

Exhibit D-8
to AMD's Invalidity Contentions

Invalidity Chart for U.S. Patent No. 11,811,582
MultiMATLAB System

The MultiMATLAB system (“MultiMATLAB”) was available no later than November 1997. *See, e.g.,* Menon et al., *MultiMATLAB: Integrating MATLAB with High-Performance Parallel Computing*, Proceedings of the 1997 ACM/IEEE conference on Supercomputing, November 1997. The ’582 patent is not entitled to a priority date earlier than at least January 23, 2023. MultiMATLAB, as well as each of the publications listed below relating to that system, are therefore prior art to the ’582 patent under at least pre-AIA 35 U.S.C. §§ 102(a), 102(b) and/or 103(a), or under AIA 35 U.S.C. §§ 102(a) and/or 103. MultiMATLAB, and the printed publications related to MultiMATLAB, anticipate and/or renders obvious, alone or in combination with other references, each of the Asserted Claims of the ’582 patent as described in the chart below and in the main invalidity contentions document to which this chart is annexed. In the chart below, each statement that MultiMATLAB discloses a claim element also is a contention that one or more printed publications regarding MultiMATLAB discloses that claim element. AMD hereby incorporates by reference all statements and reservations of rights in the main invalidity contentions document.

AMD also incorporates by reference and reserves the right to rely on the claim chart for MultiMATLAB served on Plaintiff by the defendant(s) in other cases, such as *Advanced Cluster Systems, Inc., v. Nvidia Corporation et al.*, No. 1:19-cv-2032-MN-CJB (D. Del.), including descriptions in that claim chart regarding the state of the prior art, reasons or motivations to combine prior art references, and explanations of why certain references were anticipated or obvious.

Evidence of the structure, ability, features, and functionality of MultiMATLAB, as well as availability and/or use, can be found in the following document(s) and thing(s):

- Trefethen et al., *MultiMATLAB: MATLAB on Multiple Processors*, Technical Report 96-239, Cornell Theory Center, 1996, available at <https://ecommons.cornell.edu/server/api/core/bitstreams/79478a13-e65a-4526-8926-ca72d3fa3940/content> (“Trefethen”).¹
- Menon et al., *MultiMATLAB: Integrating MATLAB with High-Performance Parallel Computing*, Proceedings of the 1997 ACM/IEEE conference on Supercomputing, November 1997 (“Menon”).
- *RS/6000 SP: Planning vol. 1, hardware and physical environment: Introducing the RS/6000 SP* (2001), at-
https://web.archive.org/web/20010726184202/http://www.rs6000.ibm.com/resource/aix_resource/sp_books/hardware/planguide1/da709ms/t07.html (“RS6000”).

¹ This Invalidity Chart cites to the section of this source, rather than the page number.

- *RS/6000 SP: Planning vol. 1, hardware and physical environment: 375 MHz POWER3 SMP High Node (F/C 2058)* (2001), at-
https://web.archive.org/web/20010726183314/http://www.rs6000.ibm.com/resource/aix_resource/sp_books/hardware/planguide1/da709mst08.html#HDRHIGH-PW3-2 ("RS6000-2").
- *RS/6000 SP: Planning vol. 1, hardware and physical environment* (2002) ("RS6000 2002").
- IBM, *IBM Parallel Environment for AIX: Operation and Use, Volume 1, Using the Parallel Operating Environment*, October 1998 ("POEref").
- MATLAB: The Language of Technical Computing – External Interfaces, Version 6 (2003) ("MATLAB External Interfaces").

Evidence of the structure, ability, features, and functionality of the MPI protocol, which was used as part of the MultiMATLAB system or would have been obvious to combine with MultiMATLAB, can be found in the following document(s) and thing(s):

- MPI: *A Message-Passing Interface Standard*, May 5, 1994 ("MPI Standard 1994")
- MPI: *A Message-Passing Interface Standard*, June 12, 1995 ("MPI Standard 1995")
- Gropp et al., *A high-performance, portable implementation of the MPI message passing interface standard*, published in *Parallel Computing*, Vol. 22, 1996 ("MPI Implementation").
- Cownie, Gropp, "A Standard Interface for Debugger Access to Message Queue Information in MPI," *EuroPVM/MPI 1999: Recent Advances in Parallel Virtual Machine and Message Passing Interface*, September 1999 ("Cownie")

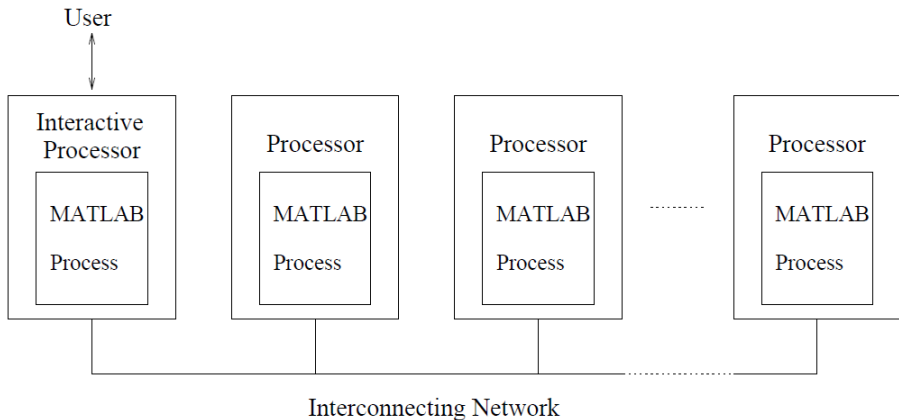
To the extent MultiMATLAB does not disclose one or more limitations of the claims, it would have been obvious to combine the teachings of MultiMATLAB with the knowledge of one of ordinary skill in the art and with one or more of the references below to render the claims at issue in the '768 patent invalid, including the Matlab*P system, for example, which describes "MultiMatlab-like" features in its own implementation:

- Long Yin Choy and Alan Edelman. *MATLAB*P 2.0: A unified parallel MATLAB*, Technical Report, Massachusetts Institute of Technology, January 2003, available at <http://libraries.mit.edu/dspace-mit/> ("Choy 1").
- Ron Choy and Alan Edelman, *Parallel MATLAB: Doing It Right*, *Proceedings of the IEEE*, Vol. 93, No. 2, February 2005, 331-341 ("Choy 2").
- R. Choy, *Parallel MATLAB survey*, 2001, available at: <http://theory.lcs.mit.edu/clv/survey.html> ("Choy 3").
- Long Yin Choy, *MATLAB*P 2.0: Interactive Supercomputing Made Practical*, Masters Thesis, Massachusetts Institute of Technology, 2002 ("Choy Thesis").
- *Interactive Supercomputing Star-P User Guide*, 2006 ("Star-P User Guide").
- U.S. Pat. No. 7,814,462 ("Husbands").

- MPI Specification, 1995.
- Cramer, Board, *"The Development and Integration of a Distributed 3D FFT for a Cluster of Workstations,"* Proceedings of the 4th Annual Linux Showcase and Conference, USENIX, 2000 ("Cramer")
- FFTW User's Manual, Version 2.1.5, March 2003 ("FFTW Manual")
- U.S. Patent Application Publication No. 2003/0195938, filed on January 10, 2003, issued on October 16, 2003 ("Howard")
- "FFT algorithms and their adaptation to parallel processing," by Chu et al., published in Linear Algebra and its Applications in 1998 ("Chu")
- Agarwal, et al., "A High Performance Parallel Algorithm for 1-D FFT, Supercomputing '94, 1994 ("Agarwal")
- "Maple V," PC Magazine, August 1992, pp. 419-425 ("PC Magazine 1").
- "Maple V Looks Better With Improved Graphics," PC Magazine, July 1993, p. 50 ("PC Magazine 2").
- U.S. Patent No. 7,162,590 ("Pruvost").
- MapleSoft Application Demo
- U.S. Pat. No. 7,694,158 ("Melpignano").
- Anshuman Nayak, et al., "A Library based compiler to execute MATLAB Programs on a Heterogeneous Platform" ("Nayak")
- Systems and publications discussed in Distributed Maple claim charts
- Systems and publications discussed in FFTW claim charts
- Systems and publications discussed in Intel Paragon Maple charts
- Systems and publications discussed in Matlab*P claim charts

Additional motivation to combine the MultiMATLAB system with the Matlab*P system and with the teachings of the references identified above exist for the reasons provided in the Matlab*P system and references claim charts, as well as described in the cover pleading.

'582 Claim	Prior Art Reference – MultiMATLAB
[1PRE] A computer cluster comprising:	To the extent that the preamble of claim 1 is considered a claim limitation, MultiMATLAB discloses it. See, e.g.: MultiMATLAB[1] is a general extension of the MATLAB environment to any distributed memory multiprocessors. This paper presents a new MultiMATLAB system designed to provide high-performance on multiprocessors while maintaining the functionality and usability of the MATLAB environment. This

'582 Claim	Prior Art Reference – MultiMATLAB
	system will enable users to access high-performance parallel routines from within the MATLAB environment, to extent the environment with new parallel routines, and to use these routines to develop parallel applications with the MATLAB language. We discuss a general MultiMATLAB architecture, present two implementations based upon the MPI communication standard [2], and demonstrate the use of this system. Menon at 1. The system should be easily portable onto any parallel platform that allows MATLAB to run on each processor. This includes any network of workstations supporting MATLAB. Since most modern fine-grain multiprocessors are built using commodity processors, it should also include most modern parallel platforms. <i>Id.</i> at 2.  The diagram illustrates the MultiMATLAB Architecture. At the top, a 'User' is connected via a double-headed arrow to an 'Interactive Processor'. This 'Interactive Processor' contains a 'MATLAB Process' box. To its right are three 'Processor' boxes, each containing a 'MATLAB Process' box. A horizontal line with a dotted segment in the middle connects the bottom of the 'Interactive Processor' to the bottom of the first 'Processor'. Another horizontal line with a dotted segment in the middle connects the bottom of the last 'Processor' to the bottom of the 'Interactive Processor'. Below these lines is the label 'Interconnecting Network'. Figure 1: MultiMATLAB Architecture <i>Id.</i> at Fig. 1; see discussion at Fig. 1. In the MultiMatlab architecture, illustrated in Figure 1, each processor in a parallel platform individually runs a Matlab process. Each process is then provided with the ability to communicate

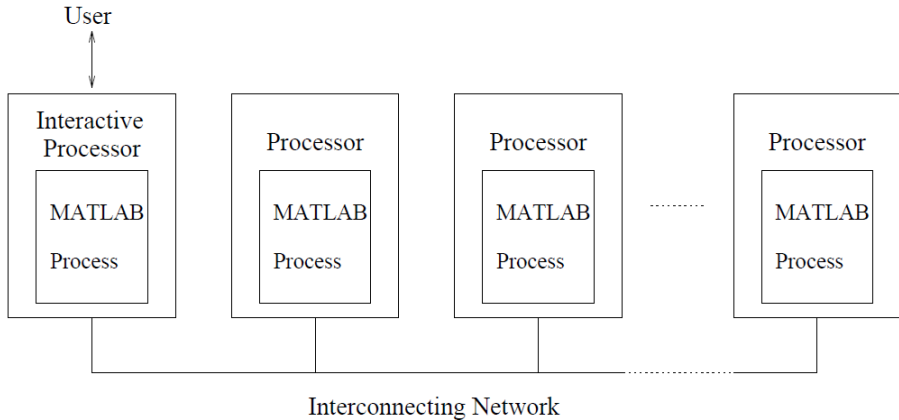
'582 Claim	Prior Art Reference – MultiMATLAB
	with other processes through a communication layer that runs over the parallel platform's interconnection network. The user interacts directly with one Matlab process, called the interactive process, and operates within that process's Matlab environment. Other Matlab processes await commands from the interactive process. These other processes are used either by explicitly sending Matlab commands to them or by executing routines that, in turn, use them. <i>Id.</i> at 3. We present two implementations of the MultiMATLAB architecture. Both implementations use MPI as their underlying communication substrate. . . . One implementation of MultiMATLAB, designed to run on a generic network of Unix workstations, is based upon MPICH [5], a popular public domain version of MPI developed at Argonne National Laboratory and Mississippi State University. The second was designed for the IBM SP2, a modern high performance distributed memory multiprocessor. This implementation is based upon MPI-F [6], IBM's proprietary version of MPI specifically optimized for the SP2. <i>Id.</i> at 5. The MPICH implementation of MultiMATLAB is based upon the P4 [7] communication subsystem, allowing it to run over a network of workstations connected by TCP/IP. In particular, this implementation should run on any platform providing both MATLAB and MPICH. To date, we have successfully run it on IBM RS6000 and Sun Sparc workstations and the IBM SP2. <i>Id.</i> at 5. The MPI-F implementation of MultiMATLAB was design specifically for the IBM SP2. MPI-F is a highly optimized implementation of the MPI standard running over the SP2's high performance switch. This implementation permits access to the switch in user space. In this implementation, all MATLAB processes are started via POE, IBM's Parallel Operating Environment. For each of p MATLAB processes, POE assigns an identification number between 0 and p-1. The process numbered 0 is considered the interactive MATLAB. All other processes wait for commands from the interactive MATLAB process.

'582 Claim	Prior Art Reference – MultiMATLAB
	This implementation was designed to demonstrate the performance potential of the MultiMATLAB system. In particular, it provides a platform in which parallel libraries or applications written in MPI can easily be integrated into MATLAB on the SP2. <i>Id.</i> at 6. In this section, we describe different MultiMATLAB implementations of the conjugate gradients algorithm on the IBM SP2 at the Cornell Theory Center. <i>Id.</i> at 12. Figures 7 and 8 compare the performance of three different MultiMatlab implementations with a stand-alone C implementation using MPI-F on up to 32 processors of the IBM SP2. <i>Id.</i> at 13; Figs. 7 and 8. MATLAB®, a commercial product of The MathWorks, Inc., has become one of the principal languages of desktop scientific computing. A system is described that enables one to run MATLAB conveniently on multiple processors. Using short, MATLAB-style commands like Eval, Send, Recv, Bcast, Min, and Sum, the user operating within one MATLAB session can start MATLAB processes on other machines and then pass commands and data between between these various processes in a fashion that maintains MATLAB's traditional user-friendliness. Multi-processor graphics is also supported. The system currently runs under MPICH on an IBM SP2 or a network of Unix workstations, and extensions are planned to networks of PCs. MultiMATLAB is potentially useful for education in parallel programming, for prototyping parallel algorithms, and for fast and convenient execution of easily parallelizable numerical computations on multiple processors. Trefethen at Abstract. The vision of the MultiMATLAB project has been to bridge this gap. Think of a user who finds him- or herself computing happily in MATLAB, but frustrated by the time it takes to rerun the program for six different boundary conditions, or a dozen different parameter choices, or a hundred different initial guesses. Such a user's problems might be solved by a system that makes it convenient to spawn MATLAB processes on multiple processors of a parallel computer or a network of workstations or PCs. In many cases the needs for communication between the processors are rather

'582 Claim	Prior Art Reference – MultiMATLAB
	small. Convenience of spreading the problem across machines and collecting the results numerically or graphically is paramount. <i>Id.</i> at 1.2. Suppose the first author is sitting at her workstation in the Theory Center, connected to a node of the IBM SP2, running MATLAB. <i>Id.</i> at 2.1. Our own first experiments were carried out in 1993 (A. E. Trefethen). By making use of a Fortran wrapper based on IBM's message passing environment (MPL), we ran MATLAB on multiple nodes of an IBM SP-1. <i>Id.</i> at 5. MultiMATLAB can be summarized in a few words. We run MATLAB processes on multiple processors, with full access to all the usual capabilities such as Toolboxes. These processes communicate via simple MATLAB-style commands built on MPI, with all message-passing details hidden as far as possible from the user. Both master/slave and SPMD paradigms are implemented, and attention is paid to multiprocessor graphics. All of this happens without any changes in the MATLAB architecture; indeed, we have not had access to the MATLAB source code. It is a straightforward matter to install our current software on any network of Unix workstations or SP2 system, provided that all the nodes are licensed to run MATLAB and there is a shared file system. We expect that extensions to networks of PCs running Windows, based on appropriate implementations of MPI, are not far behind. <i>Id.</i> at 6. Essentially, an SP system is a collection of RS/6000 processors grouped into a number of frames. Each frame of an SP system can contain from two to 16 RS/6000 processors. POERef at 1.

'582 Claim	Prior Art Reference – MultiMATLAB
	The PE is a distributed memory message passing system. It runs on the RS/6000 platform using the AIX operating system (Version 4.2.1). Specifically, you can use the PE to execute parallel programs on: • any configuration of the IBM RS/6000 SP as described in the <i>IBM Parallel System Support Programs for AIX: Installation and Migration Guide</i> Essentially, an SP system is a collection of RS/6000 processors grouped into a number of <i>frames</i>. Each frame of an SP system can contain from two to 16 RS/6000 processors. • a networked cluster of RS/6000 processors, including a single processor or a single workstation. • a <i>mixed system</i>. In a mixed system, additional RS/6000 processors supplement the processors of an SP system. POEref at 1. The IBM RS/6000 SP is IBM's family of scalable, parallel computing solutions. It provides a state-of-the-art parallel computing system and industry-leading application enablers and applications. The RS/6000 SP runs the AIX operating system along with the Parallel System Support Programs (PSSP) system software on the control workstation and all processor nodes. For complete information on SP system software issues, see IBM RS/6000 SP: Planning Volume 2, Control Workstation and Software Environment. The scalable architecture of the SP system, its high-performance communication, POWER3, POWER2, and PowerPC Architecture give you the power to handle data-intensive, compute-intensive and I/O-intensive jobs with ease. You can execute both serial and parallel applications simultaneously, while managing your system from a single workstation. RS6000 at 1. The basic components of the RS/6000 SP system are: *Processor nodes (includes SP-attached servers) * Frames with integral power subsystems * Switches * Extension nodes * Control workstations (a high availability option is available) * Network connectivity adapters* External disk drives

'582 Claim	Prior Art Reference – MultiMATLAB
	The IBM RS/6000 SP System is scalable from one to 128 processor nodes. Up to sixteen thin, eight wide, or four high processor nodes can be mounted in a tall frame while a short frame can hold up to eight thin or four wide nodes. RS6000 at 2. Alternatively, to the extent that the preamble of claim 1 is considered a claim limitation, it is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the '582 patent as charted for this claim limitation, and/or MultiMATLAB combined with the knowledge of one of ordinary skill in the art. Motivations to combine may come from the knowledge of the person of ordinary skill, or from the known problems and predictable solutions as embodied in these references. For example, to the extent any one of the MultiMATLAB printed publications lacks this element, it would have been obvious to combine with another of the MultiMATLAB printed publications that discloses this element, because they expressly cite to each other and/or to the same system, were published by overlapping research teams and/or in similar locations, and represent combinations of prior art elements according to known methods to yield predictable results. Further motivation to combine references and additional details may be found in the invalidity contentions cover pleading.
[1A] a first processor; a second processor; a third processor;	MultiMATLAB discloses this claim limitation. See, e.g.: MultiMATLAB[1] is a general extension of the MATLAB environment to any distributed memory multiprocessors. This paper presents a new MultiMATLAB system designed to provide high-performance on multiprocessors while maintaining the functionality and usability of the MATLAB environment. This system will enable users to access high-performance parallel routines from within the MATLAB environment, to extent the environment with new parallel routines, and to use these routines to develop parallel applications with the MATLAB language. We discuss a general MultiMATLAB architecture, present two implementations based upon the MPI communication standard [2], and demonstrate the use of this system. Menon at 1. The system should be easily portable onto any parallel platform that allows MATLAB to run on each processor. This includes any network of workstations supporting MATLAB. Since most

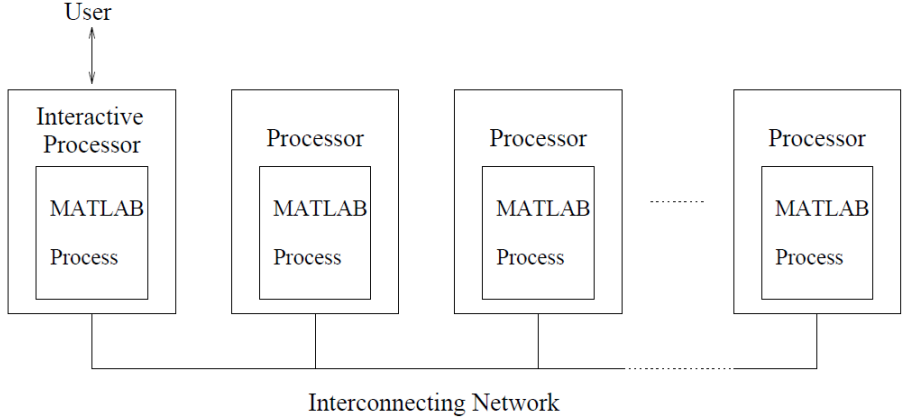
'582 Claim	Prior Art Reference – MultiMATLAB
	modern fine-grain multiprocessors are built using commodity processors, it should also include most modern parallel platforms. <i>Id.</i> at 2.  The diagram illustrates the MultiMATLAB architecture. At the top, a 'User' is connected via a double-headed arrow to an 'Interactive Processor'. This processor contains a 'MATLAB Process'. Below it, there are three 'Processor' blocks, each containing a 'MATLAB Process'. These processors are connected to a common 'Interconnecting Network' at the bottom. Dotted lines indicate that there can be more than three processors in the system. Figure 1: MultiMATLAB Architecture <i>Id.</i> at Fig. 1; see discussion at Fig. 1. In the MultiMatlab architecture, illustrated in Figure 1, each processor in a parallel platform individually runs a Matlab process. Each process is then provided with the ability to communicate with other processes through a communication layer that runs over the parallel platform's interconnection network. The user interacts directly with one Matlab process, called the interactive process, and operates within that process's Matlab environment. Other Matlab processes await commands from the interactive process. These other processes are used either by explicitly sending Matlab commands to them or by executing routines that, in turn, use them. <i>Id.</i> at 3. We present two implementations of the MultiMATLAB architecture. Both implementations use MPI as their underlying communication substrate. . . . One implementation of MultiMATLAB, designed

'582 Claim	Prior Art Reference – MultiMATLAB
	to run on a generic network of Unix workstations, is based upon MPICH [5], a popular public domain version of MPI developed at Argonne National Laboratory and Mississippi State University. The second was designed for the IBM SP2, a modern high performance distributed memory multiprocessor. This implementation is based upon MPI-F [6], IBM's proprietary version of MPI specifically optimized for the SP2. <i>Id.</i> at 5. The MPICH implementation of MultiMATLAB is based upon the P4 [7] communication subsystem, allowing it to run over a network of workstations connected by TCP/IP. In particular, this implementation should run on any platform providing both MATLAB and MPICH. To date, we have successfully run it on IBM RS6000 and Sun Sparc workstations and the IBM SP2. <i>Id.</i> at 5. The MPI-F implementation of MultiMATLAB was design specifically for the IBM SP2. MPI-F is a highly optimized implementation of the MPI standard running over the SP2's high performance switch. This implementation permits access to the switch in user space. In this implementation, all MATLAB processes are started via POE, IBM's Parallel Operating Environment. For each of p MATLAB processes, POE assigns an identification number between 0 and p-1. The process numbered 0 is considered the interactive MATLAB. All other processes wait for commands from the interactive MATLAB process. This implementation was designed to demonstrate the performance potential of the MultiMATLAB system. In particular, it provides a platform in which parallel libraries or applications written in MPI can easily be integrated into MATLAB on the SP2. <i>Id.</i> at 6. In this section, we describe different MultiMATLAB implementations of the conjugate gradients algorithm on the IBM SP2 at the Cornell Theory Center.

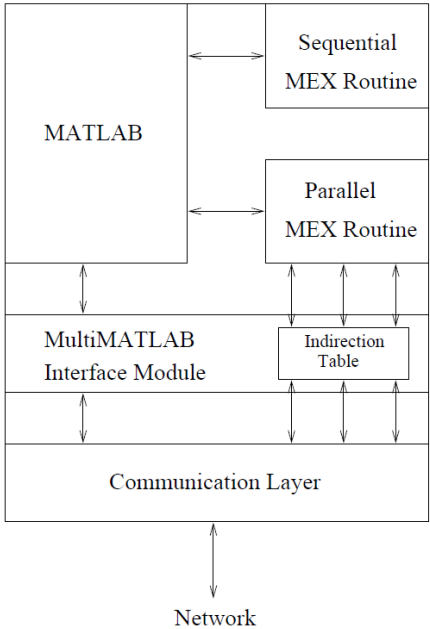
'582 Claim	Prior Art Reference – MultiMATLAB
	<i>Id.</i> at 12. Figures 7 and 8 compare the performance of three different MultiMatlab implementations with a stand-alone C implementation using MPI-F on up to 32 processors of the IBM SP2. <i>Id.</i> at 13; Figs. 7 and 8. MATLAB®, a commercial product of The MathWorks, Inc., has become one of the principal languages of desktop scientific computing. A system is described that enables one to run MATLAB conveniently on multiple processors. Using short, MATLAB-style commands like Eval, Send, Recv, Bcast, Min, and Sum, the user operating within one MATLAB session can start MATLAB processes on other machines and then pass commands and data between between these various processes in a fashion that maintains MATLAB's traditional user-friendliness. Multi-processor graphics is also supported. The system currently runs under MPICH on an IBM SP2 or a network of Unix workstations, and extensions are planned to networks of PCs. MultiMATLAB is potentially useful for education in parallel programming, for prototyping parallel algorithms, and for fast and convenient execution of easily parallelizable numerical computations on multiple processors. Trefethen at Abstract. The vision of the MultiMATLAB project has been to bridge this gap. Think of a user who finds him- or herself computing happily in MATLAB, but frustrated by the time it takes to rerun the program for six different boundary conditions, or a dozen different parameter choices, or a hundred different initial guesses. Such a user's problems might be solved by a system that makes it convenient to spawn MATLAB processes on multiple processors of a parallel computer or a network of workstations or PCs. In many cases the needs for communication between the processors are rather small. Convenience of spreading the problem across machines and collecting the results numerically or graphically is paramount. <i>Id.</i> at 1.2. Suppose the first author is sitting at her workstation in the Theory Center, connected to a node of the IBM SP2, running MATLAB.

'582 Claim	Prior Art Reference – MultiMATLAB
	<i>Id.</i> at 2.1. Our own first experiments were carried out in 1993 (A. E. Trefethen). By making use of a Fortran wrapper based on IBM's message passing environment (MPL), we ran MATLAB on multiple nodes of an IBM SP-1. <i>Id.</i> at 5. MultiMATLAB can be summarized in a few words. We run MATLAB processes on multiple processors, with full access to all the usual capabilities such as Toolboxes. These processes communicate via simple MATLAB-style commands built on MPI, with all message-passing details hidden as far as possible from the user. Both master/slave and SPMD paradigms are implemented, and attention is paid to multiprocessor graphics. All of this happens without any changes in the MATLAB architecture; indeed, we have not had access to the MATLAB source code. It is a straightforward matter to install our current software on any network of Unix workstations or SP2 system, provided that all the nodes are licensed to run MATLAB and there is a shared file system. We expect that extensions to networks of PCs running Windows, based on appropriate implementations of MPI, are not far behind. <i>Id.</i> at 6. Essentially, an SP system is a collection of RS/6000 processors grouped into a number of frames. Each frame of an SP system can contain from two to 16 RS/6000 processors. POERef at 1. The IBM RS/6000 SP is IBM's family of scalable, parallel computing solutions. It provides a state-of-the-art parallel computing system and industry-leading application enablers and applications. The RS/6000 SP runs the AIX operating system along with the Parallel System Support Programs (PSSP) system software on the control workstation and all processor nodes. For complete information on SP system software issues, see IBM RS/6000 SP: Planning Volume 2, Control Workstation and Software Environment.

'582 Claim	Prior Art Reference – MultiMATLAB
	The scalable architecture of the SP system, its high-performance communication, POWER3, POWER2, and PowerPC Architecture give you the power to handle data-intensive, compute-intensive and I/O-intensive jobs with ease. You can execute both serial and parallel applications simultaneously, while managing your system from a single workstation. RS6000 at 1; <i>see also</i> RS6000-2 at 1-3. The basic components of the RS/6000 SP system are: *Processor nodes (includes SP-attached servers) * Frames with integral power subsystems * Switches * Extension nodes * Control workstations (a high availability option is available) * Network connectivity adapters* External disk drives The IBM RS/6000 SP System is scalable from one to 128 processor nodes. Up to sixteen thin, eight wide, or four high processor nodes can be mounted in a tall frame while a short frame can hold up to eight thin or four wide nodes. RS6000 at 2. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the '582 patent as charted for this claim limitation, and/or MultiMATLAB combined with the knowledge of one of ordinary skill in the art. Motivations to combine may come from the knowledge of the person of ordinary skill, or from the known problems and predictable solutions as embodied in these references. For example, to the extent any one of the MultiMATLAB printed publications lacks this element, it would have been obvious to combine with another of the MultiMATLAB printed publications that discloses this element, because they expressly cite to each other and/or to the same system, were published by overlapping research teams and/or in similar locations, and represent combinations of prior art elements according to known methods to yield predictable results. Further motivation to combine references and additional details may be found in the invalidity contentions cover pleading.
[1B] at least one computer-readable medium in communication with	MultiMATLAB discloses this claim limitation. See, e.g.:

'582 Claim	Prior Art Reference – MultiMATLAB
at least one of the first processor, the second processor, or the third processor; a first kernel residing in the at least one computer-readable medium, said first kernel configured to translate user commands into calls for execution of matrix operations on an array of data elements;	In particular, MATLAB offers several features that simplify numerical computing. First, it provides users with easy access to an extensive library of high quality numerical routines. Second, it is an intuitive, interactive environment that enables users to use these routines in a dynamic manner well suited to scientific problem solving. Third, it provides a high-level matrix based language that permits users to express computations in an exceptionally concise manner via matrix or vector formulations and often with a fraction of the code required in conventional languages such as C or Fortran. Fourth, MATLAB's graphical and visualization tools offer a simple but powerful mechanism to manipulate and analyze data more effectively. Menon at 1.  The diagram illustrates the MultiMATLAB Architecture. At the top, a 'User' is shown with a double-headed arrow pointing down to an 'Interactive Processor'. This processor contains a 'MATLAB Process'. Below it, a horizontal line represents the 'Interconnecting Network'. Connected to this network are several 'Processor' blocks, each containing a 'MATLAB Process'. The first processor is connected directly, followed by a dotted line, and then the last processor. All processors are connected to the same network line. Figure 1: MultiMATLAB Architecture <i>Id.</i> at Fig. 1; see discussion at Fig. 1. In the MultiMatlab architecture, illustrated in Figure 1, each processor in a parallel platform individually runs a Matlab process. Each process is then provided with the ability to communicate with other processes through a communication layer that runs over the parallel platform's interconnection network. The user interacts directly with one Matlab process, called the interactive process, and operates within that process's Matlab environment. Other Matlab processes await

'582 Claim	Prior Art Reference – MultiMATLAB
	commands from the interactive process. These other processes are used either by explicitly sending Matlab commands to them or by executing routines that, in turn, use them. The key component in this architecture is the MultiMatlab interface module. The interface module, shown in Figure 2, is responsible for initializing an underlying communication layer, such as MPI [2], PVM [3], BLACS [4], or any other package available on the platform, and exposing it to the rest of the system. MEX routines, also shown in the Figure 2, are executables originally written in C or Fortran that may be run directly from Matlab. All parallel functionality in the MultiMatlab system is provided through MEX routines. As a simple example, the MultiMatlab system provides users with an Eval routine for parallel evaluation. This routine allows users to execute commands on the other processes. On the interactive process, the Eval routine is implemented as a MEX routine which accepts a vector of Matlab process IDs and a Matlab command. This routine sends the commands over the underlying communication layer to the processes corresponding to the specified IDs. On non-interactive processes, a separate top-level MEX routine is immediately run upon initialization of the system. This MEX routine runs in a loop in which it waits for Matlab commands to arrive from the interactive process and executes them in its own Matlab environment. All parallel MEX routines access the underlying communication layer and, hence, the network, through the MultiMatlab interface module. It is important to note that the interface module exists in the corresponding Matlab process's address space. For this reason, the interface module does not impose any additional context switch when parallel MEX routines access the communication layer. This is of particular importance when the parallel platform provides access to the network in user space. This enables parallel applications to access the network without operating system intervention as is often required for best performance. By residing within the Matlab process's address space, the MultiMatlab interface module is able to preserve this property for parallel MEX routines. <i>Id.</i> at 3-4.

'582 Claim	Prior Art Reference – MultiMATLAB
	 Figure 2: MATLAB Process Address Space <i>Id.</i> at Fig. 2; see discussion at Fig. 2. The MultiMatlab interface module provides MEX routines access to the underlying communication layer via an indirection table. Upon initialization, the interface module builds a table of pointers to all functions and data in the underlying communication layer that need to be exposed to parallel MEX routines. <i>Id.</i> at 5. In this implementation, the interactive MATLAB process is started by the user as a normal MATLAB session. The user is provided with a Start command that initializes the MultiMATLAB

'582 Claim	Prior Art Reference – MultiMATLAB
	system. This command builds a P4 process group file of remote hosts, which are either explicitly specified by the user or taken from a default list, and the initializes MPICH. <i>Id.</i> at 5. The MPI-F implementation of MultiMATLAB was design specifically for the IBM SP2. MPI-F is a highly optimized implementation of the MPI standard running over the SP2's high performance switch. This implementation permits access to the switch in user space. In this implementation, all MATLAB processes are started via POE, IBM's Parallel Operating Environment. For each of p MATLAB processes, POE assigns an identification number between 0 and p-1. The process numbered 0 is considered the interactive MATLAB. All other processes wait for commands from the interactive MATLAB process. This implementation was designed to demonstrate the performance potential of the MultiMATLAB system. In particular, it provides a platform in which parallel libraries or applications written in MPI can easily be integrated into MATLAB on the SP2. <i>Id.</i> at 6.

'582 Claim	Prior Art Reference – MultiMATLAB
	<pre> r = b; rho = r'*r; k = 0; while((k < kmax)) k = k+1; if (k == 1) p = r; else beta = rho/oldrho; p = r + beta*p; end w = A * p; alpha = rho/(p'*w); x = x + alpha * p; r = r - alpha * w; oldrho = rho; rho = r'*r; end </pre> <pre> r_local = b_local; rho = Sum(r_local'*r_local); k = 0; while((k < kmax)) k = k+1; if (k == 1) p_local = r_local; else beta = rho/oldrho; p_local = r_local + beta*p_local; end p = Gather(p_local); w_local = A_local * p; alpha = rho/(Sum(p_local'*w_local)); x_local = x_local + alpha * p_local; r_local = r_local - alpha * w_local; oldrho = rho; rho = Sum(r_local'*r_local); end </pre> a. Sequential MATLAB b. Message Passing MATLAB Figure 4: Conjugate Gradients In this section, we describe different MultiMATLAB implementations of the conjugate gradients algorithm on the IBM SP2 at the Cornell Theory Center. The conjugate gradients algorithm is iterative method to solve a linear system of the form $Ax = b$, where A is a symmetric positive definite matrix and x and b are vectors. The computational core of this method is a single matrix-vector multiplication at each iteration. The sequential MATLAB code, adapted from [13], is shown in Figure 4a. ... Figure 4b illustrates a parallel MATLAB implementation of conjugate gradients that uses the SPMD message passing described in Section 5.2. In this case, the matrix A and the different vectors are each distributed by row over all processes as in Figure 6. In each iteration, a process only does the computation required for its local data. However, communication is required for two different operations: the dot-products needed to compute ρ and α and the matrix-vector multiply needed

'582 Claim	Prior Art Reference – MultiMATLAB
	to compute <code>w_local</code>. The dot products only require communication of a single value over all processes. On the other hand, the matrix-vector multiplication needed for <code>w_local</code> requires the global vector <code>p</code>, and, thus, more expensive communication. <i>Id.</i> at 11-13. Figures 7 and 8 compare the performance of three different MultiMatlab implementations with a stand-alone C implementation using MPI-F on up to 32 processors of the IBM SP2. <i>Id.</i> at 13. MATLAB®, a commercial product of The MathWorks, Inc., has become one of the principal languages of desktop scientific computing. A system is described that enables one to run MATLAB conveniently on multiple processors. Using short, MATLAB-style commands like Eval, Send, Recv, Bcast, Min, and Sum, the user operating within one MATLAB session can start MATLAB processes on other machines and then pass commands and data between between these various processes in a fashion that maintains MATLAB's traditional user-friendliness. Multi-processor graphics is also supported. The system currently runs under MPICH on an IBM SP2 or a network of Unix workstations, and extensions are planned to networks of PCs. MultiMATLAB is potentially useful for education in parallel programming, for prototyping parallel algorithms, and for fast and convenient execution of easily parallelizable numerical computations on multiple processors. Trefethen at Abstract. MATLAB® is a high-level language, and a problem-solving environment, for mathematical and scientific calculations. It originated in the late 1970s with an attempt by Cleve Moler to provide interactive access to the Fortran linear algebra software packages EISPACK and LINPACK. Soon a programming language emerged (programs conventionally have the extension <code>.m</code> and are called "m-files") containing dozens of high-level commands such as <code>svd</code> (singular value decomposition), <code>fft</code> (fast Fourier transform), and <code>roots</code> (polynomial zero-finding). Graphical commands were built into the language, and a company called The MathWorks, Inc. was formed in 1984 by Moler and John Little, now based in Natick, Massachusetts. <i>Id.</i> at 1.1.

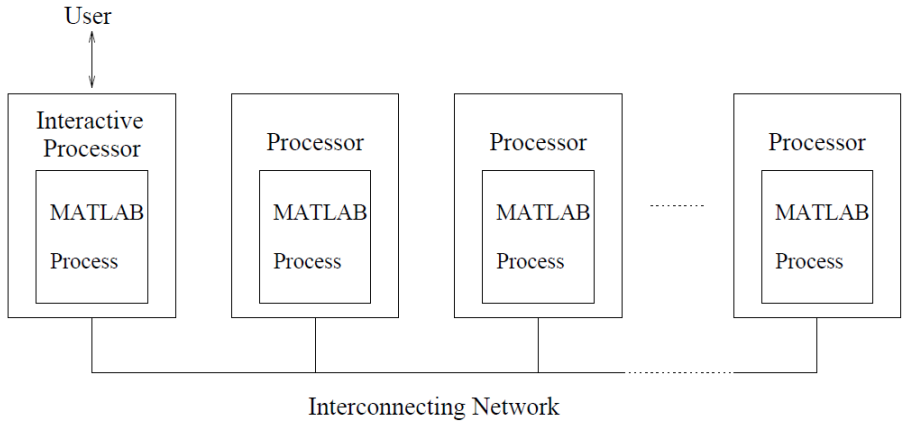
'582 Claim	Prior Art Reference – MultiMATLAB
	The fifth is MATLAB's easy extensibility via packages of m-files known as Toolboxes. Many Toolboxes have been produced over the years, both by The MathWorks and by others, covering application areas such as optimization, signal processing, fuzzy logic, partial differential equations, and mathematical finance. Finally, perhaps the most interesting reason for MATLAB's success may be that from the beginning, the whole language has been built around real or complex vectors and matrices (including sparse matrices) as the fundamental data type. <i>Id.</i> The vision of the MultiMATLAB project has been to bridge this gap. Think of a user who finds him- or herself computing happily in MATLAB, but frustrated by the time it takes to rerun the program for six different boundary conditions, or a dozen different parameter choices, or a hundred different initial guesses. Such a user's problems might be solved by a system that makes it convenient to spawn MATLAB processes on multiple processors of a parallel computer or a network of workstations or PCs. In many cases the needs for communication between the processors are rather small. Convenience of spreading the problem across machines and collecting the results numerically or graphically is paramount. <i>Id.</i> at 1.2. The standard MultiMATLAB command for executing commands on one or more processors is Eval. If the user now types Eval('sqrt(2)') then the MATLAB command sqrt(2) is executed on all six processors. The result is six repetitions of 1.4142. <i>Id.</i>

'582 Claim	Prior Art Reference – MultiMATLAB
	<pre> 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; </pre> Process 0 creates the variable <code>a</code> with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of <code>a</code>, and sends it along to process 2; and so on. If there are six processors the command <code>Eval('cycle')</code> produces the output <pre> a = 1 a = 2 a = 4 a = 8 a = 16 a = 32 </pre> The processes run asynchronously, but since each <code>Send</code> command is only executed after the corresponding <code>Recv</code> has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning. <i>Id.</i> at 2.3. We can do this by taking a data-parallel approach in a simplistic fashion. We have developed a number of routines such as <code>Distribute</code> and <code>Collect</code> that allow a user to distribute a matrix or to collect a set of matrices into one large matrix. These functions operate using a mask that indicates which processors hold which portions of the matrix. This allows us also to develop routines such as <code>Shift</code> and <code>Copy</code> that are useful in data-parallel computing, keeping the communication to a more abstract level. <i>Id.</i> at 2.5. The MultiMATLAB <code>Start</code> command builds a P4 process group file of remote hosts, which are either explicitly specified by the user or taken from a default list, and then initializes MPICH. MATLAB processes are then started on the remote hosts. Each process iterates over a simple loop, waiting for and executing commands received from the user's interactive MATLAB process. <i>Id.</i> at 4.

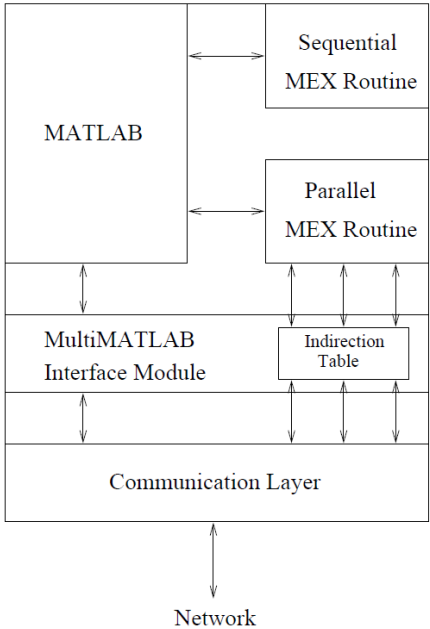
'582 Claim	Prior Art Reference – MultiMATLAB
	The IBM RS/6000 SP is IBM's family of scalable, parallel computing solutions. It provides a state-of-the-art parallel computing system and industry-leading application enablers and applications. The RS/6000 SP runs the AIX operating system along with the Parallel System Support Programs (PSSP) system software on the control workstation and all processor nodes. For complete information on SP system software issues, see IBM RS/6000 SP: Planning Volume 2, Control Workstation and Software Environment. The scalable architecture of the SP system, its high-performance communication, POWER3, POWER2, and PowerPC Architecture give you the power to handle data-intensive, compute-intensive and I/O-intensive jobs with ease. You can execute both serial and parallel applications simultaneously, while managing your system from a single workstation. RS6000 at 1. The basic components of the RS/6000 SP system are: *Processor nodes (includes SP-attached servers) * Frames with integral power subsystems * Switches * Extension nodes * Control workstations (a high availability option is available) * Network connectivity adapters* External disk drives The IBM RS/6000 SP System is scalable from one to 128 processor nodes. Up to sixteen thin, eight wide, or four high processor nodes can be mounted in a tall frame while a short frame can hold up to eight thin or four wide nodes. RS6000 at 2. Hard disk drives for the SP system can be either of two types as follows: Internal (contained within the node) External (mounted separately, outside of the node) RS6000 at 6; <i>see also</i> RS6000-2 at 3.

'582 Claim	Prior Art Reference – MultiMATLAB
	MATLAB [?] is one of the most widely used tools in scientific and technical computing. It started in 1970s as an interactive interface to EISPACK [?] and LINPACK [?], a set of eigenvalue and linear system solution routines. It has since grown to a feature rich product utilizing modern numerical libraries like ATLAS [?] and FFTW [?], and with toolboxes in a number of application areas, for example, financial mathematics, neural networks, and control theory. It has a built-in interpreted language that is similar to C and HPF, and the flexible matrix indexing makes it very suitable for programming matrix problems. It is a very useful tool to a wide audience. For example, it comes in very handy as a matrix calculator for simple problems, because of its easy-to-use command syntax. Its strong graphical capabilities makes it a good data analysis tool. Also researchers have been known to build very complex systems using MATLAB scripts. Choy 1 at 1. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the '582 patent as charted for this claim limitation, and/or MultiMATLAB combined with the knowledge of one of ordinary skill in the art. Motivations to combine may come from the knowledge of the person of ordinary skill, or from the known problems and predictable solutions as embodied in these references. For example, to the extent any one of the MultiMATLAB printed publications lacks this element, it would have been obvious to combine with another of the MultiMATLAB printed publications that discloses this element, because they expressly cite to each other and/or to the same system, were published by overlapping research teams and/or in similar locations, and represent combinations of prior art elements according to known methods to yield predictable results. Further motivation to combine references and additional details may be found in the invalidity contentions cover pleading.

'582 Claim	Prior Art Reference – MultiMATLAB
[1C] a first cluster node module residing in the at least one computer-readable medium, said first cluster node module configured to communicate with the first kernel and receive commands from a user interface;	MultiMATLAB discloses this claim limitation. See, e.g.: In particular, MATLAB offers several features that simplify numerical computing. First, it provides users with easy access to an extensive library of high quality numerical routines. Second, it is an intuitive, interactive environment that enables users to use these routines in a dynamic manner well suited to scientific problem solving. Third, it provides a high-level matrix based language that permits users to express computations in an exceptionally concise manner via matrix or vector formulations and often with a fraction of the code required in conventional languages such as C or Fortran. Fourth, MATLAB's graphical and visualization tools offer a simple but powerful mechanism to manipulate and analyze data more effectively. Menon at 1. The MultiMATLAB architecture now provides direct access from MATLAB to vendor-optimized communication libraries. <i>Id.</i> at 2. Users are provided interactive access to all processors in a multiprocessor within the MATLAB environment. This permits steering of parallel computation. <i>Id.</i> at 2.

'582 Claim	Prior Art Reference – MultiMATLAB
	 Figure 1: MultiMATLAB Architecture <i>Id.</i> at Fig. 1; see discussion at Fig. 1. In the MultiMatlab architecture, illustrated in Figure 1, each processor in a parallel platform individually runs a Matlab process. Each process is then provided with the ability to communicate with other processes through a communication layer that runs over the parallel platform's interconnection network. The user interacts directly with one Matlab process, called the interactive process, and operates within that process's Matlab environment. Other Matlab processes await commands from the interactive process. These other processes are used either by explicitly sending Matlab commands to them or by executing routines that, in turn, use them. The key component in this architecture is the MultiMatlab interface module. The interface module, shown in Figure 2, is responsible for initializing an underlying communication layer, such as MPI [2], PVM [3], BLACS [4], or any other package available on the platform, and exposing it to the rest of the system. MEX routines, also shown in the Figure 2, are executables originally written in C or Fortran that may be run directly from Matlab. All parallel functionality in the MultiMatlab system is provided through MEX routines. As a simple example, the MultiMatlab system provides users with an Eval

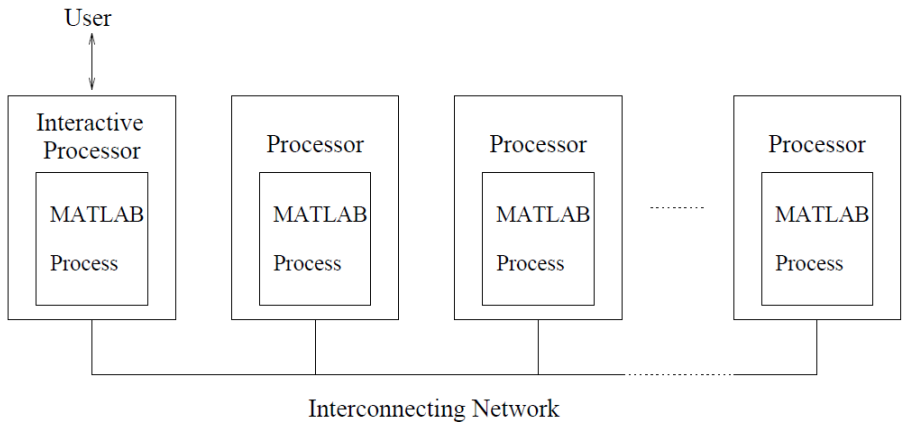
'582 Claim	Prior Art Reference – MultiMATLAB
	routine for parallel evaluation. This routine allows users to execute commands on the other processes. On the interactive process, the Eval routine is implemented as a MEX routine which accepts a vector of Matlab process IDs and a Matlab command. This routine sends the commands over the underlying communication layer to the processes corresponding to the specified IDs. On non-interactive processes, a separate top-level MEX routine is immediately run upon initialization of the system. This MEX routine runs in a loop in which it waits for Matlab commands to arrive from the interactive process and executes them in its own Matlab environment. All parallel MEX routines access the underlying communication layer and, hence, the network, through the MultiMatlab interface module. It is important to note that the interface module exists in the corresponding Matlab process's address space. For this reason, the interface module does not impose any additional context switch when parallel MEX routines access the communication layer. This is of particular importance when the parallel platform provides access to the network in user space. This enables parallel applications to access the network without operating system intervention as is often required for best performance. By residing within the Matlab process's address space, the MultiMatlab interface module is able to preserve this property for parallel MEX routines. The MultiMatlab interface module provides MEX routines access to the underlying communication layer via an indirection table. Upon initialization, the interface module builds a table of pointers to all functions and data in the underlying communication layer that need to be exposed to parallel MEX routines. When a parallel MEX routine is first loaded and executed, it accesses the MultiMatlab interface module through a function call to Matlab and is given the location of this table. This is the only time the MEX routine needs to directly access the interface module. Once the table is obtained, the MEX routine has the capability to access any function provided by the underlying communication layer through its corresponding offset in the table. In general, the MultiMatlab interface module adds an overhead of one load instruction per communication layer function call in a parallel MEX routine. As the execution time of an instruction is minimal compared to the latency of communicating a single packet of data, this overhead should be negligible. <i>Id.</i> at 3-5.

'582 Claim	Prior Art Reference – MultiMATLAB
	 Figure 2: MATLAB Process Address Space <i>Id.</i> at Fig. 2; see discussion at Fig. 2. For example, in Matlab's Partial Differential Equation Toolkit, users may interactively create and solve rather general PDE systems using the finite element method. The PDE toolkit provides an intuitive graphical interface allowing users to specify geometries, differential equations, and boundary conditions. <i>Id.</i> at 6. However, modifying and recompiling libraries is not a viable option when source code is not available. As an alternative, parallel libraries could be pushed down to the underlying communication layer. For example, on the IBM SP2, we can push Parallel ESSL [8], a parallel numerical library optimized for that platform, into the communication layer along with MPI-F. As a

'582 Claim	Prior Art Reference – MultiMATLAB
	result, parallel MEX routines on the SP2 may directly call Parallel ESSL functions in addition to MPI functions. In this case, the library is not reimplemented as a MEX routine. Instead, the MultiMATLAB interface module provides access to the library to all parallel MEX routines. <i>Id.</i> at 7. MATLAB®, a commercial product of The MathWorks, Inc., has become one of the principal languages of desktop scientific computing. A system is described that enables one to run MATLAB conveniently on multiple processors. Using short, MATLAB-style commands like Eval, Send, Recv, Beast, Min, and Sum, the user operating within one MATLAB session can start MATLAB processes on other machines and then pass commands and data between between these various processes in a fashion that maintains MATLAB's traditional user-friendliness. Multi-processor graphics is also supported. The system currently runs under MPICH on an IBM SP2 or a network of Unix workstations, and extensions are planned to networks of PCs. MultiMATLAB is potentially useful for education in parallel programming, for prototyping parallel algorithms, and for fast and convenient execution of easily parallelizable numerical computations on multiple processors. Trefethen at Abstract. The fifth is MATLAB's easy extensibility via packages of m-files known as Toolboxes. Many Toolboxes have been produced over the years, both by The MathWorks and by others, covering application areas such as optimization, signal processing, fuzzy logic, partial differential equations, and mathematical finance. Finally, perhaps the most interesting reason for MATLAB's success may be that from the beginning, the whole language has been built around real or complex vectors and matrices (including sparse matrices) as the fundamental data type. <i>Id.</i> at 1.1. MultiMATLAB is built upon MPI (Message Passing Interface), a highly functional and portable message passing standard [7, 13]. Here is a brief description of how the system is put together. The system is written using MPICH, a popular and freely available implementation of MPI developed at Argonne National Laboratory and Mississippi State University [6]. In particular, MultiMATLAB uses the P4 communication layer within MPICH, allowing it to run over a

'582 Claim	Prior Art Reference – MultiMATLAB
	heterogeneous network of workstations. In building upon MPICH, we believe we have developed a portable and extensible system, in that anyone can freely get a copy of the software and it will run on many systems. Versions of MPICH are beginning to become available that run on PCs running Windows, and we expect soon to experiment with MultiMATLAB on those platforms. The MultiMATLAB Start command builds a P4 process group file of remote hosts, which are either explicitly specified by the user or taken from a default list, and then initializes MPICH. MATLAB processes are then started on the remote hosts. Each process iterates over a simple loop, waiting for and executing commands received from the user's interactive MATLAB process. <i>Id.</i> at 4. The IBM RS/6000 SP is IBM's family of scalable, parallel computing solutions. It provides a state-of-the-art parallel computing system and industry-leading application enablers and applications. The RS/6000 SP runs the AIX operating system along with the Parallel System Support Programs (PSSP) system software on the control workstation and all processor nodes. For complete information on SP system software issues, see IBM RS/6000 SP: Planning Volume 2, Control Workstation and Software Environment. The scalable architecture of the SP system, its high-performance communication, POWER3, POWER2, and PowerPC Architecture give you the power to handle data-intensive, compute-intensive and I/O-intensive jobs with ease. You can execute both serial and parallel applications simultaneously, while managing your system from a single workstation. RS6000 at 1. The basic components of the RS/6000 SP system are: *Processor nodes (includes SP-attached servers) * Frames with integral power subsystems * Switches * Extension nodes * Control workstations (a high availability option is available) * Network connectivity adapters* External disk drives

'582 Claim	Prior Art Reference – MultiMATLAB
	The IBM RS/6000 SP System is scalable from one to 128 processor nodes. Up to sixteen thin, eight wide, or four high processor nodes can be mounted in a tall frame while a short frame can hold up to eight thin or four wide nodes. RS6000 at 2. Hard disk drives for the SP system can be either of two types as follows: Internal (contained within the node) External (mounted separately, outside of the node) RS6000 at 6; <i>see also</i> RS6000-2 at 3. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the '582 patent as charted for this claim limitation, and/or MultiMATLAB combined with the knowledge of one of ordinary skill in the art. Motivations to combine may come from the knowledge of the person of ordinary skill, or from the known problems and predictable solutions as embodied in these references. For example, to the extent any one of the MultiMATLAB printed publications lacks this element, it would have been obvious to combine with another of the MultiMATLAB printed publications that discloses this element, because they expressly cite to each other and/or to the same system, were published by overlapping research teams and/or in similar locations, and represent combinations of prior art elements according to known methods to yield predictable results. Further motivation to combine references and additional details may be found in the invalidity contentions cover pleading.
[1D] a second kernel residing in the at least one computer-readable medium, said second kernel configured to execute the matrix operations on the array of data	MultiMATLAB discloses this claim limitation. See, e.g.: In particular, MATLAB offers several features that simplify numerical computing. First, it provides users with easy access to an extensive library of high quality numerical routines. Second, it is an intuitive, interactive environment that enables users to use these routines in a dynamic manner well suited to scientific problem solving. Third, it provides a high-level matrix based language that permits users to express computations in an exceptionally concise manner via matrix or vector formulations and often with a fraction

'582 Claim	Prior Art Reference – MultiMATLAB
elements and return a first result using the second processor;	of the code required in conventional languages such as C or Fortran. Fourth, MATLAB's graphical and visualization tools offer a simple but powerful mechanism to manipulate and analyze data more effectively. Menon at 1.  The diagram illustrates the MultiMATLAB architecture. At the top, a 'User' is connected via a double-headed arrow to an 'Interactive Processor'. This processor contains a 'MATLAB Process'. Below it, there are several 'Processor' blocks, each containing a 'MATLAB Process'. These processors are connected to a common 'Interconnecting Network' at the bottom. The first processor is connected to the network, followed by a gap, and then another processor connected to the network. Ellipses indicate additional processors in the system. Figure 1: MultiMATLAB Architecture <i>Id.</i> at Fig. 1; see discussion at Fig. 1. In the MultiMatlab architecture, illustrated in Figure 1, each processor in a parallel platform individually runs a Matlab process. Each process is then provided with the ability to communicate with other processes through a communication layer that runs over the parallel platform's interconnection network. The user interacts directly with one Matlab process, called the interactive process, and operates within that process's Matlab environment. Other Matlab processes await commands from the interactive process. These other processes are used either by explicitly sending Matlab commands to them or by executing routines that, in turn, use them. The key component in this architecture is the MultiMatlab interface module. The interface module, shown in Figure 2, is responsible for initializing an underlying communication layer, such as MPI

'582 Claim	Prior Art Reference – MultiMATLAB
	[2], PVM [3], BLACS [4], or any other package available on the platform, and exposing it to the rest of the system. MEX routines, also shown in the Figure 2, are executables originally written in C or Fortran that may be run directly from Matlab. All parallel functionality in the MultiMatlab system is provided through MEX routines. As a simple example, the MultiMatlab system provides users with an Eval routine for parallel evaluation. This routine allows users to execute commands on the other processes. On the interactive process, the Eval routine is implemented as a MEX routine which accepts a vector of Matlab process IDs and a Matlab command. This routine sends the commands over the underlying communication layer to the processes corresponding to the specified IDs. On non-interactive processes, a separate top-level MEX routine is immediately run upon initialization of the system. This MEX routine runs in a loop in which it waits for Matlab commands to arrive from the interactive process and executes them in its own Matlab environment. All parallel MEX routines access the underlying communication layer and, hence, the network, through the MultiMatlab interface module. It is important to note that the interface module exists in the corresponding Matlab process's address space. For this reason, the interface module does not impose any additional context switch when parallel MEX routines access the communication layer. This is of particular importance when the parallel platform provides access to the network in user space. This enables parallel applications to access the network without operating system intervention as is often required for best performance. By residing within the Matlab process's address space, the MultiMatlab interface module is able to preserve this property for parallel MEX routines. <i>Id.</i> at 3-4. The MPI-F implementation of MultiMATLAB was design specifically for the IBM SP2. MPI-F is a highly optimized implementation of the MPI standard running over the SP2's high performance switch. This implementation permits access to the switch in user space. In this implementation, all MATLAB processes are started via POE, IBM's Parallel Operating Environment. For each of p MATLAB processes, POE assigns an identification number between 0

'582 Claim	Prior Art Reference – MultiMATLAB
	and p-1. The process numbered 0 is considered the interactive MATLAB. All other processes wait for commands from the interactive MATLAB process. This implementation was designed to demonstrate the performance potential of the MultiMATLAB system. In particular, it provides a platform in which parallel libraries or applications written in MPI can easily be integrated into MATLAB on the SP2. <i>Id.</i> at 6. <pre style="width: 45%;"> r = b; rho = r'*r; k = 0; while((k < kmax)) k = k+1; if (k == 1) p = r; else beta = rho/oldrho; p = r + beta*p; end w = A * p; alpha = rho/(p'*w); x = x + alpha * p; r = r - alpha * w; oldrho = rho; rho = r'*r; end a. Sequential MATLAB </pre> <pre style="width: 45%;"> r_local = b_local; rho = Sum(r_local'*r_local); k = 0; while((k < kmax)) k = k+1; if (k == 1) p_local = r_local; else beta = rho/oldrho; p_local = r_local + beta*p_local; end p = Gather(p_local); w_local = A_local * p; alpha = rho/(Sum(p_local'*w_local)); x_local = x_local + alpha * p_local; r_local = r_local - alpha * w_local; oldrho = rho; rho = Sum(r_local'*r_local); end b. Message Passing MATLAB </pre> Figure 4: Conjugate Gradients In this section, we describe different MultiMATLAB implementations of the conjugate gradients algorithm on the IBM SP2 at the Cornell Theory Center. The conjugate gradients algorithm is iterative method to solve a linear system of the form $Ax = b$, where A is a symmetric positive definite matrix and x and b are vectors. The computational core of this method is a single matrix-

'582 Claim	Prior Art Reference – MultiMATLAB
	vector multiplication at each iteration. The sequential MATLAB code, adapted from [13], is shown in Figure 4a. ... Figure 4b illustrates a parallel MATLAB implementation of conjugate gradients that uses the SPMD message passing described in Section 5.2. In this case, the matrix A and the different vectors are each distributed by row over all processes as in Figure 6. In each iteration, a process only does the computation required for its local data. However, communication is required for two different operations: the dot-products needed to compute rho and alpha and the matrix-vector multiply needed to compute w_local. The dot products only require communication of a single value over all processes. On the other hand, the matrix-vector multiplication needed for w_local requires the global vector p, and, thus, more expensive communication. <i>Id.</i> at 11-13. MATLAB®, a commercial product of The MathWorks, Inc., has become one of the principal languages of desktop scientific computing. A system is described that enables one to run MATLAB conveniently on multiple processors. Using short, MATLAB-style commands like Eval, Send, Recv, Beast, Min, and Sum, the user operating within one MATLAB session can start MATLAB processes on other machines and then pass commands and data between between these various processes in a fashion that maintains MATLAB's traditional user-friendliness. Multi-processor graphics is also supported. The system currently runs under MPICH on an IBM SP2 or a network of Unix workstations, and extensions are planned to networks of PCs. MultiMATLAB is potentially useful for education in parallel programming, for prototyping parallel algorithms, and for fast and convenient execution of easily parallelizable numerical computations on multiple processors. Trefethen at Abstract. More general point-to-point communication is accomplished by send and receive commands, which can be executed on any of the MATLAB processes. For example, the sequence $x = [\pi \pi^2];$

'582 Claim	Prior Art Reference – MultiMATLAB
	Send(3,x) Eval(3, 'Recv') passes a message containing a 2-vector from the master process to process 3, leading to the output 3.1416 9.8696 An optional argument can be added in Recv to specify the source. <i>Id.</i> at 2.3; <i>see also id.</i> at 2.1. <pre> 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; </pre> Process 0 creates the variable <code>a</code> with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of <code>a</code>, and sends it along to process 2; and so on. If there are six processors the command <code>Eval('cycle')</code> produces the output <pre> a = 1 a = 2 a = 4 a = 8 a = 16 a = 32 </pre> The processes run asynchronously, but since each <code>Send</code> command is only executed after the corresponding <code>Recv</code> has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning. <i>Id.</i> at 2.3. We can do this by taking a data-parallel approach in a simplistic fashion. We have developed a number of routines such as <code>Distribute</code> and <code>Collect</code> that allow a user to distribute a matrix or to collect a set of matrices into one large matrix. These functions operate using a mask that indicates which processors hold which portions of the matrix. This allows us also to develop routines such as <code>Shift</code>

'582 Claim	Prior Art Reference – MultiMATLAB
	and Copy that are useful in data-parallel computing, keeping the communication to a more abstract level. <i>Id.</i> at 2.5. The MultiMATLAB Start command builds a P4 process group file of remote hosts, which are either explicitly specified by the user or taken from a default list, and then initializes MPICH. MATLAB processes are then started on the remote hosts. Each process iterates over a simple loop, waiting for and executing commands received from the user's interactive MATLAB process. <i>Id.</i> at 4. The IBM RS/6000 SP is IBM's family of scalable, parallel computing solutions. It provides a state-of-the-art parallel computing system and industry-leading application enablers and applications. The RS/6000 SP runs the AIX operating system along with the Parallel System Support Programs (PSSP) system software on the control workstation and all processor nodes. For complete information on SP system software issues, see IBM RS/6000 SP: Planning Volume 2, Control Workstation and Software Environment. The scalable architecture of the SP system, its high-performance communication, POWER3, POWER2, and PowerPC Architecture give you the power to handle data-intensive, compute-intensive and I/O-intensive jobs with ease. You can execute both serial and parallel applications simultaneously, while managing your system from a single workstation. RS6000 at 1. The basic components of the RS/6000 SP system are: *Processor nodes (includes SP-attached servers) * Frames with integral power subsystems * Switches * Extension nodes * Control workstations (a high availability option is available) * Network connectivity adapters* External disk drives The IBM RS/6000 SP System is scalable from one to 128 processor nodes. Up to sixteen thin, eight wide, or four high processor nodes can be mounted in a tall frame while a short frame can hold up to eight thin or four wide nodes.

'582 Claim	Prior Art Reference – MultiMATLAB
	RS6000 at 2. Hard disk drives for the SP system can be either of two types as follows: Internal (contained within the node) External (mounted separately, outside of the node) RS6000 at 6; <i>see also</i> RS6000-2 at 3. MATLAB [?] is one of the most widely used tools in scientific and technical computing. It started in 1970s as an interactive interface to EISPACK [?] and LINPACK [?], a set of eigenvalue and linear system solution routines. It has since grown to a feature rich product utilizing modern numerical libraries like ATLAS [?] and FFTW [?], and with toolboxes in a number of application areas, for example, financial mathematics, neural networks, and control theory. It has a built-in interpreted language that is similar to C and HPF, and the flexible matrix indexing makes it very suitable for programming matrix problems. It is a very useful tool to a wide audience. For example, it comes in very handy as a matrix calculator for simple problems, because of its easy-to-use command syntax. Its strong graphical capabilities makes it a good data analysis tool. Also researchers have been known to build very complex systems using MATLAB scripts. Choy 1 at 1. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the '582 patent as charted for this claim limitation, and/or MultiMATLAB combined with the knowledge of one of ordinary skill in the art. Motivations to combine may

'582 Claim	Prior Art Reference – MultiMATLAB
	come from the knowledge of the person of ordinary skill, or from the known problems and predictable solutions as embodied in these references. For example, to the extent any one of the MultiMATLAB printed publications lacks this element, it would have been obvious to combine with another of the MultiMATLAB printed publications that discloses this element, because they expressly cite to each other and/or to the same system, were published by overlapping research teams and/or in similar locations, and represent combinations of prior art elements according to known methods to yield predictable results. Further motivation to combine references and additional details may be found in the invalidity contentions cover pleading.
[1E] a second cluster node module residing in the at least one computer-readable medium, said second cluster node module configured to send commands to the second kernel and communicate with the first cluster node module;	MultiMATLAB discloses this claim limitation. <i>See</i> disclosures at [1A]. [1C], <i>supra</i>. See, also, e.g.: More general point-to-point communication is accomplished by send and receive commands, which can be executed on any of the MATLAB processes. For example, the sequence x = [pi pi^2]; Send(3,x) Eval(3, 'Recv') passes a message containing a 2-vector from the master process to process 3, leading to the output 3.1416 9.8696 An optional argument can be added in Recv to specify the source. <i>Id.</i> at 2.3; <i>see also id.</i> at 2.1.

'582 Claim	Prior Art Reference – MultiMATLAB
	<pre> 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; </pre> Process 0 creates the variable <code>a</code> with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of <code>a</code>, and sends it along to process 2; and so on. If there are six processors the command <code>Eval('cycle')</code> produces the output <pre> a = 1 a = 2 a = 4 a = 8 a = 16 a = 32 </pre> The processes run asynchronously, but since each <code>Send</code> command is only executed after the corresponding <code>Recv</code> has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning. <i>Id.</i> at 2.3. Although <code>Send</code> takes a vector of processor IDs as its destination list, the underlying idea is that of point-to-point communication. For more efficient communication between multiple processes, as well as greater convenience for the programmer, MultiMATLAB also has various commands for collective communication. These commands must be evaluated simultaneously on all processes. <i>Id.</i> at 2.4. The MultiMATLAB <code>Start</code> command builds a P4 process group file of remote hosts, which are either explicitly specified by the user or taken from a default list, and then initializes MPICH. MATLAB processes are then started on the remote hosts. Each process iterates over a simple loop, waiting for and executing commands received from the user's interactive MATLAB process. <i>Id.</i> at 4.

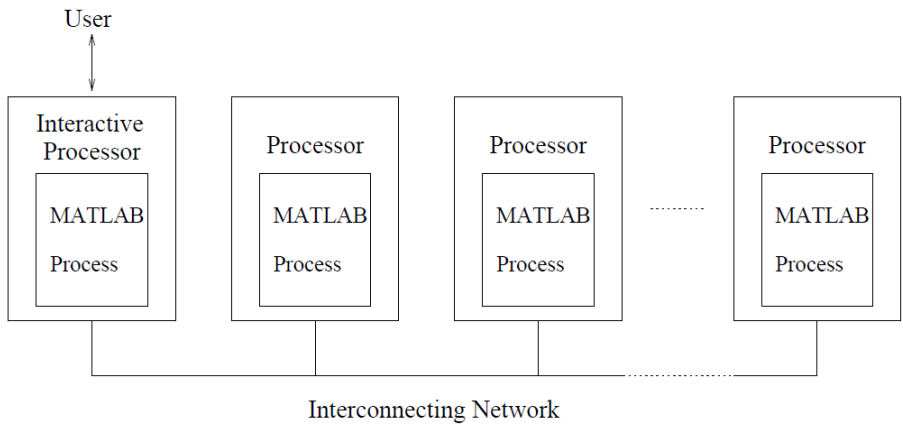
'582 Claim	Prior Art Reference – MultiMATLAB
	MultiMATLAB is built upon MPI (Message Passing Interface), a highly functional and portable message passing standard [7, 13]. Here is a brief description of how the system is put together. The system is written using MPICH, a popular and freely available implementation of MPI developed at Argonne National Laboratory and Mississippi State University [6]. In particular, MultiMATLAB uses the P4 communication layer within MPICH, allowing it to run over a heterogeneous network of workstations. In building upon MPICH, we believe we have developed a portable and extensible system, in that anyone can freely get a copy of the software and it will run on many systems. Versions of MPICH are beginning to become available that run on PCs running Windows, and we expect soon to experiment with MultiMATLAB on those platforms. <i>Id.</i> at 4. The IBM RS/6000 SP is IBM's family of scalable, parallel computing solutions. It provides a state-of-the-art parallel computing system and industry-leading application enablers and applications. The RS/6000 SP runs the AIX operating system along with the Parallel System Support Programs (PSSP) system software on the control workstation and all processor nodes. For complete information on SP system software issues, see IBM RS/6000 SP: Planning Volume 2, Control Workstation and Software Environment. The scalable architecture of the SP system, its high-performance communication, POWER3, POWER2, and PowerPC Architecture give you the power to handle data-intensive, compute-intensive and I/O-intensive jobs with ease. You can execute both serial and parallel applications simultaneously, while managing your system from a single workstation. RS6000 at 1. The basic components of the RS/6000 SP system are: *Processor nodes (includes SP-attached servers) * Frames with integral power subsystems * Switches * Extension nodes * Control workstations (a high availability option is available) * Network connectivity adapters* External disk drives

'582 Claim	Prior Art Reference – MultiMATLAB
	The IBM RS/6000 SP System is scalable from one to 128 processor nodes. Up to sixteen thin, eight wide, or four high processor nodes can be mounted in a tall frame while a short frame can hold up to eight thin or four wide nodes. RS6000 at 2. Hard disk drives for the SP system can be either of two types as follows: Internal (contained within the node) External (mounted separately, outside of the node) RS6000 at 6; <i>see also</i> RS6000-2 at 3.

'582 Claim	Prior Art Reference – MultiMATLAB
	Chapter 3 Point-to-Point Communication 3.1 Introduction Sending and receiving of messages by processes is the basic MPI communication mechanism. The basic point-to-point communication operations are send and receive. Their use is illustrated in the example below. <pre>#include "mpi.h" int main(int argc, char *argv[]) { char message[20]; int myrank; MPI_Status status; MPI_Init(&argc, &argv); MPI_Comm_rank(MPI_COMM_WORLD, &myrank); if (myrank == 0) /* code for process zero */ { strcpy(message,"Hello, there"); MPI_Send(message, strlen(message)+1, MPI_CHAR, 1, 99, MPI_COMM_WORLD); } else if (myrank == 1) /* code for process one */ { MPI_Recv(message, 20, MPI_CHAR, 0, 99, MPI_COMM_WORLD, &status); printf("received :%s:\n", message); } MPI_Finalize(); return 0; }</pre> In this example, process zero (<i>myrank</i> = 0) sends a message to process one using the send operation MPI_SEND. The operation specifies a send buffer in the sender memory from which the message data is taken. In the example above, the send buffer consists of the storage containing the variable message in the memory of process zero. The location, size and type of the send buffer are specified by the first three parameters of the send operation. The message sent will contain the 13 characters of this variable. In addition, the send operation associates an envelope with the message. This envelope specifies the 23 MPI Standard 1994 at 23.

'582 Claim	Prior Art Reference – MultiMATLAB																					
	MPI_ISEND(buf, count, datatype, dest, tag, comm, request) <table><tr><td>IN</td><td>buf</td><td>initial address of send buffer (choice)</td></tr><tr><td>IN</td><td>count</td><td>number of elements in send buffer (integer)</td></tr><tr><td>IN</td><td>datatype</td><td>datatype of each send buffer element (handle)</td></tr><tr><td>IN</td><td>dest</td><td>rank of destination (integer)</td></tr><tr><td>IN</td><td>tag</td><td>message tag (integer)</td></tr><tr><td>IN</td><td>comm</td><td>communicator (handle)</td></tr><tr><td>OUT</td><td>request</td><td>communication request (handle)</td></tr></table> int MPI_Isend(void* buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *request) MPI_ISEND(BUF, COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR) <type> BUF(*) INTEGER COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR Start a standard mode, nonblocking send. MPI Standard 1995 at 38; <i>see also</i> MPI Standard 1995 at 16-23.	IN	buf	initial address of send buffer (choice)	IN	count	number of elements in send buffer (integer)	IN	datatype	datatype of each send buffer element (handle)	IN	dest	rank of destination (integer)	IN	tag	message tag (integer)	IN	comm	communicator (handle)	OUT	request	communication request (handle)
IN	buf	initial address of send buffer (choice)																				
IN	count	number of elements in send buffer (integer)																				
IN	datatype	datatype of each send buffer element (handle)																				
IN	dest	rank of destination (integer)																				
IN	tag	message tag (integer)																				
IN	comm	communicator (handle)																				
OUT	request	communication request (handle)																				

'582 Claim	Prior Art Reference – MultiMATLAB																					
	MPI_RECV (buf, count, datatype, source, tag, comm, request) <table><tr><td>OUT</td><td>buf</td><td>initial address of receive buffer (choice)</td></tr><tr><td>IN</td><td>count</td><td>number of elements in receive buffer (integer)</td></tr><tr><td>IN</td><td>datatype</td><td>datatype of each receive buffer element (handle)</td></tr><tr><td>IN</td><td>source</td><td>rank of source (integer)</td></tr><tr><td>IN</td><td>tag</td><td>message tag (integer)</td></tr><tr><td>IN</td><td>comm</td><td>communicator (handle)</td></tr><tr><td>OUT</td><td>request</td><td>communication request (handle)</td></tr></table> int MPI_Irecv(void* buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Request *request) MPI_RECV(BUF, COUNT, DATATYPE, SOURCE, TAG, COMM, REQUEST, IERROR) <type> BUF(*) INTEGER COUNT, DATATYPE, SOURCE, TAG, COMM, REQUEST, IERROR Start a nonblocking receive. These calls allocate a communication request object and associate it with the request handle (the argument request). The request can be used later to query the status of the communication or wait for its completion. A nonblocking send call indicates that the system may start copying data out of the send buffer. The sender should not access any part of the send buffer after a nonblocking send operation is called, until the send completes. A nonblocking receive call indicates that the system may start writing data into the receive buffer. The receiver should not access any part of the receive buffer after a nonblocking receive operation is called, until the receive completes. MPI Standard 1995 at 40. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the '582 patent as charted for this claim limitation, and/or MultiMATLAB combined with the knowledge of one of ordinary skill in the art. Motivations to combine may come from the knowledge of the person of ordinary skill, or from the known problems and predictable solutions as embodied in these references. For example, to the extent any one of the MultiMATLAB printed publications lacks this element, it would have been obvious to combine with another of the MultiMATLAB printed publications that discloses this element, because they expressly cite to each other and/or to the same system, were published by overlapping research teams and/or in similar locations, and represent combinations of prior art elements according to known methods to yield predictable results. Further motivation to combine references and additional details may be found in the invalidity contentions cover pleading.	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	request	communication request (handle)
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	request	communication request (handle)																				

'582 Claim	Prior Art Reference – MultiMATLAB
[1F] a third kernel residing in the at least one computer-readable medium, said third kernel configured to receive the first result and execute additional matrix operations on the first result and return a second result to one or more of the first, second, or third cluster node modules using the third processor; and	MultiMATLAB discloses this claim limitation. <i>See</i> disclosures at [1A], [1D], <i>supra</i>. See, also, e.g.: In particular, MATLAB offers several features that simplify numerical computing. First, it provides users with easy access to an extensive library of high quality numerical routines. Second, it is an intuitive, interactive environment that enables users to use these routines in a dynamic manner well suited to scientific problem solving. Third, it provides a high-level matrix based language that permits users to express computations in an exceptionally concise manner via matrix or vector formulations and often with a fraction of the code required in conventional languages such as C or Fortran. Fourth, MATLAB's graphical and visualization tools offer a simple but powerful mechanism to manipulate and analyze data more effectively. Menon at 1.  <pre> graph TD User[User] <--> IP[Interactive Processor] subgraph Network [Interconnecting Network] IP --- P1[Processor] P1 --- P2[Processor] P2 --- P3[Processor] P3 --- P4[Processor] end subgraph IP_Box [Interactive Processor] IP_Matlab[MATLAB Process] end subgraph P1_Box [Processor] P1_Matlab[MATLAB Process] end subgraph P2_Box [Processor] P2_Matlab[MATLAB Process] end subgraph P3_Box [Processor] P3_Matlab[MATLAB Process] end subgraph P4_Box [Processor] P4_Matlab[MATLAB Process] end </pre> Figure 1: MultiMATLAB Architecture <i>Id.</i> at Fig. 1; see discussion at Fig. 1. A possible alternative is to use parallel MEX routines that provide similar functionality to high-level MATLAB operations. These parallel MEX routines would perform matrix multiplications, solve for eigenvalues, compute Fourier transforms, and so on. In fact, these MEX routines could provide an

'582 Claim	Prior Art Reference – MultiMATLAB
	interface identical to corresponding sequential MATLAB functions. In particular, they would distribute data from the interactive MATLAB process among all processes, perform the parallel computation, and collect results back to the interactive process. Menon at 10. <pre style="width: 45%;"> r = b; rho = r'*r; k = 0; while((k < kmax)) k = k+1; if (k == 1) p = r; else beta = rho/oldrho; p = r + beta*p; end w = A * p; alpha = rho/(p'*w); x = x + alpha * p; r = r - alpha * w; oldrho = rho; rho = r'*r; end a. Sequential MATLAB </pre> <pre style="width: 45%;"> r_local = b_local; rho = Sum(r_local'*r_local); k = 0; while((k < kmax)) k = k+1; if (k == 1) p_local = r_local; else beta = rho/oldrho; p_local = r_local + beta*p_local; end p = Gather(p_local); w_local = A_local * p; alpha = rho/(Sum(p_local'*w_local)); x_local = x_local + alpha * p_local; r_local = r_local - alpha * w_local; oldrho = rho; rho = Sum(r_local'*r_local); end b. Message Passing MATLAB </pre> Figure 4: Conjugate Gradients In this section, we describe different MultiMATLAB implementations of the conjugate gradients algorithm on the IBM SP2 at the Cornell Theory Center. The conjugate gradients algorithm is iterative method to solve a linear system of the form $Ax = b$, where A is a symmetric positive definite matrix and x and b are vectors. The computational core of this method is a single matrix-vector multiplication at each iteration. The sequential MATLAB code, adapted from [13], is shown in Figure 4a. ...

'582 Claim	Prior Art Reference – MultiMATLAB
	Figure 4b illustrates a parallel MATLAB implementation of conjugate gradients that uses the SPMD message passing described in Section 5.2. In this case, the matrix A and the different vectors are each distributed by row over all processes as in Figure 6. In each iteration, a process only does the computation required for its local data. However, communication is required for two different operations: the dot-products needed to compute rho and alpha and the matrix-vector multiply needed to compute w_local. The dot products only require communication of a single value over all processes. On the other hand, the matrix-vector multiplication needed for w_local requires the global vector p, and, thus, more expensive communication. <i>Id.</i> at 11-13. MATLAB®, a commercial product of The MathWorks, Inc., has become one of the principal languages of desktop scientific computing. A system is described that enables one to run MATLAB conveniently on multiple processors. Using short, MATLAB-style commands like Eval, Send, Recv, Bcast, Min, and Sum, the user operating within one MATLAB session can start MATLAB processes on other machines and then pass commands and data between between these various processes in a fashion that maintains MATLAB's traditional user-friendliness. Multi-processor graphics is also supported. The system currently runs under MPICH on an IBM SP2 or a network of Unix workstations, and extensions are planned to networks of PCs. MultiMATLAB is potentially useful for education in parallel programming, for prototyping parallel algorithms, and for fast and convenient execution of easily parallelizable numerical computations on multiple processors. Trefethen at Abstract. More general point-to-point communication is accomplished by send and receive commands, which can be executed on any of the MATLAB processes. For example, the sequence <pre> x = [pi pi^2]; Send(3,x) Eval(3, 'Recv') </pre>

'582 Claim	Prior Art Reference – MultiMATLAB
	passes a message containing a 2-vector from the master process to process 3, leading to the output 3.1416 9.8696 An optional argument can be added in Recv to specify the source. <i>Id.</i> at 2.3; <i>see also id.</i> at 2.1. <pre> 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; </pre> Process 0 creates the variable <code>a</code> with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of <code>a</code>, and sends it along to process 2; and so on. If there are six processors the command <code>Eval('cycle')</code> produces the output <pre> a = 1 a = 2 a = 4 a = 8 a = 16 a = 32 </pre> The processes run asynchronously, but since each <code>Send</code> command is only executed after the corresponding <code>Recv</code> has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning. <i>Id.</i> at 2.3. We can do this by taking a data-parallel approach in a simplistic fashion. We have developed a number of routines such as <code>Distribute</code> and <code>Collect</code> that allow a user to distribute a matrix or to collect a set of matrices into one large matrix. These functions operate using a mask that indicates which processors hold which portions of the matrix. This allows us also to develop routines such as <code>Shift</code> and <code>Copy</code> that are useful in data-parallel computing, keeping the communication to a more abstract level.

'582 Claim	Prior Art Reference – MultiMATLAB
	<i>Id.</i> at 2.5. The IBM RS/6000 SP is IBM's family of scalable, parallel computing solutions. It provides a state-of-the-art parallel computing system and industry-leading application enablers and applications. The RS/6000 SP runs the AIX operating system along with the Parallel System Support Programs (PSSP) system software on the control workstation and all processor nodes. For complete information on SP system software issues, see IBM RS/6000 SP: Planning Volume 2, Control Workstation and Software Environment. The scalable architecture of the SP system, its high-performance communication, POWER3, POWER2, and PowerPC Architecture give you the power to handle data-intensive, compute-intensive and I/O-intensive jobs with ease. You can execute both serial and parallel applications simultaneously, while managing your system from a single workstation. RS6000 at 1. The basic components of the RS/6000 SP system are: *Processor nodes (includes SP-attached servers) * Frames with integral power subsystems * Switches * Extension nodes * Control workstations (a high availability option is available) * Network connectivity adapters* External disk drives The IBM RS/6000 SP System is scalable from one to 128 processor nodes. Up to sixteen thin, eight wide, or four high processor nodes can be mounted in a tall frame while a short frame can hold up to eight thin or four wide nodes. RS6000 at 2. Hard disk drives for the SP system can be either of two types as follows: Internal (contained within the node) External (mounted separately, outside of the node) RS6000 at 6; <i>see also</i> RS6000-2 at 3.

'582 Claim	Prior Art Reference – MultiMATLAB
	Chapter 3 Point-to-Point Communication 3.1 Introduction Sending and receiving of messages by processes is the basic MPI communication mechanism. The basic point-to-point communication operations are send and receive. Their use is illustrated in the example below. <pre>#include "mpi.h" int main(int argc, char *argv[]) { char message[20]; int myrank; MPI_Status status; MPI_Init(&argc, &argv); MPI_Comm_rank(MPI_COMM_WORLD, &myrank); if (myrank == 0) /* code for process zero */ { strcpy(message,"Hello, there"); MPI_Send(message, strlen(message)+1, MPI_CHAR, 1, 99, MPI_COMM_WORLD); } else if (myrank == 1) /* code for process one */ { MPI_Recv(message, 20, MPI_CHAR, 0, 99, MPI_COMM_WORLD, &status); printf("received :%s:\n", message); } MPI_Finalize(); return 0; }</pre> In this example, process zero (<i>myrank</i> = 0) sends a message to process one using the send operation MPI_SEND. The operation specifies a send buffer in the sender memory from which the message data is taken. In the example above, the send buffer consists of the storage containing the variable message in the memory of process zero. The location, size and type of the send buffer are specified by the first three parameters of the send operation. The message sent will contain the 13 characters of this variable. In addition, the send operation associates an envelope with the message. This envelope specifies the 23 MPI Standard 1994 at 23.

'582 Claim	Prior Art Reference – MultiMATLAB																					
	MPI_ISEND(buf, count, datatype, dest, tag, comm, request) <table><tr><td>IN</td><td>buf</td><td>initial address of send buffer (choice)</td></tr><tr><td>IN</td><td>count</td><td>number of elements in send buffer (integer)</td></tr><tr><td>IN</td><td>datatype</td><td>datatype of each send buffer element (handle)</td></tr><tr><td>IN</td><td>dest</td><td>rank of destination (integer)</td></tr><tr><td>IN</td><td>tag</td><td>message tag (integer)</td></tr><tr><td>IN</td><td>comm</td><td>communicator (handle)</td></tr><tr><td>OUT</td><td>request</td><td>communication request (handle)</td></tr></table> <pre>int MPI_Isend(void* buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *request)</pre> MPI_ISEND(BUF, COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR) <type> BUF(*) INTEGER COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR Start a standard mode, nonblocking send. MPI Standard 1995 at 38; <i>see also</i> MPI Standard 1995 at 16-23.	IN	buf	initial address of send buffer (choice)	IN	count	number of elements in send buffer (integer)	IN	datatype	datatype of each send buffer element (handle)	IN	dest	rank of destination (integer)	IN	tag	message tag (integer)	IN	comm	communicator (handle)	OUT	request	communication request (handle)
IN	buf	initial address of send buffer (choice)																				
IN	count	number of elements in send buffer (integer)																				
IN	datatype	datatype of each send buffer element (handle)																				
IN	dest	rank of destination (integer)																				
IN	tag	message tag (integer)																				
IN	comm	communicator (handle)																				
OUT	request	communication request (handle)																				

'582 Claim	Prior Art Reference – MultiMATLAB																					
	MPI_RECV (buf, count, datatype, source, tag, comm, request) <table><tr><td>OUT</td><td>buf</td><td>initial address of receive buffer (choice)</td></tr><tr><td>IN</td><td>count</td><td>number of elements in receive buffer (integer)</td></tr><tr><td>IN</td><td>datatype</td><td>datatype of each receive buffer element (handle)</td></tr><tr><td>IN</td><td>source</td><td>rank of source (integer)</td></tr><tr><td>IN</td><td>tag</td><td>message tag (integer)</td></tr><tr><td>IN</td><td>comm</td><td>communicator (handle)</td></tr><tr><td>OUT</td><td>request</td><td>communication request (handle)</td></tr></table> int MPI_Irecv(void* buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Request *request) MPI_RECV(BUF, COUNT, DATATYPE, SOURCE, TAG, COMM, REQUEST, IERROR) <type> BUF(*) INTEGER COUNT, DATATYPE, SOURCE, TAG, COMM, REQUEST, IERROR Start a nonblocking receive. These calls allocate a communication request object and associate it with the request handle (the argument request). The request can be used later to query the status of the communication or wait for its completion. A nonblocking send call indicates that the system may start copying data out of the send buffer. The sender should not access any part of the send buffer after a nonblocking send operation is called, until the send completes. A nonblocking receive call indicates that the system may start writing data into the receive buffer. The receiver should not access any part of the receive buffer after a nonblocking receive operation is called, until the receive completes. MPI Standard 1995 at 40.	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	request	communication request (handle)
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	request	communication request (handle)																				

'582 Claim	Prior Art Reference – MultiMATLAB
	MATLAB [?] is one of the most widely used tools in scientific and technical computing. It started in 1970s as an interactive interface to EISPACK [?] and LINPACK [?], a set of eigenvalue and linear system solution routines. It has since grown to a feature rich product utilizing modern numerical libraries like ATLAS [?] and FFTW [?], and with toolboxes in a number of application areas, for example, financial mathematics, neural networks, and control theory. It has a built-in interpreted language that is similar to C and HPF, and the flexible matrix indexing makes it very suitable for programming matrix problems. It is a very useful tool to a wide audience. For example, it comes in very handy as a matrix calculator for simple problems, because of its easy-to-use command syntax. Its strong graphical capabilities makes it a good data analysis tool. Also researchers have been known to build very complex systems using MATLAB scripts. Choy 1 at 1. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the '582 patent as charted for this claim limitation, and/or MultiMATLAB combined with the knowledge of one of ordinary skill in the art. Motivations to combine may come from the knowledge of the person of ordinary skill, or from the known problems and predictable solutions as embodied in these references. For example, to the extent any one of the MultiMATLAB printed publications lacks this element, it would have been obvious to combine with another of the MultiMATLAB printed publications that discloses this element, because they expressly cite to each other and/or to the same system, were published by overlapping research teams and/or in similar locations, and represent combinations of prior art elements according to known methods to yield predictable results. Further motivation to combine references and additional details may be found in the invalidity contentions cover pleading.

'582 Claim	Prior Art Reference – MultiMATLAB
[1G] a third cluster node module residing in the at least one computer-readable medium, said third cluster node module configured to send commands to the third kernel and configured to communicate with the first cluster node module and the second cluster node module;	MultiMATLAB discloses this claim limitation. <i>See</i> disclosures at [1C], [1E], Menon at Fig. 1, Fig. 2; see discussion at Fig. 1, Fig. 2. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the '582 patent as charted for this claim limitation, and/or MultiMATLAB combined with the knowledge of one of ordinary skill in the art. Motivations to combine may come from the knowledge of the person of ordinary skill, or from the known problems and predictable solutions as embodied in these references. For example, to the extent any one of the MultiMATLAB printed publications lacks this element, it would have been obvious to combine with another of the MultiMATLAB printed publications that discloses this element, because they expressly cite to each other and/or to the same system, were published by overlapping research teams and/or in similar locations, and represent combinations of prior art elements according to known methods to yield predictable results. Further motivation to combine references and additional details may be found in the invalidity contentions cover pleading.
[1H] wherein the first cluster node module is configured to return, based on the second result, a final result.	MultiMATLAB discloses this claim limitation. <i>See</i> disclosures at [1G], <i>supra</i>. See, also, e.g.: In particular, MATLAB offers several features that simplify numerical computing. First, it provides users with easy access to an extensive library of high quality numerical routines. Second, it is an intuitive, interactive environment that enables users to use these routines in a dynamic manner well suited to scientific problem solving. Third, it provides a high-level matrix based language that permits users to express computations in an exceptionally concise manner via matrix or vector formulations and often with a fraction of the code required in conventional languages such as C or Fortran. Fourth, MATLAB's graphical and visualization tools offer a simple but powerful mechanism to manipulate and analyze data more effectively. Menon at 1. In the MultiMatlab architecture, illustrated in Figure 1, each processor in a parallel platform individually runs a Matlab process. Each process is then provided with the ability to communicate with other processes through a communication layer that runs over the parallel platform's

'582 Claim	Prior Art Reference – MultiMATLAB
	interconnection network. The user interacts directly with one Matlab process, called the interactive process, and operates within that process's Matlab environment. Other Matlab processes await commands from the interactive process. These other processes are used either by explicitly sending Matlab commands to them or by executing routines that, in turn, use them. The key component in this architecture is the MultiMatlab interface module. The interface module, shown in Figure 2, is responsible for initializing an underlying communication layer, such as MPI [2], PVM [3], BLACS [4], or any other package available on the platform, and exposing it to the rest of the system. MEX routines, also shown in the Figure 2, are executables originally written in C or Fortran that may be run directly from Matlab. All parallel functionality in the MultiMatlab system is provided through MEX routines. As a simple example, the MultiMatlab system provides users with an Eval routine for parallel evaluation. This routine allows users to execute commands on the other processes. On the interactive process, the Eval routine is implemented as a MEX routine which accepts a vector of Matlab process IDs and a Matlab command. This routine sends the commands over the underlying communication layer to the processes corresponding to the specified IDs. On non-interactive processes, a separate top-level MEX routine is immediately run upon initialization of the system. This MEX routine runs in a loop in which it waits for Matlab commands to arrive from the interactive process and executes them in its own Matlab environment. All parallel MEX routines access the underlying communication layer and, hence, the network, through the MultiMatlab interface module. It is important to note that the interface module exists in the corresponding Matlab process's address space. For this reason, the interface module does not impose any additional context switch when parallel MEX routines access the communication layer. This is of particular importance when the parallel platform provides access to the network in user space. This enables parallel applications to access the network without operating system intervention as is often required for best performance. By residing within the Matlab process's address space, the MultiMatlab interface module is able to preserve this property for parallel MEX routines. <i>Id.</i> at 3-4.

'582 Claim	Prior Art Reference – MultiMATLAB
	A possible alternative is to use parallel MEX routines that provide similar functionality to high-level MATLAB operations. These parallel MEX routines would perform matrix multiplications, solve for eigenvalues, compute Fourier transforms, and so on. In fact, these MEX routines could provide an interface identical to corresponding sequential MATLAB functions. In particular, they would distribute data from the interactive MATLAB process among all processes, perform the parallel computation, and collect results back to the interactive process. Menon at 10. <pre style="width: 45%;"> a. Sequential MATLAB r = b; rho = r'*r; k = 0; while((k < kmax)) k = k+1; if (k == 1) p = r; else beta = rho/oldrho; p = r + beta*p; end w = A * p; alpha = rho/(p'*w); x = x + alpha * p; r = r - alpha * w; oldrho = rho; rho = r'*r; end </pre> <pre style="width: 45%;"> b. Message Passing MATLAB r_local = b_local; rho = Sum(r_local'*r_local); k = 0; while((k < kmax)) k = k+1; if (k == 1) p_local = r_local; else beta = rho/oldrho; p_local = r_local + beta*p_local; end p = Gather(p_local); w_local = A_local * p; alpha = rho/(Sum(p_local'*w_local)); x_local = x_local + alpha * p_local; r_local = r_local - alpha * w_local; oldrho = rho; rho = Sum(r_local'*r_local); end </pre> Figure 4: Conjugate Gradients In this section, we describe different MultiMATLAB implementations of the conjugate gradients algorithm on the IBM SP2 at the Cornell Theory Center. The conjugate gradients algorithm is iterative method to solve a linear system of the form $Ax = b$, where A is a symmetric positive definite matrix and x and b are vectors. The computational core of this method is a single matrix-

'582 Claim	Prior Art Reference – MultiMATLAB
	vector multiplication at each iteration. The sequential MATLAB code, adapted from [13], is shown in Figure 4a. ... Figure 4b illustrates a parallel MATLAB implementation of conjugate gradients that uses the SPMD message passing described in Section 5.2. In this case, the matrix A and the different vectors are each distributed by row over all processes as in Figure 6. In each iteration, a process only does the computation required for its local data. However, communication is required for two different operations: the dot-products needed to compute rho and alpha and the matrix-vector multiply needed to compute w_local. The dot products only require communication of a single value over all processes. On the other hand, the matrix-vector multiplication needed for w_local requires the global vector p, and, thus, more expensive communication. <i>Id.</i> at 11-13. MATLAB®, a commercial product of The MathWorks, Inc., has become one of the principal languages of desktop scientific computing. A system is described that enables one to run MATLAB conveniently on multiple processors. Using short, MATLAB-style commands like Eval, Send, Recv, Beast, Min, and Sum, the user operating within one MATLAB session can start MATLAB processes on other machines and then pass commands and data between between these various processes in a fashion that maintains MATLAB's traditional user-friendliness. Multi-processor graphics is also supported. The system currently runs under MPICH on an IBM SP2 or a network of Unix workstations, and extensions are planned to networks of PCs. MultiMATLAB is potentially useful for education in parallel programming, for prototyping parallel algorithms, and for fast and convenient execution of easily parallelizable numerical computations on multiple processors. Trefethen at Abstract. At least six reasons for MATLAB's success can be identified. The first is an exceptionally user-friendly, intuitive syntax, favoring brevity and simplicity at all turns without being so compressed as to interfere with intelligibility. The second is the very high quality of the underlying numerical

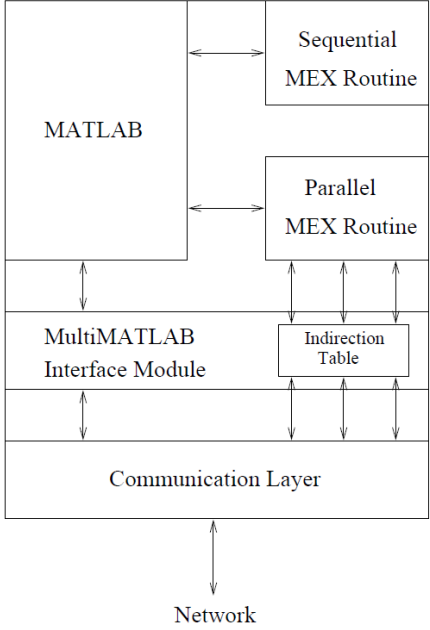
'582 Claim	Prior Art Reference – MultiMATLAB
	programs, a result of MATLAB's intimate ties from the beginning with the numerical analysis research community. The third is powerful and user-friendly graphics. <i>Id.</i> at 1.1. The vision of the MultiMATLAB project has been to bridge this gap. Think of a user who finds him- or herself computing happily in MATLAB, but frustrated by the time it takes to rerun the program for six different boundary conditions, or a dozen different parameter choices, or a hundred different initial guesses. Such a user's problems might be solved by a system that makes it convenient to spawn MATLAB processes on multiple processors of a parallel computer or a network of workstations or PCs. In many cases the needs for communication between the processors are rather small. Convenience of spreading the problem across machines and collecting the results numerically or graphically is paramount. <i>Id.</i> at 1.2. More general point-to-point communication is accomplished by send and receive commands, which can be executed on any of the MATLAB processes. For example, the sequence <pre>x = [pi pi^2]; Send(3,x) Eval(3, 'Recv')</pre> passes a message containing a 2-vector from the master process to process 3, leading to the output <pre>3.1416 9.8696</pre> An optional argument can be added in Recv to specify the source. <i>Id.</i> at 2.3; <i>see also id.</i> at 2.1.

'582 Claim	Prior Art Reference – MultiMATLAB
	<pre> 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; </pre> Process 0 creates the variable <code>a</code> with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of <code>a</code>, and sends it along to process 2; and so on. If there are six processors the command <code>Eval('cycle')</code> produces the output <pre> a = 1 a = 2 a = 4 a = 8 a = 16 a = 32 </pre> The processes run asynchronously, but since each <code>Send</code> command is only executed after the corresponding <code>Recv</code> has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning. <i>Id.</i> at 2.3. Equally often, however, it may be desirable to produce plots in a distributed fashion that are then sent to the user's screen. This can be particularly useful when one wishes to monitor the progress of computations on several processors graphically. <i>Id.</i> at 3. MultiMATLAB can be summarized in a few words. We run MATLAB processes on multiple processors, with full access to all the usual capabilities such as Toolboxes. These processes communicate via simple MATLAB-style commands built on MPI, with all message-passing details hidden as far as possible from the user. Both master/slave and SPMD paradigms are implemented, and attention is paid to multiprocessor graphics. All of this happens without any changes in the MATLAB architecture; indeed, we have not had access to the MATLAB source code. <i>Id.</i> at 6.

'582 Claim	Prior Art Reference – MultiMATLAB
	Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the '582 patent as charted for this claim limitation, and/or MultiMATLAB combined with the knowledge of one of ordinary skill in the art. Motivations to combine may come from the knowledge of the person of ordinary skill, or from the known problems and predictable solutions as embodied in these references. For example, to the extent any one of the MultiMATLAB printed publications lacks this element, it would have been obvious to combine with another of the MultiMATLAB printed publications that discloses this element, because they expressly cite to each other and/or to the same system, were published by overlapping research teams and/or in similar locations, and represent combinations of prior art elements according to known methods to yield predictable results. Further motivation to combine references and additional details may be found in the invalidity contentions cover pleading.
[3] The computer cluster of claim 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, wherein the first kernel and the first cluster node module reside in the first computer-readable medium, and wherein the second kernel and	MultiMATLAB discloses this claim limitation. See, e.g.: In particular, MATLAB offers several features that simplify numerical computing. First, it provides users with easy access to an extensive library of high quality numerical routines. Second, it is an intuitive, interactive environment that enables users to use these routines in a dynamic manner well suited to scientific problem solving. Third, it provides a high-level matrix based language that permits users to express computations in an exceptionally concise manner via matrix or vector formulations and often with a fraction of the code required in conventional languages such as C or Fortran. Fourth, MATLAB's graphical and visualization tools offer a simple but powerful mechanism to manipulate and analyze data more effectively. Menon at 1.

'582 Claim	Prior Art Reference – MultiMATLAB
the second cluster node module reside in the second computer-readable medium.	<pre> graph TD User[User] <--> IP[Interactive Processor] subgraph Processors IP P1[Processor] P2[Processor] P3[Processor] end subgraph MATLAB_Processes IP --- MP1[MATLAB Process] P1 --- MP2[MATLAB Process] P2 --- MP3[MATLAB Process] P3 --- MP4[MATLAB Process] end IP --- IN[Interconnecting Network] P1 --- IN P2 --- IN P3 --- IN </pre> The diagram illustrates the MultiMATLAB architecture. At the top, a 'User' is connected via a double-headed arrow to an 'Interactive Processor'. Below the 'Interactive Processor' is a 'MATLAB Process'. To the right of the 'Interactive Processor' are three 'Processor' blocks, each containing a 'MATLAB Process'. These components are connected to a common 'Interconnecting Network' at the bottom. Dotted lines indicate that there can be more than three processors. Figure 1: MultiMATLAB Architecture <i>Id.</i> at Fig. 1; see discussion at Fig. 1. In the MultiMatlab architecture, illustrated in Figure 1, each processor in a parallel platform individually runs a Matlab process. Each process is then provided with the ability to communicate with other processes through a communication layer that runs over the parallel platform's interconnection network. The user interacts directly with one Matlab process, called the interactive process, and operates within that process's Matlab environment. Other Matlab processes await commands from the interactive process. These other processes are used either by explicitly sending Matlab commands to them or by executing routines that, in turn, use them. The key component in this architecture is the MultiMatlab interface module. The interface module, shown in Figure 2, is responsible for initializing an underlying communication layer, such as MPI [2], PVM [3], BLACS [4], or any other package available on the platform, and exposing it to the rest of the system. MEX routines, also shown in the Figure 2, are executables originally written in C or Fortran that may be run directly from Matlab. All parallel functionality in the MultiMatlab system is provided through MEX routines. As a simple example, the MultiMatlab system provides users with an Eval

'582 Claim	Prior Art Reference – MultiMATLAB
	routine for parallel evaluation. This routine allows users to execute commands on the other processes. On the interactive process, the Eval routine is implemented as a MEX routine which accepts a vector of Matlab process IDs and a Matlab command. This routine sends the commands over the underlying communication layer to the processes corresponding to the specified IDs. On non-interactive processes, a separate top-level MEX routine is immediately run upon initialization of the system. This MEX routine runs in a loop in which it waits for Matlab commands to arrive from the interactive process and executes them in its own Matlab environment. All parallel MEX routines access the underlying communication layer and, hence, the network, through the MultiMatlab interface module. It is important to note that the interface module exists in the corresponding Matlab process's address space. For this reason, the interface module does not impose any additional context switch when parallel MEX routines access the communication layer. This is of particular importance when the parallel platform provides access to the network in user space. This enables parallel applications to access the network without operating system intervention as is often required for best performance. By residing within the Matlab process's address space, the MultiMatlab interface module is able to preserve this property for parallel MEX routines. <i>Id.</i> at 3-4.

'582 Claim	Prior Art Reference – MultiMATLAB
	 Figure 2: MATLAB Process Address Space <i>Id.</i> at Fig. 2; see discussion at Fig. 2. The MultiMatlab interface module provides MEX routines access to the underlying communication layer via an indirection table. Upon initialization, the interface module builds a table of pointers to all functions and data in the underlying communication layer that need to be exposed to parallel MEX routines. <i>Id.</i> at 5. Figures 7 and 8 compare the performance of three different MultiMatlab implementations with a stand-alone C implementation using MPI-F on up to 32 processors of the IBM SP2. <i>Id.</i> at 13.

'582 Claim	Prior Art Reference – MultiMATLAB
	MATLAB®, a commercial product of The MathWorks, Inc., has become one of the principal languages of desktop scientific computing. A system is described that enables one to run MATLAB conveniently on multiple processors. Using short, MATLAB-style commands like Eval, Send, Recv, Bcast, Min, and Sum, the user operating within one MATLAB session can start MATLAB processes on other machines and then pass commands and data between between these various processes in a fashion that maintains MATLAB's traditional user-friendliness. Multi-processor graphics is also supported. The system currently runs under MPICH on an IBM SP2 or a network of Unix workstations, and extensions are planned to networks of PCs. MultiMATLAB is potentially useful for education in parallel programming, for prototyping parallel algorithms, and for fast and convenient execution of easily parallelizable numerical computations on multiple processors. Trefethen at Abstract. MATLAB® is a high-level language, and a problem-solving environment, for mathematical and scientific calculations. It originated in the late 1970s with an attempt by Cleve Moler to provide interactive access to the Fortran linear algebra software packages EISPACK and LINPACK. Soon a programming language emerged (programs conventionally have the extension .m and are called "m-files") containing dozens of high-level commands such as svd (singular value decomposition), fft (fast Fourier transform), and roots (polynomial zero-finding). Graphical commands were built into the language, and a company called The MathWorks, Inc. was formed in 1984 by Moler and John Little, now based in Natick, Massachusetts. <i>Id.</i> at 1.1. The fifth is MATLAB's easy extensibility via packages of m-files known as Toolboxes. Many Toolboxes have been produced over the years, both by The MathWorks and by others, covering application areas such as optimization, signal processing, fuzzy logic, partial differential equations, and mathematical finance. Finally, perhaps the most interesting reason for MATLAB's success may be that from the beginning, the whole language has been built around real or complex vectors and matrices (including sparse matrices) as the fundamental data type. <i>Id.</i>

'582 Claim	Prior Art Reference – MultiMATLAB
	The vision of the MultiMATLAB project has been to bridge this gap. Think of a user who finds him- or herself computing happily in MATLAB, but frustrated by the time it takes to rerun the program for six different boundary conditions, or a dozen different parameter choices, or a hundred different initial guesses. Such a user's problems might be solved by a system that makes it convenient to spawn MATLAB processes on multiple processors of a parallel computer or a network of workstations or PCs. In many cases the needs for communication between the processors are rather small. Convenience of spreading the problem across machines and collecting the results numerically or graphically is paramount. <i>Id.</i> at 1.2. The standard MultiMATLAB command for executing commands on one or more processors is Eval. If the user now types Eval('sqrt(2)') then the MATLAB command sqrt(2) is executed on all six processors. The result is six repetitions of 1.4142. <i>Id.</i> The MultiMATLAB Start command builds a P4 process group file of remote hosts, which are either explicitly specified by the user or taken from a default list, and then initializes MPICH. MATLAB processes are then started on the remote hosts. Each process iterates over a simple loop, waiting for and executing commands received from the user's interactive MATLAB process. <i>Id.</i> at 4. The IBM RS/6000 SP is IBM's family of scalable, parallel computing solutions. It provides a state-of-the-art parallel computing system and industry-leading application enablers and applications. The RS/6000 SP runs the AIX operating system along with the Parallel System Support Programs (PSSP) system software on the control workstation and all processor nodes. For complete information on SP system software issues, see IBM RS/6000 SP: Planning Volume 2, Control Workstation and Software Environment.

'582 Claim	Prior Art Reference – MultiMATLAB
	The scalable architecture of the SP system, its high-performance communication, POWER3, POWER2, and PowerPC Architecture give you the power to handle data-intensive, compute-intensive and I/O-intensive jobs with ease. You can execute both serial and parallel applications simultaneously, while managing your system from a single workstation. RS6000 at 1. The basic components of the RS/6000 SP system are: *Processor nodes (includes SP-attached servers) * Frames with integral power subsystems * Switches * Extension nodes * Control workstations (a high availability option is available) * Network connectivity adapters* External disk drives The IBM RS/6000 SP System is scalable from one to 128 processor nodes. Up to sixteen thin, eight wide, or four high processor nodes can be mounted in a tall frame while a short frame can hold up to eight thin or four wide nodes. RS6000 at 2. Hard disk drives for the SP system can be either of two types as follows: Internal (contained within the node) External (mounted separately, outside of the node) RS6000 at 6; <i>see also</i> RS6000-2 at 3. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the '582 patent as charted for this claim limitation, and/or MultiMATLAB combined with the knowledge of one of ordinary skill in the art. Motivations to combine may come from the knowledge of the person of ordinary skill, or from the known problems and predictable solutions as embodied in these references. For example, to the extent any one of the MultiMATLAB printed publications lacks this element, it would have been obvious to combine with another of the MultiMATLAB printed publications that discloses this element, because they expressly cite to each other and/or to the same system, were published by overlapping research teams and/or in similar locations, and represent combinations of prior art elements according

'582 Claim	Prior Art Reference – MultiMATLAB
	to known methods to yield predictable results. Further motivation to combine references and additional details may be found in the invalidity contentions cover pleading.
[4] 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.	MultiMATLAB discloses this claim limitation. <i>See</i> disclosures at [1C], [1E], <i>supra</i>. See, also, e.g.: In the MultiMatlab architecture, illustrated in Figure 1, each processor in a parallel platform individually runs a Matlab process. Each process is then provided with the ability to communicate with other processes through a communication layer that runs over the parallel platform's interconnection network. The user interacts directly with one Matlab process, called the interactive process, and operates within that process's Matlab environment. Other Matlab processes await commands from the interactive process. These other processes are used either by explicitly sending Matlab commands to them or by executing routines that, in turn, use them. The key component in this architecture is the MultiMatlab interface module. The interface module, shown in Figure 2, is responsible for initializing an underlying communication layer, such as MPI [2], PVM [3], BLACS [4], or any other package available on the platform, and exposing it to the rest of the system. MEX routines, also shown in the Figure 2, are executables originally written in C or Fortran that may be run directly from Matlab. All parallel functionality in the MultiMatlab system is provided through MEX routines. As a simple example, the MultiMatlab system provides users with an Eval routine for parallel evaluation. This routine allows users to execute commands on the other processes. On the interactive process, the Eval routine is implemented as a MEX routine which accepts a vector of Matlab process IDs and a Matlab command. This routine sends the commands over the underlying communication layer to the processes corresponding to the specified IDs. On non-interactive processes, a separate top-level MEX routine is immediately run upon initialization of the system. This MEX routine runs in a loop in which it waits for Matlab commands to arrive from the interactive process and executes them in its own Matlab environment. All parallel MEX routines access the underlying communication layer and, hence, the network, through the MultiMatlab interface module.

'582 Claim	Prior Art Reference – MultiMATLAB
	It is important to note that the interface module exists in the corresponding Matlab process's address space. For this reason, the interface module does not impose any additional context switch when parallel MEX routines access the communication layer. This is of particular importance when the parallel platform provides access to the network in user space. This enables parallel applications to access the network without operating system intervention as is often required for best performance. By residing within the Matlab process's address space, the MultiMatlab interface module is able to preserve this property for parallel MEX routines. Menon at 3-4. The MultiMatlab interface module provides MEX routines access to the underlying communication layer via an indirection table. Upon initialization, the interface module builds a table of pointers to all functions and data in the underlying communication layer that need to be exposed to parallel MEX routines. <i>Id.</i> at 5. In this implementation, the interactive MATLAB process is started by the user as a normal MATLAB session. The user is provided with a Start command that initializes the MultiMATLAB system. This command builds a P4 process group file of remote hosts, which are either explicitly specified by the user or taken from a default list, and the initializes MPICH. <i>Id.</i> at 5. The MPI-F implementation of MultiMATLAB was design specifically for the IBM SP2. MPI-F is a highly optimized implementation of the MPI standard running over the SP2's high performance switch. This implementation permits access to the switch in user space. In this implementation, all MATLAB processes are started via POE, IBM's Parallel Operating Environment. For each of p MATLAB processes, POE assigns an identification number between 0 and p-1. The process numbered 0 is considered the interactive MATLAB. All other processes wait for commands from the interactive MATLAB process.

'582 Claim	Prior Art Reference – MultiMATLAB
	This implementation was designed to demonstrate the performance potential of the MultiMATLAB system. In particular, it provides a platform in which parallel libraries or applications written in MPI can easily be integrated into MATLAB on the SP2. <i>Id.</i> at 6. MATLAB®, a commercial product of The MathWorks, Inc., has become one of the principal languages of desktop scientific computing. A system is described that enables one to run MATLAB conveniently on multiple processors. Using short, MATLAB-style commands like Eval, Send, Recv, Bcast, Min, and Sum, the user operating within one MATLAB session can start MATLAB processes on other machines and then pass commands and data between these various processes in a fashion that maintains MATLAB's traditional user-friendliness. Multi-processor graphics is also supported. The system currently runs under MPICH on an IBM SP2 or a network of Unix workstations, and extensions are planned to networks of PCs. MultiMATLAB is potentially useful for education in parallel programming, for prototyping parallel algorithms, and for fast and convenient execution of easily parallelizable numerical computations on multiple processors. Trefethen at Abstract. Suppose the first author is sitting at her workstation in the Theory Center, connected to a node of the IBM SP2, running MATLAB. After a time she decides to start MATLAB on five new processors. She types Start(5) MATLAB is then started on five additional processors taken from a predetermined list. Or perhaps the second author is sitting at a machine connected to Cornell's Computer Science Department network. He types Start(['gemini'; 'orion'; 'rigel'; 'castor'; 'pollux']) Now MATLAB is started on the five processors with the names indicated.

'582 Claim	Prior Art Reference – MultiMATLAB
	<i>Id.</i> at 2.1. The MultiMATLAB Start command builds a P4 process group file of remote hosts, which are either explicitly specified by the user or taken from a default list, and then initializes MPICH. MATLAB processes are then started on the remote hosts. Each process iterates over a simple loop, waiting for and executing commands received from the user's interactive MATLAB process. <i>Id.</i> at 4. It is a straightforward matter to install our current software on any network of Unix workstations or SP2 system, provided that all the nodes are licensed to run MATLAB and there is a shared file system. We expect that extensions to networks of PCs running Windows, based on appropriate implementations of MPI, are not far behind. <i>Id.</i> at 6. In general, with POE, you invoke a parallel program from your home node and run its parallel tasks on a number of remote nodes. When you invoke a program on your home node, POE starts your Partition Manager which allocates the nodes of your partition and initializes the local environment. Depending on your hardware and configuration, the Partition Manager uses a host list file, LoadLeveler, or the SP system Resource Manager to allocate nodes. A host list file contains an explicit list of node requests, while LoadLeveler or the Resource Manager allocate nodes from one or more system pools implicitly based on their availability. On an SP system using the Resource Manager, you can also use a host list file to determine how an allocated node's resources – its SP switch adapter and CPU are used. POEref at 3. With regard to the expanded LoadLeveler function, POE now provides an option to enable you to specify whether your program will use MPI, LAPI, or both. Using this option, POE ensures that each API initializes properly and informs LoadLeveler which APIs are used so each node is set up completely.

'582 Claim	Prior Art Reference – MultiMATLAB
	For Single Program Multiple Data (SPMD) applications the Partition Manager executes the same program on all nodes. For Multiple Program Multiple Data (MPMD) applications, the Partition Manager prompts you for the name of the program to load on each node. The Partition Manager also connects standard I/O to each remote node so the parallel tasks can communicate with the home node. Although you are running tasks on remote nodes, POE allows you to continue using the standard UNIX** and AIX execution techniques with which you are already familiar. For example, you can redirect input and output, pipe the output of programs, or use shell tools. <i>Id.</i> When you invoke poe, the Partition Manager allocates processor nodes for each task and initializes the local environment. It then loads your program, and reproduces your local environment, on each processor node. The Partition Manager also passes the option list to each remote node. If your program is in a shared file system, the Partition Manager loads a copy of it on each node. If your program is in a private file system, you will have already manually copied your executable to the nodes using the mprcp or mcp command. If you are using the dynamic message passing interface, the appropriate communication subsystem library implementation (IP or US) is automatically loaded at this time. POEref at 35. <i>1) MultiMATLAB:</i> MultiMATLAB is perhaps one of the most well known parallel MATLABs. It was developed by A. Trefethen, V. Menon, C. Chang, G. Czajkowski, C. Myers, and L. Trefethen at Cornell University. In MultiMATLAB, a MATLAB user can start MATLAB processes on other machines and then pass commands and data between these various processes. The most important commands are: • Init—initialize the MultiMATLAB environment; • Nproc—returns the number of processes in the environment; equivalent of MPI Comm size(); • ID—returns the rank of the process; equivalent of MPI Comm rank(); • Send/Recv—sends and receives data between processes; • Eval—evaluates a string in every process in the environment.

'582 Claim	Prior Art Reference – MultiMATLAB
	Choy 2 at 333. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the '582 patent as charted for this claim limitation, and/or MultiMATLAB combined with the knowledge of one of ordinary skill in the art. Motivations to combine may come from the knowledge of the person of ordinary skill, or from the known problems and predictable solutions as embodied in these references. For example, to the extent any one of the MultiMATLAB printed publications lacks this element, it would have been obvious to combine with another of the MultiMATLAB printed publications that discloses this element, because they expressly cite to each other and/or to the same system, were published by overlapping research teams and/or in similar locations, and represent combinations of prior art elements according to known methods to yield predictable results. Further motivation to combine references and additional details may be found in the invalidity contentions cover pleading.
[6] 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.	MultiMATLAB discloses this claim limitation. See, e.g.: MATLAB®, a commercial product of The MathWorks, Inc., has become one of the principal languages of desktop scientific computing. A system is described that enables one to run MATLAB conveniently on multiple processors. Using short, MATLAB-style commands like Eval, Send, Recv, Bcast, Min, and Sum, the user operating within one MATLAB session can start MATLAB processes on other machines and then pass commands and data between between these various processes in a fashion that maintains MATLAB's traditional user-friendliness. Multi-processor graphics is also supported. The system currently runs under MPICH on an IBM SP2 or a network of Unix workstations, and extensions are planned to networks of PCs. MultiMATLAB is potentially useful for education in parallel programming, for prototyping parallel algorithms, and for fast and convenient execution of easily parallelizable numerical computations on multiple processors. Trefethen at Abstract. More general point-to-point communication is accomplished by send and receive commands, which can be executed on any of the MATLAB processes. For example, the sequence $x = [\pi \pi^2];$

'582 Claim	Prior Art Reference – MultiMATLAB
	Send(3,x) Eval(3, 'Recv') passes a message containing a 2-vector from the master process to process 3, leading to the output 3.1416 9.8696 An optional argument can be added in Recv to specify the source. <i>Id.</i> at 2.3; <i>see also id.</i> at 2.1. <pre> 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; </pre> Process 0 creates the variable <code>a</code> with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of <code>a</code>, and sends it along to process 2; and so on. If there are six processors the command <code>Eval('cycle')</code> produces the output <pre> a = 1 a = 2 a = 4 a = 8 a = 16 a = 32 </pre> The processes run asynchronously, but since each <code>Send</code> command is only executed after the corresponding <code>Recv</code> has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning. <i>Id.</i> at 2.3. At the user level, MultiMATLAB consists of a collection of commands such as <code>Send</code>, for example. Such a command is written as a C file called <code>Send.c</code>, which is interfaced to MATLAB via the standard MATLAB Fortran/C/C++ interface system known as MEX. Within MPI, many variants on sends and receives are defined. MultiMATLAB is currently built upon the standard send and receive

'582 Claim	Prior Art Reference – MultiMATLAB
	variants, which employ buffered communication for most messages and synchronous communication for very large ones. Our underlying MPI sends and receives are both blocking operations, to ensure that no data is overwritten, but to the MultiMATLAB programmer, the semantics is that Recv is blocking while Send is non-blocking. <i>Id.</i> at 4. The table compares round-trip latencies for a MultiMATLAB code with those for an equivalent C code using MPICH, and reveals that MultiMATLAB does add some overhead to that of MPICH. The timings were obtained on the IBM SP2, not using the high-performance switch. This occurs because MATLAB performs memory allocation for received matrices. It might be possible to alleviate this problem by maintaining a list of preallocated buffers, but we have not pursued this idea. <i>Id.</i> In the MultiMatlab architecture, illustrated in Figure 1, each processor in a parallel platform individually runs a Matlab process. Each process is then provided with the ability to communicate with other processes through a communication layer that runs over the parallel platform's interconnection network. The user interacts directly with one Matlab process, called the interactive process, and operates within that process's Matlab environment. Other Matlab processes await commands from the interactive process. These other processes are used either by explicitly sending Matlab commands to them or by executing routines that, in turn, use them. The key component in this architecture is the MultiMatlab interface module. The interface module, shown in Figure 2, is responsible for initializing an underlying communication layer, such as MPI [2], PVM [3], BLACS [4], or any other package available on the platform, and exposing it to the rest of the system. MEX routines, also shown in the Figure 2, are executables originally written in C or Fortran that may be run directly from Matlab. All parallel functionality in the MultiMatlab system is provided through MEX routines. As a simple example, the MultiMatlab system provides users with an Eval routine for parallel evaluation. This routine allows users to execute commands on the other

'582 Claim	Prior Art Reference – MultiMATLAB
	processes. On the interactive process, the Eval routine is implemented as a MEX routine which accepts a vector of Matlab process IDs and a Matlab command. This routine sends the commands over the underlying communication layer to the processes corresponding to the specified IDs. On non-interactive processes, a separate top-level MEX routine is immediately run upon initialization of the system. This MEX routine runs in a loop in which it waits for Matlab commands to arrive from the interactive process and executes them in its own Matlab environment. All parallel MEX routines access the underlying communication layer and, hence, the network, through the MultiMatlab interface module. It is important to note that the interface module exists in the corresponding Matlab process's address space. For this reason, the interface module does not impose any additional context switch when parallel MEX routines access the communication layer. This is of particular importance when the parallel platform provides access to the network in user space. This enables parallel applications to access the network without operating system intervention as is often required for best performance. By residing within the Matlab process's address space, the MultiMatlab interface module is able to preserve this property for parallel MEX routines. Menon at 3-4.

'582 Claim	Prior Art Reference – MultiMATLAB														
	Figure 2: MATLAB Process Address Space <i>Id.</i> at Fig. 2; see discussion at Fig. 2. <table><tr><th colspan="2">SPMD Communication</th></tr><tr><td>Send(pid,data)</td><td>Send data from one process to another.</td></tr><tr><td>Recv(pid)</td><td>Receive data sent from another process.</td></tr><tr><td>Barrier</td><td>Synchronize processes.</td></tr><tr><td>Bcast(pid,data)</td><td>Broadcasts data from processor pid to all processes.</td></tr><tr><td>Sum(data)</td><td>Adds data across all processors to form a global sum.</td></tr><tr><td>Collect(data)</td><td>Collects local data segments into a single global one.</td></tr></table> Table 1: Selected commands in the MultiMATLAB system	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.
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.														

'582 Claim	Prior Art Reference – MultiMATLAB																
	In order to accommodate applications requiring a finer degree of communication, we provide a set of MATLAB message passing routines analogous to those in MPI. In particular, we provide point to point and collective communication MATLAB routines operating directly on MATLAB data structures. We list a subset of these routines in Table 1. Although message passing is a relatively low-level concept, programming in MATLAB does simplify the process of developing message passing code. First, users may think in terms of matrices, matrix sections, or other MATLAB data structures instead of communication buffers. While the MATLAB communication routines are optimized for dense, real matrices that may be mapped almost directly to the underlying layer, they also work for arbitrary MATLAB data structures such as sparse matrices, cell arrays, and structures. <i>Id.</i> at 9 and Table 1. <table border="0"> <tr> <td>MPI_RECV (buf, count, datatype, source, tag, comm, request)</td><td>21</td></tr> <tr> <td>OUT buf</td><td>initial address of receive buffer (choice) 22</td></tr> <tr> <td>IN count</td><td>number of elements in receive buffer (integer) 23</td></tr> <tr> <td>IN datatype</td><td>datatype of each receive buffer element (handle) 24</td></tr> <tr> <td>IN source</td><td>rank of source (integer) 25</td></tr> <tr> <td>IN tag</td><td>message tag (integer) 26</td></tr> <tr> <td>IN comm</td><td>communicator (handle) 27</td></tr> <tr> <td>OUT request</td><td>communication request (handle) 28</td></tr> </table> <pre> int MPI_Irecv(void* buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Request *request) MPI_RECV(BUF, COUNT, DATATYPE, SOURCE, TAG, COMM, REQUEST, IERROR) <type> BUF(*) INTEGER COUNT, DATATYPE, SOURCE, TAG, COMM, REQUEST, IERROR Start a nonblocking receive. These calls allocate a communication request object and associate it with the request handle (the argument request). The request can be used later to query the status of the communication or wait for its completion. A nonblocking send call indicates that the system may start copying data out of the send buffer. The sender should not access any part of the send buffer after a nonblocking send operation is called, until the send completes. A nonblocking receive call indicates that the system may start writing data into the re- ceive buffer. The receiver should not access any part of the receive buffer after a nonblocking receive operation is called, until the receive completes. </pre>	MPI_RECV (buf, count, datatype, source, tag, comm, request)	21	OUT buf	initial address of receive buffer (choice) 22	IN count	number of elements in receive buffer (integer) 23	IN datatype	datatype of each receive buffer element (handle) 24	IN source	rank of source (integer) 25	IN tag	message tag (integer) 26	IN comm	communicator (handle) 27	OUT request	communication request (handle) 28
MPI_RECV (buf, count, datatype, source, tag, comm, request)	21																
OUT buf	initial address of receive buffer (choice) 22																
IN count	number of elements in receive buffer (integer) 23																
IN datatype	datatype of each receive buffer element (handle) 24																
IN source	rank of source (integer) 25																
IN tag	message tag (integer) 26																
IN comm	communicator (handle) 27																
OUT request	communication request (handle) 28																

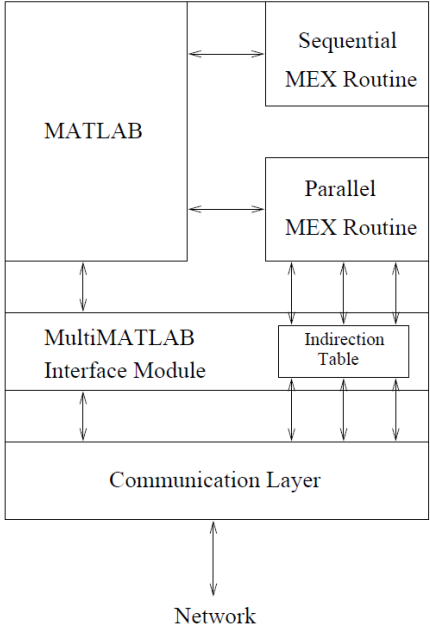
'582 Claim	Prior Art Reference – MultiMATLAB
	MPI Standard 1995 at 40. The mpilRecv[] command can also be called from within a kernel module 206. It creates a packet specifying where it expects to receive an expression and from which processor this expression is expected. Once received by its local cluster node module, this packet is decoded and its contents are stored in a message receiving queue (MRQ) 308 (FIG. 3). '582 patent 14:17–23. cator there are, conceptually at least, three distinct “message queues,” which represent the state of the MPI subsystem. These are Send Queue: represents all of the outstanding send operations. Receive Queue: represents all of the outstanding receive operations. Unexpected Message Queue: represents messages that have been sent to this process, but have not yet been received. The send and receive queues store information about all of the unfinished send and receive operations that the process has started within the communicator. These might result either from blocking operations such as MPI_Send and MPI_Recv or nonblocking operations such as MPI_Isend or MPI_Irecv. Each entry on one of these queues contains the information that was passed to the function call that initiated the operation. Nonblocking operations will remain on these queues until they have completed and have been collected by a suitable MPI_Wait, MPI_Test, or one of the related multiple completion routines. The unexpected message queue represents a different class of information, since the elements on this queue have been created by MPI calls in other processes. Therefore, less information is available about these elements. In all cases the order of the queues represents the order that the MPI subsystem will perform matching (this is important where many entries could match, for instance when wild-card tag or source is used in a receive operation). Cownie at 1-3, 6-8. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the '582 patent as charted for this claim limitation, and/or MultiMATLAB combined with the knowledge of one of ordinary skill in the art. Motivations to combine may come from the knowledge of the person of ordinary skill, or from the known problems and predictable solutions as embodied in these references. For example, to the extent any one of the MultiMATLAB printed publications lacks

'582 Claim	Prior Art Reference – MultiMATLAB
	this element, it would have been obvious to combine with another of the MultiMATLAB printed publications that discloses this element, because they expressly cite to each other and/or to the same system, were published by overlapping research teams and/or in similar locations, and represent combinations of prior art elements according to known methods to yield predictable results. Further motivation to combine references and additional details may be found in the invalidity contentions cover pleading.
[7] The computer cluster of claim 6, wherein the data structure of the first cluster node module comprises a received message queue.	MultiMATLAB discloses this claim limitation. <i>See</i> disclosures at [6], <i>supra</i>. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the '582 patent as charted for this claim limitation, and/or MultiMATLAB combined with the knowledge of one of ordinary skill in the art. Motivations to combine may come from the knowledge of the person of ordinary skill, or from the known problems and predictable solutions as embodied in these references. For example, to the extent any one of the MultiMATLAB printed publications lacks this element, it would have been obvious to combine with another of the MultiMATLAB printed publications that discloses this element, because they expressly cite to each other and/or to the same system, were published by overlapping research teams and/or in similar locations, and represent combinations of prior art elements according to known methods to yield predictable results. Further motivation to combine references and additional details may be found in the invalidity contentions cover pleading.
[8] 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	MultiMATLAB discloses this claim limitation. <i>See</i> disclosures at [7], <i>supra</i>. See, e.g.: MATLAB®, a commercial product of The MathWorks, Inc., has become one of the principal languages of desktop scientific computing. A system is described that enables one to run MATLAB conveniently on multiple processors. Using short, MATLAB-style commands like Eval, Send, Recv, Bcast, Min, and Sum, the user operating within one MATLAB session can start MATLAB processes on other machines and then pass commands and data between between these various processes in a fashion that maintains MATLAB's traditional user-friendliness. Multi-processor graphics is also supported. The system currently runs under MPICH on an IBM SP2 or a network of Unix workstations, and extensions are planned to networks of PCs. MultiMATLAB is potentially useful for education in parallel programming, for prototyping parallel algorithms, and for fast and convenient execution of easily parallelizable numerical computations on multiple processors.

'582 Claim	Prior Art Reference – MultiMATLAB
processor from which the expression is expected to be sent are stored.	Trefethen at Abstract. More general point-to-point communication is accomplished by send and receive commands, which can be executed on any of the MATLAB processes. For example, the sequence <pre>x = [pi pi^2];</pre> <pre>Send(3,x)</pre> <pre>Eval(3, 'Recv')</pre> passes a message containing a 2-vector from the master process to process 3, leading to the output <pre>3.1416 9.8696</pre> An optional argument can be added in Recv to specify the source. <i>Id.</i> at 2.3; <i>see also id.</i> at 2.1.

'582 Claim	Prior Art Reference – MultiMATLAB
	<pre> 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; </pre> Process 0 creates the variable <code>a</code> with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of <code>a</code>, and sends it along to process 2; and so on. If there are six processors the command <code>Eval('cycle')</code> produces the output <pre> a = 1 a = 2 a = 4 a = 8 a = 16 a = 32 </pre> The processes run asynchronously, but since each <code>Send</code> command is only executed after the corresponding <code>Recv</code> has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning. <i>Id.</i> at 2.3. At the user level, MultiMATLAB consists of a collection of commands such as <code>Send</code>, for example. Such a command is written as a C file called <code>Send.c</code>, which is interfaced to MATLAB via the standard MATLAB Fortran/C/C++ interface system known as MEX. Within MPI, many variants on sends and receives are defined. MultiMATLAB is currently built upon the standard send and receive variants, which employ buffered communication for most messages and synchronous communication for very large ones. Our underlying MPI sends and receives are both blocking operations, to ensure that no data is overwritten, but to the MultiMATLAB programmer, the semantics is that <code>Recv</code> is blocking while <code>Send</code> is non-blocking. <i>Id.</i> at 4. The table compares round-trip latencies for a MultiMATLAB code with those for an equivalent C code using MPICH, and reveals that MultiMATLAB does add some overhead to that of MPICH. The timings were obtained on the IBM SP2, not using the high-performance switch. This occurs because MATLAB performs memory allocation for received matrices. It might be possible to

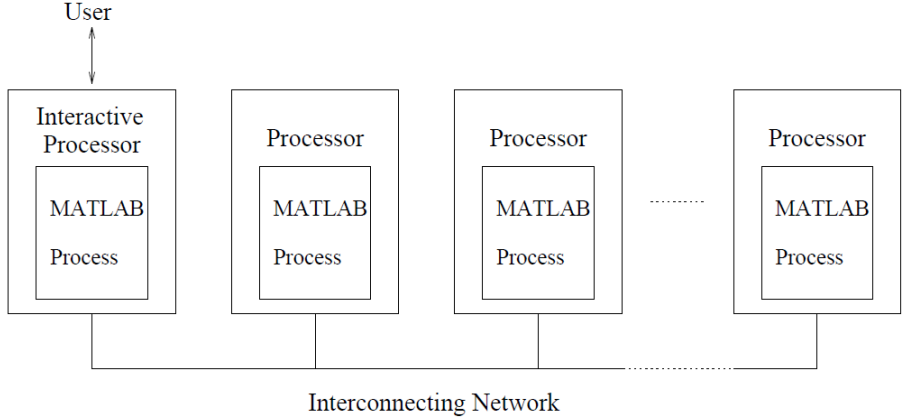
'582 Claim	Prior Art Reference – MultiMATLAB
	alleviate this problem by maintaining a list of preallocated buffers, but we have not pursued this idea. <i>Id.</i> In the MultiMatlab architecture, illustrated in Figure 1, each processor in a parallel platform individually runs a Matlab process. Each process is then provided with the ability to communicate with other processes through a communication layer that runs over the parallel platform's interconnection network. The user interacts directly with one Matlab process, called the interactive process, and operates within that process's Matlab environment. Other Matlab processes await commands from the interactive process. These other processes are used either by explicitly sending Matlab commands to them or by executing routines that, in turn, use them. The key component in this architecture is the MultiMatlab interface module. The interface module, shown in Figure 2, is responsible for initializing an underlying communication layer, such as MPI [2], PVM [3], BLACS [4], or any other package available on the platform, and exposing it to the rest of the system. MEX routines, also shown in the Figure 2, are executables originally written in C or Fortran that may be run directly from Matlab. All parallel functionality in the MultiMatlab system is provided through MEX routines. As a simple example, the MultiMatlab system provides users with an Eval routine for parallel evaluation. This routine allows users to execute commands on the other processes. On the interactive process, the Eval routine is implemented as a MEX routine which accepts a vector of Matlab process IDs and a Matlab command. This routine sends the commands over the underlying communication layer to the processes corresponding to the specified IDs. On non-interactive processes, a separate top-level MEX routine is immediately run upon initialization of the system. This MEX routine runs in a loop in which it waits for Matlab commands to arrive from the interactive process and executes them in its own Matlab environment. All parallel MEX routines access the underlying communication layer and, hence, the network, through the MultiMatlab interface module. It is important to note that the interface module exists in the corresponding Matlab process's address space. For this reason, the interface module does not impose any additional context switch when

'582 Claim	Prior Art Reference – MultiMATLAB
	parallel MEX routines access the communication layer. This is of particular importance when the parallel platform provides access to the network in user space. This enables parallel applications to access the network without operating system intervention as is often required for best performance. By residing within the Matlab process's address space, the MultiMatlab interface module is able to preserve this property for parallel MEX routines. Menon at 3-4.  Figure 2: MATLAB Process Address Space <i>Id.</i> at Fig. 2; see discussion at Fig. 2.

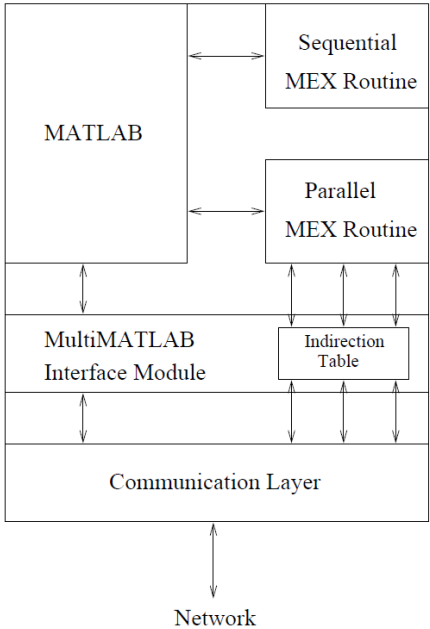
'582 Claim	Prior Art Reference – MultiMATLAB														
	<table border="1"> <thead> <tr> <th colspan="2">SPMD Communication</th></tr> </thead> <tbody> <tr> <td>Send(pid,data)</td><td>Send data from one process to another.</td></tr> <tr> <td>Recv(pid)</td><td>Receive data sent from another process.</td></tr> <tr> <td>Barrier</td><td>Synchronize processes.</td></tr> <tr> <td>Bcast(pid,data)</td><td>Broadcasts data from processor pid to all processes.</td></tr> <tr> <td>Sum(data)</td><td>Adds data across all processors to form a global sum.</td></tr> <tr> <td>Collect(data)</td><td>Collects local data segments into a single global one.</td></tr> </tbody> </table> Table 1: Selected commands in the MultiMATLAB system In order to accommodate applications requiring a finer degree of communication, we provide a set of MATLAB message passing routines analogous to those in MPI. In particular, we provide point to point and collective communication MATLAB routines operating directly on MATLAB data structures. We list a subset of these routines in Table 1. Although message passing is a relatively low-level concept, programming in MATLAB does simplify the process of developing message passing code. First, users may think in terms of matrices, matrix sections, or other MATLAB data structures instead of communication buffers. While the MATLAB communication routines are optimized for dense, real matrices that may be mapped almost directly to the underlying layer, they also work for arbitrary MATLAB data structures such as sparse matrices, cell arrays, and structures. <i>Id.</i> at 9 and Table 1.	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.
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.														

'582 Claim	Prior Art Reference – MultiMATLAB
	<pre> MPI_RECV(buf, count, datatype, source, tag, comm, request) 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 request communication request (handle) int MPI_Irecv(void* buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Request *request) MPI_RECV(BUF, COUNT, DATATYPE, SOURCE, TAG, COMM, REQUEST, IERROR) <type> BUF(*) INTEGER COUNT, DATATYPE, SOURCE, TAG, COMM, REQUEST, IERROR Start a nonblocking receive. These calls allocate a communication request object and associate it with the request handle (the argument request). The request can be used later to query the status of the communication or wait for its completion. A nonblocking send call indicates that the system may start copying data out of the send buffer. The sender should not access any part of the send buffer after a nonblocking send operation is called, until the send completes. A nonblocking receive call indicates that the system may start writing data into the re- ceive buffer. The receiver should not access any part of the receive buffer after a nonblocking receive operation is called, until the receive completes. </pre> MPI Standard 1995 at 40. The mpilRecv[] command can also be called from within a kernel module 206. It creates a packet specifying where it expects to receive an expression and from which processor this expression is expected. Once received by its local cluster node module, this packet is decoded and its contents are stored in a message receiving queue (MRQ) 308 (FIG. 3). '582 patent 14:17–23. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the '582 patent as charted for this claim limitation, and/or MultiMATLAB combined with the knowledge of one of ordinary skill in the art. Motivations to combine may come from the knowledge of the person of ordinary skill, or from the known problems and predictable solutions as embodied in these references. For example, to the extent any one of the MultiMATLAB printed publications lacks this element, it would have been obvious to combine with another of the MultiMATLAB printed publications that

'582 Claim	Prior Art Reference – MultiMATLAB
	discloses this element, because they expressly cite to each other and/or to the same system, were published by overlapping research teams and/or in similar locations, and represent combinations of prior art elements according to known methods to yield predictable results. Further motivation to combine references and additional details may be found in the invalidity contentions cover pleading.
[9] 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.	MultiMATLAB discloses this claim limitation. <i>See</i> disclosures at [1C], [1E], [6], [7], [8], <i>supra</i>. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the '582 patent as charted for this claim limitation, and/or MultiMATLAB combined with the knowledge of one of ordinary skill in the art. Motivations to combine may come from the knowledge of the person of ordinary skill, or from the known problems and predictable solutions as embodied in these references. For example, to the extent any one of the MultiMATLAB printed publications lacks this element, it would have been obvious to combine with another of the MultiMATLAB printed publications that discloses this element, because they expressly cite to each other and/or to the same system, were published by overlapping research teams and/or in similar locations, and represent combinations of prior art elements according to known methods to yield predictable results. Further motivation to combine references and additional details may be found in the invalidity contentions cover pleading.
[10] The computer cluster of claim 1, wherein the first cluster node module comprises an advanced functions module, and wherein the advanced functions module comprises a call that evaluates an expression across the	MultiMATLAB discloses this claim limitation. <i>See</i> disclosures at [1C], <i>supra</i>. See, also, e.g.: MultiMATLAB[1] is a general extension of the MATLAB environment to any distributed memory multiprocessors. This paper presents a new MultiMATLAB system designed to provide high-performance on multiprocessors while maintaining the functionality and usability of the MATLAB environment. This system will enable users to access high-performance parallel routines from within the MATLAB environment, to extent the environment with new parallel routines, and to use these routines to develop parallel applications with the MATLAB language. We discuss a general MultiMATLAB architecture, present two implementations based upon the MPI communication standard [2], and demonstrate the use of this system. Menon at 1.

'582 Claim	Prior Art Reference – MultiMATLAB
computer cluster in parallel.	In particular, MATLAB offers several features that simplify numerical computing. First, it provides users with easy access to an extensive library of high quality numerical routines. Second, it is an intuitive, interactive environment that enables users to use these routines in a dynamic manner well suited to scientific problem solving. Third, it provides a high-level matrix based language that permits users to express computations in an exceptionally concise manner via matrix or vector formulations and often with a fraction of the code required in conventional languages such as C or Fortran. Fourth, MATLAB's graphical and visualization tools offer a simple but powerful mechanism to manipulate and analyze data more effectively. <i>Id.</i> at 1.  The diagram illustrates the MultiMATLAB Architecture. At the top, a 'User' is connected via a double-headed arrow to an 'Interactive Processor'. This processor contains a 'MATLAB Process'. Below it, there are three 'Processor' blocks, each containing a 'MATLAB Process'. These processors are connected to a common 'Interconnecting Network' at the bottom. Dotted lines indicate that there are more processors than shown. Figure 1: MultiMATLAB Architecture <i>Id.</i> at Fig. 1; see discussion at Fig. 1. In the MultiMatlab architecture, illustrated in Figure 1, each processor in a parallel platform individually runs a Matlab process. Each process is then provided with the ability to communicate with other processes through a communication layer that runs over the parallel platform's interconnection network. The user interacts directly with one Matlab process, called the interactive process, and operates within that process's Matlab environment. Other Matlab processes await

'582 Claim	Prior Art Reference – MultiMATLAB
	commands from the interactive process. These other processes are used either by explicitly sending Matlab commands to them or by executing routines that, in turn, use them. The key component in this architecture is the MultiMatlab interface module. The interface module, shown in Figure 2, is responsible for initializing an underlying communication layer, such as MPI [2], PVM [3], BLACS [4], or any other package available on the platform, and exposing it to the rest of the system. MEX routines, also shown in the Figure 2, are executables originally written in C or Fortran that may be run directly from Matlab. All parallel functionality in the MultiMatlab system is provided through MEX routines. As a simple example, the MultiMatlab system provides users with an Eval routine for parallel evaluation. This routine allows users to execute commands on the other processes. On the interactive process, the Eval routine is implemented as a MEX routine which accepts a vector of Matlab process IDs and a Matlab command. This routine sends the commands over the underlying communication layer to the processes corresponding to the specified IDs. On non-interactive processes, a separate top-level MEX routine is immediately run upon initialization of the system. This MEX routine runs in a loop in which it waits for Matlab commands to arrive from the interactive process and executes them in its own Matlab environment. All parallel MEX routines access the underlying communication layer and, hence, the network, through the MultiMatlab interface module. It is important to note that the interface module exists in the corresponding Matlab process's address space. For this reason, the interface module does not impose any additional context switch when parallel MEX routines access the communication layer. This is of particular importance when the parallel platform provides access to the network in user space. This enables parallel applications to access the network without operating system intervention as is often required for best performance. By residing within the Matlab process's address space, the MultiMatlab interface module is able to preserve this property for parallel MEX routines. <i>Id.</i> at 3-4.

'582 Claim	Prior Art Reference – MultiMATLAB
	 Figure 2: MATLAB Process Address Space <i>Id.</i> at Fig. 2; see discussion at Fig. 2. The MultiMatlab interface module provides MEX routines access to the underlying communication layer via an indirection table. Upon initialization, the interface module builds a table of pointers to all functions and data in the underlying communication layer that need to be exposed to parallel MEX routines. <i>Id.</i> at 5. We present two implementations of the MultiMATLAB architecture. Both implementations use MPI as their underlying communication substrate. . . . One implementation of MultiMATLAB, designed to run on a generic network of Unix workstations, is based upon MPICH [5], a popular public domain version of MPI developed at Argonne National Laboratory and Mississippi State University.

'582 Claim	Prior Art Reference – MultiMATLAB
	The second was designed for the IBM SP2, a modern high performance distributed memory multiprocessor. This implementation is based upon MPI-F [6], IBM's proprietary version of MPI specifically optimized for the SP2. <i>Id.</i> at 5. The MPI-F implementation of MultiMATLAB was design specifically for the IBM SP2. MPI-F is a highly optimized implementation of the MPI standard running over the SP2's high performance switch. This implementation permits access to the switch in user space. In this implementation, all MATLAB processes are started via POE, IBM's Parallel Operating Environment. For each of p MATLAB processes, POE assigns an identification number between 0 and p-1. The process numbered 0 is considered the interactive MATLAB. All other processes wait for commands from the interactive MATLAB process. This implementation was designed to demonstrate the performance potential of the MultiMATLAB system. In particular, it provides a platform in which parallel libraries or applications written in MPI can easily be integrated into MATLAB on the SP2. <i>Id.</i> at 6. For example, in Matlab's Partial Differential Equation Toolkit, users may interactively create and solve rather general PDE systems using the finite element method. The PDE toolkit provides an intuitive graphical interface allowing users to specify geometries, differential equations, and boundary conditions. <i>Id.</i> at 6. Although message passing is a relatively low-level concept, programming in MATLAB does simplify the process of developing message passing code. First, users may think in terms of matrices, matrix sections, or other MATLAB data structures instead of communication buffers. While the MATLAB communication routines are optimized for dense, real matrices that may be

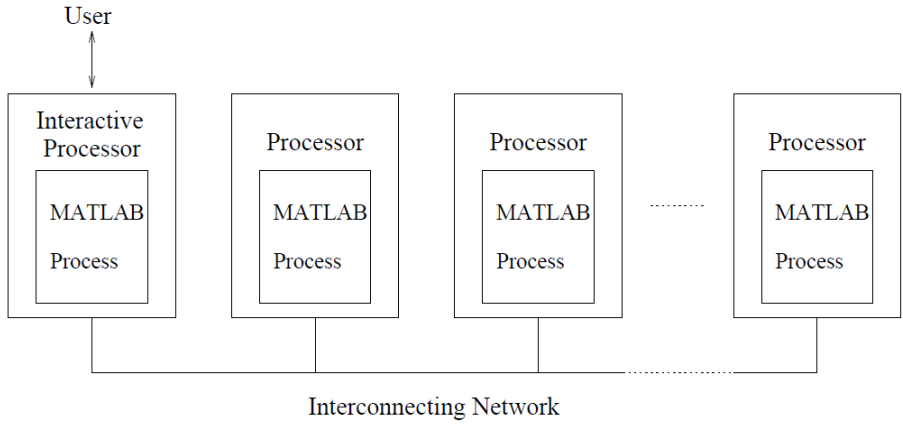
'582 Claim	Prior Art Reference – MultiMATLAB
	mapped almost directly to the underlying layer, they also work for arbitrary MATLAB data structures such as sparse matrices, cell arrays, and structures. <i>Id.</i> at 9 and Table 1. MATLAB®, a commercial product of The MathWorks, Inc., has become one of the principal languages of desktop scientific computing. A system is described that enables one to run MATLAB conveniently on multiple processors. Using short, MATLAB-style commands like Eval, Send, Recv, Beast, Min, and Sum, the user operating within one MATLAB session can start MATLAB processes on other machines and then pass commands and data between between these various processes in a fashion that maintains MATLAB's traditional user-friendliness. Multi-processor graphics is also supported. The system currently runs under MPICH on an IBM SP2 or a network of Unix workstations, and extensions are planned to networks of PCs. MultiMATLAB is potentially useful for education in parallel programming, for prototyping parallel algorithms, and for fast and convenient execution of easily parallelizable numerical computations on multiple processors. Trefethen at Abstract. MATLAB® is a high-level language, and a problem-solving environment, for mathematical and scientific calculations. It originated in the late 1970s with an attempt by Cleve Moler to provide interactive access to the Fortran linear algebra software packages EISPACK and LINPACK. Soon a programming language emerged (programs conventionally have the extension .m and are called "m-files") containing dozens of high-level commands such as svd (singular value decomposition), fft (fast Fourier transform), and roots (polynomial zero-finding). Graphical commands were built into the language, and a company called The MathWorks, Inc. was formed in 1984 by Moler and John Little, now based in Natick, Massachusetts. <i>Id.</i> at 1.1. The fifth is MATLAB's easy extensibility via packages of m-files known as Toolboxes. Many Toolboxes have been produced over the years, both by The MathWorks and by others, covering application areas such as optimization, signal processing, fuzzy logic, partial differential equations, and mathematical finance. Finally, perhaps the most interesting reason for MATLAB's success may

'582 Claim	Prior Art Reference – MultiMATLAB
	be that from the beginning, the whole language has been built around real or complex vectors and matrices (including sparse matrices) as the fundamental data type. <i>Id.</i> The vision of the MultiMATLAB project has been to bridge this gap. Think of a user who finds him- or herself computing happily in MATLAB, but frustrated by the time it takes to rerun the program for six different boundary conditions, or a dozen different parameter choices, or a hundred different initial guesses. Such a user's problems might be solved by a system that makes it convenient to spawn MATLAB processes on multiple processors of a parallel computer or a network of workstations or PCs. In many cases the needs for communication between the processors are rather small. Convenience of spreading the problem across machines and collecting the results numerically or graphically is paramount. <i>Id.</i> at 1.2. MultiMATLAB can be summarized in a few words. We run MATLAB processes on multiple processors, with full access to all the usual capabilities such as Toolboxes. These processes communicate via simple MATLAB-style commands built on MPI, with all message-passing details hidden as far as possible from the user. Both master/slave and SPMD paradigms are implemented, and attention is paid to multiprocessor graphics. All of this happens without any changes in the MATLAB architecture; indeed, we have not had access to the MATLAB source code. It is a straightforward matter to install our current software on any network of Unix workstations or SP2 system, provided that all the nodes are licensed to run MATLAB and there is a shared file system. We expect that extensions to networks of PCs running Windows, based on appropriate implementations of MPI, are not far behind. <i>Id.</i> at 6. While parallel MEX routines are capable of providing the same performance as regular MPI programs, they are, unfortunately, just as difficult to write. If users only require parallelism in a few common routines such as a linear system solver or a discrete Fourier transform, this may not be a

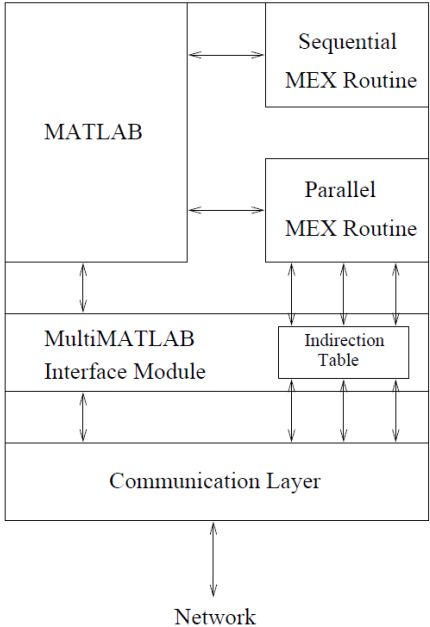
'582 Claim	Prior Art Reference – MultiMATLAB
	problem. These are the types of MEX routines that are likely to be either provided with a MultiMATLAB implementation or by a third party. Menon at 7. A possible alternative is to use parallel MEX routines that provide similar functionality to high-level MATLAB operations. These parallel MEX routines would perform matrix multiplications, solve for eigenvalues, compute Fourier transforms, and so on. In fact, these MEX routines could provide an interface identical to corresponding sequential MATLAB functions. In particular, they would distribute data from the interactive MATLAB process among all processes, perform the parallel computation, and collect results back to the interactive process. <i>Id.</i> at 10. MATLAB [?] is one of the most widely used tools in scientific and technical computing. It started in 1970s as an interactive interface to EISPACK [?] and LINPACK [?], a set of eigenvalue and linear system solution routines. It has since grown to a feature rich product utilizing modern numerical libraries like ATLAS [?] and FFTW [?], and with toolboxes in a number of application areas, for example, financial mathematics, neural networks, and control theory. It has a built-in interpreted language that is similar to C and HPF, and the flexible matrix indexing makes it very suitable for programming matrix problems. It is a very useful tool to a wide audience. For example, it comes in very handy as a matrix calculator for simple problems, because of its easy-to-use command syntax. Its strong graphical capabilities makes it a good data analysis tool. Also researchers have been known to build very complex systems using MATLAB scripts.

'582 Claim	Prior Art Reference – MultiMATLAB
	Choy 1 at 1. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the '582 patent as charted for this claim limitation, and/or MultiMATLAB combined with the knowledge of one of ordinary skill in the art. Motivations to combine may come from the knowledge of the person of ordinary skill, or from the known problems and predictable solutions as embodied in these references. For example, to the extent any one of the MultiMATLAB printed publications lacks this element, it would have been obvious to combine with another of the MultiMATLAB printed publications that discloses this element, because they expressly cite to each other and/or to the same system, were published by overlapping research teams and/or in similar locations, and represent combinations of prior art elements according to known methods to yield predictable results. Further motivation to combine references and additional details may be found in the invalidity contentions cover pleading.
[12] 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.	MultiMATLAB discloses this claim limitation. <i>See</i> disclosures at [10], [11], <i>supra</i>. See, also, e.g.: We can do this by taking a data-parallel approach in a simplistic fashion. We have developed a number of routines such as Distribute and Collect that allow a user to distribute a matrix or to collect a set of matrices into one large matrix. These functions operate using a mask that indicates which processors hold which portions of the matrix. This allows us also to develop routines such as Shift and Copy that are useful in data-parallel computing, keeping the communication to a more abstract level. Trefethen at 2.5. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the '582 patent as charted for this claim limitation, and/or MultiMATLAB combined with the knowledge of one of ordinary skill in the art. Motivations to combine may come from the knowledge of the person of ordinary skill, or from the known problems and predictable solutions as embodied in these references. For example, to the extent any one of the MultiMATLAB printed publications lacks this element, it would have been obvious to combine with another of the MultiMATLAB printed publications that discloses this element, because they expressly cite to each other and/or to the same system, were published by overlapping research teams and/or in similar locations, and represent combinations of prior art elements according

'582 Claim	Prior Art Reference – MultiMATLAB
	to known methods to yield predictable results. Further motivation to combine references and additional details may be found in the invalidity contentions cover pleading.
[13] The computer cluster of claim 1, wherein the final result comprises at least a portion of the second result.	MultiMATLAB discloses this claim limitation. <i>See</i> disclosures at [1H], <i>supra</i>. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the '582 patent as charted for this claim limitation, and/or MultiMATLAB combined with the knowledge of one of ordinary skill in the art. Motivations to combine may come from the knowledge of the person of ordinary skill, or from the known problems and predictable solutions as embodied in these references. For example, to the extent any one of the MultiMATLAB printed publications lacks this element, it would have been obvious to combine with another of the MultiMATLAB printed publications that discloses this element, because they expressly cite to each other and/or to the same system, were published by overlapping research teams and/or in similar locations, and represent combinations of prior art elements according to known methods to yield predictable results. Further motivation to combine references and additional details may be found in the invalidity contentions cover pleading.
[14] The computer cluster of claim 1, wherein the first cluster node module is configured to return the final result to the user interface.	MultiMATLAB discloses this claim limitation. <i>See</i> disclosures at [1H], <i>supra</i>. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the '582 patent as charted for this claim limitation, and/or MultiMATLAB combined with the knowledge of one of ordinary skill in the art. Motivations to combine may come from the knowledge of the person of ordinary skill, or from the known problems and predictable solutions as embodied in these references. For example, to the extent any one of the MultiMATLAB printed publications lacks this element, it would have been obvious to combine with another of the MultiMATLAB printed publications that discloses this element, because they expressly cite to each other and/or to the same system, were published by overlapping research teams and/or in similar locations, and represent combinations of prior art elements according to known methods to yield predictable results. Further motivation to combine references and additional details may be found in the invalidity contentions cover pleading.
[15] The computer cluster of claim 1, further comprising a communications	MultiMATLAB discloses this claim limitation. See, e.g.:

'582 Claim	Prior Art Reference – MultiMATLAB
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.	MultiMATLAB[1] is a general extension of the MATLAB environment to any distributed memory multiprocessors. This paper presents a new MultiMATLAB system designed to provide high-performance on multiprocessors while maintaining the functionality and usability of the MATLAB environment. This system will enable users to access high-performance parallel routines from within the MATLAB environment, to extent the environment with new parallel routines, and to use these routines to develop parallel applications with the MATLAB language. We discuss a general MultiMATLAB architecture, present two implementations based upon the MPI communication standard [2], and demonstrate the use of this system. Menon at 1.  The diagram illustrates the MultiMATLAB architecture. At the top, a 'User' is shown with a double-headed arrow pointing down to the first processor. This processor is labeled 'Interactive Processor' and contains a 'MATLAB Process' box. To its right are three more processors, each labeled 'Processor' and also containing a 'MATLAB Process' box. A dotted line indicates that there can be more than four processors. All four processors are connected to a common 'Interconnecting Network' at the bottom by vertical lines. Figure 1: MultiMATLAB Architecture <i>Id.</i> at Fig. 1; see discussion at Fig. 1. In the MultiMatlab architecture, illustrated in Figure 1, each processor in a parallel platform individually runs a Matlab process. Each process is then provided with the ability to communicate with other processes through a communication layer that runs over the parallel platform's interconnection network. The user interacts directly with one Matlab process, called the interactive process, and operates within that process's Matlab environment. Other Matlab processes await

'582 Claim	Prior Art Reference – MultiMATLAB
	commands from the interactive process. These other processes are used either by explicitly sending Matlab commands to them or by executing routines that, in turn, use them. The key component in this architecture is the MultiMatlab interface module. The interface module, shown in Figure 2, is responsible for initializing an underlying communication layer, such as MPI [2], PVM [3], BLACS [4], or any other package available on the platform, and exposing it to the rest of the system. MEX routines, also shown in the Figure 2, are executables originally written in C or Fortran that may be run directly from Matlab. All parallel functionality in the MultiMatlab system is provided through MEX routines. As a simple example, the MultiMatlab system provides users with an Eval routine for parallel evaluation. This routine allows users to execute commands on the other processes. On the interactive process, the Eval routine is implemented as a MEX routine which accepts a vector of Matlab process IDs and a Matlab command. This routine sends the commands over the underlying communication layer to the processes corresponding to the specified IDs. On non-interactive processes, a separate top-level MEX routine is immediately run upon initialization of the system. This MEX routine runs in a loop in which it waits for Matlab commands to arrive from the interactive process and executes them in its own Matlab environment. All parallel MEX routines access the underlying communication layer and, hence, the network, through the MultiMatlab interface module. It is important to note that the interface module exists in the corresponding Matlab process's address space. For this reason, the interface module does not impose any additional context switch when parallel MEX routines access the communication layer. This is of particular importance when the parallel platform provides access to the network in user space. This enables parallel applications to access the network without operating system intervention as is often required for best performance. By residing within the Matlab process's address space, the MultiMatlab interface module is able to preserve this property for parallel MEX routines. The MultiMatlab interface module provides MEX routines access to the underlying communication layer via an indirection table. Upon initialization, the interface module builds a table of pointers to all functions and data in the underlying communication layer that need to be exposed to parallel

'582 Claim	Prior Art Reference – MultiMATLAB
	MEX routines. When a parallel MEX routine is first loaded and executed, it accesses the MultiMatlab interface module through a function call to Matlab and is given the location of this table. This is the only time the MEX routine needs to directly access the interface module. Once the table is obtained, the MEX routine has the capability to access any function provided by the underlying communication layer through its corresponding o set in the table. In general, the MultiMatlab interface module adds an overhead of one load instruction per communication layer function call in a parallel MEX routine. As the execution time of an instruction is minimal compared to the latency of communicating a single packet of data, this overhead should be negligible. <i>Id.</i> at 3-5.  Figure 2: MATLAB Process Address Space <i>Id.</i> at Fig. 2; see discussion at Fig. 2.

'582 Claim	Prior Art Reference – MultiMATLAB
	We present two implementations of the MultiMATLAB architecture. Both implementations use MPI as their underlying communication substrate. . . . One implementation of MultiMATLAB, designed to run on a generic network of Unix workstations, is based upon MPICH [5], a popular public domain version of MPI developed at Argonne National Laboratory and Mississippi State University. The second was designed for the IBM SP2, a modern high performance distributed memory multiprocessor. This implementation is based upon MPI-F [6], IBM's proprietary version of MPI specifically optimized for the SP2. <i>Id.</i> at 5. The MPICH implementation of MultiMATLAB is based upon the P4 [7] communication subsystem, allowing it to run over a network of workstations connected by TCP/IP. In particular, this implementation should run on any platform providing both MATLAB and MPICH. To date, we have successfully run it on IBM RS6000 and Sun Sparc workstations and the IBM SP2. <i>Id.</i> at 5. The MPI-F implementation of MultiMATLAB was design specifically for the IBM SP2. MPI-F is a highly optimized implementation of the MPI standard running over the SP2's high performance switch. This implementation permits access to the switch in user space. In this implementation, all MATLAB processes are started via POE, IBM's Parallel Operating Environment. For each of p MATLAB processes, POE assigns an identification number between 0 and p-1. The process numbered 0 is considered the interactive MATLAB. All other processes wait for commands from the interactive MATLAB process. This implementation was designed to demonstrate the performance potential of the MultiMATLAB system. In particular, it provides a platform in which parallel libraries or applications written in MPI can easily be integrated into MATLAB on the SP2. <i>Id.</i> at 6.

'582 Claim	Prior Art Reference – MultiMATLAB														
	<table border="1" data-bbox="562 272 1732 527"> <tr> <th colspan="2">SPMD Communication</th></tr> <tr> <td>Send(pid,data)</td><td>Send data from one process to another.</td></tr> <tr> <td>Recv(pid)</td><td>Receive data sent from another process.</td></tr> <tr> <td>Barrier</td><td>Synchronize processes.</td></tr> <tr> <td>Bcast(pid,data)</td><td>Broadcasts data from processor pid to all processes.</td></tr> <tr> <td>Sum(data)</td><td>Adds data across all processors to form a global sum.</td></tr> <tr> <td>Collect(data)</td><td>Collects local data segments into a single global one.</td></tr> </table> Table 1: Selected commands in the MultiMATLAB system In order to accommodate applications requiring a finer degree of communication, we provide a set of MATLAB message passing routines analogous to those in MPI. In particular, we provide point to point and collective communication MATLAB routines operating directly on MATLAB data structures. We list a subset of these routines in Table 1. Although message passing is a relatively low-level concept, programming in MATLAB does simplify the process of developing message passing code. First, users may think in terms of matrices, matrix sections, or other MATLAB data structures instead of communication buffers. While the MATLAB communication routines are optimized for dense, real matrices that may be mapped almost directly to the underlying layer, they also work for arbitrary MATLAB data structures such as sparse matrices, cell arrays, and structures. <i>Id.</i> at 9 and Table 1. MATLAB®, a commercial product of The MathWorks, Inc., has become one of the principal languages of desktop scientific computing. A system is described that enables one to run MATLAB conveniently on multiple processors. Using short, MATLAB-style commands like Eval, Send, Recv, Bcast, Min, and Sum, the user operating within one MATLAB session can start MATLAB processes on other machines and then pass commands and data between between these various processes in a fashion that maintains MATLAB's traditional user-friendliness. Multi-processor graphics is also supported. The system currently runs under MPICH on an IBM SP2 or a network of Unix workstations, and extensions are planned to networks of PCs. MultiMATLAB is potentially useful for education in parallel programming, for prototyping parallel	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.
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.														

'582 Claim	Prior Art Reference – MultiMATLAB
	algorithms, and for fast and convenient execution of easily parallelizable numerical computations on multiple processors. Trefethen at Abstract. More general point-to-point communication is accomplished by send and receive commands, which can be executed on any of the MATLAB processes. For example, the sequence <pre>x = [pi pi^2];</pre> <pre>Send(3,x)</pre> <pre>Eval(3, 'Recv')</pre> passes a message containing a 2-vector from the master process to process 3, leading to the output <pre>3.1416 9.8696</pre> An optional argument can be added in Recv to specify the source. <i>Id.</i> at 2.3; <i>see also id.</i> at 2.1.

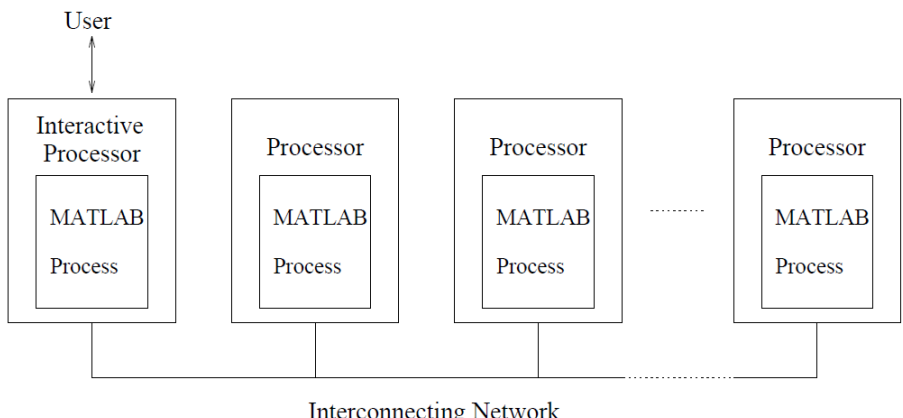
'582 Claim	Prior Art Reference – MultiMATLAB
	<pre> if ID==0 % first process: send a = 1 Send(ID+1,a) elseif ID == Nproc-1 % last process: receive and double a = 2*Recv else % middle processes: receive, double, and send a = 2*Recv Send(ID+1,a) end; </pre> Process 0 creates the variable <code>a</code> with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of <code>a</code>, and sends it along to process 2; and so on. If there are six processors the command <code>Eval('cycle')</code> produces the output <pre> a = 1 a = 2 a = 4 a = 8 a = 16 a = 32 </pre> The processes run asynchronously, but since each <code>Send</code> command is only executed after the corresponding <code>Recv</code> has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning. <i>Id.</i> at 2.3. Although <code>Send</code> takes a vector of processor IDs as its destination list, the underlying idea is that of point-to-point communication. For more efficient communication between multiple processes, as well as greater convenience for the programmer, MultiMATLAB also has various commands for collective communication. These commands must be evaluated simultaneously on all processes. <i>Id.</i> at 2.4; <i>see generally id.</i> at 2.4. We can do this by taking a data-parallel approach in a simplistic fashion. We have developed a number of routines such as <code>Distribute</code> and <code>Collect</code> that allow a user to distribute a matrix or to collect a set of matrices into one large matrix. These functions operate using a mask that indicates which processors hold which portions of the matrix. This allows us also to develop routines such as <code>Shift</code> and <code>Copy</code> that are useful in data-parallel computing, keeping the communication to a more abstract level. <i>Id.</i> at 2.5.

'582 Claim	Prior Art Reference – MultiMATLAB
	MultiMATLAB is built upon MPI (Message Passing Interface), a highly functional and portable message passing standard [7, 13]. Here is a brief description of how the system is put together. The system is written using MPICH, a popular and freely available implementation of MPI developed at Argonne National Laboratory and Mississippi State University [6]. In particular, MultiMATLAB uses the P4 communication layer within MPICH, allowing it to run over a heterogeneous network of workstations. In building upon MPICH, we believe we have developed a portable and extensible system, in that anyone can freely get a copy of the software and it will run on many systems. Versions of MPICH are beginning to become available that run on PCs running Windows, and we expect soon to experiment with MultiMATLAB on those platforms. <i>Id.</i> at 4. At the user level, MultiMATLAB consists of a collection of commands such as Send, for example. Such a command is written as a C file called Send.c, which is interfaced to MATLAB via the standard MATLAB Fortran/C/C++ interface system known as MEX. Within MPI, many variants on sends and receives are defined. MultiMATLAB is currently built upon the standard send and receive variants, which employ buffered communication for most messages and synchronous communication for very large ones. Our underlying MPI sends and receives are both blocking operations, to ensure that no data is overwritten, but to the MultiMATLAB programmer, the semantics is that Recv is blocking while Send is non-blocking. Higher-level MultiMATLAB commands are usually built on higher-level MPI commands. For example, Bcast and Min and Max and Sum are built on MPI collective communication routines, and Grid and Coord are built on MPI routines that support cartesian topologies Trefethen at 4. The basic components of the RS/6000 SP system are: *Processor nodes (includes SP-attached servers) * Frames with integral power subsystems * Switches * Extension nodes * Control workstations (a high availability option is available) * Network connectivity adapters* External disk drives

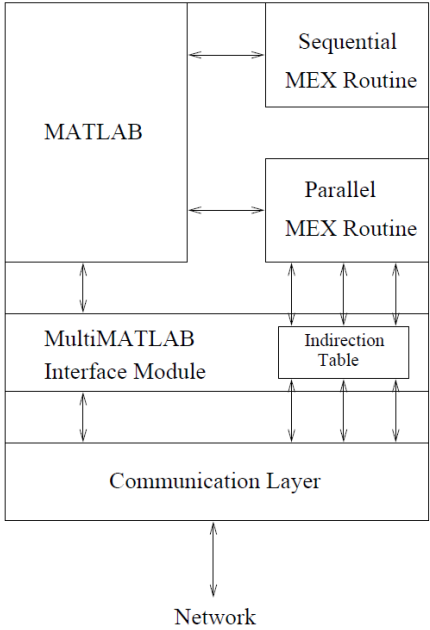
'582 Claim	Prior Art Reference – MultiMATLAB
	The IBM RS/6000 SP System is scalable from one to 128 processor nodes. Up to sixteen thin, eight wide, or four high processor nodes can be mounted in a tall frame while a short frame can hold up to eight thin or four wide nodes. RS6000 at 2. Chapter 3 Point-to-Point Communication 3.1 Introduction Sending and receiving of messages by processes is the basic MPI communication mechanism. The basic point-to-point communication operations are send and receive. Their use is illustrated in the example below. <pre> #include "mpi.h" int main(int argc, char *argv[]) { char message[20]; int myrank; MPI_Status status; MPI_Init(&argc, &argv); MPI_Comm_rank(MPI_COMM_WORLD, &myrank); if (myrank == 0) /* code for process zero */ { strcpy(message,"Hello, there"); MPI_Send(message, strlen(message)+1, MPI_CHAR, 1, 99, MPI_COMM_WORLD); } else if (myrank == 1) /* code for process one */ { MPI_Recv(message, 20, MPI_CHAR, 0, 99, MPI_COMM_WORLD, &status); printf("received :%s:\n", message); } MPI_Finalize(); return 0; } </pre> In this example, process zero (<i>myrank</i> = 0) sends a message to process one using the send operation MPI_SEND. The operation specifies a send buffer in the sender memory from which the message data is taken. In the example above, the send buffer consists of the storage containing the variable message in the memory of process zero. The location, size and type of the send buffer are specified by the first three parameters of the send operation. The message sent will contain the 13 characters of this variable. In addition, the send operation associates an envelope with the message. This envelope specifies the 23

'582 Claim	Prior Art Reference – MultiMATLAB																					
	MPI Standard 1994 at 23. MPI_ISEND(buf, count, datatype, dest, tag, comm, request) <table><tr><td>IN</td><td>buf</td><td>initial address of send buffer (choice)</td></tr><tr><td>IN</td><td>count</td><td>number of elements in send buffer (integer)</td></tr><tr><td>IN</td><td>datatype</td><td>datatype of each send buffer element (handle)</td></tr><tr><td>IN</td><td>dest</td><td>rank of destination (integer)</td></tr><tr><td>IN</td><td>tag</td><td>message tag (integer)</td></tr><tr><td>IN</td><td>comm</td><td>communicator (handle)</td></tr><tr><td>OUT</td><td>request</td><td>communication request (handle)</td></tr></table> int MPI_Isend(void* buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *request) MPI_ISEND(BUF, COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR) <type> BUF(*) INTEGER COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR Start a standard mode, nonblocking send. MPI Standard 1995 at 38; <i>see also</i> MPI Standard 1995 at 16-23.	IN	buf	initial address of send buffer (choice)	IN	count	number of elements in send buffer (integer)	IN	datatype	datatype of each send buffer element (handle)	IN	dest	rank of destination (integer)	IN	tag	message tag (integer)	IN	comm	communicator (handle)	OUT	request	communication request (handle)
IN	buf	initial address of send buffer (choice)																				
IN	count	number of elements in send buffer (integer)																				
IN	datatype	datatype of each send buffer element (handle)																				
IN	dest	rank of destination (integer)																				
IN	tag	message tag (integer)																				
IN	comm	communicator (handle)																				
OUT	request	communication request (handle)																				

'582 Claim	Prior Art Reference – MultiMATLAB																					
	MPI_RECV (buf, count, datatype, source, tag, comm, request) <table><tr><td>OUT</td><td>buf</td><td>initial address of receive buffer (choice)</td></tr><tr><td>IN</td><td>count</td><td>number of elements in receive buffer (integer)</td></tr><tr><td>IN</td><td>datatype</td><td>datatype of each receive buffer element (handle)</td></tr><tr><td>IN</td><td>source</td><td>rank of source (integer)</td></tr><tr><td>IN</td><td>tag</td><td>message tag (integer)</td></tr><tr><td>IN</td><td>comm</td><td>communicator (handle)</td></tr><tr><td>OUT</td><td>request</td><td>communication request (handle)</td></tr></table> int MPI_Irecv(void* buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Request *request) MPI_RECV(BUF, COUNT, DATATYPE, SOURCE, TAG, COMM, REQUEST, IERROR) <type> BUF(*) INTEGER COUNT, DATATYPE, SOURCE, TAG, COMM, REQUEST, IERROR Start a nonblocking receive. These calls allocate a communication request object and associate it with the request handle (the argument request). The request can be used later to query the status of the communication or wait for its completion. A nonblocking send call indicates that the system may start copying data out of the send buffer. The sender should not access any part of the send buffer after a nonblocking send operation is called, until the send completes. A nonblocking receive call indicates that the system may start writing data into the receive buffer. The receiver should not access any part of the receive buffer after a nonblocking receive operation is called, until the receive completes. MPI Standard 1995 at 40. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the '582 patent as charted for this claim limitation, and/or MultiMATLAB combined with the knowledge of one of ordinary skill in the art. Motivations to combine may come from the knowledge of the person of ordinary skill, or from the known problems and predictable solutions as embodied in these references. For example, to the extent any one of the MultiMATLAB printed publications lacks this element, it would have been obvious to combine with another of the MultiMATLAB printed publications that discloses this element, because they expressly cite to each other and/or to the same system, were published by overlapping research teams and/or in similar locations, and represent combinations of prior art elements according to known methods to yield predictable results. Further motivation to combine references and additional details may be found in the invalidity contentions cover pleading.	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	request	communication request (handle)
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	request	communication request (handle)																				

'582 Claim	Prior Art Reference – MultiMATLAB
[16] The computer cluster of claim 15, wherein one or more of the cluster node modules are configured to: accept user instructions; 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 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.	MultiMATLAB discloses this claim limitation. <i>See</i> disclosures at [15], <i>supra</i>. See, e.g.: In particular, MATLAB offers several features that simplify numerical computing. First, it provides users with easy access to an extensive library of high quality numerical routines. Second, it is an intuitive, interactive environment that enables users to use these routines in a dynamic manner well suited to scientific problem solving. Third, it provides a high-level matrix based language that permits users to express computations in an exceptionally concise manner via matrix or vector formulations and often with a fraction of the code required in conventional languages such as C or Fortran. Fourth, MATLAB's graphical and visualization tools offer a simple but powerful mechanism to manipulate and analyze data more effectively. Menon at 1. Users are provided interactive access to all processors in a multiprocessor within the MATLAB environment. This permits steering of parallel computation. <i>Id.</i> at 2.  <pre> graph TD User[User] <--> IP[Interactive Processor] subgraph IP_Box [Interactive Processor] MP1[MATLAB Process] end subgraph P1_Box [Processor] MP2[MATLAB Process] end subgraph P2_Box [Processor] MP3[MATLAB Process] end subgraph Pn_Box [Processor] MPn[MATLAB Process] end IP_Box --- IN[Interconnecting Network] P1_Box --- IN P2_Box --- IN Pn_Box --- IN </pre> Figure 1: MultiMATLAB Architecture

'582 Claim	Prior Art Reference – MultiMATLAB
	<i>Id.</i> at Fig. 1; see discussion at Fig. 1. In the MultiMatlab architecture, illustrated in Figure 1, each processor in a parallel platform individually runs a Matlab process. Each process is then provided with the ability to communicate with other processes through a communication layer that runs over the parallel platform's interconnection network. The user interacts directly with one Matlab process, called the interactive process, and operates within that process's Matlab environment. Other Matlab processes await commands from the interactive process. These other processes are used either by explicitly sending Matlab commands to them or by executing routines that, in turn, use them. The key component in this architecture is the MultiMatlab interface module. The interface module, shown in Figure 2, is responsible for initializing an underlying communication layer, such as MPI [2], PVM [3], BLACS [4], or any other package available on the platform, and exposing it to the rest of the system. MEX routines, also shown in the Figure 2, are executables originally written in C or Fortran that may be run directly from Matlab. All parallel functionality in the MultiMatlab system is provided through MEX routines. As a simple example, the MultiMatlab system provides users with an Eval routine for parallel evaluation. This routine allows users to execute commands on the other processes. On the interactive process, the Eval routine is implemented as a MEX routine which accepts a vector of Matlab process IDs and a Matlab command. This routine sends the commands over the underlying communication layer to the processes corresponding to the specified IDs. On non-interactive processes, a separate top-level MEX routine is immediately run upon initialization of the system. This MEX routine runs in a loop in which it waits for Matlab commands to arrive from the interactive process and executes them in its own Matlab environment. All parallel MEX routines access the underlying communication layer and, hence, the network, through the MultiMatlab interface module. It is important to note that the interface module exists in the corresponding Matlab process's address space. For this reason, the interface module does not impose any additional context switch when parallel MEX routines access the communication layer. This is of particular importance when the parallel platform provides access to the network in user space. This enables parallel applications to

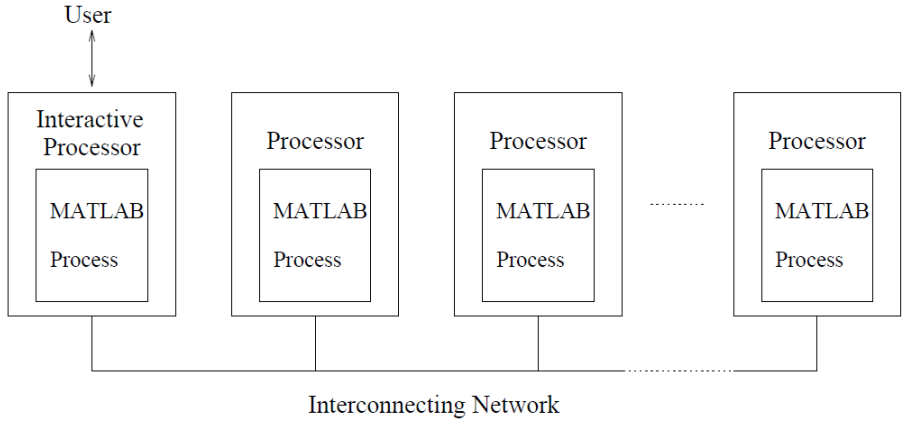
'582 Claim	Prior Art Reference – MultiMATLAB
	access the network without operating system intervention as is often required for best performance. By residing within the Matlab process's address space, the MultiMatlab interface module is able to preserve this property for parallel MEX routines. <i>Id.</i> at 3-4.  Figure 2: MATLAB Process Address Space <i>Id.</i> at Fig. 2; see discussion at Fig. 2. The MPI-F implementation of MultiMATLAB was design specifically for the IBM SP2. MPI-F is a highly optimized implementation of the MPI standard running over the SP2's high performance switch. This implementation permits access to the switch in user space.

'582 Claim	Prior Art Reference – MultiMATLAB
	In this implementation, all MATLAB processes are started via POE, IBM's Parallel Operating Environment. For each of p MATLAB processes, POE assigns an identification number between 0 and p-1. The process numbered 0 is considered the interactive MATLAB. All other processes wait for commands from the interactive MATLAB process. This implementation was designed to demonstrate the performance potential of the MultiMATLAB system. In particular, it provides a platform in which parallel libraries or applications written in MPI can easily be integrated into MATLAB on the SP2. <i>Id.</i> at 6. An important goal of the MultiMatlab system is to allow users to integrate high performance parallel routines into the Matlab environment. In particular, users should have the ability to replace computationally intensive parts of their programs with corresponding parallel routines. For example, in Matlab's Partial Differential Equation Toolkit, users may interactively create and solve rather general PDE systems using the finite element method. The PDE toolkit provides an intuitive graphical interface allowing users to specify geometries, differential equations, and boundary conditions. However, the computationally intensive core of this application consists of mesh generation and linear system solutions. With the MultiMatlab system, these computations could be done with high performance parallel routines. Menon at 6. Parallel MEX routines may be run in a SPMD fashion on any subset of MATLAB processes via the Eval command described in Section 2. The MultiMATLAB interface module provides parallel MEX routines with access to the underlying communication layer. In our implementations, MEX routines are granted access to MPI routines. Moreover, the indirection imposed by the interface is transparent to the programmer. <i>Id.</i> at 6. MATLAB®, a commercial product of The MathWorks, Inc., has become one of the principal languages of desktop scientific computing. A system is described that enables one to run MATLAB conveniently on multiple processors. Using short, MATLAB-style commands like Eval, Send, Recv, Bcast, Min, and Sum, the user operating within one MATLAB session can start MATLAB processes on other machines and then

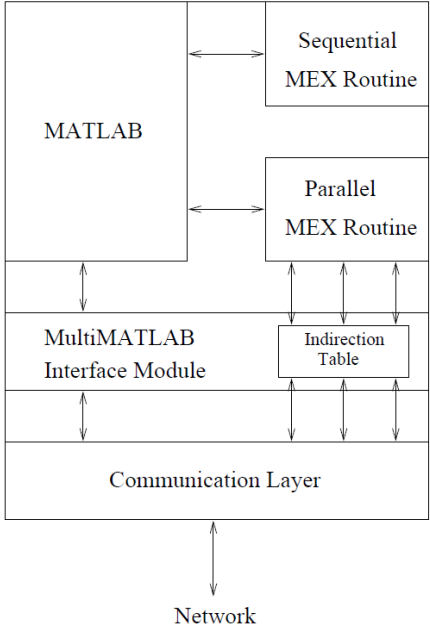
'582 Claim	Prior Art Reference – MultiMATLAB
	pass commands and data between between these various processes in a fashion that maintains MATLAB's traditional user-friendliness. Multi-processor graphics is also supported. The system currently runs under MPICH on an IBM SP2 or a network of Unix workstations, and extensions are planned to networks of PCs. MultiMATLAB is potentially useful for education in parallel programming, for prototyping parallel algorithms, and for fast and convenient execution of easily parallelizable numerical computations on multiple processors. Trefethen at Abstract. MATLAB® is a high-level language, and a problem-solving environment, for mathematical and scientific calculations. It originated in the late 1970s with an attempt by Cleve Moler to provide interactive access to the Fortran linear algebra software packages EISPACK and LINPACK. Soon a programming language emerged (programs conventionally have the extension .m and are called "m-files") containing dozens of high-level commands such as svd (singular value decomposition), fft (fast Fourier transform), and roots (polynomial zero-finding). Graphical commands were built into the language, and a company called The MathWorks, Inc. was formed in 1984 by Moler and John Little, now based in Natick, Massachusetts. <i>Id.</i> at 1.1. At least six reasons for MATLAB's success can be identified. The first is an exceptionally user-friendly, intuitive syntax, favoring brevity and simplicity at all turns without being so compressed as to interfere with intelligibility. The second is the very high quality of the underlying numerical programs, a result of MATLAB's intimate ties from the beginning with the numerical analysis research community. The third is powerful and user-friendly graphics. <i>Id.</i> at 1.1. The vision of the MultiMATLAB project has been to bridge this gap. Think of a user who finds him- or herself computing happily in MATLAB, but frustrated by the time it takes to rerun the program for six different boundary conditions, or a dozen different parameter choices, or a hundred different initial guesses. Such a user's problems might be solved by a system that makes it convenient to spawn MATLAB processes on multiple processors of a parallel computer or a network of workstations or PCs. In many cases the needs for communication between the processors are rather

'582 Claim	Prior Art Reference – MultiMATLAB
	small. Convenience of spreading the problem across machines and collecting the results numerically or graphically is paramount. <i>Id.</i> at 1.2. More general point-to-point communication is accomplished by send and receive commands, which can be executed on any of the MATLAB processes. For example, the sequence <pre>x = [pi pi^2]; Send(3,x) Eval(3, 'Recv')</pre> passes a message containing a 2-vector from the master process to process 3, leading to the output <pre>3.1416 9.8696</pre> An optional argument can be added in Recv to specify the source. <i>Id.</i> at 2.3; <i>see also id.</i> at 2.1.

'582 Claim	Prior Art Reference – MultiMATLAB
	<pre> 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; </pre> Process 0 creates the variable <code>a</code> with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of <code>a</code>, and sends it along to process 2; and so on. If there are six processors the command <code>Eval('cycle')</code> produces the output <pre> a = 1 a = 2 a = 4 a = 8 a = 16 a = 32 </pre> The processes run asynchronously, but since each <code>Send</code> command is only executed after the corresponding <code>Recv</code> has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning. <i>Id.</i> at 2.3. The MultiMATLAB Start command builds a P4 process group file of remote hosts, which are either explicitly specified by the user or taken from a default list, and then initializes MPICH. MATLAB processes are then started on the remote hosts. Each process iterates over a simple loop, waiting for and executing commands received from the user's interactive MATLAB process. <i>Id.</i> at 4. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the '582 patent as charted for this claim limitation, and/or MultiMATLAB combined with the knowledge of one of ordinary skill in the art. Motivations to combine may come from the knowledge of the person of ordinary skill, or from the known problems and predictable solutions as embodied in these references. For example, to the extent any one of the MultiMATLAB printed publications lacks this element, it would have been obvious to combine with another of the MultiMATLAB printed publications that discloses this element, because they expressly cite to each other and/or to the same system, were published by overlapping research teams and/or in similar locations, and represent combinations of prior art elements according

'582 Claim	Prior Art Reference – MultiMATLAB
	to known methods to yield predictable results. Further motivation to combine references and additional details may be found in the invalidity contentions cover pleading.
[17] 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.	MultiMATLAB discloses this claim limitation. See, e.g.: MultiMATLAB[1] is a general extension of the MATLAB environment to any distributed memory multiprocessors. This paper presents a new MultiMATLAB system designed to provide high-performance on multiprocessors while maintaining the functionality and usability of the MATLAB environment. This system will enable users to access high-performance parallel routines from within the MATLAB environment, to extent the environment with new parallel routines, and to use these routines to develop parallel applications with the MATLAB language. We discuss a general MultiMATLAB architecture, present two implementations based upon the MPI communication standard [2], and demonstrate the use of this system. Menon at 1; <i>see also</i> disclosures at [15], <i>supra</i>.  Figure 1: MultiMATLAB Architecture <i>Id.</i> at Fig. 1; see discussion at Fig. 1.

'582 Claim	Prior Art Reference – MultiMATLAB
	In the MultiMatlab architecture, illustrated in Figure 1, each processor in a parallel platform individually runs a Matlab process. Each process is then provided with the ability to communicate with other processes through a communication layer that runs over the parallel platform's interconnection network. The user interacts directly with one Matlab process, called the interactive process, and operates within that process's Matlab environment. Other Matlab processes await commands from the interactive process. These other processes are used either by explicitly sending Matlab commands to them or by executing routines that, in turn, use them. The key component in this architecture is the MultiMatlab interface module. The interface module, shown in Figure 2, is responsible for initializing an underlying communication layer, such as MPI [2], PVM [3], BLACS [4], or any other package available on the platform, and exposing it to the rest of the system. MEX routines, also shown in the Figure 2, are executables originally written in C or Fortran that may be run directly from Matlab. All parallel functionality in the MultiMatlab system is provided through MEX routines. As a simple example, the MultiMatlab system provides users with an Eval routine for parallel evaluation. This routine allows users to execute commands on the other processes. On the interactive process, the Eval routine is implemented as a MEX routine which accepts a vector of Matlab process IDs and a Matlab command. This routine sends the commands over the underlying communication layer to the processes corresponding to the specified IDs. On non-interactive processes, a separate top-level MEX routine is immediately run upon initialization of the system. This MEX routine runs in a loop in which it waits for Matlab commands to arrive from the interactive process and executes them in its own Matlab environment. All parallel MEX routines access the underlying communication layer and, hence, the network, through the MultiMatlab interface module. It is important to note that the interface module exists in the corresponding Matlab process's address space. For this reason, the interface module does not impose any additional context switch when parallel MEX routines access the communication layer. This is of particular importance when the parallel platform provides access to the network in user space. This enables parallel applications to access the network without operating system intervention as is often required for best performance.

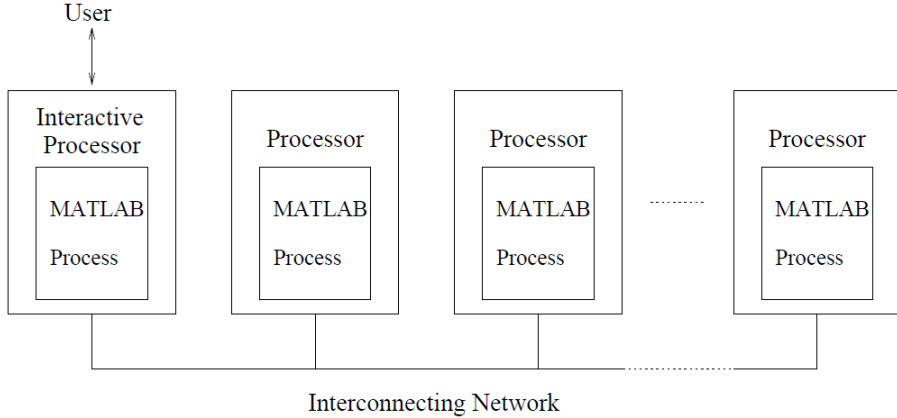
'582 Claim	Prior Art Reference – MultiMATLAB
	By residing within the Matlab process's address space, the MultiMatlab interface module is able to preserve this property for parallel MEX routines. <i>Id.</i> at 3-4.  Figure 2: MATLAB Process Address Space <i>Id.</i> at Fig. 2; see discussion at Fig. 2. We present two implementations of the MultiMATLAB architecture. Both implementations use MPI as their underlying communication substrate. . . . One implementation of MultiMATLAB, designed to run on a generic network of Unix workstations, is based upon MPICH [5], a popular public domain version of MPI developed at Argonne National Laboratory and Mississippi State University. The second was designed for the IBM SP2, a modern high performance distributed memory

'582 Claim	Prior Art Reference – MultiMATLAB
	multiprocessor. This implementation is based upon MPI-F [6], IBM's proprietary version of MPI specifically optimized for the SP2. <i>Id.</i> at 5. The MPICH implementation of MultiMATLAB is based upon the P4 [7] communication subsystem, allowing it to run over a network of workstations connected by TCP/IP. In particular, this implementation should run on any platform providing both MATLAB and MPICH. To date, we have successfully run it on IBM RS6000 and Sun Sparc workstations and the IBM SP2. <i>Id.</i> at 5. The MPI-F implementation of MultiMATLAB was design specifically for the IBM SP2. MPI-F is a highly optimized implementation of the MPI standard running over the SP2's high performance switch. This implementation permits access to the switch in user space. In this implementation, all MATLAB processes are started via POE, IBM's Parallel Operating Environment. For each of p MATLAB processes, POE assigns an identification number between 0 and p-1. The process numbered 0 is considered the interactive MATLAB. All other processes wait for commands from the interactive MATLAB process. This implementation was designed to demonstrate the performance potential of the MultiMATLAB system. In particular, it provides a platform in which parallel libraries or applications written in MPI can easily be integrated into MATLAB on the SP2. <i>Id.</i> at 6.

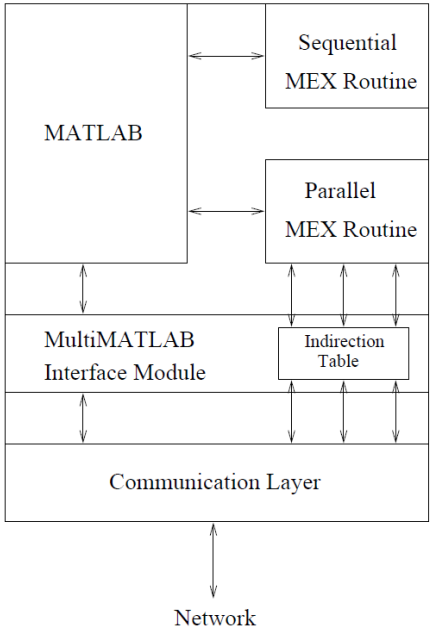
'582 Claim	Prior Art Reference – MultiMATLAB														
	<table border="1" data-bbox="562 272 1732 527"> <tr> <th colspan="2">SPMD Communication</th></tr> <tr> <td>Send(pid,data)</td><td>Send data from one process to another.</td></tr> <tr> <td>Recv(pid)</td><td>Receive data sent from another process.</td></tr> <tr> <td>Barrier</td><td>Synchronize processes.</td></tr> <tr> <td>Bcast(pid,data)</td><td>Broadcasts data from processor pid to all processes.</td></tr> <tr> <td>Sum(data)</td><td>Adds data across all processors to form a global sum.</td></tr> <tr> <td>Collect(data)</td><td>Collects local data segments into a single global one.</td></tr> </table> Table 1: Selected commands in the MultiMATLAB system In order to accommodate applications requiring a finer degree of communication, we provide a set of MATLAB message passing routines analogous to those in MPI. In particular, we provide point to point and collective communication MATLAB routines operating directly on MATLAB data structures. We list a subset of these routines in Table 1. Although message passing is a relatively low-level concept, programming in MATLAB does simplify the process of developing message passing code. First, users may think in terms of matrices, matrix sections, or other MATLAB data structures instead of communication buffers. While the MATLAB communication routines are optimized for dense, real matrices that may be mapped almost directly to the underlying layer, they also work for arbitrary MATLAB data structures such as sparse matrices, cell arrays, and structures. <i>Id.</i> at 9 and Table 1. MATLAB®, a commercial product of The MathWorks, Inc., has become one of the principal languages of desktop scientific computing. A system is described that enables one to run MATLAB conveniently on multiple processors. Using short, MATLAB-style commands like Eval, Send, Recv, Bcast, Min, and Sum, the user operating within one MATLAB session can start MATLAB processes on other machines and then pass commands and data between between these various processes in a fashion that maintains MATLAB's traditional user-friendliness. Multi-processor graphics is also supported. The system currently runs under MPICH on an IBM SP2 or a network of Unix workstations, and extensions are planned to networks of PCs. MultiMATLAB is potentially useful for education in parallel programming, for prototyping parallel	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.
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.														

'582 Claim	Prior Art Reference – MultiMATLAB
	algorithms, and for fast and convenient execution of easily parallelizable numerical computations on multiple processors. Trefethen at Abstract. Although Send takes a vector of processor IDs as its destination list, the underlying idea is that of point-to-point communication. For more efficient communication between multiple processes, as well as greater convenience for the programmer, MultiMATLAB also has various commands for collective communication. These commands must be evaluated simultaneously on all processes. <i>Id.</i> at 2.4. MultiMATLAB is built upon MPI (Message Passing Interface), a highly functional and portable message passing standard [7, 13]. Here is a brief description of how the system is put together. The system is written using MPICH, a popular and freely available implementation of MPI developed at Argonne National Laboratory and Mississippi State University [6]. In particular, MultiMATLAB uses the P4 communication layer within MPICH, allowing it to run over a heterogeneous network of workstations. In building upon MPICH, we believe we have developed a portable and extensible system, in that anyone can freely get a copy of the software and it will run on many systems. Versions of MPICH are beginning to become available that run on PCs running Windows, and we expect soon to experiment with MultiMATLAB on those platforms. <i>Id.</i> at 4. The IBM RS/6000 SP is IBM's family of scalable, parallel computing solutions. It provides a state-of-the-art parallel computing system and industry-leading application enablers and applications. The RS/6000 SP runs the AIX operating system along with the Parallel System Support Programs (PSSP) system software on the control workstation and all processor nodes. For complete information on SP system software issues, see IBM RS/6000 SP: Planning Volume 2, Control Workstation and Software Environment. The scalable architecture of the SP system, its high-performance communication, POWER3, POWER2, and PowerPC Architecture give you the power to handle data-intensive, compute-intensive and I/O-intensive

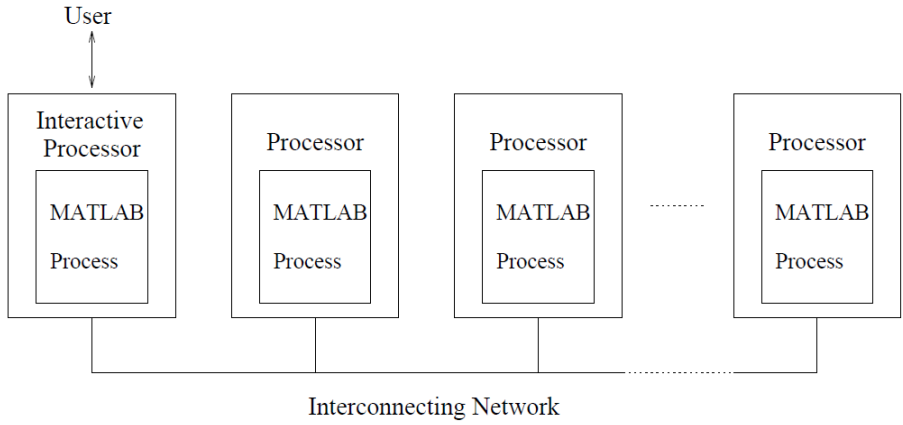
'582 Claim	Prior Art Reference – MultiMATLAB
	jobs with ease. You can execute both serial and parallel applications simultaneously, while managing your system from a single workstation. RS6000 at 1. The basic components of the RS/6000 SP system are: *Processor nodes (includes SP-attached servers) * Frames with integral power subsystems * Switches * Extension nodes * Control workstations (a high availability option is available) * Network connectivity adapters* External disk drives The IBM RS/6000 SP System is scalable from one to 128 processor nodes. Up to sixteen thin, eight wide, or four high processor nodes can be mounted in a tall frame while a short frame can hold up to eight thin or four wide nodes. RS6000 at 2. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the '582 patent as charted for this claim limitation, and/or MultiMATLAB combined with the knowledge of one of ordinary skill in the art. Motivations to combine may come from the knowledge of the person of ordinary skill, or from the known problems and predictable solutions as embodied in these references. For example, to the extent any one of the MultiMATLAB printed publications lacks this element, it would have been obvious to combine with another of the MultiMATLAB printed publications that discloses this element, because they expressly cite to each other and/or to the same system, were published by overlapping research teams and/or in similar locations, and represent combinations of prior art elements according to known methods to yield predictable results. Further motivation to combine references and additional details may be found in the invalidity contentions cover pleading.
[18] The computer cluster of claim 1, wherein at least one of the first, second, or third kernels	MultiMATLAB discloses this claim limitation. See, e.g.: In particular, MATLAB offers several features that simplify numerical computing. First, it provides users with easy access to an extensive library of high quality numerical routines. Second, it is an intuitive, interactive environment that enables users to use these routines in a dynamic manner well suited to scientific

'582 Claim	Prior Art Reference – MultiMATLAB
comprises a single-node kernel.	problem solving. Third, it provides a high-level matrix based language that permits users to express computations in an exceptionally concise manner via matrix or vector formulations and often with a fraction of the code required in conventional languages such as C or Fortran. Fourth, MATLAB's graphical and visualization tools offer a simple but powerful mechanism to manipulate and analyze data more effectively. Menon at 1.  The diagram illustrates the MultiMATLAB architecture. At the top, a 'User' is connected via a double-headed arrow to an 'Interactive Processor'. This processor contains a 'MATLAB Process'. Below it, there are several 'Processor' blocks, each also containing a 'MATLAB Process'. These processors are connected to a common 'Interconnecting Network' at the bottom. The first processor is connected to the network, followed by a gap indicated by a dotted line, and then the final processor is connected to the network. Figure 1: MultiMATLAB Architecture <i>Id.</i> at Fig. 1; see discussion at Fig. 1. In the MultiMatlab architecture, illustrated in Figure 1, each processor in a parallel platform individually runs a Matlab process. Each process is then provided with the ability to communicate with other processes through a communication layer that runs over the parallel platform's interconnection network. The user interacts directly with one Matlab process, called the interactive process, and operates within that process's Matlab environment. Other Matlab processes await commands from the interactive process. These other processes are used either by explicitly sending Matlab commands to them or by executing routines that, in turn, use them. The key component in this architecture is the MultiMatlab interface module. The interface module, shown in Figure 2, is responsible for initializing an underlying communication layer, such as MPI

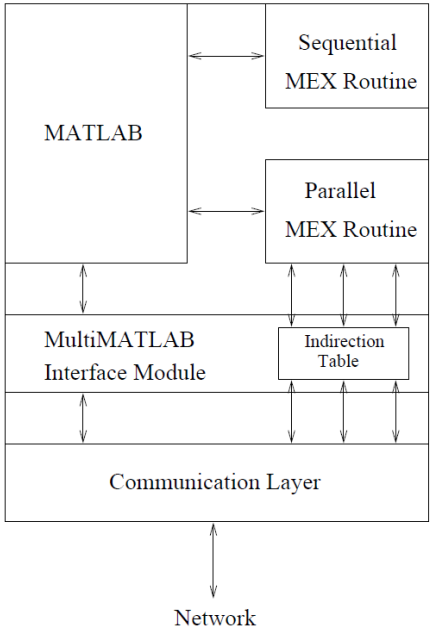
'582 Claim	Prior Art Reference – MultiMATLAB
	[2], PVM [3], BLACS [4], or any other package available on the platform, and exposing it to the rest of the system. MEX routines, also shown in the Figure 2, are executables originally written in C or Fortran that may be run directly from Matlab. All parallel functionality in the MultiMatlab system is provided through MEX routines. As a simple example, the MultiMatlab system provides users with an Eval routine for parallel evaluation. This routine allows users to execute commands on the other processes. On the interactive process, the Eval routine is implemented as a MEX routine which accepts a vector of Matlab process IDs and a Matlab command. This routine sends the commands over the underlying communication layer to the processes corresponding to the specified IDs. On non-interactive processes, a separate top-level MEX routine is immediately run upon initialization of the system. This MEX routine runs in a loop in which it waits for Matlab commands to arrive from the interactive process and executes them in its own Matlab environment. All parallel MEX routines access the underlying communication layer and, hence, the network, through the MultiMatlab interface module. It is important to note that the interface module exists in the corresponding Matlab process's address space. For this reason, the interface module does not impose any additional context switch when parallel MEX routines access the communication layer. This is of particular importance when the parallel platform provides access to the network in user space. This enables parallel applications to access the network without operating system intervention as is often required for best performance. By residing within the Matlab process's address space, the MultiMatlab interface module is able to preserve this property for parallel MEX routines. <i>Id.</i> at 3-4.

'582 Claim	Prior Art Reference – MultiMATLAB
	 Figure 2: MATLAB Process Address Space <i>Id.</i> at Fig. 2; see discussion at Fig. 2. MATLAB® is a high-level language, and a problem-solving environment, for mathematical and scientific calculations. It originated in the late 1970s with an attempt by Cleve Moler to provide interactive access to the Fortran linear algebra software packages EISPACK and LINPACK. Soon a programming language emerged (programs conventionally have the extension .m and are called "m-files") containing dozens of high-level commands such as svd (singular value decomposition), fft (fast Fourier transform), and roots (polynomial zero-finding). Graphical commands were built into the language, and a company called The MathWorks, Inc. was formed in 1984 by Moler and John Little, now based in Natick, Massachusetts. Trefethen at 1.1.

'582 Claim	Prior Art Reference – MultiMATLAB
	The fifth is MATLAB's easy extensibility via packages of m-files known as Toolboxes. Many Toolboxes have been produced over the years, both by The MathWorks and by others, covering application areas such as optimization, signal processing, fuzzy logic, partial differential equations, and mathematical finance. Finally, perhaps the most interesting reason for MATLAB's success may be that from the beginning, the whole language has been built around real or complex vectors and matrices (including sparse matrices) as the fundamental data type. <i>Id.</i> The standard MultiMATLAB command for executing commands on one or more processors is Eval. If the user now types Eval('sqrt(2)') then the MATLAB command sqrt(2) is executed on all six processors. The result is six repetitions of 1.4142. <i>Id.</i> Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the '582 patent as charted for this claim limitation, and/or MultiMATLAB combined with the knowledge of one of ordinary skill in the art. Motivations to combine may come from the knowledge of the person of ordinary skill, or from the known problems and predictable solutions as embodied in these references. For example, to the extent any one of the MultiMATLAB printed publications lacks this element, it would have been obvious to combine with another of the MultiMATLAB printed publications that discloses this element, because they expressly cite to each other and/or to the same system, were published by overlapping research teams and/or in similar locations, and represent combinations of prior art elements according to known methods to yield predictable results. Further motivation to combine references and additional details may be found in the invalidity contentions cover pleading.
[19] The computer cluster of claim 1, wherein the cluster	MultiMATLAB discloses this claim limitation. <i>See</i> disclosures at [1C], [1E], [1G], <i>supra</i>. See, also, e.g.:

'582 Claim	Prior Art Reference – MultiMATLAB
node modules communicate with one another to act as a cluster.	MultiMATLAB[1] is a general extension of the MATLAB environment to any distributed memory multiprocessors. This paper presents a new MultiMATLAB system designed to provide high-performance on multiprocessors while maintaining the functionality and usability of the MATLAB environment. This system will enable users to access high-performance parallel routines from within the MATLAB environment, to extent the environment with new parallel routines, and to use these routines to develop parallel applications with the MATLAB language. We discuss a general MultiMATLAB architecture, present two implementations based upon the MPI communication standard [2], and demonstrate the use of this system. Menon at 1.  The diagram illustrates the MultiMATLAB architecture. At the top, a 'User' is shown with a double-headed arrow pointing down to the first processor. This processor is labeled 'Interactive Processor' and contains a 'MATLAB Process'. To its right are several 'Processor' blocks, each also containing a 'MATLAB Process'. A dotted line indicates that there are more processors. All these processors are connected to a common 'Interconnecting Network' at the bottom, represented by a horizontal line with vertical connections to each processor block. Figure 1: MultiMATLAB Architecture <i>Id.</i> at Fig. 1; see discussion at Fig. 1. In the MultiMatlab architecture, illustrated in Figure 1, each processor in a parallel platform individually runs a Matlab process. Each process is then provided with the ability to communicate with other processes through a communication layer that runs over the parallel platform's interconnection network. The user interacts directly with one Matlab process, called the interactive process, and operates within that process's Matlab environment. Other Matlab processes await

'582 Claim	Prior Art Reference – MultiMATLAB
	commands from the interactive process. These other processes are used either by explicitly sending Matlab commands to them or by executing routines that, in turn, use them. The key component in this architecture is the MultiMatlab interface module. The interface module, shown in Figure 2, is responsible for initializing an underlying communication layer, such as MPI [2], PVM [3], BLACS [4], or any other package available on the platform, and exposing it to the rest of the system. MEX routines, also shown in the Figure 2, are executables originally written in C or Fortran that may be run directly from Matlab. All parallel functionality in the MultiMatlab system is provided through MEX routines. As a simple example, the MultiMatlab system provides users with an Eval routine for parallel evaluation. This routine allows users to execute commands on the other processes. On the interactive process, the Eval routine is implemented as a MEX routine which accepts a vector of Matlab process IDs and a Matlab command. This routine sends the commands over the underlying communication layer to the processes corresponding to the specified IDs. On non-interactive processes, a separate top-level MEX routine is immediately run upon initialization of the system. This MEX routine runs in a loop in which it waits for Matlab commands to arrive from the interactive process and executes them in its own Matlab environment. All parallel MEX routines access the underlying communication layer and, hence, the network, through the MultiMatlab interface module. It is important to note that the interface module exists in the corresponding Matlab process's address space. For this reason, the interface module does not impose any additional context switch when parallel MEX routines access the communication layer. This is of particular importance when the parallel platform provides access to the network in user space. This enables parallel applications to access the network without operating system intervention as is often required for best performance. By residing within the Matlab process's address space, the MultiMatlab interface module is able to preserve this property for parallel MEX routines. <i>Id.</i> at 3-4.

'582 Claim	Prior Art Reference – MultiMATLAB
	 Figure 2: MATLAB Process Address Space <i>Id.</i> at Fig. 2; see discussion at Fig. 2. We present two implementations of the MultiMATLAB architecture. Both implementations use MPI as their underlying communication substrate. . . . One implementation of MultiMATLAB, designed to run on a generic network of Unix workstations, is based upon MPICH [5], a popular public domain version of MPI developed at Argonne National Laboratory and Mississippi State University. The second was designed for the IBM SP2, a modern high performance distributed memory multiprocessor. This implementation is based upon MPI-F [6], IBM's proprietary version of MPI specifically optimized for the SP2. <i>Id.</i> at 5.

'582 Claim	Prior Art Reference – MultiMATLAB														
	The MPICH implementation of MultiMATLAB is based upon the P4 [7] communication subsystem, allowing it to run over a network of workstations connected by TCP/IP. In particular, this implementation should run on any platform providing both MATLAB and MPICH. To date, we have successfully run it on IBM RS6000 and Sun Sparc workstations and the IBM SP2. <i>Id.</i> at 5. The MPI-F implementation of MultiMATLAB was design specifically for the IBM SP2. MPI-F is a highly optimized implementation of the MPI standard running over the SP2's high performance switch. This implementation permits access to the switch in user space. In this implementation, all MATLAB processes are started via POE, IBM's Parallel Operating Environment. For each of p MATLAB processes, POE assigns an identification number between 0 and p-1. The process numbered 0 is considered the interactive MATLAB. All other processes wait for commands from the interactive MATLAB process. This implementation was designed to demonstrate the performance potential of the MultiMATLAB system. In particular, it provides a platform in which parallel libraries or applications written in MPI can easily be integrated into MATLAB on the SP2. <i>Id.</i> at 6. <table border="1" data-bbox="562 1008 1734 1265"> <thead> <tr> <th colspan="2">SPMD Communication</th></tr> </thead> <tbody> <tr> <td>Send(pid,data)</td><td>Send data from one process to another.</td></tr> <tr> <td>Recv(pid)</td><td>Receive data sent from another process.</td></tr> <tr> <td>Barrier</td><td>Synchronize processes.</td></tr> <tr> <td>Bcast(pid,data)</td><td>Broadcasts data from processor pid to all processes.</td></tr> <tr> <td>Sum(data)</td><td>Adds data across all processors to form a global sum.</td></tr> <tr> <td>Collect(data)</td><td>Collects local data segments into a single global one.</td></tr> </tbody> </table> Table 1: Selected commands in the MultiMATLAB system	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.
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.														

'582 Claim	Prior Art Reference – MultiMATLAB
	In order to accommodate applications requiring a finer degree of communication, we provide a set of MATLAB message passing routines analogous to those in MPI. In particular, we provide point to point and collective communication MATLAB routines operating directly on MATLAB data structures. We list a subset of these routines in Table 1. Although message passing is a relatively low-level concept, programming in MATLAB does simplify the process of developing message passing code. First, users may think in terms of matrices, matrix sections, or other MATLAB data structures instead of communication buffers. While the MATLAB communication routines are optimized for dense, real matrices that may be mapped almost directly to the underlying layer, they also work for arbitrary MATLAB data structures such as sparse matrices, cell arrays, and structures. <i>Id.</i> at 9 and Table 1. MATLAB®, a commercial product of The MathWorks, Inc., has become one of the principal languages of desktop scientific computing. A system is described that enables one to run MATLAB conveniently on multiple processors. Using short, MATLAB-style commands like Eval, Send, Recv, Bcast, Min, and Sum, the user operating within one MATLAB session can start MATLAB processes on other machines and then pass commands and data between between these various processes in a fashion that maintains MATLAB's traditional user-friendliness. Multi-processor graphics is also supported. The system currently runs under MPICH on an IBM SP2 or a network of Unix workstations, and extensions are planned to networks of PCs. MultiMATLAB is potentially useful for education in parallel programming, for prototyping parallel algorithms, and for fast and convenient execution of easily parallelizable numerical computations on multiple processors. Trefethen at Abstract. More general point-to-point communication is accomplished by send and receive commands, which can be executed on any of the MATLAB processes. For example, the sequence <pre>x = [pi pi^2]; Send(3,x)</pre>

'582 Claim	Prior Art Reference – MultiMATLAB
	Eval(3, 'Recv') passes a message containing a 2-vector from the master process to process 3, leading to the output 3.1416 9.8696 An optional argument can be added in Recv to specify the source. <i>Id.</i> at 2.3; <i>see also id.</i> at 2.1. Although Send takes a vector of processor IDs as its destination list, the underlying idea is that of point-to-point communication. For more efficient communication between multiple processes, as well as greater convenience for the programmer, MultiMATLAB also has various commands for collective communication. These commands must be evaluated simultaneously on all processes. <i>Id.</i> at 2.4. MultiMATLAB is built upon MPI (Message Passing Interface), a highly functional and portable message passing standard [7, 13]. Here is a brief description of how the system is put together. The system is written using MPICH, a popular and freely available implementation of MPI developed at Argonne National Laboratory and Mississippi State University [6]. In particular, MultiMATLAB uses the P4 communication layer within MPICH, allowing it to run over a heterogeneous network of workstations. In building upon MPICH, we believe we have developed a portable and extensible system, in that anyone can freely get a copy of the software and it will run on many systems. Versions of MPICH are beginning to become available that run on PCs running Windows, and we expect soon to experiment with MultiMATLAB on those platforms. <i>Id.</i> at 4; <i>see also</i> disclosures at [15], <i>supra</i>. The IBM RS/6000 SP is IBM's family of scalable, parallel computing solutions. It provides a state-of-the-art parallel computing system and industry-leading application enablers and applications. The RS/6000 SP runs the AIX operating system along with the Parallel System Support Programs (PSSP) system software on the

'582 Claim	Prior Art Reference – MultiMATLAB
	control workstation and all processor nodes. For complete information on SP system software issues, see IBM RS/6000 SP: Planning Volume 2, Control Workstation and Software Environment. The scalable architecture of the SP system, its high-performance communication, POWER3, POWER2, and PowerPC Architecture give you the power to handle data-intensive, compute-intensive and I/O-intensive jobs with ease. You can execute both serial and parallel applications simultaneously, while managing your system from a single workstation. RS6000 at 1. The basic components of the RS/6000 SP system are: *Processor nodes (includes SP-attached servers) * Frames with integral power subsystems * Switches * Extension nodes * Control workstations (a high availability option is available) * Network connectivity adapters* External disk drives The IBM RS/6000 SP System is scalable from one to 128 processor nodes. Up to sixteen thin, eight wide, or four high processor nodes can be mounted in a tall frame while a short frame can hold up to eight thin or four wide nodes. RS6000 at 2.

'582 Claim	Prior Art Reference – MultiMATLAB
	Chapter 3 Point-to-Point Communication 3.1 Introduction Sending and receiving of messages by processes is the basic MPI communication mechanism. The basic point-to-point communication operations are send and receive. Their use is illustrated in the example below. <pre>#include "mpi.h" int main(int argc, char *argv[]) { char message[20]; int myrank; MPI_Status status; MPI_Init(&argc, &argv); MPI_Comm_rank(MPI_COMM_WORLD, &myrank); if (myrank == 0) /* code for process zero */ { strcpy(message,"Hello, there"); MPI_Send(message, strlen(message)+1, MPI_CHAR, 1, 99, MPI_COMM_WORLD); } else if (myrank == 1) /* code for process one */ { MPI_Recv(message, 20, MPI_CHAR, 0, 99, MPI_COMM_WORLD, &status); printf("received :%s:\n", message); } MPI_Finalize(); return 0; }</pre> In this example, process zero (<i>myrank</i> = 0) sends a message to process one using the send operation MPI_SEND. The operation specifies a send buffer in the sender memory from which the message data is taken. In the example above, the send buffer consists of the storage containing the variable message in the memory of process zero. The location, size and type of the send buffer are specified by the first three parameters of the send operation. The message sent will contain the 13 characters of this variable. In addition, the send operation associates an envelope with the message. This envelope specifies the 23 MPI Standard 1994 at 23.

'582 Claim	Prior Art Reference – MultiMATLAB																					
	MPI_ISEND(buf, count, datatype, dest, tag, comm, request) <table><tr><td>IN</td><td>buf</td><td>initial address of send buffer (choice)</td></tr><tr><td>IN</td><td>count</td><td>number of elements in send buffer (integer)</td></tr><tr><td>IN</td><td>datatype</td><td>datatype of each send buffer element (handle)</td></tr><tr><td>IN</td><td>dest</td><td>rank of destination (integer)</td></tr><tr><td>IN</td><td>tag</td><td>message tag (integer)</td></tr><tr><td>IN</td><td>comm</td><td>communicator (handle)</td></tr><tr><td>OUT</td><td>request</td><td>communication request (handle)</td></tr></table> int MPI_Isend(void* buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *request) MPI_ISEND(BUF, COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR) <type> BUF(*) INTEGER COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR Start a standard mode, nonblocking send. MPI Standard 1995 at 38; <i>see also</i> MPI Standard 1995 at 16-23.	IN	buf	initial address of send buffer (choice)	IN	count	number of elements in send buffer (integer)	IN	datatype	datatype of each send buffer element (handle)	IN	dest	rank of destination (integer)	IN	tag	message tag (integer)	IN	comm	communicator (handle)	OUT	request	communication request (handle)
IN	buf	initial address of send buffer (choice)																				
IN	count	number of elements in send buffer (integer)																				
IN	datatype	datatype of each send buffer element (handle)																				
IN	dest	rank of destination (integer)																				
IN	tag	message tag (integer)																				
IN	comm	communicator (handle)																				
OUT	request	communication request (handle)																				

'582 Claim	Prior Art Reference – MultiMATLAB																					
	MPI_RECV (buf, count, datatype, source, tag, comm, request) <table><tr><td>OUT</td><td>buf</td><td>initial address of receive buffer (choice)</td></tr><tr><td>IN</td><td>count</td><td>number of elements in receive buffer (integer)</td></tr><tr><td>IN</td><td>datatype</td><td>datatype of each receive buffer element (handle)</td></tr><tr><td>IN</td><td>source</td><td>rank of source (integer)</td></tr><tr><td>IN</td><td>tag</td><td>message tag (integer)</td></tr><tr><td>IN</td><td>comm</td><td>communicator (handle)</td></tr><tr><td>OUT</td><td>request</td><td>communication request (handle)</td></tr></table> int MPI_Irecv(void* buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Request *request) MPI_RECV(BUF, COUNT, DATATYPE, SOURCE, TAG, COMM, REQUEST, IERR) <type> BUF(*) INTEGER COUNT, DATATYPE, SOURCE, TAG, COMM, REQUEST, IERR Start a nonblocking receive. These calls allocate a communication request object and associate it with the request handle (the argument request). The request can be used later to query the status of the communication or wait for its completion. A nonblocking send call indicates that the system may start copying data out of the send buffer. The sender should not access any part of the send buffer after a nonblocking send operation is called, until the send completes. A nonblocking receive call indicates that the system may start writing data into the receive buffer. The receiver should not access any part of the receive buffer after a nonblocking receive operation is called, until the receive completes. MPI Standard 1995 at 40. The PE is a distributed memory message passing system. It runs on the RS/6000 platform using the AIX operating system (Version 4.2.1). Specifically, you can use the PE to execute parallel programs on: - any configuration of the IBM RS/6000 SP as described in the <i>IBM Parallel System Support Programs for AIX: Installation and Migration Guide</i> Essentially, an SP system is a collection of RS/6000 processors grouped into a number of frames. Each frame of an SP system can contain from two to 16 RS/6000 processors. - a networked cluster of RS/6000 processors, including a single processor or a single workstation. - a mixed system. In a mixed system, additional RS/6000 processors supplement the processors of an SP system.	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	request	communication request (handle)
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	request	communication request (handle)																				

'582 Claim	Prior Art Reference – MultiMATLAB
	POEref at 1. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the '582 patent as charted for this claim limitation, and/or MultiMATLAB combined with the knowledge of one of ordinary skill in the art. Motivations to combine may come from the knowledge of the person of ordinary skill, or from the known problems and predictable solutions as embodied in these references. For example, to the extent any one of the MultiMATLAB printed publications lacks this element, it would have been obvious to combine with another of the MultiMATLAB printed publications that discloses this element, because they expressly cite to each other and/or to the same system, were published by overlapping research teams and/or in similar locations, and represent combinations of prior art elements according to known methods to yield predictable results. Further motivation to combine references and additional details may be found in the invalidity contentions cover pleading.
[20] 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.	MultiMATLAB discloses this claim limitation. See, e.g.: In the MultiMatlab architecture, illustrated in Figure 1, each processor in a parallel platform individually runs a Matlab process. Each process is then provided with the ability to communicate with other processes through a communication layer that runs over the parallel platform's interconnection network. The user interacts directly with one Matlab process, called the interactive process, and operates within that process's Matlab environment. Other Matlab processes await commands from the interactive process. These other processes are used either by explicitly sending Matlab commands to them or by executing routines that, in turn, use them. The key component in this architecture is the MultiMatlab interface module. The interface module, shown in Figure 2, is responsible for initializing an underlying communication layer, such as MPI [2], PVM [3], BLACS [4], or any other package available on the platform, and exposing it to the rest of the system. MEX routines, also shown in the Figure 2, are executables originally written in C or Fortran that may be run directly from Matlab. All parallel functionality in the MultiMatlab system is provided

'582 Claim	Prior Art Reference – MultiMATLAB
	through MEX routines. As a simple example, the MultiMatlab system provides users with an Eval routine for parallel evaluation. This routine allows users to execute commands on the other processes. On the interactive process, the Eval routine is implemented as a MEX routine which accepts a vector of Matlab process IDs and a Matlab command. This routine sends the commands over the underlying communication layer to the processes corresponding to the specified IDs. On non-interactive processes, a separate top-level MEX routine is immediately run upon initialization of the system. This MEX routine runs in a loop in which it waits for Matlab commands to arrive from the interactive process and executes them in its own Matlab environment. All parallel MEX routines access the underlying communication layer and, hence, the network, through the MultiMatlab interface module. It is important to note that the interface module exists in the corresponding Matlab process's address space. For this reason, the interface module does not impose any additional context switch when parallel MEX routines access the communication layer. This is of particular importance when the parallel platform provides access to the network in user space. This enables parallel applications to access the network without operating system intervention as is often required for best performance. By residing within the Matlab process's address space, the MultiMatlab interface module is able to preserve this property for parallel MEX routines. Menon at 3-4. The MultiMatlab interface module provides MEX routines access to the underlying communication layer via an indirection table. Upon initialization, the interface module builds a table of pointers to all functions and data in the underlying communication layer that need to be exposed to parallel MEX routines. <i>Id.</i> at 5. The MultiMatlab interface module provides MEX routines access to the underlying communication layer via an indirection table. Upon initialization, the interface module builds a table of pointers to all functions and data in the underlying communication layer that need to be exposed to parallel MEX routines.

'582 Claim	Prior Art Reference – MultiMATLAB
	<i>Id.</i> at 5. In this implementation, the interactive MATLAB process is started by the user as a normal MATLAB session. The user is provided with a Start command that initializes the MultiMATLAB system. This command builds a P4 process group file of remote hosts, which are either explicitly specified by the user or taken from a default list, and the initializes MPICH. <i>Id.</i> at 5. The MPI-F implementation of MultiMATLAB was design specifically for the IBM SP2. MPI-F is a highly optimized implementation of the MPI standard running over the SP2's high performance switch. This implementation permits access to the switch in user space. In this implementation, all MATLAB processes are started via POE, IBM's Parallel Operating Environment. For each of p MATLAB processes, POE assigns an identification number between 0 and p-1. The process numbered 0 is considered the interactive MATLAB. All other processes wait for commands from the interactive MATLAB process. This implementation was designed to demonstrate the performance potential of the MultiMATLAB system. In particular, it provides a platform in which parallel libraries or applications written in MPI can easily be integrated into MATLAB on the SP2. <i>Id.</i> at 6. MATLAB®, a commercial product of The MathWorks, Inc., has become one of the principal languages of desktop scientific computing. A system is described that enables one to run MATLAB conveniently on multiple processors. Using short, MATLAB-style commands like Eval, Send, Recv, Bcast, Min, and Sum, the user operating within one MATLAB session can start MATLAB processes on other machines and then pass commands and data between between these various processes in a fashion that maintains MATLAB's traditional user-friendliness. Multi-processor graphics is also supported. The system currently runs under MPICH on an IBM SP2 or a network of Unix workstations, and extensions are planned to networks of PCs. MultiMATLAB is potentially useful for education in parallel programming, for prototyping parallel

'582 Claim	Prior Art Reference – MultiMATLAB
	algorithms, and for fast and convenient execution of easily parallelizable numerical computations on multiple processors. Trefethen at Abstract. Suppose the first author is sitting at her workstation in the Theory Center, connected to a node of the IBM SP2, running MATLAB. After a time she decides to start MATLAB on five new processors. She types Start(5) MATLAB is then started on five additional processors taken from a predetermined list. Or perhaps the second author is sitting at a machine connected to Cornell's Computer Science Department network. He types Start(['gemini'; 'orion'; 'rigel'; 'castor'; 'pollux']) Now MATLAB is started on the five processors with the names indicated. <i>Id.</i> at 2.1. The MultiMATLAB Start command builds a P4 process group file of remote hosts, which are either explicitly specified by the user or taken from a default list, and then initializes MPICH. MATLAB processes are then started on the remote hosts. Each process iterates over a simple loop, waiting for and executing commands received from the user's interactive MATLAB process. <i>Id.</i> at 4. It is a straightforward matter to install our current software on any network of Unix workstations or SP2 system, provided that all the nodes are licensed to run MATLAB and there is a shared file system. We expect that extensions to networks of PCs running Windows, based on appropriate implementations of MPI, are not far behind. <i>Id.</i> at 6.

'582 Claim	Prior Art Reference – MultiMATLAB
	In general, with POE, you invoke a parallel program from your home node and run its parallel tasks on a number of remote nodes. When you invoke a program on your home node, POE starts your Partition Manager which allocates the nodes of your partition and initializes the local environment. Depending on your hardware and configuration, the Partition Manager uses a host list file, LoadLeveler, or the SP system Resource Manager to allocate nodes. A host list file contains an explicit list of node requests, while LoadLeveler or the Resource Manager allocate nodes from one or more system pools implicitly based on their availability. On an SP system using the Resource Manager, you can also use a host list file to determine how an allocated node's resources – its SP switch adapter and CPU are used. POEref at 3. With regard to the expanded LoadLeveler function, POE now provides an option to enable you to specify whether your program will use MPI, LAPI, or both. Using this option, POE ensures that each API initializes properly and informs LoadLeveler which APIs are used so each node is set up completely. For Single Program Multiple Data (SPMD) applications the Partition Manager executes the same program on all nodes. For Multiple Program Multiple Data (MPMD) applications, the Partition Manager prompts you for the name of the program to load on each node. The Partition Manager also connects standard I/O to each remote node so the parallel tasks can communicate with the home node. Although you are running tasks on remote nodes, POE allows you to continue using the standard UNIX** and AIX execution techniques with which you are already familiar. For example, you can redirect input and output, pipe the output of programs, or use shell tools. <i>Id.</i> When you invoke poe, the Partition Manager allocates processor nodes for each task and initializes the local environment. It then loads your program, and reproduces your local environment, on each processor node. The Partition Manager also passes the option list to each remote node. If your program is in a shared file system, the Partition Manager loads a copy of it on each node. If your program is in a private file system, you will have already manually copied your executable to the nodes using the mprcp or mcp command. If

'582 Claim	Prior Art Reference – MultiMATLAB
	you are using the dynamic message passing interface, the appropriate communication subsystem library implementation (IP or US) is automatically loaded at this time. POEref at 35. <i>1) MultiMATLAB:</i> MultiMATLAB is perhaps one of the most well known parallel MATLABs. It was developed by A. Trefethen, V. Menon, C. Chang, G. Czajkowski, C. Myers, and L. Trefethen at Cornell University. In MultiMATLAB, a MATLAB user can start MATLAB processes on other machines and then pass commands and data between these various processes. The most important commands are: • Init—initialize the MultiMATLAB environment; • Nproc—returns the number of processes in the environment; equivalent of MPI Comm size(); • ID—returns the rank of the process; equivalent of MPI Comm rank(); • Send/Recv—sends and receives data between processes; • Eval—evaluates a string in every process in the environment. Choy 2 at 333. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the '582 patent as charted for this claim limitation, and/or MultiMATLAB combined with the knowledge of one of ordinary skill in the art. Motivations to combine may come from the knowledge of the person of ordinary skill, or from the known problems and predictable solutions as embodied in these references. For example, to the extent any one of the MultiMATLAB printed publications lacks this element, it would have been obvious to combine with another of the MultiMATLAB printed publications that discloses this element, because they expressly cite to each other and/or to the same system, were published by overlapping research teams and/or in similar locations, and represent combinations of prior art elements according to known methods to yield predictable results. Further motivation to combine references and additional details may be found in the invalidity contentions cover pleading.