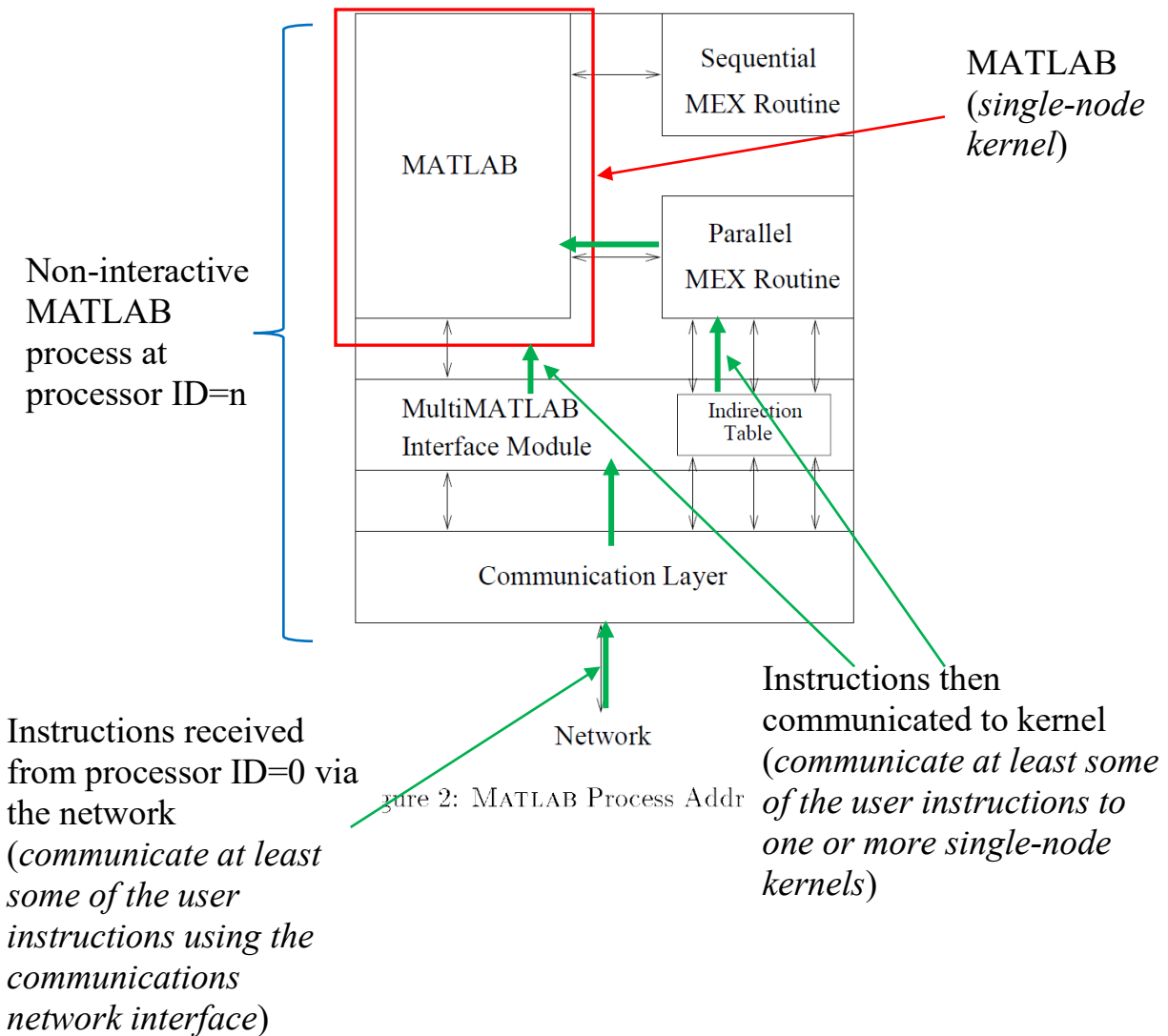
Menon describes how the interactive processor (e.g., ID=0) sends commands to the other MATLAB processes, stating that the Eval routine is “a MEX Routine” that receives “MATLAB process IDs and a MATLAB command... [and] sends the commands over the underlying communication layer to the processes corresponding to the specified IDs.” Ex.1005, 4.

the IBM SP2's interconnection network, thereby rendering obvious this limitation.

Ex.1003, ¶¶364-366.

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

Trefethen provides examples showing that commands specified in the *user instructions* are processed by the respective MATLAB processes identified by the IDs in the user instructions. Ex.1006, 3-7. For example, as discussed in [1.3.2], the user provides *user instructions*, such as “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. Ex.1006, 4; Ex.1003, ¶369.



Ex.1005, FIG. 2 (annotated); Ex.1003, ¶369

Accordingly, Menon and Trefethen teach that the MEX routines on the interactive processor (ID=0) communicate commands over the interconnection network implemented on an IBM SP2, and once received by each non-interactive node (ID=n), the commands are communicated to that MATLAB process's MATLAB (*single-node kernel*) for processing, thereby rendering obvious this limitation. Ex.1003, ¶¶367-370.

Claim 17

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

As discussed in [15.0], Menon and RS6000 teach the nodes communicating with each other over the IBM SP2's message-passing network.

RS6000 further describes that each node has “five [] adapter slots” that can include “Ethernet,” “Token Ring,” “ATM”, “Wide Area Network (WAN)” and more. Ex.1007, 2.

Accordingly, Menon and Trefethen teach MultiMATLAB on an IBM SP2 in view of RS6000 teaching the IBM SP2 having at least a local area network (LAN) and wide area network (WAN), rendering obvious this limitation. Ex.1003, ¶¶371-372.

Claim 18

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

In limitation [1.3.1], [1.5.1], and [1.7.1], the MATLAB kernel in the MATLAB processes on the first non-interactive processor, the second non-interactive processor, and the third non-interactive processor of the MultiMATLAB architecture teaches a first kernel, a second kernel, and a third kernel. As discussed therein, Menon teaches that “each processor ... individually

runs a MATLAB process,” which includes MATLAB. Ex.1005, 3, FIG. 1 (showing each processor having a MATLAB process), FIG. 2 (showing a MATLAB process having MATLAB).

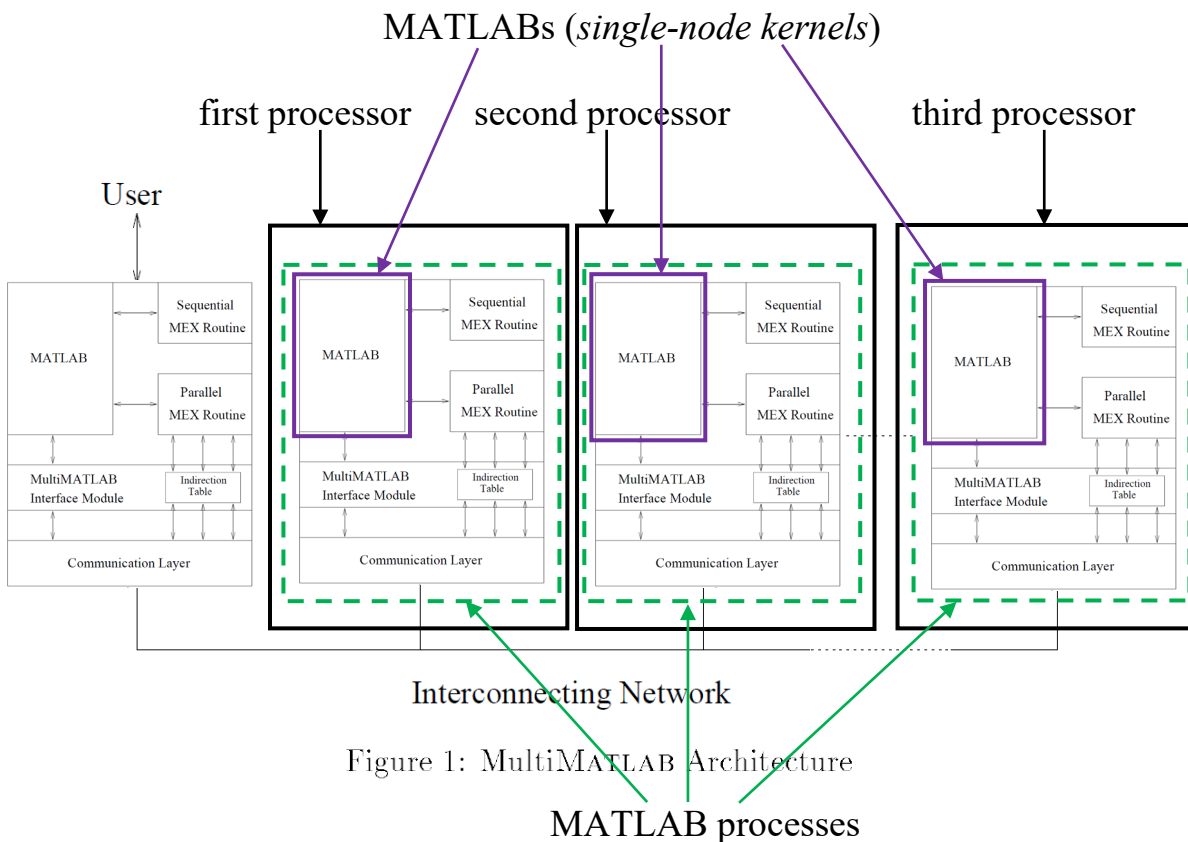


Figure 1: MultiMATLAB Architecture

Ex.1005, FIGs. 1 and 2 (combined and annotated); Ex.1003, ¶374

A POSITA would recognize and understand that, as shown in the combination figure of Figures 1 and 2 of Menon, MATLAB (the *first kernel*) in the MATLAB process on the first non-interactive processor executes only on the first non-interactive processor, MATLAB (the *second kernel*) in the MATLAB process on the second non-interactive processor executes only on the second non-

interactive processor, and MATLAB (the *third kernel*) in the MATLAB process on the third non-interactive processor executes only on the third non-interactive processor, rendering obvious this limitation. Ex.1003, ¶¶373-374.

Claim 19

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

As discussed in [1.4.1], Menon discloses each MATLAB process has a set of components (MEX routines, a MultiMATLAB interface module, indirection table and communication layer) (*cluster node modules*) to communicate with the other nodes.

Menon describes that MultiMATLAB is a “parallel platform” implemented on an IBM SP2 to execute “parallel programs on ... a networked cluster.” Ex.1005, 3, 6; Ex.1008, 1. That is, the MATLAB processes of MultiMATLAB executing commands in parallel are a group of processor nodes communicating with each other to accomplish a mathematical computation as if they were a single computer (*act as a cluster*), rendering obvious this limitation. Ex.1005, 6; Ex.1008, 1; Ex.1003, ¶¶375-382.

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

1. POEref

“Operation and Use, Volume 1, Using the Parallel Operating Environment”

(“POEref,” Ex.1008) is IBM product documentation for “the IBM Parallel Environment (PE)” and “its Parallel Operating Environment (POE).” Menon discusses using POE to initialize the MATLAB processes. Ex.1008, ix; Ex.1005, 6; Ex.1003, ¶134.

2. Combining POEref with Menon

a) Analogous art

POEref is analogous prior art pertaining to the same field of endeavor as the ’582 patent, namely, “the field of cluster computing” and is reasonably pertinent to the problem of the ’582 patent, namely, executing a single program on multiple nodes. Ex.1008, ix (“PE Software is designed to run on an...RS/6000 network cluster”); Ex.1001, 1:14-15; Ex.1003, ¶¶163-165.

b) Motivation to combine

A POSITA would have combined the teachings of POEref with Menon because Menon suggests the combination by teaching that MultiMATLAB uses “POE, IBM's Parallel Operating Environment.” Ex.1005, 6. A POSITA seeking to understand MultiMATLAB would be motivated to review the documentation corresponding to the systems expressly discussed in Menon. Ex.1005, 6.

Menon describes using POE to initialize the MATLAB processes. Ex.1005, 6. POEref describes that the “Parallel Operating Environment (POE)” provides the command “poe” “to load and execute programs on remote nodes” and “initializ[e]

the local environment.” Ex.1008, ix, 28.

Menon and POEref together represent a combination of prior art elements (Menon describing the MultiMATLAB architecture using IBM’s POE and POEref “describ[ing] ... [the] Parallel Operating Environment (POE)”) according to known methods (as taught by Menon) to yield the predictable result of an implementation of MultiMATLAB utilizing the Parallel Operating Environment. Ex.1005, 6; Ex.1008, ix, 28.

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 MultiMATLAB. Ex.1005, 6, 14 (“These results demonstrate that the MultiMATLAB system can be an effective platform for parallel computing”). *See* Ex.1003, ¶¶166-170.

Claim 4

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

The ’582 patent describes a cluster configuration module as “includ[ing] program code for initializing a plurality of cluster node modules to add cluster computing support to computer systems.” Ex.1001, 22:4-6. Likewise, Menon teaches that a user starts a MATLAB process on each processor of the

MultiMATLAB architecture by using IBM's POE. Ex.1005, 6 ("all MATLAB processes are started via POE, IBM's Parallel Operating Environment").

POEref describes that the initialization process is started using the "poe" command. Ex.1008, 28 ("When you invoke poe, the Partition Manager allocates processor nodes for each task and initializes the local environment"). The poe initialization "loads your program, and reproduces your local environment, on each processor node." Ex.1008, 28.

Accordingly, Menon teaches that a user starts a MATLAB process on each processor of the MultiMATLAB architecture by using IBM's POE, an execution environment that includes a program code for loading a program on each one of multiple processor nodes, as taught by POEref, and that each MATLAB process on each non-interactive processor is initialized, includes its MEX routines, a MultiMATLAB interface module, indirection table, and a communication layer (*cluster node module*), rendering obvious this limitation. Ex.1003, ¶¶392-397.

Claim 20

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

Menon teaches that a user starts a MATLAB process on each processor of the MultiMATLAB architecture by using IBM's POE. Ex.1005, 6 ("all MATLAB processes are started via POE, IBM's Parallel Operating Environment"). Menon

also teaches that the user interacts directly with the interactive processor. Ex.1005, 3.

POEref describes that the initialization process is started using the “poe” command. Ex.1008, 28 (“When you invoke poe, the Partition Manager allocates processor nodes for each task and initializes the local environment”). The poe initialization “loads your program, and reproduces your local environment, on each processor node.” Ex.1008, 28.

Menon describes that when using POE, “[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. All other processes wait for commands from the interactive MATLAB process.” “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.

Accordingly, Menon and POEref teach using IBM’s POE to start all the MATLAB processes on the processors of the MultiMATLAB architecture, where the cluster node module on each non-interactive processor is immediately run upon initialization of the system and awaits commands, rendering obvious this limitation. Ex.1003, ¶¶398-402.

XII. SOTERA STIPULATION

Petitioners each have filed the following stipulation at the district court:

Sotera Stipulation: Petitioner hereby stipulates that, if this proceeding is instituted, it will not pursue in the parallel district court case, any grounds for invalidating the '582 patent that was raised or could have reasonably been raised in this IPR. Ex.1042; Ex.1046.

XIII. CONCLUSION

Petitioners have established a reasonable likelihood that the challenged claims are unpatentable.

Respectfully submitted,

Dated: April 29, 2025
HAYNES AND BOONE, LLP
2801 N. Harwood Street, Suite 2300
Dallas, Texas 75201
Customer No. 27683

/Andrew S. Ehmke/
Andrew S. Ehmke
Lead Counsel for Petitioner
Registration No. 50,271

XIV. MANDATORY NOTICES

A. Real party-in-interest

Pursuant to 37 C.F.R. § 42.8(b)(1), Petitioners certify that the real parties-in-interest are Intel Corporation and Advanced Micro Devices, Inc.

B. Related matters

Pursuant to 37 C.F.R. § 42.8(b)(2), to the best knowledge of Petitioners, the '582 patent is or was involved in the following cases:

Case Caption	Number	Court	Filed	Status
<i>Advanced Cluster Systems, Inc. v. Intel Corporation</i>	7:24-cv-00245	W.D. Tex.	9/26/24	Pending
<i>Advanced Cluster Systems, Inc. v. Advanced Micro Devices, Inc.</i>	7:24-cv-00244	W.D. Tex.	9/26/24	Pending

To the best knowledge of Petitioners, patents related to the '582 patent are or were involved in the following cases:

<i>Advanced Cluster Systems, Inc. v. NVIDIA Corporation</i>	1:19-cv-02032	D. DE	10/28/19	Dismissed
<i>NVIDIA Corporation v. Advanced Cluster Systems, Inc.</i>	IPR2021-00019	PTAB	10/6/20	Institution Denied
<i>NVIDIA Corporation v. Advanced Cluster Systems, Inc.</i>	IPR2021-00020	PTAB	10/10/20	Institution Denied

<i>Intel Corp. v. Advanced Cluster Systems, Inc.</i>	IPR2025-00794	PTAB	3/28/25	Pending
<i>Intel Corp. v. Advanced Cluster Systems, Inc.</i>	IPR2025-00795	PTAB	3/28/25	Pending
<i>Advanced Micro Devices, Inc. v. Advanced Cluster Systems, Inc.</i>	IPR2025-00862	PTAB	4/16/25	Pending
<i>Advanced Micro Devices, Inc. v. Advanced Cluster Systems, Inc.</i>	IPR2025-00863	PTAB	4/16/25	Pending

C. Lead and back-up counsel and service information

Lead Counsel

Andrew S. Ehmke
HAYNES AND BOONE, LLP
2801 N. Harwood Street, Suite 2300
Dallas, TX 75201

Phone: 214-651-5116
Fax: 214-200-0853
andy.ehmke.ipr@haynesboone.com
USPTO Reg. No. 50,271

Back-up Counsel

David L. McCombs
HAYNES AND BOONE, LLP
2801 N. Harwood Street, Suite 2300
Dallas, TX 75201

Phone: 972-651-5533
Fax: 214-200-0853
david.mccombs.ipr@haynesboone.com
USPTO Reg. No. 32,271

Dan Smith
HAYNES AND BOONE, LLP
2801 N. Harwood Street, Suite 2300
Dallas, TX 75201

Phone: 972-739-8634
Fax: 214-200-0853
dan.smith.ipr@haynesboone.com
USPTO Reg. No. 71,278

Dagim Tilahun
HAYNES AND BOONE, LLP
888 16th Street NW, Suite 300
Washington, DC 20006-4103

Phone: 202-654-4582
Fax: 214-200-0853
dagim.tilahun.ipr@haynesboone.com
USPTO Reg. No. 75,722

Brian E. Ferguson
WINSTON & STRAWN LLP
1901 L Street, N.W.
Washington, D.C. 20036

Phone: 202-282-5200
beferguson@winston.com
AMD-IPR@winston.com
USPTO Reg. No. 36,801

Chaoxuan Charles Liu
WINSTON & STRAWN LLP
2121 North Pearl Street, Ste. 900
Dallas, TX 75201

Phone: 214-453-6500
USPTO Reg. No. 76,616
ccliu@winston.com

James Kappos
WINSTON & STRAWN LLP
2121 North Pearl Street, Ste. 900
Dallas, TX 75201

Phone: 214-453-6500
USPTO Reg. No. 78,587
jkappos@winston.com

Please address all correspondence in this proceeding to lead and back-up counsel. Petitioners consent to service in this proceeding by email at the addresses above.

CERTIFICATE OF WORD COUNT

Pursuant to 37 C.F.R. § 42.24(d), Petitioners hereby certify, in accordance with and reliance on the word count provided by the word-processing system used to prepare this Petition, that the number of words in this paper is 13,608. Pursuant to 37 C.F.R. § 42.24(d), this word count excludes the table of contents, table of authorities, mandatory notices under § 42.8, certificate of service, certificate of word count, appendix of exhibits, and any claim listing.

Dated: April 29, 2025

/Andrew S. Ehmke/

Andrew S. Ehmke

Lead Counsel

Registration No. 50,271

CERTIFICATE OF SERVICE

The undersigned certifies that, in accordance with 37 C.F.R. § 42.6(e) and 37 C.F.R. § 42.105, service was made on Patent Owner as detailed below.

Date of service April 29, 2025

Manner of service FEDERAL EXPRESS

Documents served Petition for *Inter Partes* Review Under 35 U.S.C. § 312 and 37 C.F.R. § 42.104 of U.S. 11,811,582;
Petitioners' Powers of Attorney;
Petitioners' Exhibit List;
Exhibits 1001-1049.

Persons served Knobbe, Martens, Olson & Bear, LLP
2040 Main Street, Fourteenth Floor
Irvine, CA 92614

/Andrew S. Ehmke/
Andrew S. Ehmke
Lead Counsel
Registration No. 50,271