Exhibit C-8
to AMD's Invalidity Contentions

Invalidity Chart for U.S. Patent No. 11,570,034
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 '034 patent is not entitled to a priority date earlier than at least July 13, 2021. MultiMATLAB, as well as each of the publications listed below relating to that system, are therefore prior art to the '034 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 '034 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 is also a contention that one or more printed publications regarding MultMATLAB disclose 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/da709mst07.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").

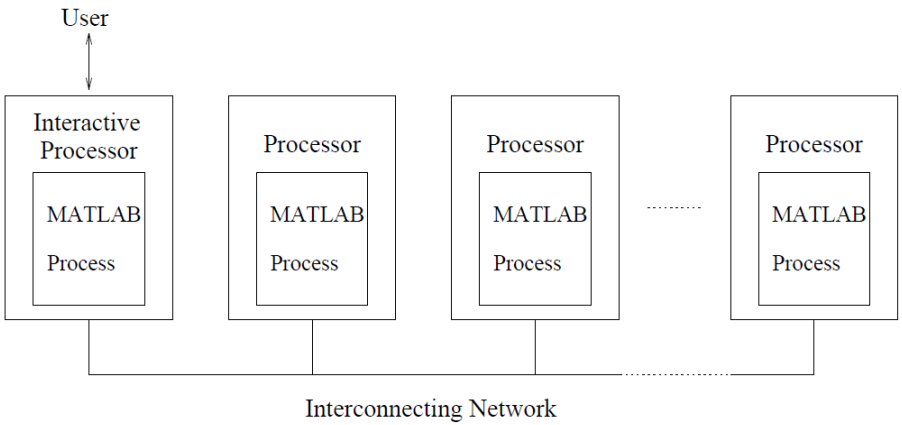
To the extent MultiMATLAB does not disclose one or more limitations of the claims, it would have been obvious to combine the teachings of MultiMATLAB with the knowledge of one of ordinary skill in the art and with one or more of the references below to render the claims at issue in the '768 patent invalid, 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:/theorv.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 at., 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.

’034 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 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

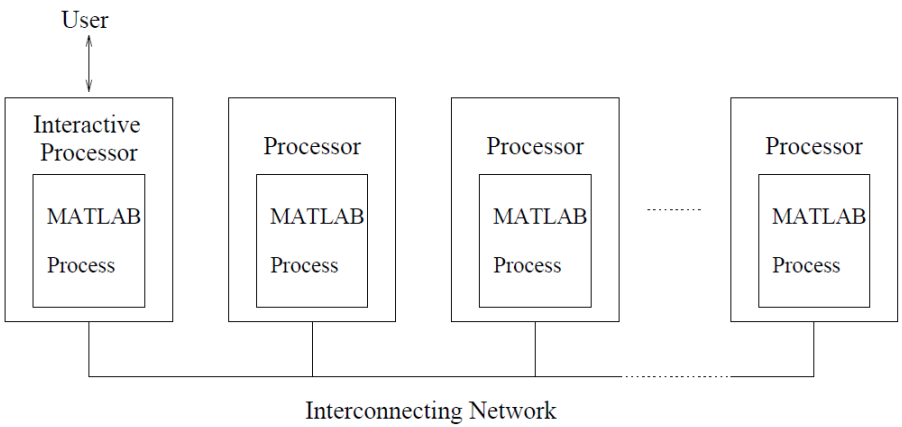
’034 Claim	Prior Art Reference – MultiMATLAB
	parallel applications with the MATLAB language. We discuss a general MultiMATLAB architecture, present two implementations based upon the MPI communication standard [2], and demonstrate the use of this system. Menon at 1. The system should be easily portable 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. *Id.* at 2.  Figure 1: MultiMATLAB Architecture *Id.* 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

’034 Claim	Prior Art Reference – MultiMATLAB
	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. *Id.* 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. *Id.* 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. *Id.* 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.

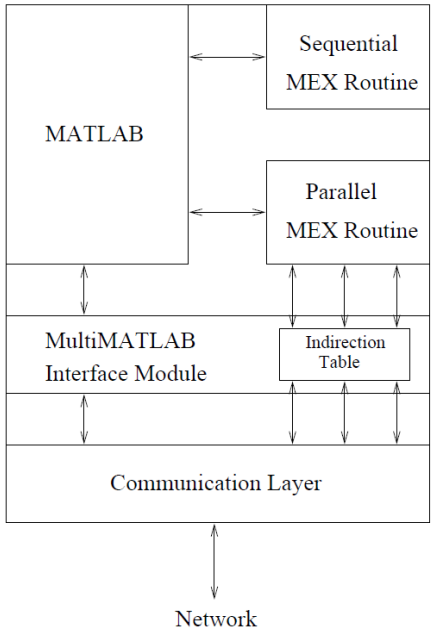
’034 Claim	Prior Art Reference – MultiMATLAB
	This implementation was designed to demonstrate the performance potential of the MultiMATLAB system. In particular, it provides a platform in which parallel libraries or applications written in MPI can easily be integrated into MATLAB on the SP2. *Id.* at 6. In this section, we describe different MultiMATLAB implementations of the conjugate gradients algorithm on the IBM SP2 at the Cornell Theory Center. *Id.* 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. *Id.* at 13; Figs. at 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

’034 Claim	Prior Art Reference – MultiMATLAB
	small. Convenience of spreading the problem across machines and collecting the results numerically or graphically is paramount. *Id.* 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. *Id.* 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. *Id.* 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. *Id.* 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.

’034 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 *IBM Parallel System Support Programs for AIX: Installation and Migration Guide* 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. POEref 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, 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 ’034 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

’034 Claim	Prior Art Reference – MultiMATLAB
	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 plurality of nodes, wherein each of the plurality of nodes comprises a hardware processor, wherein one or more of the nodes are configured to receive a command to start a cluster initialization process for the computer cluster, and wherein each of the nodes is configured to access a non-transitory computer-readable medium comprising program code for a single-node kernel that, when executed, is capable of causing the hardware processor to evaluate mathematical expressions; and	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.  Figure 1: MultiMATLAB Architecture *Id.* at Fig. 1; see discussion at Fig. 1.

’034 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.

’034 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. *Id.* at 3-4.  Figure 2: MATLAB Process Address Space *Id.* 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. *Id.* at 5.

’034 Claim	Prior Art Reference – MultiMATLAB
	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. *Id.* 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. *Id.* at 6.

’034 Claim	Prior Art Reference – MultiMATLAB

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

  w = A * p;

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

a. Sequential MATLAB

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

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

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

b. Message Passing MATLAB

Figure 4: Conjugate Gradients

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 rho and alpha and the matrix-vector multiply needed

’034 Claim	Prior Art Reference – MultiMATLAB
	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. *Id.* 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. *Id.* 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 .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 zerofinding). 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. *Id.* at 1.1.

’034 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. *Id.* 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. *Id.* 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. 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 a sitting at a machine connected to Cornell’s Computer Science Department network. He types Start([’gemini’; ’orion’; ’rigel’; ’castor’; ’pollux’])

’034 Claim	Prior Art Reference – MultiMATLAB

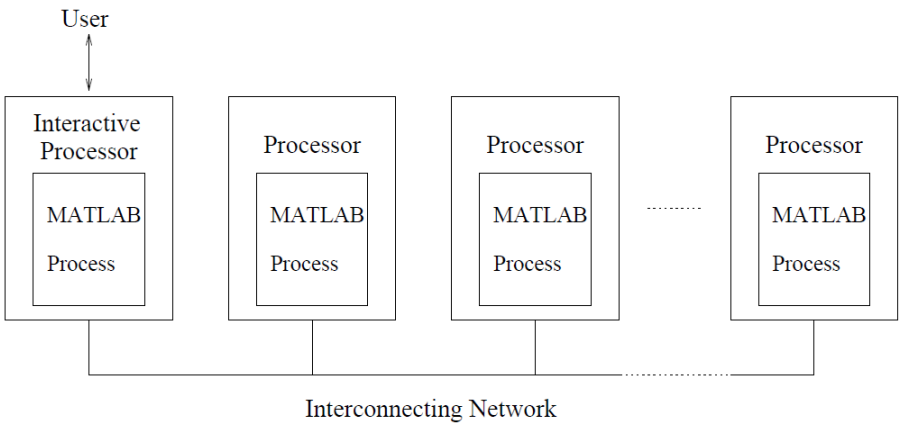
	Now MATLAB is started on the five processors with the names indicated.
	
	Id. at 2.1.
	
	The standard MultiMATLAB command for executing commands on one or more processors is Eval. If the user now types
	
	Eval(’sqrt(2)’)
	
	then the MATLAB command sqrt(2) is executed on all six processors. The result is six repetitions of 1.4142.
	
	Id. at 2.1.
	
	`if ID==0 % first process: send`
	`  a = 1`
	`  Send(ID+1,a)`
	`elseif ID == Nproc-1 % last process: receive and double`
	`  a = 2*Recv`
	`else % middle processes: receive, double, and send`
	`  a = 2*Recv`
	`  Send(ID+1,a)`
	`end;`
	
	Process 0 creates the variable `a` with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of `a`, and sends it along to process 2; and so on. If there are six processors the command `Eval( 'cycle' )` produces the output
	
	`a = 1`
	`a = 2`
	`a = 4`
	`a = 8`
	`a = 16`
	`a = 32`
	
	The processes run asynchronously, but since each `Send` command is only executed after the corresponding `Recv` has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning.
	
	Id. 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

’034 Claim	Prior Art Reference – MultiMATLAB
	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. *Id.* 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. Trefethen 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. *Id.* at 6. 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.

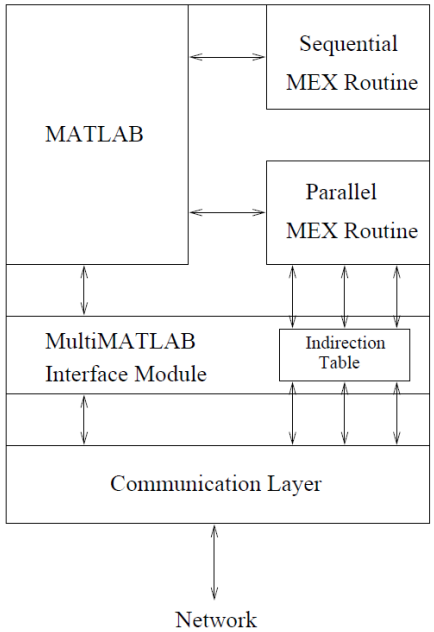
’034 Claim	Prior Art Reference – MultiMATLAB
	RS6000 at 1; *see also* RS6000-2 at 1-3. The basic components of the RS/6000 SP system are: *Processor nodes (includes SP-attached servers) * Frames with integral power subsystems * Switches * Extension nodes * Control workstations (a high availability option 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. 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; *see also* RS6000-2 at 3. 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, LoadLeverler, 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.

’034 Claim	Prior Art Reference – MultiMATLAB
	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. *1) MultiMATLAB:* 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.

’034 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 ’034 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.

’034 Claim	Prior Art Reference – MultiMATLAB
[1B] a communications network interface for the nodes to communicate results of mathematical expression evaluation with each other using a peer-to-peer architecture;	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.  Figure 1: MultiMATLAB Architecture *Id.* 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

’034 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. 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.

’034 Claim	Prior Art Reference – MultiMATLAB
	Id. at 3-4.  Figure 2: MATLAB Process Address Space *Id.* 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. *Id.* at 5.

’034 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. *Id.* 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. *Id.* at 6. SPMD Communication Send(pid,data) — Send data from one process to another. Recv(pid) — Receive data sent from another process. Barrier — Synchronize processes. Bcast(pid,data) — Broadcasts data from processor pid to all processes. Sum(data) — Adds data across all processors to form a global sum. Collect(data) — Collects local data segments into a single global one. Table 1: Selected commands in the MultiMATLAB system

’034 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. *Id.* 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 x = [pi pi^2]; Send(3,x)

’034 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.
	
	Id. at 2.3; *see also id.* at 2.1.
	
	`if ID==0 % first process: send`
	`  a = 1`
	`  Send(ID+1,a)`
	`elseif ID == Nproc-1 % last process: receive and double`
	`  a = 2*Recv`
	`else % middle processes: receive, double, and send`
	`  a = 2*Recv`
	`  Send(ID+1,a)`
	`end;`
	
	Process 0 creates the variable `a` with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of `a`, and sends it along to process 2; and so on. If there are six processors the command `Eval( 'cycle' )` produces the output
	
	`a = 1`
	`a = 2`
	`a = 4`
	`a = 8`
	`a = 16`
	`a = 32`
	
	The processes run asynchronously, but since each `Send` command is only executed after the corresponding `Recv` has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning.
	
	Id. at 2.3.
	
	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.

’034 Claim	Prior Art Reference – MultiMATLAB
	Id. at 2.4; *see generally id.* 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. *Id.* 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

’034 Claim	Prior Art Reference – MultiMATLAB
	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; *see also* RS6000 2002 generally.

’034 Claim	Prior Art Reference – MultiMATLAB

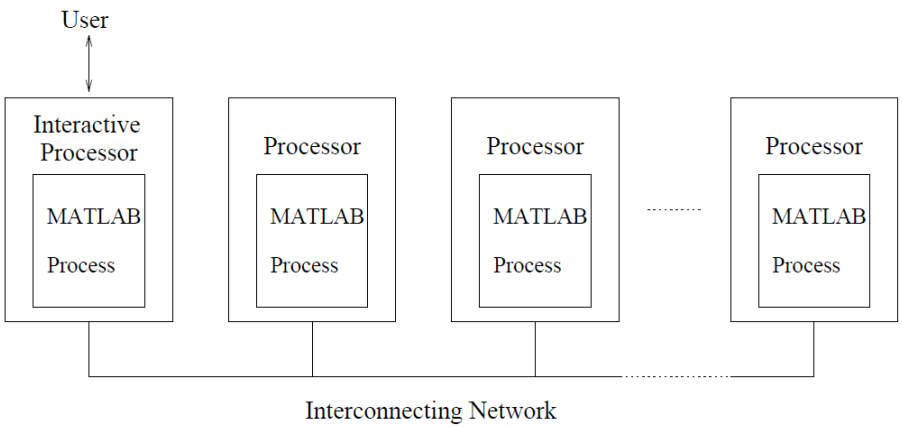
	Chapter 3
	
	Point-to-Point Communication
	
	3.1 Introduction
	
	Sending and receiving of messages by processes is the basic MPI communication mechanism. The basic point-to-point communication operations are **send** and **receive**. Their use is illustrated in the example below.
	
	`#include "mpi.h"`
	`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;`
	`}`
	
	In this example, process zero (myrank = 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.

’034 Claim	Prior Art Reference – MultiMATLAB

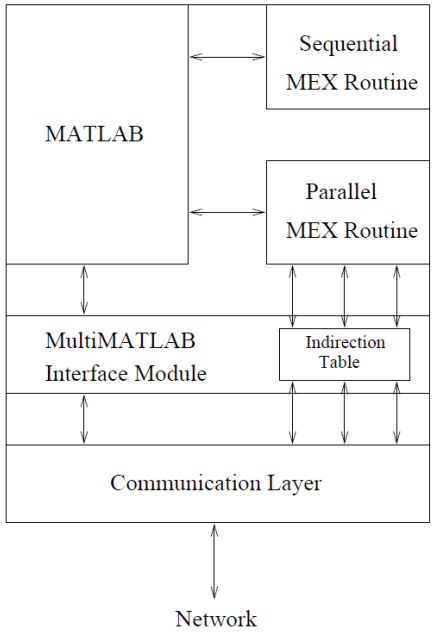
	MPI_ISEND(buf, count, datatype, dest, tag, comm, request)
	IN buf initial address of send buffer (choice)
	IN count number of elements in send buffer (integer)
	IN datatype datatype of each send buffer element (handle)
	IN dest rank of destination (integer)
	IN tag message tag (integer)
	IN comm communicator (handle)
	OUT request communication request (handle)
	
	`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; *see also* MPI Standard 1995 at 16-23.

’034 Claim	Prior Art Reference – MultiMATLAB

	MPI_IRECV (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_IRECV(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 ’034 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.

’034 Claim	Prior Art Reference – MultiMATLAB
[1C1] wherein the plurality of nodes comprises: a first node comprising a first hardware processor configured to access a first memory comprising program code for a user interface and program code for a first single-node kernel, the first single-node kernel configured to interpret user instructions and distribute calls to at least one of a plurality of other nodes for execution; and	MultiMATLAB discloses this claim limitation. *See* disclosures at [1A], *supra*. 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.  Figure 1: MultiMATLAB Architecture *Id.* at Fig. 1; see discussion at Fig. 1. > Users are provided interactive access to all processors in a multiprocessor within the MATLAB environment. This permits steering of parallel computation.

’034 Claim	Prior Art Reference – MultiMATLAB
	Id. at 2. 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

’034 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. *Id.* at 3-4.  Figure 2: MATLAB Process Address Space *Id.* at Fig. 2; see discussion at Fig. 2. 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.

’034 Claim	Prior Art Reference – MultiMATLAB
	Id. 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. *Id.* 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. *Id.* at 6.

’034 Claim	Prior Art Reference – MultiMATLAB

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

  w = A * p;

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

a. Sequential MATLAB

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

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

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

b. Message Passing MATLAB

Figure 4: Conjugate Gradients

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 rho and alpha and the matrix-vector multiply needed

’034 Claim	Prior Art Reference – MultiMATLAB
	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. *Id.* 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. 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. *Id.* 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

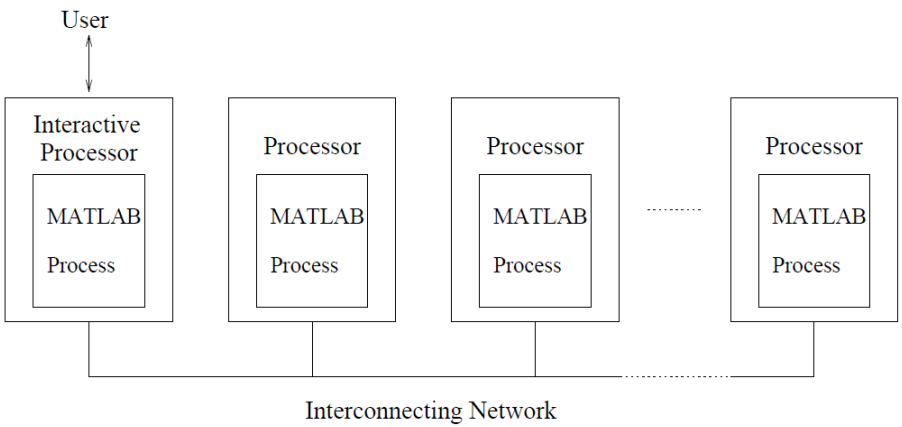
’034 Claim	Prior Art Reference – MultiMATLAB
	small. Convenience of spreading the problem across machines and collecting the results numerically or graphically is paramount. *Id.* at 1.2. More general point-to-point communication is accomplished by send and receive commands, which can be executed on any of the MATLAB processes. For example, the sequence 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. *Id.* at 2.3; *see also id.* at 2.1.

’034 Claim	Prior Art Reference – MultiMATLAB

	`if ID==0 % first process: send`
	`  a = 1`
	`  Send(ID+1,a)`
	`elseif ID == Nproc-1 % last process: receive and double`
	`  a = 2*Recv`
	`else % middle processes: receive, double, and send`
	`  a = 2*Recv`
	`  Send(ID+1,a)`
	`end;`
	
	Process 0 creates the variable `a` with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of `a`, and sends it along to process 2; and so on. If there are six processors the command `Eval( 'cycle' )` produces the output
	
	`a = 1`
	`a = 2`
	`a = 4`
	`a = 8`
	`a = 16`
	`a = 32`
	
	The processes run asynchronously, but since each `Send` command is only executed after the corresponding `Recv` has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning.
	
	Id. 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.
	
	Id. 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

’034 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. 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; *see also* RS6000-2 at 3. *See also* MPI Standard 1995 at 16-23, 38, 40.

’034 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 ’034 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.

’034 Claim	Prior Art Reference – MultiMATLAB
[1C2] a second node comprising a second hardware processor with a plurality of processing cores, wherein the second node is configured to receive calls from the first node, execute at least a first mathematical expression evaluation, and communicate a result of the first mathematical expression evaluation to a third node;	MultiMATLAB discloses this claim limitation. *See* disclosures at [1A], [1C1], *supra*. 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.  Figure 1: MultiMATLAB Architecture *Id.* 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

’034 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. *Id.* at 3-4.

’034 Claim	Prior Art Reference – MultiMATLAB
	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. *Id.* at 6. a. Sequential MATLAB b. Message Passing MATLAB Figure 4: Conjugate Gradients

```
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
```

```
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
```

’034 Claim	Prior Art Reference – MultiMATLAB
	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 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. *Id.* 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.

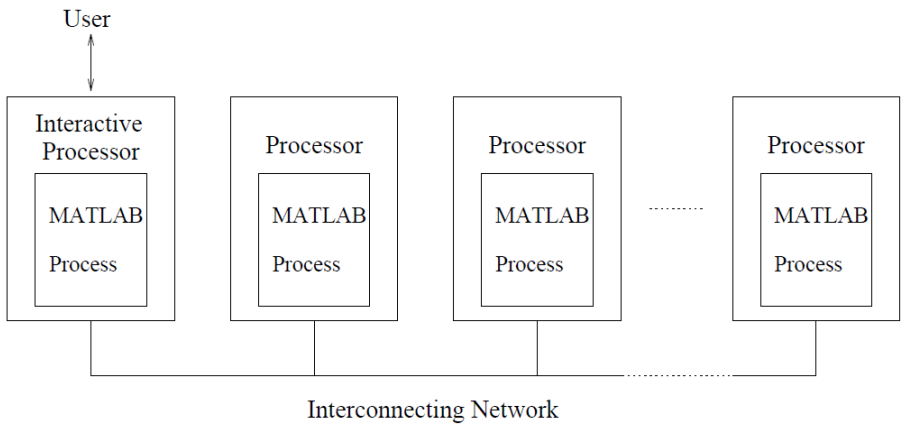
’034 Claim	Prior Art Reference – MultiMATLAB

	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.
	
	Id. at 2.3; *see also id.* at 2.1.
	
	`if ID==0 % first process: send`
	`  a = 1`
	`  Send(ID+1,a)`
	`elseif ID == Nproc-1 % last process: receive and double`
	`  a = 2*Recv`
	`else % middle processes: receive, double, and send`
	`  a = 2*Recv`
	`  Send(ID+1,a)`
	`end;`
	
	Process 0 creates the variable `a` with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of `a`, and sends it along to process 2; and so on. If there are six processors the command `Eval( 'cycle' )` produces the output
	
	`a = 1`
	`a = 2`
	`a = 4`
	`a = 8`
	`a = 16`
	`a = 32`
	
	The processes run asynchronously, but since each `Send` command is only executed after the corresponding `Recv` has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning.

’034 Claim	Prior Art Reference – MultiMATLAB
	Id. 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. *Id.* 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. RS6000 at 2.

’034 Claim	Prior Art Reference – MultiMATLAB
	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; *see also* RS6000-2 at 3. *See also* RS6000 2002 at 3, 11, 12, 19. *See also* MPI Standard 1995 at 16-23, 38, 40. 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 ’034 patent as charted for this claim limitation, and/or

’034 Claim	Prior Art Reference – MultiMATLAB
	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.
[1C3] wherein the third node comprises a third hardware processor with a plurality of processing cores, wherein the third node is configured to receive the result of the first mathematical expression evaluation from the second node, execute at least a second mathematical expression evaluation using the received result, and communicate the result of the second mathematical	MultiMATLAB discloses this claim limitation. *See* disclosures at [1A]. 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.

’034 Claim	Prior Art Reference – MultiMATLAB
expression evaluation to the first node;	 Figure 1: MultiMATLAB Architecture *Id.* 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 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.

’034 Claim	Prior Art Reference – MultiMATLAB

	r = b;
	rho = r'*r;
	k = 0;
	while((k < kmax))
	  k = k+1;
	  if (k == 1)
	  p = r;
	  else
	  beta = rho/oldrho;
	  p = r + beta*p;
	  end
	
	  w = A * p;
	
	  alpha = rho/(p'*w);
	  x = x + alpha * p;
	  r = r - alpha * w;
	  oldrho = rho;
	  rho = r'*r;
	end
	
	a. Sequential MATLAB
	
	r_local = b_local;
	rho = Sum(r_local'*r_local);
	k = 0;
	while((k < kmax))
	  k = k+1;
	  if (k == 1)
	  p_local = r_local;
	  else
	  beta = rho/oldrho;
	  p_local = r_local + beta*p_local;
	  end
	
	  p = Gather(p_local);
	  w_local = A_local * p;
	
	  alpha = rho/(Sum(p_local'*w_local));
	  x_local = x_local + alpha * p_local;
	  r_local = r_local - alpha * w_local;
	  oldrho = rho;
	  rho = Sum(r_local'*r_local);
	end
	
	b. Message Passing MATLAB
	
	Figure 4: Conjugate Gradients
	
	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 rho and alpha and the matrix-vector multiply needed

’034 Claim	Prior Art Reference – MultiMATLAB
	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. *Id.* 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 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.

’034 Claim	Prior Art Reference – MultiMATLAB

	Id. at 2.3; *see also id.* at 2.1.
	
	`if ID==0 % first process: send`
	`  a = 1`
	`  Send(ID+1,a)`
	`elseif ID == Nproc-1 % last process: receive and double`
	`  a = 2*Recv`
	`else % middle processes: receive, double, and send`
	`  a = 2*Recv`
	`  Send(ID+1,a)`
	`end;`
	
	Process 0 creates the variable `a` with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of `a`, and sends it along to process 2; and so on. If there are six processors the command `Eval( 'cycle' )` produces the output
	
	`a = 1`
	`a = 2`
	`a = 4`
	`a = 8`
	`a = 16`
	`a = 32`
	
	The processes run asynchronously, but since each `Send` command is only executed after the corresponding `Recv` has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning.
	
	Id. at 2.3.
	
	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.

’034 Claim	Prior Art Reference – MultiMATLAB
	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; *see also* RS6000-2 at 3. *See also* RS6000 2002 at 3, 11, 12, 19. *See also* MPI Standard 1995 at 16-23, 38, 40.

’034 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 ’034 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.

’034 Claim	Prior Art Reference – MultiMATLAB
[1D] wherein the first node is configured to return the result of the second mathematical expression evaluation to the user interface;	MultiMATLAB discloses this claim limitation. *See* disclosures at [1C3], *supra*. 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 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

’034 Claim	Prior Art Reference – MultiMATLAB
	> 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. *Id.* at 3-4. > 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.

’034 Claim	Prior Art Reference – MultiMATLAB

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

  w = A * p;

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

a. Sequential MATLAB

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

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

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

b. Message Passing MATLAB

Figure 4: Conjugate Gradients

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 rho and alpha and the matrix-vector multiply needed

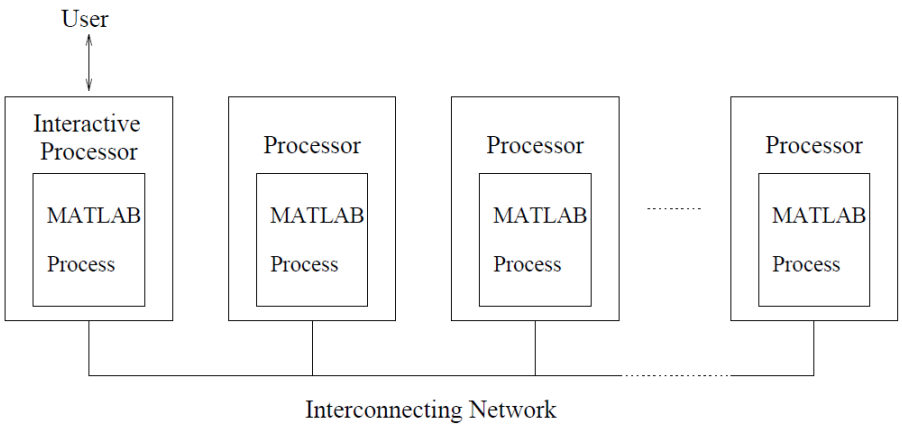
’034 Claim	Prior Art Reference – MultiMATLAB
	to compute w_local. The dot products only require communication of a single value over all processes. On the other hand, the matrix-vector multiplication needed for w_local requires the global vector p, and, thus, more expensive communication. *Id.* at 11-13. 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. 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. *Id.* 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

’034 Claim	Prior Art Reference – MultiMATLAB
	small. Convenience of spreading the problem across machines and collecting the results numerically or graphically is paramount. *Id.* 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 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. *Id.* at 2.3; *see also id.* at 2.1.

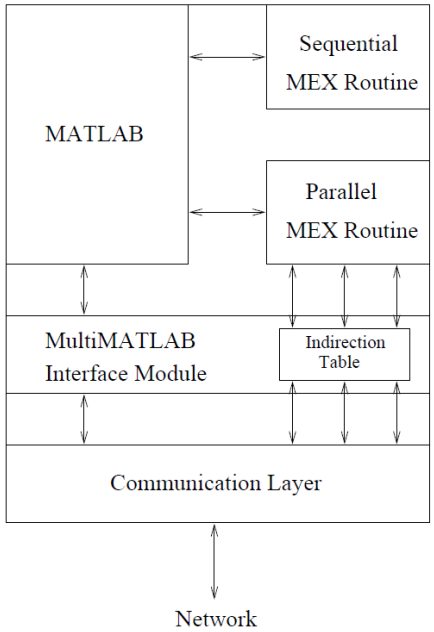
’034 Claim	Prior Art Reference – MultiMATLAB

	`if ID==0 % first process: send`
	`  a = 1`
	`  Send(ID+1,a)`
	`elseif ID == Nproc-1 % last process: receive and double`
	`  a = 2*Recv`
	`else % middle processes: receive, double, and send`
	`  a = 2*Recv`
	`  Send(ID+1,a)`
	`end;`
	
	Process 0 creates the variable `a` with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of `a`, and sends it along to process 2; and so on. If there are six processors the command `Eval( 'cycle' )` produces the output
	
	`a = 1`
	`a = 2`
	`a = 4`
	`a = 8`
	`a = 16`
	`a = 32`
	
	The processes run asynchronously, but since each `Send` command is only executed after the corresponding `Recv` has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning.
	
	Id. 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.
	
	Id. 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.
	
	Id. at 6.

’034 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 ’034 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.
[1E] wherein one or more of the nodes are configured to: accept user instructions; after accepting user instructions, communicate at least some of the user instructions using the communications network interface for the nodes to communicate with each other; and after communicating at least some of the user instructions using the communications network 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. > Users are provided interactive access to all processors in a multiprocessor within the MATLAB environment. This permits steering of parallel computation. *Id.* at 2.

’034 Claim	Prior Art Reference – MultiMATLAB
communicate at least some of the user instructions to one or more single-node kernels.	 Figure 1: MultiMATLAB Architecture *Id.* 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

’034 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. *Id.* at 3-4.

’034 Claim	Prior Art Reference – MultiMATLAB
	 Figure 2: MATLAB Process Address Space *Id.* 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. 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.

’034 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. *Id.* 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. *Id.* 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

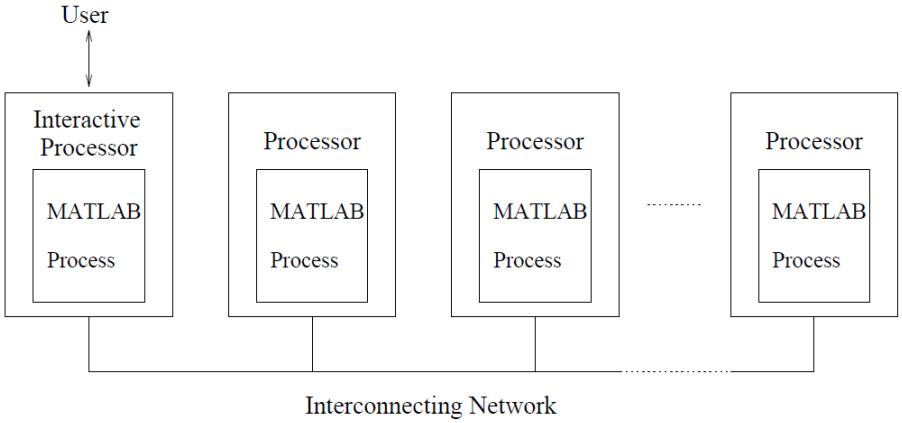
’034 Claim	Prior Art Reference – MultiMATLAB
	algorithms, and for fast and convenient execution of easily parallelizable numerical computations on multiple processors. Trefethen at Abstract. MATLAB® is a high-level language, and a problem-solving environment, for mathematical and scientific calculations. 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 zerofinding). 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. *Id.* 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. *Id.* 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. *Id.* at 1.2.

’034 Claim	Prior Art Reference – MultiMATLAB

	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.
	
	Id. at 2.3; *see also id.* at 2.1.
	
	`if ID==0 % first process: send`
	`  a = 1`
	`  Send(ID+1,a)`
	`elseif ID == Nproc-1 % last process: receive and double`
	`  a = 2*Recv`
	`else % middle processes: receive, double, and send`
	`  a = 2*Recv`
	`  Send(ID+1,a)`
	`end;`
	
	Process 0 creates the variable `a` with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of `a`, and sends it along to process 2; and so on. If there are six processors the command `Eval( 'cycle' )` produces the output
	
	`a = 1`
	`a = 2`
	`a = 4`
	`a = 8`
	`a = 16`
	`a = 32`
	
	The processes run asynchronously, but since each `Send` command is only executed after the corresponding `Recv` has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning.

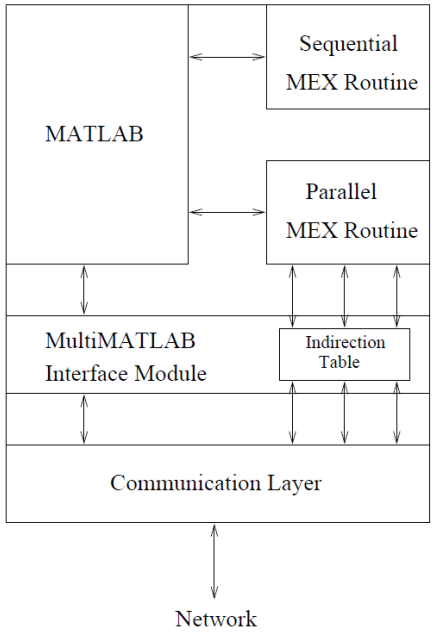
’034 Claim	Prior Art Reference – MultiMATLAB
	Id. at 2.3. The MultiMATLAB Start command builds a P4 process group file of remote hosts, which are either explicitly specified by the user or taken from a default list, and then initializes MPICH. MATLAB processes are then started on the remote hosts. Each process iterates over a simple loop, waiting for and executing commands received from the user’s interactive MATLAB process. *Id.* at 4. *See also* MPI Standard 1995 at 16-23, 38, 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 ’034 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.
[2] The computer cluster of claim 1, wherein the communications network interface comprises at least one of: a local bus, a local area network (LAN), a wide area network (WAN), a storage area network	MultiMATLAB discloses this claim limitation. See, e.g.: 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. *Id.* at 5; *see also* disclosures at [1B], *supra*. 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

’034 Claim	Prior Art Reference – MultiMATLAB
(SAN), a TCP/IP interface over Ethernet, or a wireless network interface.	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. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the ’034 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.

’034 Claim	Prior Art Reference – MultiMATLAB
[3] The computer cluster of claim 1, wherein each of the nodes comprises one or more cluster node modules.	MultiMATLAB discloses this claim limitation. *See* disclosures at [1B], *supra.* 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. > The MultiMATLAB architecture now provides direct access from MATLAB to vendor-optimized communication libraries. *Id.* at 2.  Figure 1: MultiMATLAB Architecture

’034 Claim	Prior Art Reference – MultiMATLAB
	Id. 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

’034 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. 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 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. *Id.* at 3-5.

’034 Claim	Prior Art Reference – MultiMATLAB
	 Figure 2: MATLAB Process Address Space *Id.* at Fig. 2; see discussion at Fig. 2. As a result, programmers familiar with MPI can conveniently develop parallel MEX routines for MATLAB with standard MPI function calls and expect the roughly same performance as similar native C programs. Moreover, libraries and applications already available in MPI can be easily rewritten and recompiled as MEX routines by simply providing a gateway function. 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 result, parallel MEX routines on the SP2 may directly call Parallel ESSL functions in addition to

’034 Claim	Prior Art Reference – MultiMATLAB
	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. *Id.* at 7. 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. Trefethen 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 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. *Id.* 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

’034 Claim	Prior Art Reference – MultiMATLAB
	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. *See also* MPI Standard 1995 at 16-23, 38, 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 ’034 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 1, wherein the computer cluster includes the user interface.	MultiMATLAB discloses this claim limitation. *See* disclosures at [1A], [1C1], *supra*. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the ’034 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

’034 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.
[10] The computer cluster of claim 1, wherein the cluster initialization process comprises establishing communication among two or more of the nodes.	MultiMATLAB discloses this claim limitation. 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

’034 Claim	Prior Art Reference – MultiMATLAB
	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. *Id.* 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. *Id.* 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. *Id.* at 5.

’034 Claim	Prior Art Reference – MultiMATLAB
	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. *Id.* 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 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)

’034 Claim	Prior Art Reference – MultiMATLAB
	MATLAB is then started on five additional processors taken from a predetermined list. Or perhaps the second author is a 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. *Id.* 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. *Id.* 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. *Id.* 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, LoadLeverler, 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.

’034 Claim	Prior Art Reference – MultiMATLAB
	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. *Id.* 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.

’034 Claim	Prior Art Reference – MultiMATLAB
	1) MultiMATLAB: 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 ’034 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.
[24] The computer cluster of claim 1, wherein the plurality of nodes are configured to communicate with	MultiMATLAB discloses this claim limitation. *See, e.g.*, disclosures at [1A], [1B], [1C1], *supra*. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the ’034 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

’034 Claim	Prior Art Reference – MultiMATLAB
one another to interpret and translate commands for execution by a plurality of single-node kernels.	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.
[25PRE] A computer cluster comprising:	To the extent that the preamble of claim 25 is considered a claim limitation, MultiMATLAB discloses it. *See* disclosures at [1PRE], *supra*. Alternatively, to the extent that the preamble of claim 25 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 ’034 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.
[25A] a plurality of nodes, wherein one or more of the nodes are configured to receive: a command to start a cluster initialization process for the computer cluster, wherein the cluster initialization process	MultiMATLAB discloses this claim limitation. *See* disclosures at [1A], *supra*. 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

’034 Claim	Prior Art Reference – MultiMATLAB
comprises establishing communication among two or more of the nodes; and an instruction from a user interface or a script; and	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. 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

’034 Claim	Prior Art Reference – MultiMATLAB
	all functions and data in the underlying communication layer that need to be exposed to parallel MEX routines. *Id.* 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. *Id.* 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. *Id.* 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. *Id.* at 6.

’034 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. 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 a 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. *Id.* at 2.1. For simplicity, the examples above call Eval with an explicit MATLAB command as an argument string. For most applications, however, a user will want to execute a program (an m-file) rather than a single line of text. A command such as

’034 Claim	Prior Art Reference – MultiMATLAB
	Eval(’filename’) achieves this effect. Trefethen 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. *Id.* 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. *Id.* 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, LoadLeverler, 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

’034 Claim	Prior Art Reference – MultiMATLAB
	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. *Id.* 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.

’034 Claim	Prior Art Reference – MultiMATLAB
	1) MultiMATLAB: 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 ’034 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.
[25B] a communications network interface for the nodes to communicate results of mathematical	MultiMATLAB discloses this claim limitation. *See* disclosures at [1B], *supra*. 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

’034 Claim	Prior Art Reference – MultiMATLAB
expression evaluation with each other using asynchronous calls;	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. Menon at 3-4.

’034 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. *Id.* 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. *Id.* at 6. SPMD Communication Send(pid,data) — Send data from one process to another. Recv(pid) — Receive data sent from another process. Barrier — Synchronize processes. Bcast(pid,data) — Broadcasts data from processor pid to all processes. Sum(data) — Adds data across all processors to form a global sum. Collect(data) — Collects local data segments into a single global one. Table 1: Selected commands in the MultiMATLAB system

’034 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.
	
	Id. at 9 and Table 1.
	
	``
	r = b;
	rho = r'*r;
	k = 0;
	while((k < kmax))
	  k = k+1;
	  if (k == 1)
	  p = r;
	  else
	  beta = rho/oldrho;
	  p = r + beta*p;
	  end
	
	  w = A * p;
	
	  alpha = rho/(p'*w);
	  x = x + alpha * p;
	  r = r - alpha * w;
	  oldrho = rho;
	  rho = r'*r;
	end
	``
	a. Sequential MATLAB
	
	``
	r_local = b_local;
	rho = Sum(r_local'*r_local);
	k = 0;
	while((k < kmax))
	  k = k+1;
	  if (k == 1)
	  p_local = r_local;
	  else
	  beta = rho/oldrho;
	  p_local = r_local + beta*p_local;
	  end
	
	  p = Gather(p_local);
	  w_local = A_local * p;
	
	  alpha = rho/(Sum(p_local'*w_local));
	  x_local = x_local + alpha * p_local;
	  r_local = r_local - alpha * w_local;
	  oldrho = rho;
	  rho = Sum(r_local'*r_local);
	end
	``
	b. Message Passing MATLAB
	
	Figure 4: Conjugate Gradients

’034 Claim	Prior Art Reference – MultiMATLAB
	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 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. *Id.* 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.

’034 Claim	Prior Art Reference – MultiMATLAB

	`if ID==0 % first process: send`
	`  a = 1`
	`  Send(ID+1,a)`
	`elseif ID == Nproc-1 % last process: receive and double`
	`  a = 2*Recv`
	`else % middle processes: receive, double, and send`
	`  a = 2*Recv`
	`  Send(ID+1,a)`
	`end;`
	
	Process 0 creates the variable `a` with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of `a`, and sends it along to process 2; and so on. If there are six processors the command `Eval( 'cycle' )` produces the output
	
	`a = 1`
	`a = 2`
	`a = 4`
	`a = 8`
	`a = 16`
	`a = 32`
	
	The processes run asynchronously, but since each `Send` command is only executed after the corresponding `Recv` has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning.
	
	Id. at 2.3.
	
	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.
	
	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

’034 Claim	Prior Art Reference – MultiMATLAB
	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. *Id.* at 4. Another subtle issue arises because of the nature of asynchronous communications. MPI calls may initiate operations that continue asynchronously after the call returned. MPI Standard 1995 at 13. With suitable hardware, the transfer of data out of the sender memory may proceed concurrently with computations done at the sender after the send was initiated and before it completed. MPI Standard 1995 at 36.

’034 Claim	Prior Art Reference – MultiMATLAB

	MPI_ISEND(buf, count, datatype, dest, tag, comm, request)
	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)
	
	`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.

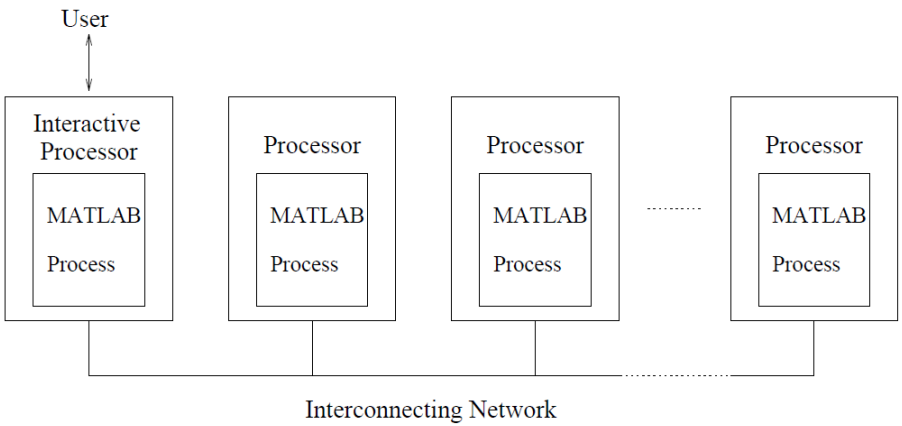
’034 Claim	Prior Art Reference – MultiMATLAB

	MPI_IRECV (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_IRECV(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 ’034 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.

’034 Claim	Prior Art Reference – MultiMATLAB
[25C] wherein each of the nodes is configured to access a non-transitory computer-readable medium comprising program code for a single-node kernel that, when executed, is capable of causing a hardware processor to evaluate mathematical expressions wherein the plurality of nodes comprises: a first node comprising a first hardware processor configured to access a first memory comprising program code for a user interface and program code for a first single-node kernel, the first single-node kernel configured to interpret user instructions and	MultiMATLAB discloses this claim limitation. *See* disclosures at [1A], *supra*. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the ’034 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.

’034 Claim	Prior Art Reference – MultiMATLAB
distribute calls to at least one of a plurality of other nodes for execution; and	
[25D] a second node comprising a second hardware processor with a plurality of processing cores, wherein the second node is configured to receive calls from the first node, execute at least a first mathematical expression evaluation, and communicate a result of mathematical expression evaluation to a third node; wherein	MultiMATLAB discloses this claim limitation. *See* disclosures at [1C2], *supra*. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the ’034 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.
[25E] the third node comprises a third hardware processor with a plurality of processing cores, wherein the third node is configured to	MultiMATLAB discloses this claim limitation. *See* disclosures at [1C3], *supra*. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the ’034 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

’034 Claim	Prior Art Reference – MultiMATLAB
receive the result of mathematical expression evaluation from the second node, execute at least a second mathematical expression evaluation using the received result, and communicate the result of the second mathematical expression evaluation to the first node;	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.
[25F] wherein the first node is configured to return the result of the second mathematical expression evaluation to the user interface or the script;	MultiMATLAB discloses this claim limitation. *See* disclosures at [1D], *supra*. 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.

’034 Claim	Prior Art Reference – MultiMATLAB
	 Figure 1: MultiMATLAB Architecture *Id.* at Fig. 1; see discussion at Fig. 1. To extend this notion to parallel numerical computing, we provide users with the ability to interactively develop parallel MATLAB computations and functions. In particular, functions written in the MATLAB language, called m-files, may then be used in a fashion similar to parallel MEX routines. *Id.* at 8. 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. *Id.* at 10.

’034 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. 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. *Id.* at 1.1. For simplicity, the examples above call Eval with an explicit MATLAB command as an argument string. For most applications, however, a user will want to execute a program (an m-file) rather than a single line of text. A command such as Eval(’filename’) achieves this effect. Trefethen at 2.1. 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

’034 Claim	Prior Art Reference – MultiMATLAB

	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.
	
	Id. at 2.3; *see also id.* at 2.1.
	
	`if ID==0 % first process: send`
	`  a = 1`
	`  Send(ID+1,a)`
	`elseif ID == Nproc-1 % last process: receive and double`
	`  a = 2*Recv`
	`else % middle processes: receive, double, and send`
	`  a = 2*Recv`
	`  Send(ID+1,a)`
	`end;`
	
	Process 0 creates the variable `a` with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of `a`, and sends it along to process 2; and so on. If there are six processors the command `Eval( 'cycle' )` produces the output
	
	`a = 1`
	`a = 2`
	`a = 4`
	`a = 8`
	`a = 16`
	`a = 32`
	
	The processes run asynchronously, but since each `Send` command is only executed after the corresponding `Recv` has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning.
	
	Id. at 2.3.
	
	Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the ’034 patent as charted for this claim limitation, and/or

’034 Claim	Prior Art Reference – MultiMATLAB
	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.
[25G] wherein one or more of the nodes are configured to: accept user instructions; after accepting user instructions, communicate at least some of the user instructions using the communications network interface for the nodes to communicate with each other; and after communicating at least some of the user instruction using the communications network interface, communicate at least some of the user	MultiMATLAB discloses this claim limitation. *See* disclosures at [1E], *supra*. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the ’034 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.

’034 Claim	Prior Art Reference – MultiMATLAB
instructions to one or more since-node kernels.	
[27] The computer cluster of claim 25, 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.: 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. *Id.* at 5; *see also* disclosures at [1B], *supra*. 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

’034 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, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the ’034 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.
[28PRE] A computer cluster comprising:	To the extent that the preamble of claim 28 is considered a claim limitation, MultiMATLAB discloses it. *See* disclosures at [1PRE], [25PRE], *supra*. Alternatively, to the extent that the preamble of claim 28 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 ’034 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.

’034 Claim	Prior Art Reference – MultiMATLAB
[28A] a plurality of nodes, wherein one or more of the nodes are configured to receive: a command to start a cluster initialization process for the computer cluster, wherein the cluster initialization process comprises establishing communication among two or more of the nodes; and an instruction from a user interface or a script; and	MultiMATLAB discloses this claim limitation. *See* disclosures at [1A], [25A], *supra*. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the ’034 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.
[28B] a communications network interface for the nodes to communicate results of mathematical expression evaluation with each other, wherein the communications network interface	MultiMATLAB discloses this claim limitation. *See* disclosures at [25B], [27], *supra*. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the ’034 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

’034 Claim	Prior Art Reference – MultiMATLAB
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;	to known methods to yield predictable results. Further motivation to combine references and additional details may be found in the invalidity contentions cover pleading.
[28C] wherein each of the nodes is configured to access a non-transitory computer-readable medium comprising program code for a single-node kernel that, when executed, is capable of causing a hardware processor to evaluate mathematical expressions	MultiMATLAB discloses this claim limitation. *See* disclosures at [1A], [25C], *supra*. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the ’034 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.
[28D1] wherein the plurality of nodes comprises:	MultiMATLAB discloses this claim limitation. *See* disclosures at [1C1], [25C], *supra*. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the ’034 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

’034 Claim	Prior Art Reference – MultiMATLAB
a first node comprising a first hardware processor configured to access a first memory comprising program code for a user interface and program code for a first single-node kernel, the first single-node kernel configured to interpret user instructions and distribute calls to at least one of a plurality of other nodes for execution; and	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.
[28D2] a second node comprising a second hardware processor with a plurality of processing cores, wherein the second node is configured to receive calls from the first node, execute at least a first	MultiMATLAB discloses this claim limitation. *See* disclosures at [1C2], [25D], *supra*. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the ’034 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

’034 Claim	Prior Art Reference – MultiMATLAB
mathematical expression evaluation, and communicate a result of mathematical expression evaluation to a third node; wherein	to known methods to yield predictable results. Further motivation to combine references and additional details may be found in the invalidity contentions cover pleading.
[28D3] the third node comprises a third hardware processor with a plurality of processing cores, wherein the third node is configured to receive the result of mathematical expression evaluation from the second node, execute at least a second mathematical expression evaluation using the received result, and communicate the result of the second mathematical expression evaluation to the first node; and	MultiMATLAB discloses this claim limitation. *See* disclosures at [1C3], [25E], *supra*. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the ’034 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.

’034 Claim	Prior Art Reference – MultiMATLAB
[28E] wherein the first node is configured to return the result of the second mathematical expression evaluation to the user interface or the script;	MultiMATLAB discloses this claim limitation. *See* disclosures at [1D], [25F], *supra*. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the ’034 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.
[28F] wherein one or more of the nodes are configured to: accept user instructions; after accepting user instructions, communicate at least some of the user instructions using the communications network interface for the nodes to communicate with each other; and after communicating at least some of the user	MultiMATLAB discloses this claim limitation. *See* disclosures at [1E], [25G], *supra*. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the ’034 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.

’034 Claim	Prior Art Reference – MultiMATLAB
instructions using the communications network interface, communicate at least some of the user instructions to one or more single-node kernels.	
[30PRE] A computer cluster node for evaluating expressions in parallel with other computer cluster nodes, the computer cluster node comprising:	To the extent that the preamble of claim 30 is considered a claim limitation, MultiMATLAB discloses it. *See* disclosures at [1PRE], [25PRE], [28PRE], [1A], [25A], [28A], *supra*. Alternatively, to the extent that the preamble of claim 30 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 ’034 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.
[30A] a hardware processor configured to access one or more non-transitory memory devices comprising program code for a single-node kernel that, when executed,	MultiMATLAB discloses this claim limitation. *See* disclosures at [1A], [1C1], [1C2], [1C3], [25C], [25D], [25E], [25F], [28C], [28D1], [28D2], [28D3], [28E], *supra*. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the ’034 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

’034 Claim	Prior Art Reference – MultiMATLAB
causes the hardware processor to interpret user instructions, to evaluate mathematical expressions, and to produce results of mathematical expression evaluation, wherein the hardware processor comprises multiple processor cores; a user connection interface configured to receive a command to start a cluster initialization process for a computer cluster;	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.
[30B] a communications network interface to communicate results of evaluation with other computer cluster nodes using a peer-to-peer architecture, wherein the communications network interface	MultiMATLAB discloses this claim limitation. *See* disclosures at [1B], [2], [25B], [27], [28B], *supra*. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the ’034 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

’034 Claim	Prior Art Reference – MultiMATLAB
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; and	to known methods to yield predictable results. Further motivation to combine references and additional details may be found in the invalidity contentions cover pleading.
[30C1] program code that, when executed, is capable of causing the hardware processor to: receive calls from a second node comprising a second hardware processor configures to access a second memory comprising program code for a user interface and program code for a second single-node kernel, the second single-node kernel configured to	MultiMATLAB discloses this claim limitation. *See* disclosures at [1A], [1C1], [1C2], [1C3], [25C], [25D], [25E], [25F], [28C], [28D1], [28D2], [28D3], [28E], *supra*. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the ’034 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.

’034 Claim	Prior Art Reference – MultiMATLAB
interpret user instructions and distribute calls to at least one of a plurality of other nodes for execution;	
[30C2] execute, using the hardware processor, at least a first mathematical expression evaluation; and communicate a result of the first mathematical expression evaluation to a third node comprising	MultiMATLAB discloses this claim limitation. *See* disclosures at [1C2], [1C3], [25D], [25E], [25F], *supra*. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the ’034 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.
[30C3] a third hardware processor with a plurality of processing cores, wherein the third node is configured to receive the result of mathematical expression evaluation from the computer cluster node, execute at least a second	MultiMATLAB discloses this claim limitation. *See* disclosures at [1C2], [1C3], [25E], [25F], [28D2], [28D3], *supra*. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the ’034 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

’034 Claim	Prior Art Reference – MultiMATLAB
mathematical expression evaluation using the result of the first mathematical expression evaluation, and communicate a result of the second mathematical expression evaluation to the first node;	to known methods to yield predictable results. Further motivation to combine references and additional details may be found in the invalidity contentions cover pleading.
[30D] wherein the user connection interface is configured to return at least one result of mathematical expression evaluation to a user interface or a script; and	MultiMATLAB discloses this claim limitation. *See* disclosures at [1D], [25F], *supra*. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the ’034 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.
[30E] wherein the computer cluster node is configured to: accept user instructions; after accepting user	MultiMATLAB discloses this claim limitation. *See* disclosures at [1E], [25G], [28F], *supra*. Alternatively, this claim limitation is obvious in light of MultiMATLAB itself, MultiMATLAB combined with any of the other references charted with respect to the ’034 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

’034 Claim	Prior Art Reference – MultiMATLAB
instructions, communicate at least some of the user instructions using the communications network interface for the nodes 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 the single-node kernel.	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.