

U.S. Patent No. 10,333,768 (“’768 patent”)**MultiMatlab P References**

The MultiMatlab References, as defined below, are prior art to as prior art to U.S. Patent No. 10,333,768 (“’768 patent”) at least under 35 U.S.C. § 102(a) and (b), and alone or with other references, renders obvious each and every asserted claim, as set forth below. To the extent the MultiMatlab References do not disclose one or more limitations of the claims, it would have been obvious to combine the teachings of the MultiMatlab References 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. Additional exemplary combinations and reasons for obviousness are described in the cover pleading, and are incorporated by reference herein.

The MultiMatlab References include at least the following:

- Trefethen et al., *MultiMATLAB: MATLAB on Multiple Processors*, Technical Report 96-239, Cornell Theory Center, 1996, available at <http://www.cs.cornell.edu/Info/People/Int/multimatlab.html>. (“Trefethen 1996”).
- Trefethen, Menon, *MultiMATLAB: Integrating MATLAB With High-Performance Parallel Computing*, Proceedings of the 1997 ACM/IEEE conference on Supercomputing, November 1997 (“Trefethen 1997”).

Trefethen 1996 describes the MultiMATLAB system as implemented by the named authors at Cornell University. The Trefethen paper is an extension of the system described in Trefethen 1996: “In [1] Trefethen, et al. present MultiMATLAB, a general extension of the Matlab environment to distributed memory multiprocessors, as a first step in this direction.... In this paper, we present a new MultiMATLAB architecture redesigned to achieve [further goals].” Trefethen 1997 at 2. Accordingly, to the extent the Trefethen 1996 or Trefethen 1997 does not expressly or inherently disclose a limitation, such limitation would have been obvious in view of the other because both references include overlapping named authors and describe the MultiMATLAB system. A person of ordinary skill in the art would have been motivated to combine the teachings of these two references because they describe the MultiMatlab system and include overlapping authors.

Further, for the reasons below and as provided in these invalidity contentions, the MultiMatlab references in view of one or more of the following references renders obvious the asserted claims. For example, several of the references below expressly describe the MultiMatlab system and further describe implementations as “MultiMatlab-like.” See Choy Thesis at 34 (“In conjunction with Parry Husbands, we developed a prototype implementation of a MultiMatlab-like, distributed Matlab package in Matlab*P, which we call the PPENGINE.”).

- Long Yin Choy and Alan Edelman. *MATLAB*P 2.0: A unified parallel MATLAB*, Technical Report, Massachusetts Institute of Technology, January 2003, available at <http://libraries.mit.edu/dspace-mit/>. (“Choy 1”).
- Ron Choy and Alan Edelman, *Parallel MATLAB: Doing It Right*, Proceedings of the IEEE, Vol. 93, No. 2, February 2005, 331-341 (“Choy 2”).
- R. Choy, *Parallel MATLAB survey*, 2001, available at: <http://theory.lcs.mit.edu/cly/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”).
- U.S. Patent Application Publication No. 2003/0195938, filed on January 10, 2003, issued on October 16, 2003 (“Howard”).
- “FFT algorithms and their adaptation to parallel processing,” by Chu et al., published in Linear Algebra and its Applications in 1998 (“Chu”).
- Agarwal, et al., “A High Performance Parallel Algorithm for 1-D FFT, Supercomputing ’94, 1994 (“Agarwal”).
- “Maple V,” PC Magazine, August 1992, pp. 419-425 (“PC Magazine 1”).
- “Maple V Looks Better With Improved Graphics,” PC Magazine, July 1993, p. 50 (“PC Magazine 2”).
- U.S. Patent No. 7,162,590 (“Pruvost”).
- MapleSoft Application Demo
- U.S. Pat. No. 7,694,158 (“Melpignano”).
- Anshuman Nayak, et al., “A Library based compiler to execute MATLAB Programs on a Heterogeneous Platform” (“Nayak”).
- Systems and publications discussed in Distributed Maple claim charts
- Systems and publications discussed in FFTW claim charts
- Systems and publications discussed in Intel Paragon Maple charts

Asserted Claim	Prior Art Disclosure
I. pre. A computer cluster comprising:	To the extent the preamble is limiting, the MultiMatlab References disclose “a computer cluster comprising.” MultiMatlab discloses a computer cluster for parallelizing Matlab computer algebra system (CAS) kernels. MultiMatlab was described in printed publications and was publicly used before the earliest alleged priority

Asserted Claim	Prior Art Disclosure
	<p>date of the patents-in-suit.</p> <p>Abstract:</p> <p>MATLAB®, a commercial product of The MathWorks, Inc., has become one of the principal languages of desktop scientific computing. A system is described that enables one to run MATLAB conveniently on multiple processors. Using short, MATLAB-style commands like Eval, Send, Recv, Beast, Min, and Sum, the user operating within one MATLAB session can start MATLAB processes on other machines and then pass commands and data between 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.</p> <p>Trefethen 1996, Abstract.</p> <div><p>User</p><pre>graph TD User --> IP[Interactive Processor] subgraph IP MP1[MATLAB Process] end IP --- IN[Interconnecting Network] subgraph P1 [Processor] MP2[MATLAB Process] end subgraph P2 [Processor] MP3[MATLAB Process] end subgraph P3 [Processor] MP4[MATLAB Process] end IN --- P1 IN --- P2 IN --- P3 P1 -.- P2 P2 -.- P3</pre><p>The diagram illustrates the MultiMATLAB architecture. At the top, a 'User' is connected via a double-headed arrow to an 'Interactive Processor'. Inside the 'Interactive Processor' is a 'MATLAB Process'. Below this, a horizontal line represents the 'Interconnecting Network'. Four 'Processor' boxes are connected to this network. Each 'Processor' box contains a 'MATLAB Process'. The first processor is connected to the network by a solid line, while the others are connected by dashed lines. Dotted lines also connect the individual processors in a sequence.</p></div> <p>Figure 1: MultiMATLAB Architecture</p> <p>Trefethen 1997 at 3.</p>

Asserted Claim	Prior Art Disclosure
	<p>MultiMatlab could run on an IBM SP2 or Unix workstation network. Trefethen 1996 at Abstract. The SP2 “consists of up to several hundred nodes which are all RISC 6000 superscalar processors. (Superscalar means that more than one instruction per clock cycle can be performed.) The SP2 can be considered a distributed memory machine because each node has its own local memory.”</p> <p>https://public.wsu.edu/~ari/miscellaneous/sp2_guide.html (last visited August 9, 2021). “There can be 4 to 16 processors per frame (or cabinet) and 1 to 16 frames per system. Frames can consist of BOTH thin and wide nodes. The main difference between thin and wide nodes is that generally wide nodes have the capacity for larger memory and a larger memory cache. Standard SP2 systems could contain up to 256 nodes. Up to 512 nodes can be ordered from IBM by special request. (ORNL has a 16 node SP2, Maui has a 400 node SP2, and Cornell has a 512 node machine.) An optional high performance switch (HPS) for inter-processor communication is usually included on most systems. Such a switch can interconnect up to 16 nodes with up to 16 other nodes. This high performance switch has a 40 MB/sec peak bi-directional bandwidth. More than one HPS may be included when the number of nodes in the SP2 machine is greater than 32. The IBM SP2 is fairly robust in that it has concurrent maintenance, which simply means each processor node can be removed, repaired, and rebooted without interrupting operations on other nodes.” <i>Id.</i></p> <p><u>Additional Evidence of Obviousness</u></p>

Asserted Claim	Prior Art Disclosure
	<p style="text-align: center;">II. MATLAB*P 2.0 OVERVIEW</p> <p>MATLAB*P 2.0 is a parallel MATLAB using the backend support approach, aimed at widespread circulation among a general audience. In order to achieve this, we took ideas from the approaches used by other software found in the survey. For example, the <i>embarrassingly parallel</i> approach allow simplistic, yet useful, division of work into multiple MATLAB sessions. The <i>message passing</i> approach, which is a superset of the <i>embarrassingly parallel</i> approach, allows finer control between the MATLAB sessions.</p> <p>In order to make MATLAB*P useful for a wider range of audience, these other approaches are incorporated into the system as well.</p> <p>Choy 1 at 1.</p>

Asserted Claim	Prior Art Disclosure
	<div><p>The diagram illustrates the architecture of MATLAB*P 2.0. A central dashed box represents 'Server #0'. Inside this box, at the top, are three boxes labeled 'PBLAS', 'EFTW', and 'ScalAPACK', with a horizontal line to their right. Below these is a 'Package Manager' box. To the right of the 'Package Manager' is a 'Client Manager' box. Below the 'Package Manager' is a 'Matrix Manager' box. To the left of the 'Matrix Manager' are three boxes labeled 'Matrix 1', 'Matrix 2', and 'Matrix 3', with a horizontal line below them. To the right of the 'Matrix Manager' is a 'Server Manager' box. Below the 'Server Manager' are four boxes labeled 'Server #1', 'Server #2', 'Server #3', and 'Server #4', with a horizontal line to their right. Arrows indicate connections: from 'PBLAS', 'EFTW', and 'ScalAPACK' to 'Package Manager'; from 'Matrix 1', 'Matrix 2', and 'Matrix 3' to 'Matrix Manager'; from 'Client Manager' to 'Proxy' and 'Client'; from 'Server Manager' to 'Server #1', 'Server #2', 'Server #3', and 'Server #4'; and from 'Proxy' to 'Client'.</p></div> <p>Fig. 1. Structure of MATLAB*P 2.0</p> <p>Choy 1 at 2.</p>

Asserted Claim	Prior Art Disclosure
	<p><i>B. Data Parallel/Embarassingly Parallel Operations</i></p> <p>One of the goals of the project is to make the software to be useful to as wide an audience as possible. In order to achieve this, we found that it would be fruitful to combine other parallel MATLAB approaches into MATLAB*P, to provide a unified parallel MATLAB framework.</p> <p>In conjunction with Parry Husbands, we developed a prototype implementation of a MultiMATLAB-like, distributed MATLAB package in MATLAB*P, which we call the PPEngine. With this package and associated m-files, we can run multiple MATLAB processes on the backend and evaluate MATLAB functions in parallel on dense matrices.</p> <p>Choy 1 at 2.</p>

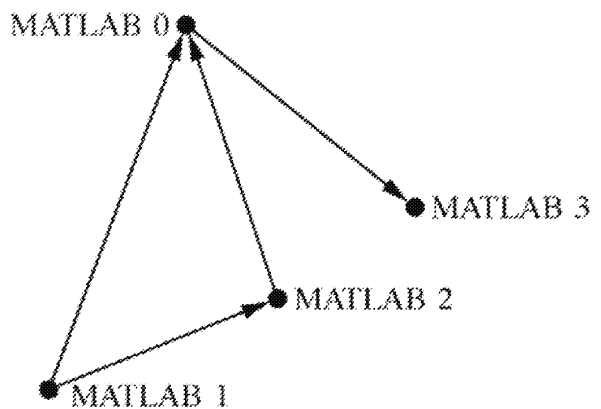
Asserted Claim	Prior Art Disclosure
	<p>The system works by starting up MATLAB engine instances on each node through calls to the MATLAB engine interface. From that point on, MATLAB commands can be relayed to the MATLAB engine.</p> <p>Examples of the usage of the PPEngine system:</p> <pre data-bbox="614 576 1024 673">>> % Example 1 >> a = 1:100*p; >> b = mm('chi2rnd',a);</pre> <p>The first example creates a distributed matrix of length 100, then fill it with random values from the chi-square distribution through calls to the function chi2rnd from MATLAB statistics toolbox.</p> <pre data-bbox="614 941 1024 1079">>> % Example 2 >> a = rand(100,100*p); >> b = rand(100,100*p); >> c = mm('plus',a,b);</pre> <p>Choy 1 at 3.</p>
1.a. a plurality of nodes, wherein each of the plurality of nodes comprises a	<p>The MultiMatlab References disclose “a plurality of nodes, wherein each of the plurality of nodes comprises a hardware processor.”</p> <p>MultiMatlab comprised a cluster with multiple computer nodes, each with at least one hardware processor:</p>

Asserted Claim	Prior Art Disclosure
hardware processor,	<div><pre>graph TD User[User] <--> IP[Interactive Processor] subgraph IP_Box [Interactive Processor] MP1[MATLAB Process] end subgraph P1_Box [Processor] MP2[MATLAB Process] end subgraph P2_Box [Processor] MP3[MATLAB Process] end subgraph P3_Box [Processor] MP4[MATLAB Process] end IP_Box --- IN[Interconnecting Network] P1_Box --- IN P2_Box --- IN P3_Box --- IN</pre></div> <p>Figure 1: MultiMATLAB Architecture</p> <p>Trefethen 1997 at 3.</p> <p><u>Additional Evidence of Obviousness</u></p>

Asserted Claim	Prior Art Disclosure
	<div><p>The diagram illustrates the architecture of MATLAB*P 2.0. A central dashed box represents 'Server #0'. Inside this box, at the top, are three boxes labeled 'PBLAS', 'EFTW', and 'ScaLAPACK', with a horizontal ellipsis to their right. Below these is a 'Package Manager' box. To the right of the 'Package Manager' is a 'Client Manager' box. Below the 'Package Manager' is a 'Matrix Manager' box. To the right of the 'Matrix Manager' is a 'Server Manager' box. Below the 'Matrix Manager' are three boxes labeled 'Matrix 1', 'Matrix 2', and 'Matrix 3', with a horizontal ellipsis below them. Arrows point from 'PBLAS', 'EFTW', and 'ScaLAPACK' to the 'Package Manager'. Arrows point from 'Matrix 1', 'Matrix 2', and 'Matrix 3' to the 'Matrix Manager'. Arrows point from the 'Client Manager' to the 'Package Manager' and the 'Matrix Manager'. Arrows point from the 'Server Manager' to the 'Package Manager' and the 'Matrix Manager'. Below the dashed box, there are four boxes labeled 'Server #1', 'Server #2', 'Server #3', and 'Server #4', with a horizontal ellipsis to their right. Arrows point from each of these boxes to the 'Server Manager' box. To the right of the dashed box, there are two boxes labeled 'Proxy' and 'Client', with a vertical line connecting them. Arrows point from the 'Client Manager' box to both the 'Proxy' and 'Client' boxes.</p></div> <p>Fig. 1. Structure of MATLAB*P 2.0</p> <p>Choy 1 at 2.</p>

Asserted Claim	Prior Art Disclosure
	<p data-bbox="588 318 849 354"><i>A. Test Platform</i></p> <ul data-bbox="631 380 1539 773" style="list-style-type: none"> <li data-bbox="631 380 1539 456">• Beowulf cluster with 9 nodes (2 nodes are used in the tests). <li data-bbox="631 467 1539 544">• Dual-processor nodes, each with two 1.533GHz Athlon MP processors. <li data-bbox="631 555 1539 631">• 1GB DDR RAM on each node. No swapping occurred during benchmarks. <li data-bbox="631 643 1539 719">• Fast ethernet (100Mbps/sec) interconnect. Intel Etherfast 410T switch. <li data-bbox="631 730 957 773">• Linux 2.4.18-4smp <p data-bbox="588 812 720 841">Choy 1 at 4.</p>

Asserted Claim	Prior Art Disclosure
	<p data-bbox="588 305 825 337"><i>A. Test Platform</i></p> <ul style="list-style-type: none"> <li data-bbox="627 362 1144 394">• Beowulf cluster with 15 nodes. <li data-bbox="627 402 1410 475">• Dual-processor nodes, each with two Intel Xeon 2.4 GHz with Hyperthreading. <li data-bbox="627 483 1410 678">• All tests are run with one process per processor. Also, the minimum number of nodes are used. So if a test is using four processors, two nodes are used. Thus, in this case we are using two shared-memory pairs of processors. <li data-bbox="627 686 1410 760">• Two gigabytes of RAM per node. No swapping occurred during the tests. <li data-bbox="627 768 1410 841">• Fast Ethernet (100 Mb/s) interconnect. Intel Etherfast 410T switch. <p data-bbox="588 865 750 898">Choy 2 at 337.</p>

Asserted Claim	Prior Art Disclosure
	 <p>Fig. 2. Message Passing.</p> <p>Choy 2 at 333.</p>
<p>1.b. wherein one or more of the nodes are configured to receive a command to start a cluster initialization process for the computer cluster,</p>	<p>The MultiMatlab References disclose “wherein one or more of the nodes are configured to receive a command to start a cluster initialization process for the computer cluster.”</p> <p>Each MultiMatlab cluster included at least one node that received user commands to initialize the cluster:</p> <p>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.</p> <p>Trefethen 1997 at 3.</p>

Asserted Claim	Prior Art Disclosure
	<p>The MultiMATLAB <code>start</code> command builds a P4 process group file of remote hosts, which are either explicitly specified by the user or taken from a default list, and then initializes MPICH. MATLAB processes are then started on the remote hosts. Each process iterates over a simple loop, waiting for and executing commands received from the user's interactive MATLAB process. The user may use a <code>quit</code> command to shut down the slaves and exit MultiMATLAB. Additionally, if the user quits MATLAB during a MultiMATLAB session, the slaves are automatically shut down.</p> <p>Trefethen 1996 at 11.</p> <pre>start(5)</pre> <p>MATLAB is then started on five additional processors taken from a predetermined list. Or perhaps the second author is sitting at a machine connected to Cornell's Computer Science Department network. He types</p> <pre>start(['gemini'; 'orion'; 'rigel'; 'castor'; 'pollux'])</pre> <p>Now MATLAB is started on the five processors with the names indicated. (Some names could be repeated, in which case multiple MATLAB processes would be started on a single processor.) In either case, when all the processes are started the message is returned,</p> <pre>5 MultiMATLAB processes running.</pre> <p>This total number of processors can subsequently be accessed by the MultiMATLAB command <code>Nproc</code>.</p> <p>Trefethen 1996 at 4.</p> <p><u>Additional Evidence of Obviousness</u></p>

Asserted Claim	Prior Art Disclosure
	<p data-bbox="584 310 1446 597"><i>1) MultiMATLAB:</i> MultiMATLAB is perhaps one of the most well known parallel MATLABs. It was developed by A. Trefethen, V. Menon, C. Chang, G. Czajkowski, C. Myers, and L. Trefethen at Cornell University. In MultiMATLAB, a MATLAB user can start MATLAB processes on other machines and then pass commands and data between these various processes. The most important commands are:</p> <ul data-bbox="633 613 1446 987" style="list-style-type: none"> • 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. <p data-bbox="584 987 720 1019">Choy 2 at 3.</p>

Asserted Claim	Prior Art Disclosure
	<p>In one embodiment, when server software 105 is initialized, one processor may be designated the head processor and all or some of the remaining processors may be designated slave processors that operate under the control of the head processor. The head processor may serve as the central processor and may receive the commands from the client. Thus, in one embodiment, only the client manager module 205 on the head processor is used. When the head processor receives a command from the client, the server manager 207 on the head processor may send messages to the “slave” processors to perform the desired operation. The library managers 209 on each of the processors may call the appropriate routine in libraries 215.</p> <p>Husbands at 11:10-22.</p>

Asserted Claim	Prior Art Disclosure
	<p>There are three different procedures for starting up the server, for different system setups:</p> <ol style="list-style-type: none">1. Procedure A is for systems where the client machine can connect to the server machine directly.2. Procedure B is for systems where the server exists on a private network which is exposed to the client only through a frontend machine. In this situation the frontend machine has no firewall. A typical example of this setup is a server running on a Beowulf cluster on a private network, and a client machine outside of that cluster.3. Procedure C is for systems similar to that in Procedure B, except that a firewall exists on the frontend. <p>Choy Thesis at 52.</p>

Asserted Claim	Prior Art Disclosure
	<p>Starting Star-P on a Linux Client System</p> <p>Your system administrator will usually have installed the Star-P software on the systems (client(s) and server) you will be running on in advance. The default location of the starp software is <i>/usr/local/starp/version</i>. Assuming this install location is in your shell path, then the following sequence will start the Star-P client (on a system named <i>sauroon</i>) and connect to the Star-P server configured by the administrator, which happens to be a system named <i>hope2.americas.sgi.com</i>.</p> <pre>sauroon\$ starp sgr@hope2.americas.sgi.com's password: *****</pre> <p style="text-align: center;">< M A T L A B > Copyright 1984-2005 The MathWorks, Inc. Version 7.0.4.352 (R14) Service Pack 2 January 29, 2005</p> <p>To get started, type one of these: <i>helpwin</i>, <i>helpdesk</i>, or <i>demo</i>. For product information, visit www.mathworks.com.</p> <p>Connecting to Star-P Server with 2 processes</p> <p>Star-P Client. (C) MIT 2002-2005. (C) Interactive Supercomputing, LLC 2004,2005. All Rights Reserved. Evaluation copy. Not to be redistributed. By using this software you agree to the terms and conditions described in AGREEMENT. Type help AGREEMENT</p> <p>Star-P User Guide at 15.</p>
I.c. and wherein each of the nodes is configured to access a non-transitory computer-readable	<p>The MultiMatlab References disclose “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.”</p> <p>MultiMatlab references disclose a cluster with multiple computer nodes, each of which included at least one</p>

Asserted Claim	Prior Art Disclosure
medium comprising program code for a single-node kernel that, when executed, is capable of causing the hardware processor to evaluate mathematical expressions; and	<p>computer-readable medium such as a disk and / or RAM memory. This at least one computer-readable medium stored the software components of MultiMatlab nodes, including the CAS (Computer Algebra System) single-node kernel program code (e.g., a MATLAB kernel) and additional components described below. In addition, the software could reside on a server and/or a shared directory in communication with the one or more processors, as was common in the art. MATLAB was a single-node kernel because it was designed to execute on a single computer. Likewise, Maple and Mathematica kernels could be used with implementations of MultiMatlab, e.g., the Star P implementation. These CAS kernels were single-node kernels designed to execute on a single computer. CAS kernels, including MATLAB, Maple and Mathematica kernels, were command interpreters, and translated user instructions in a high level computational language (e.g., Maple and Mathematica commands) into low level code for execution by each processor in the cluster. The processor executing the CAS kernel code evaluated mathematical expressions presented by the user to the CAS kernel through its user interface and translated by the kernel into machine-level code.</p> <p>The diagram illustrates the MultiMATLAB Architecture. At the top, a 'User' is connected via a double-headed arrow to an 'Interactive Processor'. Below the 'Interactive Processor' is a box labeled 'MATLAB Process'. To the right of the 'Interactive Processor' are three 'Processor' boxes, each containing a 'MATLAB Process' box. A horizontal dotted line connects the first 'Processor' to the last 'Processor'. All four boxes (Interactive Processor and the three Processors) are connected to a horizontal line labeled 'Interconnecting Network' at the bottom.</p> <p>Figure 1: MultiMATLAB Architecture</p> <p>Trefethen 1997 at 3.</p>

Asserted Claim	Prior Art Disclosure
	<p>Now MATLAB is started on the five processors with the names indicated. (Some names could be repeated, in which case multiple MATLAB processes would be started on a single processor.) In either case, when all the processes are started the message is returned,</p> <pre>5 MultiMATLAB processes running.</pre> <p>This total number of processors can subsequently be accessed by the MultiMATLAB command <code>nproc</code>.</p> <p>The standard MultiMATLAB command for executing commands on one or more processors is <code>Eval</code>. If the user now types</p> <pre>Eval('sqrt(2)')</pre> <p>then the MATLAB command <code>sqrt(2)</code> is executed on all six processors. The result is six repetitions of 1.4142, which is not very interesting. On the other hand the command</p> <pre>Eval('ID')</pre> <p>calls the MultiMATLAB command <code>ID</code> on each of the processors running. This command returns the number of the current process, an integer from 0 to <code>nproc-1</code>. Running it on all nodes might give the result</p> <pre>ans = 0 ans = 1 ans = 5 ans = 2 ans = 3 ans = 4</pre> <p>Trefethen 1996 at 4.</p>

Asserted Claim	Prior Art Disclosure
	<p data-bbox="1015 326 1123 347" style="text-align: center;">Abstract</p> <p data-bbox="627 370 1522 732">MATLAB is the most popular scientific computing environment available on uniprocessors today. Unfortunately, no such environment is currently available for multiprocessors. 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 extend 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. Preliminary results indicate that the MultiMATLAB system can offer the full performance of the underlying multiprocessor to the MATLAB environment.</p> <p data-bbox="579 764 810 792">Trefethen 1997 at 1.</p> <p data-bbox="605 841 1701 1068">On the other hand, the MATLAB version using SPMD message passing, shown in Figure 4b, demonstrates similar overhead with respect to C versions as in the sequential case. Note, however, that as the number of processors is increased, the memory requirement of each processor is decreased, and the overhead between MATLAB and C decreases significantly. This behavior is consistent with that of the sequential case. In particular, for a 1024 by 1024 matrix, this MATLAB code exhibits almost linear speedup up to 32 processors, while the C versions achieve a slower rate of speedup, and none beyond 16 processors.</p> <p data-bbox="579 1078 825 1105">Trefethen 1997 at 14.</p> <p data-bbox="579 1143 1019 1170"><u>Additional Evidence of Obviousness</u></p>

Asserted Claim	Prior Art Disclosure
	<p>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.</p> <p>Because of its roots in serial numerical libraries, MATLAB has always been a serial program. However, as modern en-</p> <p>Choy 1 at 1.</p>

Asserted Claim	Prior Art Disclosure
	<p>Scientific computing software applications are widely used in the science and engineering community as computing and modeling tools. As used herein, the term scientific computing software application refers to a software application capable of solving computational problems that are input by a user. Examples of commercially available scientific computing software applications include MATLAB™, produced and sold by The MathWorks, Natick, Mass., Maple™, produced and sold by Maplesoft, Waterloo, Ontario, Canada, and Mathematica™, produced and sold by Wolfram Research, Champaign, Ill. These software applications are useful in a variety of disciplines, including, for example, engineering, mathematics, physics, life sciences, chemistry, chemical engineering, social sciences, finance, economics, and others. It should be appreciated that while referred to as a scientific computing software application, scientific software applications are not limited to solving problems or performing calculations of a scientific nature, as any suitable type of problems or calculations may be processed by the scientific computing software application.</p> <p>Husbands at 3:37-56.</p> <p>“The kernel is the base of Maple’s system. It contains fundamental and primitive commands: the Maple language interpreter (which converts the commands you type into machine instructions your computer processor can understand), algorithms for numerical calculation, and routines to display results and perform other input and output operations.... The Maple kernel implements the most frequently used routines for integer and rational arithmetic and simple polynomial calculations.”</p> <p>Maple Guide at 88.</p>

Asserted Claim	Prior Art Disclosure
	<p>Some application programs include an interpreter that executes instructions provided to the program by a user, a script, or another source. Such an interpreter is sometimes called a “kernel” because, for example, the interpreter can manage at least some hardware resources of a computer system and/or can manage communications between those resources and software (for example, the provided instructions, which can include a high-level programming language). Some software programs include a kernel that is designed to communicate with a single node. An example of a software package that includes a kernel that is designed to communicate with a single node is Mathematica® from Wolfram Research, Inc. (“Mathematica”). Mathematics software packages from other vendors and other types of software can also include such a kernel.</p> <p>’289, 1:33-48.</p> <p>A kernel module 206 typically includes program code for interpreting high-level code, commands, and/or instructions supplied by a user or a script into low-level code, such as, for example, machine language or assembly language. In one</p> <p>’289, 23:1-4.</p>

Asserted Claim	Prior Art Disclosure
	<p>5.2 Embedding Mathematica</p> <p>The use of Mathematica in mathematical education sessions is twofold see Section 3:</p> <ol style="list-style-type: none"> 1. Using the notebook facility to display mathematical hypertext documents on screen (“presenter”), 2. Using the interpreter to run program demonstrations and support interactive exercises (“executor”). <p>Buchberger at 28.</p>
<p>I.d. a mechanism for the nodes to communicate results of mathematical expression evaluation with each other using a peer-to-peer architecture;</p>	<p>The MultiMatlab References disclose “a mechanism for the nodes to communicate results of mathematical expression evaluation with each other using a peer-to-peer architecture.”</p> <p>MultiMatlab discloses a mechanism for node-to-node communication, including peer-to-peer messaging of the results of mathematical expression evaluation, using the MPI messaging protocol and/or remote function calls. Implementations could use other protocols, such as PVM and BLACS. The “cluster node module” components enabling and involved in the peer-to-peer communication and scheduling mechanism comprised one or more interface modules, parallel server modules, and communication libraries, e.g., MPI libraries, and enabled the peer-to-peer architecture. The interface modules comprised one or more of the distributed server / client modules and could further comprise specialized computational libraries, such as BLAS and/or FFTW.</p> <p>One or more of these components comprise the claimed “mechanism for the nodes to communicate” on each node:</p> <p>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.</p> <p>Trefethen 1997 at 3.</p>

Asserted Claim	Prior Art Disclosure														
	<div><p>The diagram illustrates the MultiMATLAB architecture. At the top, a box labeled 'MATLAB' is connected by bidirectional arrows to two boxes: 'Sequential MEX Routine' and 'Parallel MEX Routine'. Below these, a 'MultiMATLAB Interface Module' is connected to 'MATLAB' and the 'Parallel MEX Routine' by bidirectional arrows. To the right of the interface module is an 'Indirection Table', also connected by bidirectional arrows. Below these components is a 'Communication Layer', which is connected to the 'MultiMATLAB Interface Module' and the 'Indirection Table' by bidirectional arrows. At the bottom, a 'Network' is connected to the 'Communication Layer' by a bidirectional arrow.</p></div> <p>Trefethen 1997 at 4.</p> <table><tr><th colspan="2">SPMD Communication</th></tr><tr><td>Send(pid,data)</td><td>Send data from one process to another.</td></tr><tr><td>Recv(pid)</td><td>Receive data sent from another process.</td></tr><tr><td>Barrier</td><td>Synchronize processes.</td></tr><tr><td>Bcast(pid,data)</td><td>Broadcasts data from processor pid to all processes.</td></tr><tr><td>Sum(data)</td><td>Adds data across all processors to form a global sum.</td></tr><tr><td>Collect(data)</td><td>Collects local data segments into a single global one.</td></tr></table> <p>Table 1: Selected commands in the MultiMATLAB system Trefethen 1997 at 9.</p>	SPMD Communication		Send(pid,data)	Send data from one process to another.	Recv(pid)	Receive data sent from another process.	Barrier	Synchronize processes.	Bcast(pid,data)	Broadcasts data from processor pid to all processes.	Sum(data)	Adds data across all processors to form a global sum.	Collect(data)	Collects local data segments into a single global one.
SPMD Communication															
Send(pid,data)	Send data from one process to another.														
Recv(pid)	Receive data sent from another process.														
Barrier	Synchronize processes.														
Bcast(pid,data)	Broadcasts data from processor pid to all processes.														
Sum(data)	Adds data across all processors to form a global sum.														
Collect(data)	Collects local data segments into a single global one.														

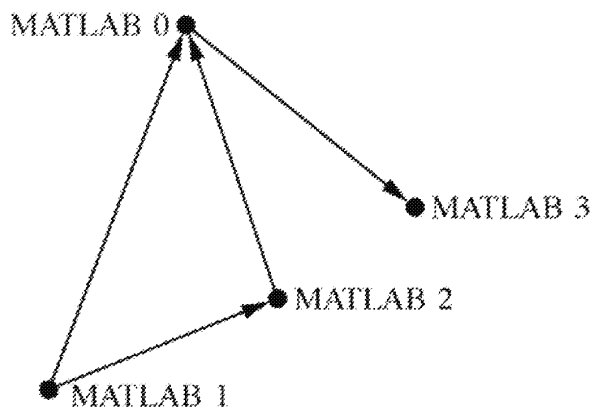
Asserted Claim	Prior Art Disclosure
	<pre> if ID==0 % first process: send a = 1 Send(ID+1,a) elseif ID == Nproc-1 % last process: receive and double a = 2*Recv else % middle processes: receive, double, and send a = 1*Recv Send(ID+1,a) end; </pre> <p>Process 0 creates the variable <code>a</code> with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of <code>a</code>, and sends it along to process 2; and so on. If there are six processors the command <code>Eval('cycle')</code> produces the output</p> <pre> a = 1 a = 2 a = 4 a = 8 a = 16 a = 32 </pre> <p>The processes run asynchronously, but since each <code>send</code> command is only executed after the corresponding <code>recv</code> has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning.</p> <p>Trefethen 1996 at 7.</p>

Asserted Claim	Prior Art Disclosure
	<div><pre>graph TD User[User] <--> IP[Interactive Processor] subgraph IP_Box [] direction TB IP_Label[Interactive Processor] MP1[MATLAB Process] end subgraph P1_Box [] direction TB P1_Label[Processor] MP2[MATLAB Process] end subgraph P2_Box [] direction TB P2_Label[Processor] MP3[MATLAB Process] end subgraph P3_Box [] direction TB P3_Label[Processor] MP4[MATLAB Process] end IP_Box --- IN[Interconnecting Network] P1_Box --- IN P2_Box --- IN P3_Box --- IN</pre></div> <p>Figure 1: MultiMATLAB Architecture</p> <p>Trefethen 1997 at 3.</p> <p>The term “a mechanism for the nodes to communicate” is governed by 35 U.S.C. § 112(6). The recited function to communicate user instructions and results of mathematical expression evaluation with other computer cluster nodes using a peer-to-peer architecture. The corresponding structure is Message-Passing Interface (“MPI”) as described at col 6:3-8; RMQ received message queue and MRQ message receiving queue of a cluster node module as described at col. 14:9-28 configured to execute the process described at col. 25:9-47.</p> <p>The function is disclosed by MultiMatlab as described in this exhibit. With regard to structure, for the reasons stated above it would have been obvious to implement MultiMatlab with the MPI communication library, not least because MultiMatlab supports MPI:</p> <p>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.</p> <p>Trefethen 1996 at 11.</p>

Asserted Claim	Prior Art Disclosure
	<p>The MPI protocol defines the standard <code>mpiIrecv</code> function that the '768 specification identifies with the MRQ queue (see col. 14:9-16):</p> <pre> 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. MPI Specification at 40. The standard <code>mpiIrecv</code> function includes a receive buffer, which comprises a message receiving queue. <i>Id.</i> The '768 specification associates the RMQ with the standard MPI <code>mpiTest</code> function (see col. 14:17-28), an implementation of which therefore discloses the RMQ: </pre>

Asserted Claim	Prior Art Disclosure									
	<p>MPLTEST(request, flag, status)</p> <table><tr><td>INOUT</td><td>request</td><td>communication request (handle)</td></tr><tr><td>OUT</td><td>flag</td><td>true if operation completed (logical)</td></tr><tr><td>OUT</td><td>status</td><td>status object (Status)</td></tr></table> <p>int MPI_Test(MPI_Request *request, int *flag, MPI_Status *status)</p> <p>MPI_TEST(REQUEST, FLAG, STATUS, IERROR)</p> <p>LOGICAL FLAG</p> <p>INTEGER REQUEST, STATUS(MPI_STATUS_SIZE), IERROR</p> <p>A call to MPI_TEST returns flag = true if the operation identified by request is complete. In such a case, the status object is set to contain information on the completed operation; if the communication object was created by a nonblocking send or receive, then it is deallocated and the request handle is set to MPIREQUEST_NULL. The call returns flag = false, otherwise. In this case, the value of the status object is undefined. MPI_TEST is a local operation.</p> <p>The return status object for a receive operation carries information that can be accessed as described in section 3.2.5. The status object for a send operation carries information as described in section 3.2.5. MPI_TEST_CANCELLED (3.2.5.1.1)</p> <p>MPI Specification at 42.</p> <p>Moreover, message receiving queues were a standard implementation of MPI.</p> <p>Since each MPI communicator represents an independent communication space, the highest level of the user's model is the communicator. Within a communi-</p>	INOUT	request	communication request (handle)	OUT	flag	true if operation completed (logical)	OUT	status	status object (Status)
INOUT	request	communication request (handle)								
OUT	flag	true if operation completed (logical)								
OUT	status	status object (Status)								

Asserted Claim	Prior Art Disclosure
	<p>cator there are, conceptually at least, three distinct “message queues,” which represent the state of the MPI subsystem. These are</p> <p>Send Queue: represents all of the outstanding send operations. Receive Queue: represents all of the outstanding receive operations. Unexpected Message Queue: represents messages that have been sent to this process, but have not yet been received.</p> <p>The send and receive queues store information about all of the unfinished send and receive operations that the process has started within the communicator. These might result either from blocking operations such as <code>MPI_Send</code> and <code>MPI_Recv</code> or nonblocking operations such as <code>MPI_Isend</code> or <code>MPI_Irecv</code>. Each entry on one of these queues contains the information that was passed to the function call that initiated the operation. Nonblocking operations will remain on these queues until they have completed and have been collected by a suitable <code>MPI_Wait</code>, <code>MPI_Test</code>, or one of the related multiple completion routines. Cownie 1999 at 1-2.</p> <p>As described by Cownie and known to a POSITA as a standard MPI operation and approach, there is at least one receive queue associated with the <code>mpiRecv</code> command, and implementing at least an instance of the MRQ. Likewise, entries in the MRQ are removed when the receive operation is complete and is collected when a corresponding <code>mpiTest</code> routine returns a “true” flag. The removal is implemented by comparing a queue of <code>mpiTest</code> results (an instance of the RMQ) with the MRQ. <i>See, e.g., MPI Specification at 38-43; Cownie 1999 at 1-2, 6-8.</i></p> <p><u>Additional Evidence of Obviousness</u></p>

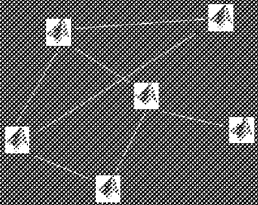
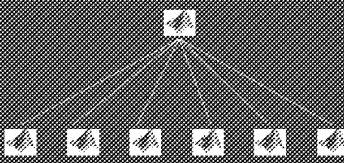
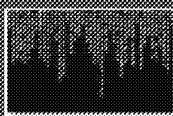
Asserted Claim	Prior Art Disclosure
	 <p>The diagram shows four nodes labeled MATLAB 0, MATLAB 1, MATLAB 2, and MATLAB 3. MATLAB 0 is at the top, MATLAB 1 is at the bottom left, MATLAB 2 is at the bottom right, and MATLAB 3 is to the right of MATLAB 2. Directed arrows indicate the flow of message passing: from MATLAB 1 to MATLAB 0, from MATLAB 0 to MATLAB 2, from MATLAB 2 to MATLAB 3, and from MATLAB 2 back to MATLAB 0.</p> <p>Fig. 2. Message Passing.</p> <p>Choy 2 at 333.</p>

Asserted Claim	Prior Art Disclosure
	<p><i>B. MultiMATLAB/MultiOctave Mode</i></p> <p>One of the goals of the project is to make the software to be useful to as wide an audience as possible. In order to achieve this, we found that it would be fruitful to combine other parallel MATLAB approaches into MATLAB * P, to provide a unified parallel MATLAB framework.</p> <p>In conjunction with P. Husbands, we developed a prototype implementation of a MultiMATLAB [35]-like, distributed MATLAB package in MATLAB * P, which we call the PPEngine. With this package and associated m-files, we can run multiple MATLAB processes on the back end and evaluate MATLAB functions in parallel on dense matrices.</p> <p>The system works by starting up MATLAB engine instances on each node through calls to the MATLAB engine interface. From that point on, MATLAB commands can be relayed to the MATLAB engine. The command is do this is <i>mm</i>.</p> <p>Choy 2 at 336</p>

Asserted Claim	Prior Art Disclosure
	<div><p>The diagram illustrates the architecture of MATLAB*P 2.0. A central dashed box represents 'Server #0'. Inside this box, at the top, are three boxes labeled 'PBLAS', 'FFTW', and 'ScaLAPACK', each with a line connecting to a 'Package Manager' box below them. To the left of the 'Package Manager' are three boxes labeled 'Matrix 1', 'Matrix 2', and 'Matrix 3', each with a line connecting to a 'Matrix Manager' box. To the right of the 'Package Manager' is a 'Client Manager' box. Below the 'Matrix Manager' is a 'Server Manager' box. Outside the 'Server #0' box, to the right, is a 'Proxy' box connected to the 'Client Manager', which in turn is connected to a 'Client' box. Below the 'Server #0' box are four boxes labeled 'Server #1', 'Server #2', 'Server #3', and 'Server #4'. Lines connect the 'Server Manager' box to each of these four server boxes. Ellipses (...) are used to indicate that there can be more than one instance of PBLAS, FFTW, ScaLAPACK, Matrix, and Server.</p></div> <p>Fig. 1. Structure of MATLAB*P 2.0</p> <p>Choy 1 at 2.</p>

Asserted Claim	Prior Art Disclosure
	<p>Server software 105 may operate in any suitable way to receive commands from the client and perform operations in parallel, as the invention is not limited in this respect. FIG. 2 is but one example of a suitable configuration of server software 105 for at least some embodiments. In FIG. 2, parallel server 201 includes a plurality of processors, 203a, 203b, . . . , 203n, and a memory 213. Each of processors 203a-203n includes four software modules: client managers 205a, 205b, . . . , 205n; server managers 207a, 207b, . . . , 207n; library managers 209a, 209b, . . . , 209n; and data managers 211a, 211b, . . . , 211n. Memory 213 of parallel server 201 stores a plurality of software libraries 215a, 215b, 215c, . . . , 215n that include routines for performing various operations.</p> <p>Client managers 205 interface with clients, and provides routines for reading commands and arguments from a client and sending results and other data back to the client. Server managers 207 handle communications between server processes executing on different processors of parallel server 201. That is, server managers 207 manage the transmission of data between the processors of parallel server 201 and collect results and error codes from the processors. Library managers</p> <p>Husbands at 10:46-67.</p>

Asserted Claim	Prior Art Disclosure
	<p data-bbox="599 310 1375 735">In one embodiment, when server software 105 is initialized, one processor may be designated the head processor and all or some of the remaining processors may be designated slave processors that operate under the control of the head processor. The head processor may serve as the central processor and may receive the commands from the client. Thus, in one embodiment, only the client manager module 205 on the head processor is used. When the head processor receives a command from the client, the server manager 207 on the head processor may send messages to the “slave” processors to perform the desired operation. The library managers 209 on each of the processors may call the appropriate routine in libraries 215.</p> <p data-bbox="599 748 1375 976">The server managers 207 on the processors 203 may communicate with each other in any suitable way, as the invention is not limited in this respect. In one embodiment, the message passage interface (MPI) application programming interface (API) may be used. Alternatively, the parallel virtual machine (PVM) API or any other suitable form of communication may be used.</p> <p data-bbox="599 989 842 1016">Husbands at 11:10-30.</p>

Asserted Claim	Prior Art Disclosure
	<div><div><h3>Parallel MATLABs</h3><div><div><div>CMTM</div><div>DP-Toolbox</div><div>MatlabMPI</div><div>MATmarks</div><div>MPITB/PVMTB</div><div>MultiMATLAB</div><div>Parallel Toolbox for MATLAB</div></div><div></div></div><div><div><div>DistributePP</div><div>MATLAB Parallelization Toolkit</div><div>MULTI Toolbox</div><div>Paralize</div><div>Parmatlab</div><div>Plab</div><div>PMI</div></div><div></div></div><div><div><div>Dlab</div><div>Paramat</div><div>PLAPACK</div><div>Matpar</div><div>MATLAB*P</div><div>Netsolve</div></div><div> </div></div><div><div><div>CONLAB Compiler</div><div>FALCON</div><div>MATCH</div><div>Menhir</div><div>Otter</div><div>ParAL</div><div>RTExpress</div></div><div>  </div></div></div></div> <div><p>Choy 3 at 20.</p><p>To the extent ACS contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of MultiMatlab and the knowledge of a POSITA, it would have been obvious to combine it with one or more of the other references in these invalidity contentions. For example, MultiMatlab and Matlab*P publications reference MPI and PVM-based parallel processing systems as comparable cluster computing systems, motivating a POSITA to look to such systems for modifications relating to this claim element. <i>See, e.g.,</i> Choy 2 at 334. Accordingly, it would have been obvious to modify MultiMatlab in view of the teachings of PVM and MPI publications, such as those referenced in these contentions, as well as references and systems implementing aspects of PVM or MPI. As another example, because MultiMatlab is directed to cluster</p></div>

Asserted Claim	Prior Art Disclosure
	<p>computing, it would have been obvious to combine with one or more of the prior art references or systems authored by the alleged inventors of the patents in suit or their collaborators, including patents, applications, articles and code by Messrs. Dager, Tannenbaum and Decyk. These references and code, including those related to the Appleseed cluster, Pooch, MacMPI and related projects, are likewise directed to cluster computing using an existing message passing library and a cluster management tool. Moreover, these prior art publications teach peer-to-peer communications between cluster nodes, and use of these cluster computing tools with CAS kernels. Given the similarity of their teaching, it would have been obvious to combine one of more of these systems or publications with MultiMatlab.</p> <p>In addition, this claim is rendered obvious by Intel Paragon Maple in view of Cramer, Choy, FFTW, Howard, and/or Chu for the reasons stated in the Cover Pleading, section IV.</p>
I.e. wherein the plurality of nodes comprises:	The MultiMatlab References disclose “wherein the plurality of nodes comprises,” for the reasons stated below.
I.f. 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,	<p>The MultiMatlab References disclose “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.”</p> <p>MultiMatlab discloses a root node computer with a processor and memory storing a MATLAB or other CAS kernel, and included a user interface:</p>

Asserted Claim	Prior Art Disclosure
	<div><p>The diagram illustrates the MultiMATLAB architecture. At the top, a 'User' is connected via a bidirectional arrow to an 'Interactive Processor'. This processor contains a 'MATLAB Process'. Below it are three 'Processor' blocks, each containing a 'MATLAB Process'. A dashed line connects the first three processors, and a dotted line connects the last one, indicating a network topology. All processors are connected to a common 'Interconnecting Network' at the bottom.</p><p>Figure 1: MultiMATLAB Architecture</p><p>Trefethen 1997 at 3.</p><p>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.</p><p>Trefethen 1997 at 3.</p></div>

Asserted Claim	Prior Art Disclosure
	<p>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.</p> <p>Trefethen 1997 at 3.</p> <p>The diagram illustrates a system architecture. At the top, a large box labeled 'MATLAB' is connected by horizontal double-headed arrows to two boxes on its right: 'Sequential MEX Routine' and 'Parallel MEX Routine'. Below 'MATLAB' is a box labeled 'MultiMATLAB Interface Module'. A vertical double-headed arrow connects 'MATLAB' to this module. To the right of the 'MultiMATLAB Interface Module' is a box labeled 'Indirection Table'. A vertical double-headed arrow connects the 'MultiMATLAB Interface Module' to the 'Indirection Table'. Below the 'Indirection Table' is a box labeled 'Communication Layer'. A vertical double-headed arrow connects the 'Indirection Table' to the 'Communication Layer'. Below the 'Communication Layer' is a box labeled 'Network'. A vertical double-headed arrow connects the 'Communication Layer' to the 'Network'.</p> <p>Trefethen 1997 at 4.</p>

Asserted Claim	Prior Art Disclosure
	<p>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 Trefethen 1997 at 6.</p> <p>Further, the '768 patent admits that a Matlab process includes a single-node kernel.</p> <p>For purposes of illustration, some embodiments are described herein in the context of cluster computing with Mathematica software. The present disclosure is not limited to a single software program; the systems and methods can be used with other application software such as, for example, Maple®, MATLAB®, MathCAD®, Apple Shake®, Apple® Compressor, IDL®, other applications employing an interpreter or a kernel, Microsoft Excel®, Adobe After Effects®, Adobe Premiere®, Adobe Photoshop®, Apple Final Cut Pro®, and Apple iMovie®. Some figures and/or descriptions, however, relate to embodiments of computer clusters running Mathematica. The system can include a variety of uses, including but not limited to students, educators, scientists, engineers, mathematicians, researchers, and technicians. It is also recognized that in other embodiments, the systems and methods can be implemented as a single module and/or implemented in conjunction with a variety of other modules. Moreover, the specific implementations described herein are set forth in order to illustrate, and not to limit, the disclosure.</p> <p>'768 patent, 4:14-33.</p>

Asserted Claim	Prior Art Disclosure
	<p>MultiMatlab could run on an IBM SP2 or Unix workstation network. Trefethen 1996 at Abstract. The SP2 “consists of up to several hundred nodes which are all RISC 6000 superscalar processors. (Superscalar means that more than one instruction per clock cycle can be performed.) The SP2 can be considered a distributed memory machine because each node has its own local memory.”</p> <p>https://public.wsu.edu/~ari/miscellaneous/sp2_guide.html (last visited August 9, 2021). “There can be 4 to 16 processors per frame (or cabinet) and 1 to 16 frames per system. Frames can consist of BOTH thin and wide nodes. The main difference between thin and wide nodes is that generally wide nodes have the capacity for larger memory and a larger memory cache. Standard SP2 systems could contain up to 256 nodes. Up to 512 nodes can be ordered from IBM by special request. (ORNL has a 16 node SP2, Maui has a 400 node SP2, and Cornell has a 512 node machine.) An optional high performance switch (HPS) for inter-processor communication is usually included on most systems. Such a switch can interconnect up to 16 nodes with up to 16 other nodes. This high performance switch has a 40 MB/sec peak bi-directional bandwidth. More than one HPS may be included when the number of nodes in the SP2 machine is greater than 32. The IBM SP2 is fairly robust in that it has concurrent maintenance, which simply means each processor node can be removed, repaired, and rebooted without interrupting operations on other nodes.” <i>Id.</i></p> <p><u>Additional Evidence of Obviousness</u></p>

Asserted Claim	Prior Art Disclosure
	<p data-bbox="592 313 1034 341"><i>B. MultiMATLAB/MultiOctave Mode</i></p> <p data-bbox="592 358 1293 524">One of the goals of the project is to make the software to be useful to as wide an audience as possible. In order to achieve this, we found that it would be fruitful to combine other parallel MATLAB approaches into MATLAB * P, to provide a unified parallel MATLAB framework.</p> <p data-bbox="592 529 1293 727">In conjunction with P. Husbands, we developed a prototype implementation of a MultiMATLAB [35]-like, distributed MATLAB package in MATLAB * P, which we call the PPEngine. With this package and associated m-files, we can run multiple MATLAB processes on the back end and evaluate MATLAB functions in parallel on dense matrices.</p> <p data-bbox="592 732 1293 898">The system works by starting up MATLAB engine instances on each node through calls to the MATLAB engine interface. From that point on, MATLAB commands can be relayed to the MATLAB engine. The command is do this is <i>mm</i>.</p> <p data-bbox="592 902 743 930">Choy 2 at 336</p>

Asserted Claim	Prior Art Disclosure
	<div><p>The diagram illustrates the architecture of MATLAB*P 2.0. A central dashed box represents 'Server #0'. Inside this box, at the top, are three boxes labeled 'PBLAS', 'PFTW', and 'ScaLAPACK', each with a line connecting to a 'Package Manager' box below them. To the left of the 'Package Manager' are three boxes labeled 'Matrix 1', 'Matrix 2', and 'Matrix 3', each with a line connecting to a 'Matrix Manager' box. To the right of the 'Matrix Manager' is a 'Server Manager' box. Above the 'Server Manager' is a 'Client Manager' box. Below the 'Server Manager' are four boxes labeled 'Server #1', 'Server #2', 'Server #3', and 'Server #4'. Lines connect the 'Server Manager' to each of these four server boxes. Outside the 'Server #0' box, to the right, is a 'Proxy' box connected to a 'Client' box. Lines also connect the 'Client Manager' to both the 'Proxy' and 'Client' boxes.</p></div> <p>Fig. 1. Structure of MATLAB*P 2.0</p> <p>Choy 1 at 2.</p>

Asserted Claim	Prior Art Disclosure
	<p>Server software 105 may operate in any suitable way to receive commands from the client and perform operations in parallel, as the invention is not limited in this respect. FIG. 2 is but one example of a suitable configuration of server software 105 for at least some embodiments. In FIG. 2, parallel server 201 includes a plurality of processors, 203a, 203b, . . . , 203n, and a memory 213. Each of processors 203a-203n includes four software modules: client managers 205a, 205b, . . . , 205n; server managers 207a, 207b, . . . , 207n; library managers 209a, 209b, . . . , 209n; and data managers 211a, 211b, . . . , 211n. Memory 213 of parallel server 201 stores a plurality of software libraries 215a, 215b, 215c, . . . , 215n that include routines for performing various operations.</p> <p>Client managers 205 interface with clients, and provides routines for reading commands and arguments from a client and sending results and other data back to the client. Server managers 207 handle communications between server processes executing on different processors of parallel server 201. That is, server managers 207 manage the transmission of data between the processors of parallel server 201 and collect results and error codes from the processors. Library managers</p> <p>Husbands at 10:46-67.</p> <p>The hardware processor on the first node, e.g., the MultiMatlab root node, accesses the node's memory comprising the user interface program code. Each CAS kernel, e.g., Maple, Mathematica, or Matlab, further includes program code for the computational single-node kernel. <i>See</i> 1.c.</p>
I.g. the first single-node kernel configured to	The MultiMatlab References disclose “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.”

Asserted Claim	Prior Art Disclosure
interpret user instructions and distribute calls to at least one of a plurality of other nodes for execution; and	<p>Each CAS kernel, for example a MATLAB kernel on the root node and remote nodes, is a command interpreter and translates user instructions in a high level computational language (e.g., Maple and Mathematica commands) into low level code for execution by each processor in the cluster. When configured in a MultiMatlab cluster, the kernel distributes calls, for example as MPI messages or remote calls, to other nodes for execution. This can be done in the MultiMATLAB mode, or in backend support mode, where the kernel accepts instructions and distributes them to remote kernels for execution. Each kernel is stored on disk and/or in memory on each cluster node.</p> <p>Figure 4b illustrates a parallel MATLAB implementation of conjugate gradients that uses the SPMD message passing described in Section 5.2. In this case, the matrix A and the different vectors are each distributed by row over all processes as in Figure 6. In each iteration, a process only does the computation required for its local data. However, communication is required for two different operations: the dot-products needed to compute ρ and α and the matrix-vector multiply needed to compute w_{local}. The dot products only require Trefethen 1997 at 11.</p> <p>The following code further illustrates the first MATLAB kernel interpreting user instructions and distributing calls to cluster nodes for execution:</p>

Asserted Claim	Prior Art Disclosure
	<pre> r_local = b_local; rho = Sum(r_local'*r_local); k = 0; while((k < kmax)) k = k+1; if (k == 1) p_local = r_local; else beta = rho/oldrho; p_local = r_local + beta*p_local; end p = Gather(p_local); w_local = A_local * p; alpha = rho/(Sum(p_local'*w_local)); x_local = x_local + alpha * p_local; r_local = r_local - alpha * w_local; oldrho = rho; rho = Sum(r_local'*r_local); end </pre> <p>b. Message Passing MATLAB</p> <p>Trefethen 1997 at 11.</p> <p>SPMD programs can be built upon <code>send</code> and <code>recv</code> commands. Typically the program contains <code>if</code> and <code>else</code> commands that specify different actions for different processes. For example, suppose the m-file</p>

Asserted Claim	Prior Art Disclosure
	<p><code>cycle.m</code> consists of the following program:</p> <pre> if ID==0 % first process: send a = 1 Send(ID+1,a) elseif ID == Nproc-1 % last process: receive and double a = 2*Recv else % middle processes: receive, double, and send a = 2*Recv Send(ID+1,a) end; </pre> <p>Process 0 creates the variable <code>a</code> with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of <code>a</code>, and sends it along to process 2; and so on. If there are six processors the command <code>Eval('cycle')</code> produces the output</p> <pre> a = 1 a = 2 a = 4 a = 8 a = 16 a = 32 </pre> <p>The processes run asynchronously, but since each <code>Send</code> command is only executed after the corresponding <code>Recv</code> has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning.</p> <p>Trefethen 1996 at 6-7.</p> <p><u>Additional Evidence of Obviousness</u></p>

Asserted Claim	Prior Art Disclosure
	<p>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.</p> <p>Because of its roots in serial numerical libraries, MATLAB has always been a serial program. However, as modern en-</p> <p>Choy 1 at 1.</p> <p>“The kernel is the base of Maple’s system. It contains fundamental and primitive commands: the Maple language interpreter (which converts the commands you type into machine instructions your computer processor can understand), algorithms for numerical calculation, and routines to display results and perform other input and output operations.... The Maple kernel implements the most frequently used routines for integer and rational arithmetic and simple polynomial calculations.”</p> <p>Maple Guide at 88.</p>

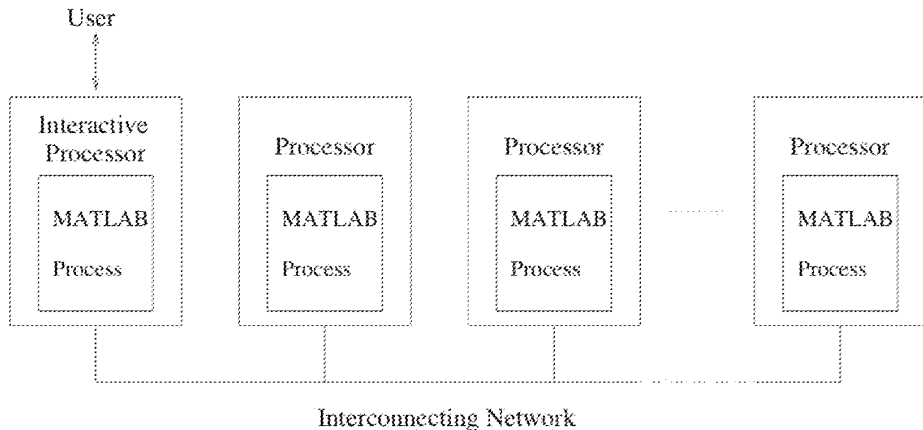
Asserted Claim	Prior Art Disclosure
	<p>Some application programs include an interpreter that executes instructions provided to the program by a user, a script, or another source. Such an interpreter is sometimes called a “kernel” because, for example, the interpreter can manage at least some hardware resources of a computer system and/or can manage communications between those resources and software (for example, the provided instructions, which can include a high-level programming language). Some software programs include a kernel that is designed to communicate with a single node. An example of a software package that includes a kernel that is designed to communicate with a single node is Mathematica® from Wolfram Research, Inc. (“Mathematica”). Mathematics software packages from other vendors and other types of software can also include such a kernel.</p> <p>’289, 1:33-48.</p> <p>A kernel module 206 typically includes program code for interpreting high-level code, commands, and/or instructions supplied by a user or a script into low-level code, such as, for example, machine language or assembly language. In one</p> <p>’289, 23:1-4.</p>

Asserted Claim	Prior Art Disclosure
	<p data-bbox="612 305 1090 337">5.2 Embedding Mathematica</p> <p data-bbox="612 375 1526 431">The use of Mathematica in mathematical education sessions is twofold see Section 3:</p> <ol data-bbox="644 480 1526 621" style="list-style-type: none"> <li data-bbox="644 480 1526 537">1. Using the notebook facility to display mathematical hypertext documents on screen (“presenter”), <li data-bbox="644 561 1526 621">2. Using the interpreter to run program demonstrations and support interactive exercises (“executor”). <p data-bbox="579 646 784 678">Buchberger at 28.</p> <p data-bbox="595 719 1034 751"><i>B. MultiMATLAB/MultiOctave Mode</i></p> <p data-bbox="595 768 1293 930">One of the goals of the project is to make the software to be useful to as wide an audience as possible. In order to achieve this, we found that it would be fruitful to combine other parallel MATLAB approaches into MATLAB * P, to provide a unified parallel MATLAB framework.</p> <p data-bbox="595 938 1293 1141">In conjunction with P. Husbands, we developed a prototype implementation of a MultiMATLAB [35]-like, distributed MATLAB package in MATLAB * P, which we call the PPEngine. With this package and associated m-files, we can run multiple MATLAB processes on the back end and evaluate MATLAB functions in parallel on dense matrices.</p> <p data-bbox="595 1149 1293 1312">The system works by starting up MATLAB engine instances on each node through calls to the MATLAB engine interface. From that point on, MATLAB commands can be relayed to the MATLAB engine. The command to do this is <i>mm</i>.</p> <p data-bbox="579 1320 741 1352">Choy 2 at 336</p>

Asserted Claim	Prior Art Disclosure
	<p>We present MATLAB *P [15] as an example of what could be a “right” parallel MATLAB. MATLAB * P is a parallel MATLAB using the back-end support approach, aimed at widespread circulation among a general audience. In order to achieve this, we took ideas from the approaches used by other software found in the survey. For example, the <i>embarrassingly parallel</i> approach allow simple, yet useful, division of work into multiple MATLAB sessions. The <i>message passing</i> approach, which is a superset of the embarrassingly parallel approach, allows finer control between the MATLAB sessions.</p> <p>The main idea in MATLAB *P is that data exist on the parallel server as distributed matrices. Any operations on a distributed matrix (which exists in the MATLAB front end only as a handle) will be relayed to the server transparently. The server calls the appropriate routine from a parallel numerical library (e.g. ScaLAPACK, FFTW, . . .) and the results stay on the server until explicitly requested.</p> <p>Choy 2 at 336.</p> <p>The following code further illustrates the first MATLAB kernel interpreting user instructions and distributing calls to cluster nodes for execution:</p>

Asserted Claim	Prior Art Disclosure
	<p>The system works by starting up MATLAB engine instances on each node through calls to the MATLAB engine interface. From that point on, MATLAB commands can be relayed to the MATLAB engine.</p> <p>Examples of the usage of the PPEngine system:</p> <pre data-bbox="599 544 980 641">>> % Example 1 >> a = 1:100*p; >> b = mm('chi2rnd',a);</pre> <p>The first example creates a distributed matrix of length 100, then fill it with random values from the chi-square distribution through calls to the function chi2rnd from MATLAB statistics toolbox.</p> <p>Choy 1 at 3.</p>

Asserted Claim	Prior Art Disclosure
	<p>Server software 105 may operate in any suitable way to receive commands from the client and perform operations in parallel, as the invention is not limited in this respect. FIG. 2 is but one example of a suitable configuration of server software 105 for at least some embodiments. In FIG. 2, parallel server 201 includes a plurality of processors, 203a, 203b, . . . , 203n, and a memory 213. Each of processors 203a-203n includes four software modules: client managers 205a, 205b, . . . , 205n; server managers 207a, 207b, . . . , 207n; library managers 209a, 209b, . . . , 209n; and data managers 211a, 211b, . . . , 211n. Memory 213 of parallel server 201 stores a plurality of software libraries 215a, 215b, 215c, . . . , 215n that include routines for performing various operations.</p> <p>Client managers 205 interface with clients, and provides routines for reading commands and arguments from a client and sending results and other data back to the client. Server managers 207 handle communications between server processes executing on different processors of parallel server 201. That is, server managers 207 manage the transmission of data between the processors of parallel server 201 and collect results and error codes from the processors. Library managers</p> <p>Husbands at 10:46-67.</p> <p>To the extent ACS contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of MultiMatlab and the knowledge of a POSITA, it would have been obvious to combine it with one or more of the other references in these invalidity contentions. For example, MultiMatlab and Matlab *P publications reference MPI and PVM-based parallel processing systems as comparable cluster computing systems, motivating a POSITA to look to such systems for modifications relating to this claim element. <i>See, e.g., Choy 2 at 334.</i> Accordingly, it would have been obvious to modify MultiMatlab in view of the teachings of PVM and MPI publications, such as those referenced in these contentions, as well as references and systems</p>

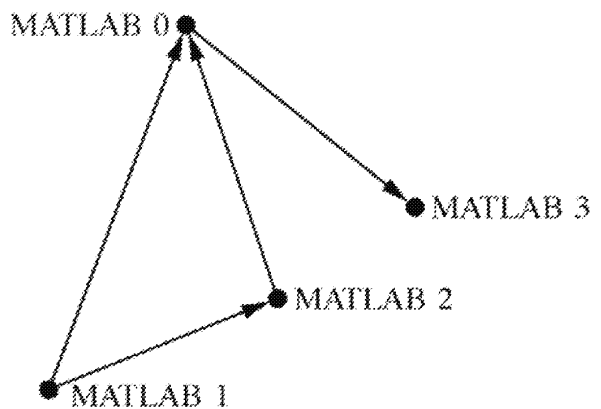
Asserted Claim	Prior Art Disclosure
	<p>implementing aspects of PVM or MPI. As another example, because MultiMatlab is directed to cluster computing, it would have been obvious to combine with one or more of the prior art references or systems authored by the alleged inventors of the patents in suit or their collaborators, including patents, applications, articles and code by Messrs. Dager, Tannenbaum and Decyk. These references and code, including those related to the Appleseed cluster, Pooch, MacMPI and related projects, are likewise directed to cluster computing using an existing message passing library and a cluster management tool. Moreover, these prior art publications teach peer-to-peer communications between cluster nodes, and use of these cluster computing tools with CAS kernels. Given the similarity of their teaching, it would have been obvious to combine one of more of these systems or publications with MultiMatlab.</p> <p>In addition, this claim is rendered obvious by Intel Paragon Maple in view of Cramer, Choy, FFTW, Howard, and/or Chu for the reasons stated in the Cover Pleading, section IV.</p>
<p>I.h. a second node comprising a second hardware processor with a plurality of processing cores,</p>	<p>The MultiMatlab References disclose “a second node comprising a second hardware processor with a plurality of processing cores.”</p>  <pre> graph TD User[User] <--> IP[Interactive Processor] subgraph Nodes IP --- P1[Processor] P1 --- P2[Processor] P2 --- P3[Processor] P3 --- P4[Processor] end P1 --- IN[Interconnecting Network] P2 --- IN P3 --- IN P4 --- IN subgraph Processes IP --- MP1[MATLAB Process] P1 --- MP2[MATLAB Process] P2 --- MP3[MATLAB Process] P3 --- MP4[MATLAB Process] end </pre> <p>Figure 1: MultiMATLAB Architecture</p>

Asserted Claim	Prior Art Disclosure
	<p>Trefethen 1997 at 3.</p> <p>MultiMatlab could run on an IBM SP2 or Unix workstation network. Trefethen 1996 at Abstract. The SP2 “consists of up to several hundred nodes which are all RISC 6000 superscalar processors. (Superscalar means that more than one instruction per clock cycle can be performed.) The SP2 can be considered a distributed memory machine because each node has its own local memory.”</p> <p>https://public.wsu.edu/~ari/miscellaneous/sp2_guide.html (last visited August 9, 2021). “There can be 4 to 16 processors per frame (or cabinet) and 1 to 16 frames per system. Frames can consist of BOTH thin and wide nodes. The main difference between thin and wide nodes is that generally wide nodes have the capacity for larger memory and a larger memory cache. Standard SP2 systems could contain up to 256 nodes. Up to 512 nodes can be ordered from IBM by special request. (ORNL has a 16 node SP2, Maui has a 400 node SP2, and Cornell has a 512 node machine.) An optional high performance switch (HPS) for inter-processor communication is usually included on most systems. Such a switch can interconnect up to 16 nodes with up to 16 other nodes. This high performance switch has a 40 MB/sec peak bi-directional bandwidth. More than one HPS may be included when the number of nodes in the SP2 machine is greater than 32. The IBM SP2 is fairly robust in that it has concurrent maintenance, which simply means each processor node can be removed, repaired, and rebooted without interrupting operations on other nodes.” <i>Id.</i></p> <p><u>Additional Evidence of Obviousness</u></p> <p>Matlab *P was configured to run on computers, including the computer comprising “a second node,” that included multiple processing cores:</p>

Asserted Claim	Prior Art Disclosure
	<div><p>The diagram illustrates the architecture of MATLAB*P 2.0. A central dashed box represents 'Server #0'. Inside this box, at the top, are three boxes labeled 'PBLAS', 'EFTW', and 'ScaLAPACK', with a horizontal ellipsis to their right. Below these is a 'Package Manager' box. To the right of the 'Package Manager' is a 'Client Manager' box. Below the 'Package Manager' is a 'Matrix Manager' box. To the left of the 'Matrix Manager' are three boxes labeled 'Matrix 1', 'Matrix 2', and 'Matrix 3', with a horizontal ellipsis below them. Arrows point from 'PBLAS', 'EFTW', and 'ScaLAPACK' to the 'Package Manager'. Arrows point from 'Matrix 1', 'Matrix 2', and 'Matrix 3' to the 'Matrix Manager'. Arrows point from the 'Package Manager' and 'Client Manager' to a 'Server Manager' box located to the right of the 'Matrix Manager'. Outside the dashed box, at the bottom, are four boxes labeled 'Server #1', 'Server #2', 'Server #3', and 'Server #4', with a horizontal ellipsis to their right. Arrows point from each of these server boxes to the 'Server Manager' box. To the right of the dashed box, outside it, are two boxes: 'Proxy' at the top and 'Client' at the bottom, connected by a vertical line. Arrows point from the 'Client Manager' box to both the 'Proxy' and 'Client' boxes.</p></div> <p>Fig. 1. Structure of MATLAB*P 2.0</p> <p>Choy 1 at 2.</p>

Asserted Claim	Prior Art Disclosure
	<p data-bbox="588 318 849 354"><i>A. Test Platform</i></p> <ul data-bbox="631 380 1539 773" style="list-style-type: none"> <li data-bbox="631 380 1539 456">• Beowulf cluster with 9 nodes (2 nodes are used in the tests). <li data-bbox="631 467 1539 544">• Dual-processor nodes, each with two 1.533GHz Athlon MP processors. <li data-bbox="631 555 1539 631">• 1GB DDR RAM on each node. No swapping occurred during benchmarks. <li data-bbox="631 643 1539 719">• Fast ethernet (100Mbps/sec) interconnect. Intel Etherfast 410T switch. <li data-bbox="631 730 957 773">• Linux 2.4.18-4smp <p data-bbox="588 812 720 841">Choy 1 at 4.</p>

Asserted Claim	Prior Art Disclosure
	<p data-bbox="588 305 825 337"><i>A. Test Platform</i></p> <ul style="list-style-type: none"> <li data-bbox="627 362 1144 394">• Beowulf cluster with 15 nodes. <li data-bbox="627 402 1410 475">• Dual-processor nodes, each with two Intel Xeon 2.4 GHz with Hyperthreading. <li data-bbox="627 483 1410 678">• All tests are run with one process per processor. Also, the minimum number of nodes are used. So if a test is using four processors, two nodes are used. Thus, in this case we are using two shared-memory pairs of processors. <li data-bbox="627 686 1410 760">• Two gigabytes of RAM per node. No swapping occurred during the tests. <li data-bbox="627 768 1410 841">• Fast Ethernet (100 Mb/s) interconnect. Intel Etherfast 410T switch. <p data-bbox="588 865 750 898">Choy 2 at 337.</p>

Asserted Claim	Prior Art Disclosure
	 <p>Fig. 2. Message Passing.</p> <p>Choy 2 at 333.</p> <p>In addition, during this timeframe, multi-core processors were prevalent in the marketplace, and it would have been a matter of mere design choice to use computers that contained hardware processors with multiple processing cores. The '768 patent itself admits that dual-core processors were conventional in the prior art, as it references the well-known Intel Core 2 Duo processors as a multi-core example. <i>See, e.g.</i>, 7:10-32. It would have been obvious to use Intel Core 2 Duo processors with MultiMatlab. In addition, as established for claim 39, it was known to use hardware processors with a plurality of processing cores comprising DSPs, FPGAs and other types of processors. <i>See</i> claim 39.</p>
I.i. wherein the second node is configured to receive calls from	The MultiMatlab References disclose “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.”

Asserted Claim	Prior Art Disclosure
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;	<p>MultiMatlab implemented peer-to-peer communications between nodes using MPI message passing or through a distributed server process. When configured in a MultiMatlab cluster, each CAS kernel or parallel compute library receives calls from the root process and executes code evaluating mathematical expressions. Each node sends computation results to other nodes, including the second node communicating with the third node. The third node accesses the results provided by the second node and performs further mathematical evaluations. Eventually, the results are returned to the root process and user interface. This can be done in the MultiMATLAB mode, or in backend support mode.</p> <p>The diagram illustrates the MultiMATLAB Architecture. At the top, a 'User' is connected via a double-headed arrow to an 'Interactive Processor'. Below this, there are four boxes representing processors. The first box is labeled 'Interactive Processor' and contains a 'MATLAB Process'. The next three boxes are labeled 'Processor' and each also contains a 'MATLAB Process'. A dashed line connects the four processor boxes, representing an 'Interconnecting Network'. Below the network line, the text 'Interconnecting Network' is written. Below the diagram, the caption 'Figure 1: MultiMATLAB Architecture' is present. At the bottom of the disclosure section, the text 'Trefethen 1997 at 3.' is written.</p>

Asserted Claim	Prior Art Disclosure
	<p>Figure 4b illustrates a parallel MATLAB implementation of conjugate gradients that uses the SPMD message passing described in Section 5.2. In this case, the matrix A and the different vectors are each distributed by row over all processes as in Figure 6. In each iteration, a process only does the computation required for its local data. However, communication is required for two different operations: the dot-products needed to compute ρ and α and the matrix-vector multiply needed 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.</p> <p>Trefethen 1997 at 11, 13.</p> <p>The following code further illustrates the second MATLAB kernel receiving calls from the first kernel, evaluating an expression, and communicating results to other nodes, including a third node:</p>

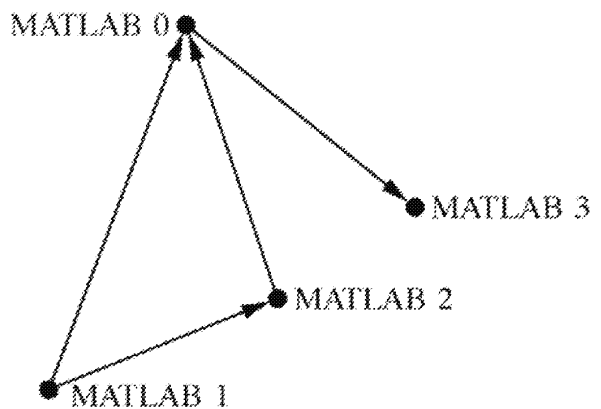
Asserted Claim	Prior Art Disclosure
	<pre> r_local = b_local; rho = Sum(r_local'*r_local); k = 0; while((k < kmax)) k = k+1; if (k == 1) p_local = r_local; else beta = rho/oldrho; p_local = r_local + beta*p_local; end p = Gather(p_local); w_local = A_local * p; alpha = rho/(Sum(p_local'*w_local)); x_local = x_local + alpha * p_local; r_local = r_local - alpha * w_local; oldrho = rho; rho = Sum(r_local'*r_local); end </pre> <p>b. Message Passing MATLAB</p> <p>Trefethen 1997 at 11.</p>

MultiMATLAB implemented MPI message passing that allowed any node to pass computation results to any other node, as further illustrated by the following code:

Asserted Claim	Prior Art Disclosure
	<pre>if ID==0 % first process: send a = 1 Send(ID+1,a) elseif ID == Nproc-1 % last process: receive and double a = 2*Recv else % middle processes: receive, double, and send a = 2*Recv Send(ID+1,a) end;</pre> <p>Process 0 creates the variable <code>a</code> with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of <code>a</code>, and sends it along to process 2; and so on. If there are six processors the command <code>Eval('cycle')</code> produces the output</p> <pre>a = 1 a = 2 a = 4 a = 8 a = 16 a = 32</pre> <p>The processes run asynchronously, but since each <code>Send</code> command is only executed after the corresponding <code>Recv</code> has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning.</p> <p>Trefethen 1996 at 7.</p> <p><u>Additional Evidence of Obviousness</u></p>

Asserted Claim	Prior Art Disclosure
	<p style="text-align: center;">Structure of the system (2)</p> <ul style="list-style-type: none"> • Client (C++) <ul style="list-style-type: none"> – Attaches to MATLAB through MEX interface – Relays commands to server and process returns • Server (C++/MPI) <ul style="list-style-type: none"> – Manipulates/processes data (matrices) – Performs the actual parallel computation by calling the appropriate library routines <p>Choy 3 at 36.</p>

Asserted Claim	Prior Art Disclosure
	<p>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.</p> <p>Because of its roots in serial numerical libraries, MATLAB has always been a serial program. However, as modern en-</p> <p>Choy 1 at 1.</p>

Asserted Claim	Prior Art Disclosure
	 <p>The diagram shows four nodes labeled MATLAB 0, MATLAB 1, MATLAB 2, and MATLAB 3. MATLAB 0 is at the top, MATLAB 1 is at the bottom left, MATLAB 2 is at the bottom right, and MATLAB 3 is to the right of MATLAB 2. Arrows indicate message passing: MATLAB 1 points to MATLAB 0, MATLAB 0 points to MATLAB 2, MATLAB 2 points to MATLAB 3, and MATLAB 3 points back to MATLAB 0.</p> <p>Fig. 2. Message Passing.</p> <p>Choy 2 at 333.</p>

Asserted Claim	Prior Art Disclosure
	<p><i>B. MultiMATLAB/MultiOctave Mode</i></p> <p>One of the goals of the project is to make the software to be useful to as wide an audience as possible. In order to achieve this, we found that it would be fruitful to combine other parallel MATLAB approaches into MATLAB * P, to provide a unified parallel MATLAB framework.</p> <p>In conjunction with P. Husbands, we developed a prototype implementation of a MultiMATLAB [35]-like, distributed MATLAB package in MATLAB * P, which we call the PPEngine. With this package and associated m-files, we can run multiple MATLAB processes on the back end and evaluate MATLAB functions in parallel on dense matrices.</p> <p>The system works by starting up MATLAB engine instances on each node through calls to the MATLAB engine interface. From that point on, MATLAB commands can be relayed to the MATLAB engine. The command is do this is <i>mm</i>.</p> <p>Choy 2 at 336</p>

Asserted Claim	Prior Art Disclosure
	<p>We present MATLAB *P [15] as an example of what could be a “right” parallel MATLAB. MATLAB * P is a parallel MATLAB using the back-end support approach, aimed at widespread circulation among a general audience. In order to achieve this, we took ideas from the approaches used by other software found in the survey. For example, the <i>embarrassingly parallel</i> approach allow simple, yet useful, division of work into multiple MATLAB sessions. The <i>message passing</i> approach, which is a superset of the embarrassingly parallel approach, allows finer control between the MATLAB sessions.</p> <p>The main idea in MATLAB *P is that data exist on the parallel server as distributed matrices. Any operations on a distributed matrix (which exists in the MATLAB front end only as a handle) will be relayed to the server transparently. The server calls the appropriate routine from a parallel numerical library (e.g. ScaLAPACK, FFTW, . . .) and the results stay on the server until explicitly requested.</p> <p>Choy 2 at 336.</p>

Asserted Claim	Prior Art Disclosure
	<p>Server software 105 may operate in any suitable way to receive commands from the client and perform operations in parallel, as the invention is not limited in this respect. FIG. 2 is but one example of a suitable configuration of server software 105 for at least some embodiments. In FIG. 2, parallel server 201 includes a plurality of processors, 203a, 203b, . . . , 203n, and a memory 213. Each of processors 203a-203n includes four software modules: client managers 205a, 205b, . . . , 205n; server managers 207a, 207b, . . . , 207n; library managers 209a, 209b, . . . , 209n; and data managers 211a, 211b, . . . , 211n. Memory 213 of parallel server 201 stores a plurality of software libraries 215a, 215b, 215c, . . . , 215n that include routines for performing various operations.</p> <p>Client managers 205 interface with clients, and provides routines for reading commands and arguments from a client and sending results and other data back to the client. Server managers 207 handle communications between server processes executing on different processors of parallel server 201. That is, server managers 207 manage the transmission of data between the processors of parallel server 201 and collect results and error codes from the processors. Library managers</p> <p>Husbands at 10:46-67.</p>

Asserted Claim	Prior Art Disclosure
	<p>The server itself is divided in four self-contained parts:</p> <ol style="list-style-type: none">1. Client Connection Manager Client Connection Manager is responsible for communications with the client. It provides functions for reading commands and arguments from the client and sending the results back to the client. It is only used in the head server process.2. Server Connection Manager Server Connection Manager takes care of communication between server processes. It mainly controls broadcasting of commands and arguments from head

Asserted Claim	Prior Art Disclosure
	<p>process to the slave processes, and collection of results and error codes. Also it provides rank and size information to the processes.</p> <p>3. Package Manager</p> <p>Package Manager is responsible for maintaining a list of available packages and functions provided by them. When initiated by the server process, Package Manager will also perform the actual call to the functions.</p> <p>4. Matrix Manager</p> <p>Matrix Manager contains all the functions needed to create, delete and change the matrices on the server processes. It maintains a mapping from client-side matrix identifiers to actual matrices on the server. It is also responsible for performing garbage collection.</p> <p>Choy Thesis at 26, 28</p> <p>The following additional example illustrates an FFT calculation on a Matlab *P cluster, resulting in communications of intermediate results between the nodes in the cluster:</p>

Asserted Claim	Prior Art Disclosure
	<p style="text-align: center;">Example 3</p> <p><code>% FFT2 in 4 lines</code></p> <ul style="list-style-type: none"> • <code>A = randn(10000,10000*p);</code> • <code>B = mm('fft',A);</code> • <code>C = B.{'</code> • <code>D = mm('fft',C);</code> • <code>F = D.{'</code> <p>Choy 3 at 47.</p> <p>The following additional example further illustrates MPI message passing between Matlab*P cluster nodes:</p>

Asserted Claim	Prior Art Disclosure
	<pre data-bbox="631 316 1422 982">function result = simpletest(X) import mpi.*; MPI.init; if MPI.COMM_WORLD.rank == 0; data = [4 2]; MPI.COMM_WORLD.send(data, 2, 1, 13); result = 9; elseif MPI.COMM_WORLD.rank == 1 got = MPI.COMM_WORLD.recv(2, 0, 13); result = got(1) + got(2); end MPI.fnlz;</pre> <p data-bbox="579 1036 735 1062">Choy 3 at 49.</p> <p data-bbox="579 1101 1849 1352">To the extent ACS contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of MultiMatlab and the knowledge of a POSITA, it would have been obvious to combine it with one or more of the other references in these invalidity contentions. For example, MultiMatlab and Matlab *P publications reference MPI and PVM-based parallel processing systems as comparable cluster computing systems, motivating a POSITA to look to such systems for modifications relating to this claim element. <i>See, e.g.,</i> Choy 2 at 334. Accordingly, it would have been obvious to modify MultiMatlab in view of the teachings of PVM and MPI publications, such as those referenced in these contentions, as well as references and systems implementing aspects of PVM or MPI. As another example, because MultiMatlab is directed to cluster</p>

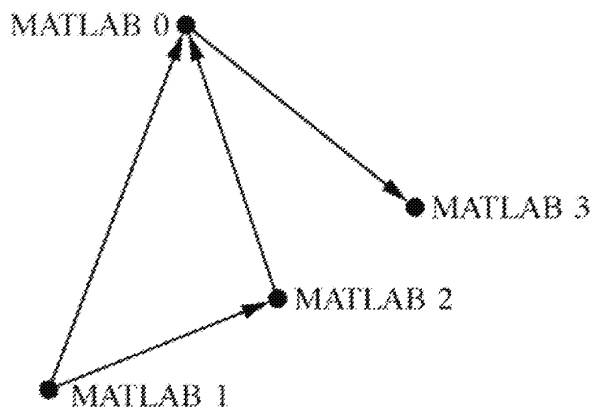
Asserted Claim	Prior Art Disclosure
	<p>computing, it would have been obvious to combine with one or more of the prior art references or systems authored by the alleged inventors of the patents in suit or their collaborators, including patents, applications, articles and code by Messrs. Dager, Tannenbaum and Decyk. These references and code, including those related to the Appleseed cluster, Pooch, MacMPI and related projects, are likewise directed to cluster computing using an existing message passing library and a cluster management tool. Moreover, these prior art publications teach peer-to-peer communications between cluster nodes, and use of these cluster computing tools with CAS kernels. Given the similarity of their teaching, it would have been obvious to combine one of more of these systems or publications with MultiMatlab.</p> <p>In addition, this claim is rendered obvious by Intel Paragon Maple in view of Cramer, Choy, FFTW, Howard, and/or Chu for the reasons stated in the Cover Pleading, section IV.</p>
I.j. wherein the third node comprises a third hardware processor with a plurality of processing cores,	<p>The MultiMatlab References disclose “wherein the third node comprises a third hardware processor with a plurality of processing cores.”</p> <p>MultiMatlab was configured to run on computers, including the computer comprising “the third node,” that included multiple processing cores:</p>

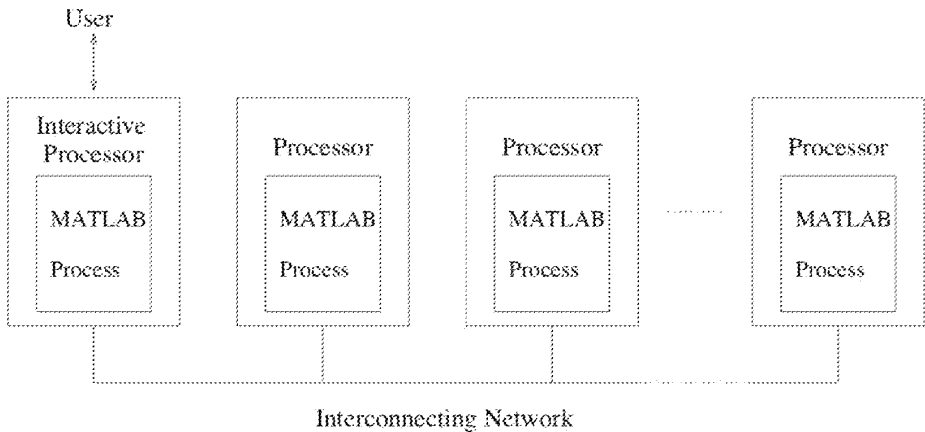
Asserted Claim	Prior Art Disclosure
	<div><pre>graph TD User[User] <--> IP[Interactive Processor] subgraph Nodes IP P1[Processor] P2[Processor] P3[Processor] end IP --- IN[Interconnecting Network] P1 --- IN P2 --- IN P3 --- IN subgraph Processes subgraph IP_Process [] direction TB IP_MATLAB[MATLAB Process] end subgraph P1_Process [] direction TB P1_MATLAB[MATLAB Process] end subgraph P2_Process [] direction TB P2_MATLAB[MATLAB Process] end subgraph P3_Process [] direction TB P3_MATLAB[MATLAB Process] end end IP --- IP_Process P1 --- P1_Process P2 --- P2_Process P3 --- P3_Process</pre></div> <p>Figure 1: MultiMATLAB Architecture</p> <p>Trefethen 1997 at 3.</p> <p>MultiMatlab could run on an IBM SP2 or Unix workstation network. Trefethen 1996 at Abstract. The SP2 “consists of up to several hundred nodes which are all RISC 6000 superscalar processors. (Superscalar means that more than one instruction per clock cycle can be performed.) The SP2 can be considered a distributed memory machine because each node has its own local memory.” https://public.wsu.edu/~ari/miscellaneous/sp2_guide.html (last visited August 9, 2021). “There can be 4 to 16 processors per frame (or cabinet) and 1 to 16 frames per system. Frames can consist of BOTH thin and wide nodes. The main difference between thin and wide nodes is that generally wide nodes have the capacity for larger memory and a larger memory cache. Standard SP2 systems could contain up to 256 nodes. Up to 512 nodes can be ordered from IBM by special request. (ORNL has a 16 node SP2, Maui has a 400 node SP2, and Cornell has a 512 node machine.) An optional high performance switch (HPS) for inter-processor communication is usually included on most systems. Such a switch can interconnect up to 16 nodes with up to 16 other nodes. This high performance switch has a 40 MB/sec peak bi-directional bandwidth. More than one HPS may be included when the number of nodes in the SP2 machine is greater than 32. The IBM SP2 is fairly robust in that it has concurrent maintenance, which simply means each processor node can be removed,</p>

Asserted Claim	Prior Art Disclosure
	<p>repaired, and rebooted without interrupting operations on other nodes.” <i>Id.</i></p> <p><u>Additional Evidence of Obviousness</u></p> <p>The diagram illustrates the architecture of MATLAB*P 2.0. A central dashed box represents 'Server #0'. Inside this box, at the top, are three boxes labeled 'PBLAS', 'FFTW', and 'SciLAPACK', with an ellipsis to their right. Arrows from these three boxes point to a 'Package Manager' box. Below the Package Manager is a 'Matrix Manager' box. To the left of the Matrix Manager are three boxes labeled 'Matrix 1', 'Matrix 2', and 'Matrix 3', with an ellipsis below them. Arrows from these three boxes point to the Matrix Manager. To the right of the Matrix Manager is a 'Server Manager' box. Arrows from the Package Manager, Matrix Manager, and Server Manager all point to the Server Manager. To the right of the Server Manager is a 'Client Manager' box. Arrows from the Client Manager and Server Manager point to a 'Client' box. Above the Client box is a 'Proxy' box, with an arrow pointing from the Client to the Proxy. Below the Server Manager box, outside the dashed box, are four boxes labeled 'Server #1', 'Server #2', 'Server #3', and 'Server #4', with an ellipsis to their right. Arrows from each of these four boxes point to the Server Manager box inside the dashed box.</p> <p>Fig. 1. Structure of MATLAB*P 2.0</p> <p>Choy 1 at 2.</p>

Asserted Claim	Prior Art Disclosure
	<p data-bbox="588 318 849 354"><i>A. Test Platform</i></p> <ul data-bbox="631 380 1539 773" style="list-style-type: none"> <li data-bbox="631 380 1539 456">• Beowulf cluster with 9 nodes (2 nodes are used in the tests). <li data-bbox="631 467 1539 544">• Dual-processor nodes, each with two 1.533GHz Athlon MP processors. <li data-bbox="631 555 1539 631">• 1GB DDR RAM on each node. No swapping occurred during benchmarks. <li data-bbox="631 643 1539 719">• Fast ethernet (100Mbps/sec) interconnect. Intel Etherfast 410T switch. <li data-bbox="631 730 957 773">• Linux 2.4.18-4smp <p data-bbox="588 812 720 841">Choy 1 at 4.</p>

Asserted Claim	Prior Art Disclosure
	<p data-bbox="588 305 825 337"><i>A. Test Platform</i></p> <ul style="list-style-type: none"> <li data-bbox="627 362 1144 394">• Beowulf cluster with 15 nodes. <li data-bbox="627 402 1410 475">• Dual-processor nodes, each with two Intel Xeon 2.4 GHz with Hyperthreading. <li data-bbox="627 483 1410 678">• All tests are run with one process per processor. Also, the minimum number of nodes are used. So if a test is using four processors, two nodes are used. Thus, in this case we are using two shared-memory pairs of processors. <li data-bbox="627 686 1410 760">• Two gigabytes of RAM per node. No swapping occurred during the tests. <li data-bbox="627 768 1410 841">• Fast Ethernet (100 Mb/s) interconnect. Intel Etherfast 410T switch. <p data-bbox="588 865 750 898">Choy 2 at 337.</p>

Asserted Claim	Prior Art Disclosure
	 <p>Fig. 2. Message Passing.</p> <p>Choy 2 at 333.</p> <p>In addition, during this timeframe, multi-core processors were prevalent in the marketplace, and it would have been a matter of mere design choice to use computers that contained hardware processors with multiple processing cores. The '768 patent itself admits that dual-core processors were conventional in the prior art, as it references the well-known Intel Core 2 Duo processors as a multi-core example. <i>See, e.g.</i>, 7:10-32. It would have been obvious to use Intel Core 2 Duo processors with MultiMatlab. In addition, as established for claim 39, it was known to use hardware processors with a plurality of processing cores comprising DSPs, FPGAs and other types of processors. <i>See</i> claim 39.</p>
I.k. wherein the third node is configured to receive the result of	The MultiMatlab References disclose “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 expression evaluation to the first node.”

Asserted Claim	Prior Art Disclosure
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 expression evaluation to the first node;	<p>MultiMatlab discloses implementing peer-to-peer communications between nodes using MPI message passing or through a distributed server process. When configured in a MultiMatlab cluster, each CAS kernel or parallel compute library receives calls from the root process and executes code evaluating mathematical expressions. Each node sends computation results to other nodes, including the second node communicating with the third node. The third node accesses the results provided by the second node and performs further mathematical evaluations. Eventually, the results are returned to the root process and user interface. This can be done in the MultiMATLAB mode, or in backend support mode.</p>  <p>Figure 1: MultiMATLAB Architecture</p> <p>Trefethen 1997 at 3.</p>

Asserted Claim	Prior Art Disclosure
	<p>Figure 4b illustrates a parallel MATLAB implementation of conjugate gradients that uses the SPMD message passing described in Section 5.2. In this case, the matrix A and the different vectors are each distributed by row over all processes as in Figure 6. In each iteration, a process only does the computation required for its local data. However, communication is required for two different operations: the dot-products needed to compute ρ and α and the matrix-vector multiply needed 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.</p> <p>Trefethen 1997 at 11, 13.</p> <p>The following code further illustrates the second MATLAB kernel receiving calls from the first kernel, evaluating an expression, and communicating results to other nodes, including a third node, which can then perform additional computation and return result to the root process for display to the user:</p>

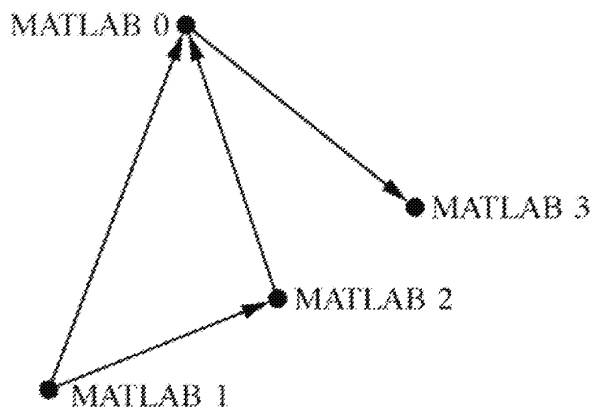
Asserted Claim	Prior Art Disclosure
	<pre> r_local = b_local; rho = Sum(r_local'*r_local); k = 0; while((k < kmax)) k = k+1; if (k == 1) p_local = r_local; else beta = rho/oldrho; p_local = r_local + beta*p_local; end p = Gather(p_local); w_local = A_local * p; alpha = rho/(Sum(p_local'*w_local)); x_local = x_local + alpha * p_local; r_local = r_local - alpha * w_local; oldrho = rho; rho = Sum(r_local'*r_local); end </pre> <p>b. Message Passing MATLAB</p> <p>Trefethen 1997 at 11.</p>

Page 85 of 254

Asserted Claim	Prior Art Disclosure
	<pre> if ID==0 % first process: send a = 1 Send(ID+1,a) elseif ID == Nproc-1 % last process: receive and double a = 2*Recv else % middle processes: receive, double, and send a = 2*Recv Send(ID+1,a) end; </pre> <p>Process 0 creates the variable <code>a</code> with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of <code>a</code>, and sends it along to process 2; and so on. If there are six processors the command <code>Eval('cycle')</code> produces the output</p> <pre> a = 1 a = 2 a = 4 a = 8 a = 16 a = 32 </pre> <p>The processes run asynchronously, but since each <code>Send</code> command is only executed after the corresponding <code>Recv</code> has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning.</p> <p>Trefethen 1996 at 7.</p> <p><u>Additional Evidence of Obviousness</u></p>

Asserted Claim	Prior Art Disclosure
	<p style="text-align: center;">Structure of the system (2)</p> <ul style="list-style-type: none"> • Client (C++) <ul style="list-style-type: none"> – Attaches to MATLAB through MEX interface – Relays commands to server and process returns • Server (C++/MPI) <ul style="list-style-type: none"> – Manipulates/processes data (matrices) – Performs the actual parallel computation by calling the appropriate library routines <p>Choy 3 at 36.</p>

Asserted Claim	Prior Art Disclosure
	<p>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.</p> <p>Because of its roots in serial numerical libraries, MATLAB has always been a serial program. However, as modern en-</p> <p>Choy 1 at 1.</p>

Asserted Claim	Prior Art Disclosure
	 <p>The diagram shows four nodes labeled MATLAB 0, MATLAB 1, MATLAB 2, and MATLAB 3. MATLAB 0 is at the top, MATLAB 1 is at the bottom left, MATLAB 2 is at the bottom right, and MATLAB 3 is to the right of MATLAB 2. Directed arrows indicate the flow of message passing: from MATLAB 1 to MATLAB 0, from MATLAB 0 to MATLAB 2, from MATLAB 2 to MATLAB 3, and from MATLAB 2 back to MATLAB 0.</p> <p>Fig. 2. Message Passing.</p> <p>Choy 2 at 333.</p>

Asserted Claim	Prior Art Disclosure
	<p><i>B. MultiMATLAB/MultiOctave Mode</i></p> <p>One of the goals of the project is to make the software to be useful to as wide an audience as possible. In order to achieve this, we found that it would be fruitful to combine other parallel MATLAB approaches into MATLAB * P, to provide a unified parallel MATLAB framework.</p> <p>In conjunction with P. Husbands, we developed a prototype implementation of a MultiMATLAB [35]-like, distributed MATLAB package in MATLAB * P, which we call the PPEngine. With this package and associated m-files, we can run multiple MATLAB processes on the back end and evaluate MATLAB functions in parallel on dense matrices.</p> <p>The system works by starting up MATLAB engine instances on each node through calls to the MATLAB engine interface. From that point on, MATLAB commands can be relayed to the MATLAB engine. The command is do this is <i>mm</i>.</p> <p>Choy 2 at 336</p>

Asserted Claim	Prior Art Disclosure
	<p>We present MATLAB *P [15] as an example of what could be a “right” parallel MATLAB. MATLAB * P is a parallel MATLAB using the back-end support approach, aimed at widespread circulation among a general audience. In order to achieve this, we took ideas from the approaches used by other software found in the survey. For example, the <i>embarrassingly parallel</i> approach allow simple, yet useful, division of work into multiple MATLAB sessions. The <i>message passing</i> approach, which is a superset of the embarrassingly parallel approach, allows finer control between the MATLAB sessions.</p> <p>The main idea in MATLAB *P is that data exist on the parallel server as distributed matrices. Any operations on a distributed matrix (which exists in the MATLAB front end only as a handle) will be relayed to the server transparently. The server calls the appropriate routine from a parallel numerical library (e.g. ScaLAPACK, FFTW, . . .) and the results stay on the server until explicitly requested.</p> <p>Choy 2 at 336.</p>

Asserted Claim	Prior Art Disclosure
	<p>Server software 105 may operate in any suitable way to receive commands from the client and perform operations in parallel, as the invention is not limited in this respect. FIG. 2 is but one example of a suitable configuration of server software 105 for at least some embodiments. In FIG. 2, parallel server 201 includes a plurality of processors, 203a, 203b, . . . , 203n, and a memory 213. Each of processors 203a-203n includes four software modules: client managers 205a, 205b, . . . , 205n; server managers 207a, 207b, . . . , 207n; library managers 209a, 209b, . . . , 209n; and data managers 211a, 211b, . . . , 211n. Memory 213 of parallel server 201 stores a plurality of software libraries 215a, 215b, 215c, . . . , 215n that include routines for performing various operations.</p> <p>Client managers 205 interface with clients, and provides routines for reading commands and arguments from a client and sending results and other data back to the client. Server managers 207 handle communications between server processes executing on different processors of parallel server 201. That is, server managers 207 manage the transmission of data between the processors of parallel server 201 and collect results and error codes from the processors. Library managers</p> <p>Husbands at 10:46-67.</p>

Asserted Claim	Prior Art Disclosure
	<p>The server itself is divided in four self-contained parts:</p> <ol style="list-style-type: none"><li data-bbox="599 386 972 418">1. Client Connection Manager <p>Client Connection Manager is responsible for communications with the client.</p> <ul style="list-style-type: none"><li data-bbox="599 500 1595 581">. It provides functions for reading commands and arguments from the client and sending the results back to the client. It is only used in the head server process. <ol style="list-style-type: none"><li data-bbox="599 621 972 654">2. Server Connection Manager <p>Server Connection Manager takes care of communication between server processes. It mainly controls broadcasting of commands and arguments from head</p>

Asserted Claim	Prior Art Disclosure
	<p>process to the slave processes, and collection of results and error codes. Also it provides rank and size information to the processes.</p> <p>3. Package Manager</p> <p>Package Manager is responsible for maintaining a list of available packages and functions provided by them. When initiated by the server process, Package Manager will also perform the actual call to the functions.</p> <p>4. Matrix Manager</p> <p>Matrix Manager contains all the functions needed to create, delete and change the matrices on the server processes. It maintains a mapping from client-side matrix identifiers to actual matrices on the server. It is also responsible for performing garbage collection.</p> <p>Choy Thesis at 26, 28</p> <p>The following additional example illustrates an FFT calculation on a Matlab *P cluster, resulting in communications of intermediate results between the nodes in the cluster, and the communication of final results from the cluster nodes to the root process for display to the user:</p>

Asserted Claim	Prior Art Disclosure
	<p style="text-align: center;">Example 3</p> <p><code>% FFT2 in 4 lines</code></p> <ul style="list-style-type: none"> • <code>A = randn(10000,10000*p);</code> • <code>B = mm('fft',A);</code> • <code>C = B.{'</code> • <code>D = mm('fft',C);</code> • <code>F = D.{'</code> <p>Choy 3 at 47.</p> <p>The following additional example further illustrates MPI message passing between Matlab*P cluster nodes:</p>

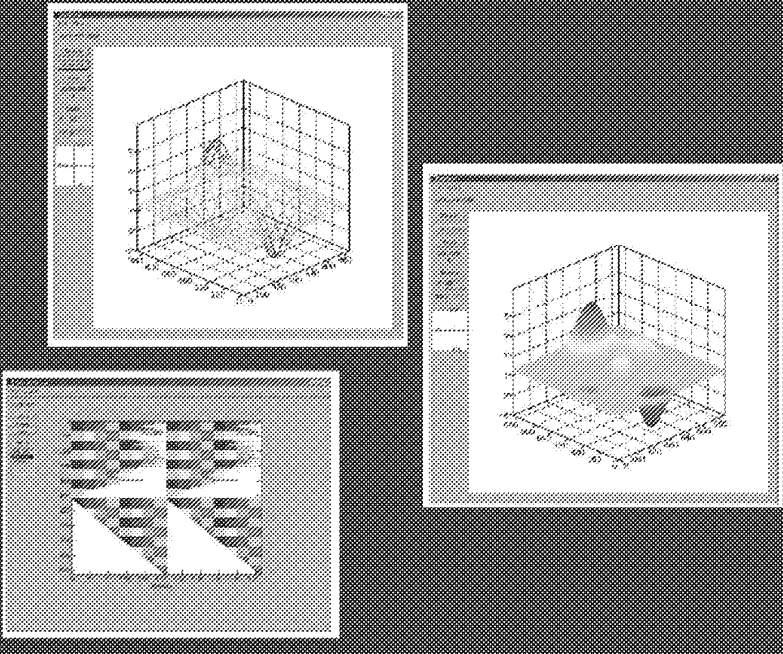
Asserted Claim	Prior Art Disclosure
	<pre data-bbox="631 316 1422 982">function result = simpletest(X) import mpi.*; MPI.init; if MPI.COMM_WORLD.rank == 0; data = [4 2]; MPI.COMM_WORLD.send(data, 2, 1, 13); result = 9; elseif MPI.COMM_WORLD.rank == 1 got = MPI.COMM_WORLD.recv(2, 0, 13); result = got(1) + got(2); end MPI.fnlz;</pre> <p data-bbox="582 1036 735 1063">Choy 3 at 49.</p> <p data-bbox="582 1101 1854 1352">To the extent ACS contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of MultiMatlab and the knowledge of a POSITA, it would have been obvious to combine it with one or more of the other references in these invalidity contentions. For example, MultiMatlab and Matlab *P publications reference MPI and PVM-based parallel processing systems as comparable cluster computing systems, motivating a POSITA to look to such systems for modifications relating to this claim element. <i>See, e.g.,</i> Choy 2 at 334. Accordingly, it would have been obvious to modify MultiMatlab in view of the teachings of PVM and MPI publications, such as those referenced in these contentions, as well as references and systems implementing aspects of PVM or MPI. As another example, because MultiMatlab is directed to cluster</p>

Asserted Claim	Prior Art Disclosure
	<p>computing, it would have been obvious to combine with one or more of the prior art references or systems authored by the alleged inventors of the patents in suit or their collaborators, including patents, applications, articles and code by Messrs. Dager, Tannenbaum and Decyk. These references and code, including those related to the Appleseed cluster, Pooch, MacMPI and related projects, are likewise directed to cluster computing using an existing message passing library and a cluster management tool. Moreover, these prior art publications teach peer-to-peer communications between cluster nodes, and use of these cluster computing tools with CAS kernels. Given the similarity of their teaching, it would have been obvious to combine one of more of these systems or publications with MultiMatlab.</p> <p>In addition, this claim is rendered obvious by Intel Paragon Maple in view of Cramer, Choy, FFTW, Howard, and/or Chu for the reasons stated in the Cover Pleading, section IV.</p>
I.I. wherein the first node is configured to return the result of the second mathematical expression evaluation to the user interface;	<p>The MultiMatlab References disclose “wherein the first node is configured to return the result of the second mathematical expression evaluation to the user interface.”</p> <p>MultiMatlab root process, e.g., a MATLAB CAS kernel or a client process on the root node, returns the result of the second mathematical evaluation to the user interface, where it can be displayed in a numeric, text, or graphical format:</p>

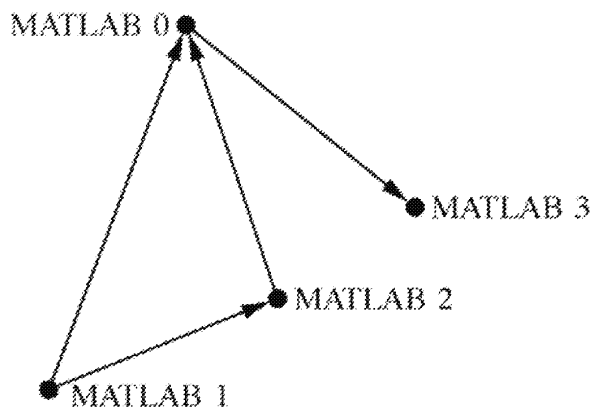
Asserted Claim	Prior Art Disclosure
	<div><pre>graph TD User[User] <--> IP[Interactive Processor] subgraph IP_Box [Interactive Processor] MP1[MATLAB Process] end subgraph P1_Box [Processor] MP2[MATLAB Process] end subgraph P2_Box [Processor] MP3[MATLAB Process] end subgraph P3_Box [Processor] MP4[MATLAB Process] end IP_Box --- IN[Interconnecting Network] P1_Box --- IN P2_Box --- IN P3_Box --- IN</pre></div> <p>Figure 1: MultiMATLAB Architecture</p> <p>Trefethen 1997 at 3.</p>

Asserted Claim	Prior Art Disclosure
	<pre> if ID==0 % first process: send a = 1 Send(ID+1,a) elseif ID == Nproc-1 % last process: receive and double a = 2*Recv else % middle processes: receive, double, and send a = 2*Recv Send(ID+1,a) end; </pre> <p>Process 0 creates the variable <code>a</code> with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of <code>a</code>, and sends it along to process 2; and so on. If there are six processors the command <code>Eval('cycle')</code> produces the output</p> <pre> a = 1 a = 2 a = 4 a = 8 a = 16 a = 32 </pre> <p>The processes run asynchronously, but since each <code>Send</code> command is only executed after the corresponding <code>Recv</code> has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning.</p> <p>Trefethen 1996 at 7.</p> <p><u>Additional Evidence of Obviousness</u></p>

Asserted Claim	Prior Art Disclosure
	<div data-bbox="711 315 1511 386"><h2>Structure of the system (2)</h2></div> <ul data-bbox="614 467 1606 917" style="list-style-type: none">• Client (C++)<ul style="list-style-type: none">– Attaches to MATLAB through MEX interface– Relays commands to server and process returns• Server (C++/MPI)<ul style="list-style-type: none">– Manipulates/processes data (matrices)– Performs the actual parallel computation by calling the appropriate library routines <p data-bbox="582 958 735 987">Choy 3 at 36.</p>

Asserted Claim	Prior Art Disclosure
	 <p data-bbox="582 966 733 998">Choy 3 at 53.</p>

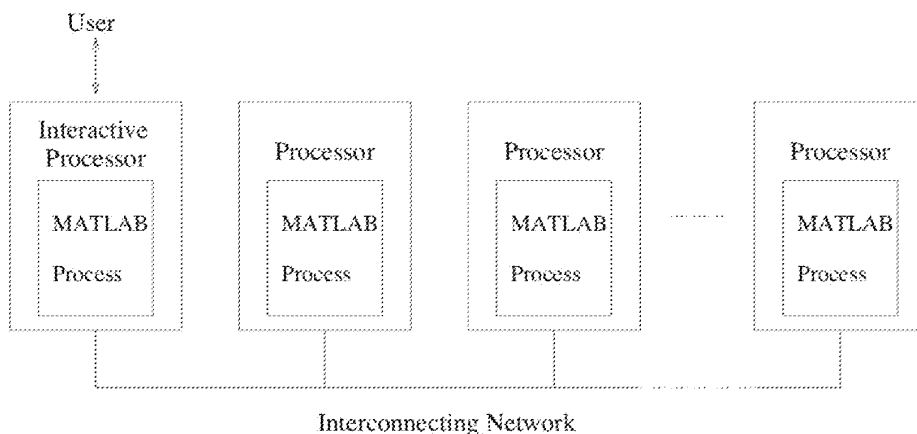
Asserted Claim	Prior Art Disclosure
	<p>MATLAB [?] is one of the most widely used tools in scientific and technical computing. It started in 1970s as an interactive interface to EISPACK [?] and LINPACK [?], a set of eigenvalue and linear system solution routines. It has since grown to a feature rich 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.</p> <p>Because of its roots in serial numerical libraries, MATLAB has always been a serial program. However, as modern en-</p> <p>Choy 1 at 1.</p>

Asserted Claim	Prior Art Disclosure
	 <p>The diagram shows four nodes labeled MATLAB 0, MATLAB 1, MATLAB 2, and MATLAB 3. MATLAB 0 is at the top, MATLAB 1 is at the bottom left, MATLAB 2 is at the bottom right, and MATLAB 3 is to the right of MATLAB 2. Directed arrows indicate the flow of message passing: from MATLAB 1 to MATLAB 0, from MATLAB 0 to MATLAB 2, from MATLAB 2 to MATLAB 1, and from MATLAB 0 to MATLAB 3.</p> <p>Fig. 2. Message Passing.</p> <p>Choy 2 at 333.</p>

Asserted Claim	Prior Art Disclosure
	<div><p>The diagram illustrates the architecture of MATLAB*P 2.0. A central dashed box represents 'Server #0'. Inside this box, at the top, are three boxes labeled 'PBLAS', 'FFTW', and 'ScaLAPACK', each with a line pointing to a 'Package Manager' box below them. To the right of the 'Package Manager' is a 'Client Manager' box. Below the 'Package Manager' is a 'Matrix Manager' box, which has three boxes labeled 'Matrix 1', 'Matrix 2', and 'Matrix 3' pointing to it. To the right of the 'Matrix Manager' is a 'Server Manager' box. Below the 'Server Manager' are four boxes labeled 'Server #1', 'Server #2', 'Server #3', and 'Server #4'. Lines connect the 'Server Manager' box to each of these four server boxes. Outside the 'Server #0' box, there is a 'Proxy' box and a 'Client' box. The 'Proxy' box has a line pointing to the 'Client Manager' box, and the 'Client' box has a line pointing to the 'Server Manager' box.</p></div> <p>Fig. 1. Structure of MATLAB*P 2.0</p> <p>Choy 1 at 2.</p> <p>To the extent ACS contends that this claim element is not expressly or inherently disclosed or is not obvious in view of MultiMatlab and the knowledge of a POSITA, it would have been obvious to combine it with one or more of the other references in these invalidity contentions. For example, MultiMatlab and Matlab *P publications reference MPI and PVM-based parallel processing systems as comparable cluster computing systems, motivating a POSITA to look to such systems for modifications relating to this claim element. <i>See, e.g.,</i> Choy 2 at 334. Accordingly, it would have been obvious to modify MultiMatlab in view of the teachings of PVM and MPI publications, such as those referenced in these contentions, as well as references and systems</p>

ACS_0065315

Asserted Claim	Prior Art Disclosure
	<p>implementing aspects of PVM or MPI. As another example, because MultiMatlab is directed to cluster computing, it would have been obvious to combine with one or more of the prior art references or systems authored by the alleged inventors of the patents in suit or their collaborators, including patents, applications, articles and code by Messrs. Dager, Tannenbaum and Decyk. These references and code, including those related to the Appleseed cluster, Pooch, MacMPI and related projects, are likewise directed to cluster computing using an existing message passing library and a cluster management tool. Moreover, these prior art publications teach peer-to-peer communications between cluster nodes, and use of these cluster computing tools with CAS kernels. Given the similarity of their teaching, it would have been obvious to combine one of more of these systems or publications with MultiMatlab.</p> <p>In addition, this claim is rendered obvious by Intel Paragon Maple in view of Cramer, Choy, FFTW, Howard, and/or Chu for the reasons stated in the Cover Pleading, section IV.</p>
I.m. wherein one or more of the nodes are configured to:	The MultiMatlab References disclose “wherein one or more of the nodes are configured to,” for the reasons below:
I.n. accept user instructions;	<p>The MultiMatlab References disclose “accept user instructions.”</p> <p>Each cluster node accepts instructions from the user interface, which may be provided by the root MATLAB kernel, or root client / server process. For example the node may receive MPI and/or MATLAB and other commands from the user interface and send these commands to the local and remote nodes using the peer-to-peer messaging mechanism. The remote cluster nodes accept these instructions upon receipt of the messages containing the instructions and may send them to other nodes:</p>

Asserted Claim	Prior Art Disclosure
	<p>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.</p> <p>Trefethen 1997 at 3.</p>  <pre> graph TD User[User] <--> IP[Interactive Processor] subgraph Processors IP P1[Processor] P2[Processor] P3[Processor] end IP --- P1 P1 --- P2 P2 --- P3 P3 --- IN[Interconnecting Network] IN --- IP subgraph MATLAB_Processes IP --- MP1[MATLAB Process] P1 --- MP2[MATLAB Process] P2 --- MP3[MATLAB Process] P3 --- MP4[MATLAB Process] end </pre> <p>Figure 1: MultiMATLAB Architecture</p> <p>Trefethen 1997 at 3.</p>

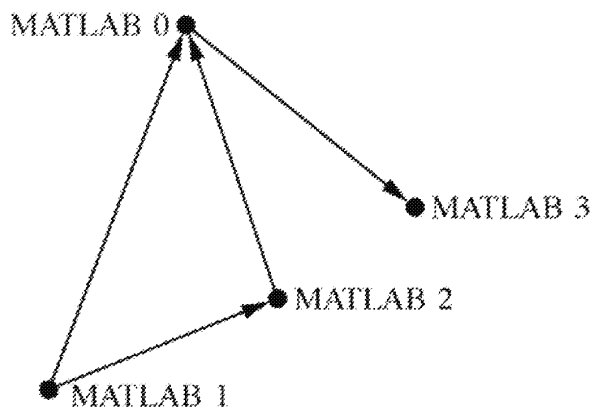
Asserted Claim	Prior Art Disclosure
	<p>MATLAB is then started on five additional processors taken from a predetermined list. Or perhaps the second author is sitting at a machine connected to Cornell's Computer Science Department network. He types</p> <pre>Start(['gemini'; 'orion'; 'rigel'; 'castor'; 'pollux'])</pre> <p>Now MATLAB is started on the five processors with the names indicated. (Some names could be repeated, in which case multiple MATLAB processes would be started on a single processor.) In either case, when all the processes are started the message is returned,</p> <pre>6 MultiMATLAB processes running.</pre> <p>This total number of processors can subsequently be accessed by the MultiMATLAB command <code>Nproc</code>.</p> <p>Trefethen 1996 at 4.</p> <p>Figure 4b illustrates a parallel MATLAB implementation of conjugate gradients that uses the SPMD message passing described in Section 5.2. In this case, the matrix A and the different vectors are each distributed by row over all processes as in Figure 6. In each iteration, a process only does the computation required for its local data. However, communication is required for two different operations: the dot-products needed to compute ρ and α and the matrix-vector multiply needed 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.</p> <p>Trefethen 1997 at 11, 13.</p> <p>The following code further illustrates the root node accepting user instructions, and passing user instructions to root and remote kernels and/or computational libraries for processing:</p>

Asserted Claim	Prior Art Disclosure
	<pre> r_local = b_local; rho = Sum(r_local'*r_local); k = 0; while((k < kmax)) k = k+1; if (k == 1) p_local = r_local; else beta = rho/oldrho; p_local = r_local + beta*p_local; end p = Gather(p_local); w_local = A_local * p; alpha = rho/(Sum(p_local'*w_local)); x_local = x_local + alpha * p_local; r_local = r_local - alpha * w_local; oldrho = rho; rho = Sum(r_local'*r_local); end </pre> <p>b. Message Passing MATLAB</p> <p>Trefethen 1997 at 11.</p>

Asserted Claim	Prior Art Disclosure
	<div><div><div><div><div><div>Process 0</div><div>Process 1</div><div>Process 2</div><div></div><div>Process P</div></div></div><div><div>A_local</div><div></div></div></div></div><div><div><div>0</div><div>1</div><div>2</div><div></div><div>P</div></div><div><div>x_local</div><div></div></div></div></div> <div><div>A</div><div>x</div></div>

Asserted Claim	Prior Art Disclosure
	<pre> if ID==0 % first process: send a = 1 Send(ID+1,a) elseif ID == Nproc-1 % last process: receive and double a = 2*Recv else % middle processes: receive, double, and send a = 2*Recv Send(ID+1,a) end; </pre> <p>Process 0 creates the variable <code>a</code> with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of <code>a</code>, and sends it along to process 2; and so on. If there are six processors the command <code>Eval('cycle')</code> produces the output</p> <pre> a = 1 a = 2 a = 4 a = 8 a = 16 a = 32 </pre> <p>The processes run asynchronously, but since each <code>Send</code> command is only executed after the corresponding <code>Recv</code> has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning.</p> <p>Trefethen 1996 at 7.</p> <p><u>Additional Evidence of Obviousness</u></p>

Asserted Claim	Prior Art Disclosure
	<div data-bbox="711 316 1509 389"><h2>Structure of the system (2)</h2></div> <ul style="list-style-type: none">• Client (C++)<ul style="list-style-type: none">– Attaches to MATLAB through MEX interface– Relays commands to server and process returns• Server (C++/MPI)<ul style="list-style-type: none">– Manipulates/processes data (matrices)– Performs the actual parallel computation by calling the appropriate library routines <p data-bbox="582 958 733 990">Choy 3 at 36.</p>

Asserted Claim	Prior Art Disclosure
	 <p data-bbox="625 795 1013 836">Fig. 2. Message Passing.</p> <p data-bbox="582 893 754 925">Choy 2 at 333.</p>

Asserted Claim	Prior Art Disclosure
	<p><i>B. MultiMATLAB/MultiOctave Mode</i></p> <p>One of the goals of the project is to make the software to be useful to as wide an audience as possible. In order to achieve this, we found that it would be fruitful to combine other parallel MATLAB approaches into MATLAB * P, to provide a unified parallel MATLAB framework.</p> <p>In conjunction with P. Husbands, we developed a prototype implementation of a MultiMATLAB [35]-like, distributed MATLAB package in MATLAB * P, which we call the PPEngine. With this package and associated m-files, we can run multiple MATLAB processes on the back end and evaluate MATLAB functions in parallel on dense matrices.</p> <p>The system works by starting up MATLAB engine instances on each node through calls to the MATLAB engine interface. From that point on, MATLAB commands can be relayed to the MATLAB engine. The command is do this is <i>mm</i>.</p> <p>Choy 2 at 336</p>

Asserted Claim	Prior Art Disclosure
	<p>We present MATLAB *P [15] as an example of what could be a “right” parallel MATLAB. MATLAB * P is a parallel MATLAB using the back-end support approach, aimed at widespread circulation among a general audience. In order to achieve this, we took ideas from the approaches used by other software found in the survey. For example, the <i>embarrassingly parallel</i> approach allow simple, yet useful, division of work into multiple MATLAB sessions. The <i>message passing</i> approach, which is a superset of the embarrassingly parallel approach, allows finer control between the MATLAB sessions.</p> <p>The main idea in MATLAB *P is that data exist on the parallel server as distributed matrices. Any operations on a distributed matrix (which exists in the MATLAB front end only as a handle) will be relayed to the server transparently. The server calls the appropriate routine from a parallel numerical library (e.g. ScaLAPACK, FFTW, . . .) and the results stay on the server until explicitly requested.</p> <p>Choy 2 at 336.</p>

Asserted Claim	Prior Art Disclosure
	<p>Server software 105 may operate in any suitable way to receive commands from the client and perform operations in parallel, as the invention is not limited in this respect. FIG. 2 is but one example of a suitable configuration of server software 105 for at least some embodiments. In FIG. 2, parallel server 201 includes a plurality of processors, 203a, 203b, . . . , 203n, and a memory 213. Each of processors 203a-203n includes four software modules: client managers 205a, 205b, . . . , 205n; server managers 207a, 207b, . . . , 207n; library managers 209a, 209b, . . . , 209n; and data managers 211a, 211b, . . . , 211n. Memory 213 of parallel server 201 stores a plurality of software libraries 215a, 215b, 215c, . . . , 215n that include routines for performing various operations.</p> <p>Client managers 205 interface with clients, and provides routines for reading commands and arguments from a client and sending results and other data back to the client. Server managers 207 handle communications between server processes executing on different processors of parallel server 201. That is, server managers 207 manage the transmission of data between the processors of parallel server 201 and collect results and error codes from the processors. Library managers</p> <p>Husbands at 10:46-67.</p>

Asserted Claim	Prior Art Disclosure
	<p>The server itself is divided in four self-contained parts:</p> <ol style="list-style-type: none"><li data-bbox="599 386 972 418">1. Client Connection Manager <p data-bbox="633 448 1595 480">Client Connection Manager is responsible for communications with the client.</p> <p data-bbox="633 500 1595 581">It provides functions for reading commands and arguments from the client and sending the results back to the client. It is only used in the head server process.</p> <ol style="list-style-type: none"><li data-bbox="599 621 972 654">2. Server Connection Manager <p data-bbox="633 683 1595 764">Server Connection Manager takes care of communication between server processes. It mainly controls broadcasting of commands and arguments from head</p>

Asserted Claim	Prior Art Disclosure
	<p>process to the slave processes, and collection of results and error codes. Also it provides rank and size information to the processes.</p> <p>3. Package Manager</p> <p>Package Manager is responsible for maintaining a list of available packages and functions provided by them. When initiated by the server process, Package Manager will also perform the actual call to the functions.</p> <p>4. Matrix Manager</p> <p>Matrix Manager contains all the functions needed to create, delete and change the matrices on the server processes. It maintains a mapping from client-side matrix identifiers to actual matrices on the server. It is also responsible for performing garbage collection.</p> <p>Choy Thesis at 26, 28.</p> <p>The following additional example illustrates an FFT calculation on a Matlab *P cluster, including the acceptance of user instructions, communication of instructions, computation and return of results:</p>

Asserted Claim	Prior Art Disclosure
	<p style="text-align: center;">Example 3</p> <p><code>% FFT2 in 4 lines</code></p> <ul style="list-style-type: none"> • <code>A = randn(10000,10000*p);</code> • <code>B = mm('fft',A);</code> • <code>C = B.{'</code> • <code>D = mm('fft',C);</code> • <code>F = D.{'</code> <p>Choy 3 at 47.</p> <p>The following additional example further illustrates MPI message passing between Matlab*P cluster nodes:</p>

Asserted Claim	Prior Art Disclosure
	<pre data-bbox="631 321 1425 987"> function result = simpletest(X) import mpi.*; MPI.init; if MPI.COMM_WORLD.rank == 0; data = [4 2]; MPI.COMM_WORLD.send(data, 2, 1, 13); result = 9; elseif MPI.COMM_WORLD.rank == 1 got = MPI.COMM_WORLD.recv(2, 0, 13); result = got(1) + got(2); end MPI.fnlz; </pre> <p data-bbox="584 1040 735 1065">Choy 3 at 49.</p> <p data-bbox="584 1105 1858 1354">To the extent ACS contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of MultiMatlab and the knowledge of a POSITA, it would have been obvious to combine it with one or more of the other references in these invalidity contentions. For example, MultiMatlab and Matlab *P publications reference MPI and PVM-based parallel processing systems as comparable cluster computing systems, motivating a POSITA to look to such systems for modifications relating to this claim element. <i>See, e.g.,</i> Choy 2 at 334. Accordingly, it would have been obvious to modify MultiMatlab in view of the teachings of PVM and MPI publications, such as those referenced in these contentions, as well as references and systems implementing aspects of PVM or MPI. As another example, because MultiMatlab is directed to cluster</p>

Asserted Claim	Prior Art Disclosure
	<p>computing, it would have been obvious to combine with one or more of the prior art references or systems authored by the alleged inventors of the patents in suit or their collaborators, including patents, applications, articles and code by Messrs. Dager, Tannenbaum and Decyk. These references and code, including those related to the Appleseed cluster, Pooch, MacMPI and related projects, are likewise directed to cluster computing using an existing message passing library and a cluster management tool. Moreover, these prior art publications teach peer-to-peer communications between cluster nodes, and use of these cluster computing tools with CAS kernels. Given the similarity of their teaching, it would have been obvious to combine one of more of these systems or publications with MultiMatlab.</p> <p>In addition, this claim is rendered obvious by Intel Paragon Maple in view of Cramer, Choy, FFTW, Howard, and/or Chu for the reasons stated in the Cover Pleading, section IV.</p>
<p>I.o. after accepting user instructions, communicate at least some of the user instructions using the mechanism for the nodes to communicate with each other; and</p>	<p>The MultiMatlab References disclose “after accepting user instructions, communicate at least some of the user instructions using the mechanism for the nodes to communicate with each other.”</p> <p>Each cluster node accepts instructions from the user interface, which may be provided by the root MATLAB kernel, or root client / server process. For example the node may receive MPI and/or MATLAB and other commands from the user interface and send these commands to the local and remote nodes using the peer-to-peer messaging mechanism. The remote cluster nodes accept these instructions upon receipt of the messages containing the instructions and may communicate them to other nodes after accepting them from the root node:</p> <p>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.</p> <p>Trefethen 1997 at 3.</p>

Asserted Claim	Prior Art Disclosure
	<p>2.2. Put, Get</p> <p>So far, we have not communicated between processes except to send screen output to the master process. Of course, a nontrivial MultiMATLAB system depends on such communication.</p> <p>One form of communication we have implemented is puts and gets, executable solely by the master MATLAB process. For example, the command</p> <pre>Put (1:4, 'A'),</pre> <p>sends the matrix A from the master process 0 to processes 1 through 4; an optional argument permits the name of A to be changed at the destination. The command</p> <pre>Get (3, 'B'),</pre> <p>gets the matrix B back from process 3 to the master.</p> <p>Trefethen 1996 at 6.</p> <p>Matlab *P implemented MPI message passing that allowed any node to pass computation instructions or results to any other node, as further illustrated by the following code:</p>

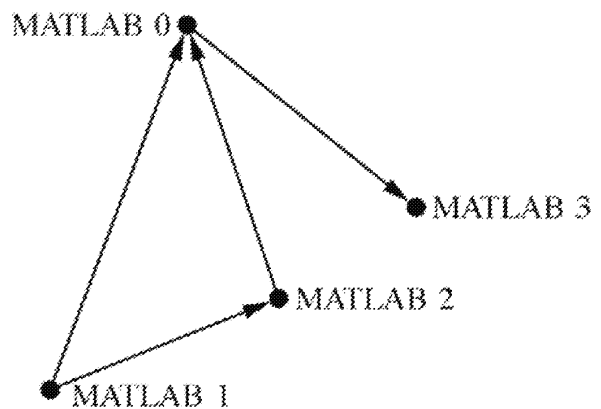
Asserted Claim	Prior Art Disclosure
	<pre> if ID==0 % first process: send a = 1 Send(ID+1,a) elseif ID == Nproc-1 % last process: receive and double a = 2*Recv else % middle processes: receive, double, and send a = 2*Recv Send(ID+1,a) end; </pre> <p>Process 0 creates the variable <i>a</i> with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of <i>a</i>, and sends it along to process 2; and so on. If there are six processors the command <code>Eval('cycle')</code> produces the output</p> <pre> a = 1 a = 2 a = 4 a = 8 a = 16 a = 32 </pre> <p>The processes run asynchronously, but since each <code>Send</code> command is only executed after the corresponding <code>Recv</code> has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning.</p> <p>Trefethen 1996 at 7.</p>

Asserted Claim	Prior Art Disclosure
	<div><pre>graph TD User[User] <--> IP[Interactive Processor] subgraph IP_Box [] direction TB IP_Label[Interactive Processor] MP1[MATLAB Process] end subgraph P1_Box [] direction TB P1_Label[Processor] MP2[MATLAB Process] end subgraph P2_Box [] direction TB P2_Label[Processor] MP3[MATLAB Process] end subgraph P3_Box [] direction TB P3_Label[Processor] MP4[MATLAB Process] end IP_Box --- P1_Box P1_Box --- P2_Box P2_Box --- P3_Box P3_Box --- IN[Interconnecting Network] IN --- IP_Box</pre></div> <p>Figure 1: MultiMATLAB Architecture</p> <p>Trefethen 1997 at 3.</p> <p>Figure 4b illustrates a parallel MATLAB implementation of conjugate gradients that uses the SPMD message passing described in Section 5.2. In this case, the matrix A and the different vectors are each distributed by row over all processes as in Figure 6. In each iteration, a process only does the computation required for its local data. However, communication is required for two different operations: the dot-products needed to compute ρ and α and the matrix-vector multiply needed 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.</p> <p>Trefethen 1997 at 11, 13.</p> <p>The following code further illustrates the root node accepting user instructions, and passing user instructions to root and remote kernels and/or computational libraries for processing:</p>

Asserted Claim	Prior Art Disclosure
	<pre> r_local = b_local; rho = Sum(r_local'*r_local); k = 0; while((k < kmax)) k = k+1; if (k == 1) p_local = r_local; else beta = rho/oldrho; p_local = r_local + beta*p_local; end p = Gather(p_local); w_local = A_local * p; alpha = rho/(Sum(p_local'*w_local)); x_local = x_local + alpha * p_local; r_local = r_local - alpha * w_local; oldrho = rho; rho = Sum(r_local'*r_local); end </pre> <p>b. Message Passing MATLAB</p> <p>Trefethen 1997 at 11.</p>

Additional Evidence of Obviousness

Asserted Claim	Prior Art Disclosure
	<div data-bbox="711 315 1511 386"><h2>Structure of the system (2)</h2></div> <div data-bbox="612 464 1608 919"><ul style="list-style-type: none">• Client (C++)<ul style="list-style-type: none">– Attaches to MATLAB through MEX interface– Relays commands to server and process returns• Server (C++/MPI)<ul style="list-style-type: none">– Manipulates/processes data (matrices)– Performs the actual parallel computation by calling the appropriate library routines</div> <p data-bbox="582 956 735 987">Choy 3 at 36.</p>

Asserted Claim	Prior Art Disclosure
	 <p>The diagram shows four nodes labeled MATLAB 0, MATLAB 1, MATLAB 2, and MATLAB 3. MATLAB 0 is at the top, MATLAB 1 is at the bottom left, MATLAB 2 is at the bottom right, and MATLAB 3 is to the right of MATLAB 2. Arrows indicate message passing: MATLAB 1 points to MATLAB 0, MATLAB 0 points to MATLAB 2, MATLAB 2 points to MATLAB 3, and MATLAB 3 points back to MATLAB 0.</p> <p>Fig. 2. Message Passing.</p> <p>Choy 2 at 333.</p>

Asserted Claim	Prior Art Disclosure
	<p><i>B. MultiMATLAB/MultiOctave Mode</i></p> <p>One of the goals of the project is to make the software to be useful to as wide an audience as possible. In order to achieve this, we found that it would be fruitful to combine other parallel MATLAB approaches into MATLAB * P, to provide a unified parallel MATLAB framework.</p> <p>In conjunction with P. Husbands, we developed a prototype implementation of a MultiMATLAB [35]-like, distributed MATLAB package in MATLAB * P, which we call the PPEngine. With this package and associated m-files, we can run multiple MATLAB processes on the back end and evaluate MATLAB functions in parallel on dense matrices.</p> <p>The system works by starting up MATLAB engine instances on each node through calls to the MATLAB engine interface. From that point on, MATLAB commands can be relayed to the MATLAB engine. The command is do this is <i>mm</i>.</p> <p>Choy 2 at 336</p>

Asserted Claim	Prior Art Disclosure
	<p>We present MATLAB *P [15] as an example of what could be a “right” parallel MATLAB. MATLAB * P is a parallel MATLAB using the back-end support approach, aimed at widespread circulation among a general audience. In order to achieve this, we took ideas from the approaches used by other software found in the survey. For example, the <i>embarrassingly parallel</i> approach allow simple, yet useful, division of work into multiple MATLAB sessions. The <i>message passing</i> approach, which is a superset of the embarrassingly parallel approach, allows finer control between the MATLAB sessions.</p> <p>The main idea in MATLAB *P is that data exist on the parallel server as distributed matrices. Any operations on a distributed matrix (which exists in the MATLAB front end only as a handle) will be relayed to the server transparently. The server calls the appropriate routine from a parallel numerical library (e.g. ScaLAPACK, FFTW, . . .) and the results stay on the server until explicitly requested.</p> <p>Choy 2 at 336.</p>

Asserted Claim	Prior Art Disclosure
	<p>Server software 105 may operate in any suitable way to receive commands from the client and perform operations in parallel, as the invention is not limited in this respect. FIG. 2 is but one example of a suitable configuration of server software 105 for at least some embodiments. In FIG. 2, parallel server 201 includes a plurality of processors, 203a, 203b, . . . , 203n, and a memory 213. Each of processors 203a-203n includes four software modules: client managers 205a, 205b, . . . , 205n; server managers 207a, 207b, . . . , 207n; library managers 209a, 209b, . . . , 209n; and data managers 211a, 211b, . . . , 211n. Memory 213 of parallel server 201 stores a plurality of software libraries 215a, 215b, 215c, . . . , 215n that include routines for performing various operations.</p> <p>Client managers 205 interface with clients, and provides routines for reading commands and arguments from a client and sending results and other data back to the client. Server managers 207 handle communications between server processes executing on different processors of parallel server 201. That is, server managers 207 manage the transmission of data between the processors of parallel server 201 and collect results and error codes from the processors. Library managers</p> <p>Husbands at 10:46-67.</p>

Asserted Claim	Prior Art Disclosure
	<p>The server itself is divided in four self-contained parts:</p> <ol style="list-style-type: none"><li data-bbox="599 386 972 418">1. Client Connection Manager <p data-bbox="633 448 1595 480">Client Connection Manager is responsible for communications with the client.</p> <p data-bbox="633 500 1595 581">It provides functions for reading commands and arguments from the client and sending the results back to the client. It is only used in the head server process.</p> <ol style="list-style-type: none"><li data-bbox="599 621 972 654">2. Server Connection Manager <p data-bbox="633 683 1595 764">Server Connection Manager takes care of communication between server processes. It mainly controls broadcasting of commands and arguments from head</p>

Asserted Claim	Prior Art Disclosure
	<p>process to the slave processes, and collection of results and error codes. Also it provides rank and size information to the processes.</p> <p>3. Package Manager</p> <p>Package Manager is responsible for maintaining a list of available packages and functions provided by them. When initiated by the server process, Package Manager will also perform the actual call to the functions.</p> <p>4. Matrix Manager</p> <p>Matrix Manager contains all the functions needed to create, delete and change the matrices on the server processes. It maintains a mapping from client-side matrix identifiers to actual matrices on the server. It is also responsible for performing garbage collection.</p> <p>Choy Thesis at 26, 28</p> <p>The following additional example illustrates an FFT calculation on a Matlab *P cluster, including the acceptance of user instructions, communication of instructions, computation and return of results:</p>

Asserted Claim	Prior Art Disclosure
	<div data-bbox="991 329 1347 410" data-label="Section-Header"> <h2>Example 3</h2> </div> <div data-bbox="636 485 1062 540" data-label="Text"> <p>% FFT2 in 4 lines</p> </div> <div data-bbox="633 565 1338 943" data-label="List-Group"> <ul style="list-style-type: none"> • <code>A = randn(10000,10000*p);</code> • <code>B = mm('fft',A);</code> • <code>C = B.';</code> • <code>D = mm('fft',C);</code> • <code>F = D.';</code> </div> <div data-bbox="571 979 741 1019" data-label="Text"> <p>Choy 3 at 47.</p> </div> <div data-bbox="571 1044 1813 1083" data-label="Text"> <p>The following additional example further illustrates MPI message passing between Matlab*P cluster nodes:</p> </div>

Asserted Claim	Prior Art Disclosure
	<pre data-bbox="631 321 1425 987">function result = simpletest(X) import mpi.*; MPI.init; if MPI.COMM_WORLD.rank == 0; data = [4 2]; MPI.COMM_WORLD.send(data, 2, 1, 13); result = 9; elseif MPI.COMM_WORLD.rank == 1 got = MPI.COMM_WORLD.recv(2, 0, 13); result = got(1) + got(2); end MPI.fnlz;</pre> <p data-bbox="584 1040 735 1065">Choy 3 at 49.</p> <p data-bbox="584 1105 1858 1354">To the extent ACS contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of MultiMatlab and the knowledge of a POSITA, it would have been obvious to combine it with one or more of the other references in these invalidity contentions. For example, MultiMatlab and Matlab *P publications reference MPI and PVM-based parallel processing systems as comparable cluster computing systems, motivating a POSITA to look to such systems for modifications relating to this claim element. <i>See, e.g.,</i> Choy 2 at 334. Accordingly, it would have been obvious to modify MultiMatlab in view of the teachings of PVM and MPI publications, such as those referenced in these contentions, as well as references and systems implementing aspects of PVM or MPI. As another example, because MultiMatlab is directed to cluster</p>

Asserted Claim	Prior Art Disclosure
	<p>computing, it would have been obvious to combine with one or more of the prior art references or systems authored by the alleged inventors of the patents in suit or their collaborators, including patents, applications, articles and code by Messrs. Dager, Tannenbaum and Decyk. These references and code, including those related to the Appleseed cluster, Pooch, MacMPI and related projects, are likewise directed to cluster computing using an existing message passing library and a cluster management tool. Moreover, these prior art publications teach peer-to-peer communications between cluster nodes, and use of these cluster computing tools with CAS kernels. Given the similarity of their teaching, it would have been obvious to combine one of more of these systems or publications with MultiMatlab.</p> <p>In addition, this claim is rendered obvious by Intel Paragon Maple in view of Cramer, Choy, FFTW, Howard, and/or Chu for the reasons stated in the Cover Pleading, section IV.</p>
<p>I.p. after communicating at least some of the user instructions using the mechanism, communicate at least some of the user instructions to one or more single-node kernels.</p>	<p>The MultiMatlab References disclose “after communicating at least some of the user instructions using the mechanism, communicate at least some of the user instructions to one or more single-node kernels.”</p> <p>Each cluster node accepts instructions from the user interface, which may be provided by the root MATLAB kernel, or root client / server process. For example the node may receive MPI and/or MATLAB and other commands from the user interface and send these commands to the local and remote nodes using the peer-to-peer messaging mechanism. The remote cluster nodes accept these instructions upon receipt of the messages containing the instructions and may send them to other nodes. The node may accept additional instructions, for example instructions for a sequence of computations, and communicate these instructions to the local or remote CAS kernels and/or compute libraries after previous instructions were already communicated:</p> <p>MultiMatlab implemented MPI message passing that allowed any node to pass computation instructions or results to any other node, including passing a sequence of such messages to different nodes and/or kernels, as illustrated by the following code:</p>

Asserted Claim	Prior Art Disclosure
	<pre> if ID==0 % first process: send a = 1 Send(ID+1,a) elseif ID == Nproc-1 % last process: receive and double a = 2*Recv else % middle processes: receive, double, and send a = 2*Recv Send(ID+1,a) end; </pre> <p>Process 0 creates the variable <code>a</code> with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of <code>a</code>, and sends it along to process 2; and so on. If there are six processors the command <code>Eval('cycle')</code> produces the output</p> <pre> a = 1 a = 2 a = 4 a = 8 a = 16 a = 32 </pre> <p>The processes run asynchronously, but since each <code>Send</code> command is only executed after the corresponding <code>Recv</code> has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning.</p> <p>Trefethen 1996 at 7.</p>

Asserted Claim	Prior Art Disclosure
	<div><p>The diagram illustrates the MultiMATLAB Architecture. At the top, a 'User' is connected via a bidirectional arrow to an 'Interactive Processor'. This 'Interactive Processor' contains a 'MATLAB Process'. Below it, an 'Interconnecting Network' connects to three 'Processor' blocks. Each 'Processor' block contains a 'MATLAB Process'. The network is represented by a horizontal line with vertical connections to each processor block.</p></div> <p>Figure 1: MultiMATLAB Architecture</p> <p>Trefethen 1997 at 3.</p> <p>Figure 4b illustrates a parallel MATLAB implementation of conjugate gradients that uses the SPMD message passing described in Section 5.2. In this case, the matrix A and the different vectors are each distributed by row over all processes as in Figure 6. In each iteration, a process only does the computation required for its local data. However, communication is required for two different operations: the dot-products needed to compute ρ and α and the matrix-vector multiply needed 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.</p> <p>Trefethen 1997 at 11, 13.</p> <p>The following code further illustrates the root node accepting user instructions, and passing user instructions as a sequence to root and remote kernels and/or computational libraries for processing:</p>

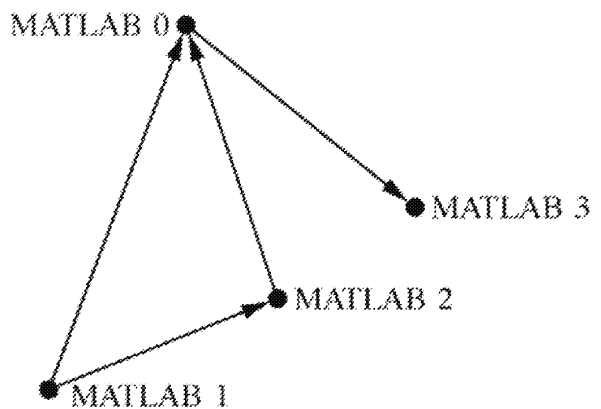
Asserted Claim	Prior Art Disclosure
	<pre> r_local = b_local; rho = Sum(r_local'*r_local); k = 0; while((k < kmax)) k = k+1; if (k == 1) p_local = r_local; else beta = rho/oldrho; p_local = r_local + beta*p_local; end p = Gather(p_local); w_local = A_local * p; alpha = rho/(Sum(p_local'*w_local)); x_local = x_local + alpha * p_local; r_local = r_local - alpha * w_local; oldrho = rho; rho = Sum(r_local'*r_local); end </pre> <p>b. Message Passing MATLAB</p> <p>Trefethen 1997 at 11.</p>

Asserted Claim	Prior Art Disclosure
	<div><div><div><div><div><div>Process 0</div><div>Process 1</div><div>Process 2</div><div></div><div>Process P</div></div></div><div><div><div>A_local</div><div></div></div><div></div></div></div></div><div><div><div>x_local</div><div></div></div><div></div></div><div><div><div>0</div><div>1</div><div>2</div><div></div><div></div><div>P</div></div><div></div></div></div>

A

x

Asserted Claim	Prior Art Disclosure
	<div data-bbox="711 316 1509 389"><h2>Structure of the system (2)</h2></div> <ul style="list-style-type: none">• Client (C++)<ul style="list-style-type: none">– Attaches to MATLAB through MEX interface– Relays commands to server and process returns• Server (C++/MPI)<ul style="list-style-type: none">– Manipulates/processes data (matrices)– Performs the actual parallel computation by calling the appropriate library routines <p data-bbox="582 958 733 990">Choy 3 at 36.</p>

Asserted Claim	Prior Art Disclosure
	 <p data-bbox="625 795 1013 836">Fig. 2. Message Passing.</p> <p data-bbox="582 893 754 925">Choy 2 at 333.</p> <p>The diagram illustrates a message-passing network with four nodes labeled MATLAB 0, MATLAB 1, MATLAB 2, and MATLAB 3. MATLAB 0 is at the top, MATLAB 1 is at the bottom left, MATLAB 2 is at the bottom right, and MATLAB 3 is to the right of MATLAB 0. Directed arrows show the flow of messages: from MATLAB 1 to MATLAB 0, from MATLAB 0 to MATLAB 2, from MATLAB 2 to MATLAB 0, and from MATLAB 0 to MATLAB 3.</p>

Asserted Claim	Prior Art Disclosure
	<p><i>B. MultiMATLAB/MultiOctave Mode</i></p> <p>One of the goals of the project is to make the software to be useful to as wide an audience as possible. In order to achieve this, we found that it would be fruitful to combine other parallel MATLAB approaches into MATLAB * P, to provide a unified parallel MATLAB framework.</p> <p>In conjunction with P. Husbands, we developed a prototype implementation of a MultiMATLAB [35]-like, distributed MATLAB package in MATLAB * P, which we call the PPEngine. With this package and associated m-files, we can run multiple MATLAB processes on the back end and evaluate MATLAB functions in parallel on dense matrices.</p> <p>The system works by starting up MATLAB engine instances on each node through calls to the MATLAB engine interface. From that point on, MATLAB commands can be relayed to the MATLAB engine. The command is do this is <i>mm</i>.</p> <p>Choy 2 at 336</p>

Asserted Claim	Prior Art Disclosure
	<p>We present MATLAB *P [15] as an example of what could be a “right” parallel MATLAB. MATLAB * P is a parallel MATLAB using the back-end support approach, aimed at widespread circulation among a general audience. In order to achieve this, we took ideas from the approaches used by other software found in the survey. For example, the <i>embarrassingly parallel</i> approach allow simple, yet useful, division of work into multiple MATLAB sessions. The <i>message passing</i> approach, which is a superset of the embarrassingly parallel approach, allows finer control between the MATLAB sessions.</p> <p>The main idea in MATLAB *P is that data exist on the parallel server as distributed matrices. Any operations on a distributed matrix (which exists in the MATLAB front end only as a handle) will be relayed to the server transparently. The server calls the appropriate routine from a parallel numerical library (e.g. ScaLAPACK, FFTW, . . .) and the results stay on the server until explicitly requested.</p> <p>Choy 2 at 336.</p>

Asserted Claim	Prior Art Disclosure
	<p>Server software 105 may operate in any suitable way to receive commands from the client and perform operations in parallel, as the invention is not limited in this respect. FIG. 2 is but one example of a suitable configuration of server software 105 for at least some embodiments. In FIG. 2, parallel server 201 includes a plurality of processors, 203a, 203b, . . . , 203n, and a memory 213. Each of processors 203a-203n includes four software modules: client managers 205a, 205b, . . . , 205n; server managers 207a, 207b, . . . , 207n; library managers 209a, 209b, . . . , 209n; and data managers 211a, 211b, . . . , 211n. Memory 213 of parallel server 201 stores a plurality of software libraries 215a, 215b, 215c, . . . , 215n that include routines for performing various operations.</p> <p>Client managers 205 interface with clients, and provides routines for reading commands and arguments from a client and sending results and other data back to the client. Server managers 207 handle communications between server processes executing on different processors of parallel server 201. That is, server managers 207 manage the transmission of data between the processors of parallel server 201 and collect results and error codes from the processors. Library managers</p> <p>Husbands at 10:46-67.</p>

Asserted Claim	Prior Art Disclosure
	<p>The server itself is divided in four self-contained parts:</p> <ol style="list-style-type: none">1. Client Connection Manager Client Connection Manager is responsible for communications with the client. It provides functions for reading commands and arguments from the client and sending the results back to the client. It is only used in the head server process.2. Server Connection Manager Server Connection Manager takes care of communication between server processes. It mainly controls broadcasting of commands and arguments from head

Asserted Claim	Prior Art Disclosure
	<p>process to the slave processes, and collection of results and error codes. Also it provides rank and size information to the processes.</p> <p>3. Package Manager</p> <p>Package Manager is responsible for maintaining a list of available packages and functions provided by them. When initiated by the server process, Package Manager will also perform the actual call to the functions.</p> <p>4. Matrix Manager</p> <p>Matrix Manager contains all the functions needed to create, delete and change the matrices on the server processes. It maintains a mapping from client-side matrix identifiers to actual matrices on the server. It is also responsible for performing garbage collection.</p> <p>Choy Thesis at 26, 28 The following additional example illustrates an FFT calculation on a Matlab *P cluster, including the acceptance of user instructions, communication of instructions, computation and return of results:</p>

Asserted Claim	Prior Art Disclosure
	<p style="text-align: center;">Example 3</p> <p><code>% FFT2 in 4 lines</code></p> <ul style="list-style-type: none"> • <code>A = randn(10000,10000*p);</code> • <code>B = mm('fft',A);</code> • <code>C = B.{'</code> • <code>D = mm('fft',C);</code> • <code>F = D.{'</code> <p>Choy 3 at 47.</p> <p>The following additional example further illustrates MPI message passing between Matlab*P cluster nodes:</p>

Asserted Claim	Prior Art Disclosure
	<pre> function result = simpletest(X) import mpi.*; MPI.init; if MPI.COMM_WORLD.rank == 0; data = [4 2]; MPI.COMM_WORLD.send(data, 2, 1, 13); result = 9; elseif MPI.COMM_WORLD.rank == 1 got = MPI.COMM_WORLD.recv(2, 0, 13); result = got(1) + got(2); end MPI.fnlz; </pre> <p>Choy 3 at 49.</p> <p>To the extent ACS contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of MultiMatlab and the knowledge of a POSITA, it would have been obvious to combine it with one or more of the other references in these invalidity contentions. For example, MultiMatlab and Matlab *P publications reference MPI and PVM-based parallel processing systems as comparable cluster computing systems, motivating a POSITA to look to such systems for modifications relating to this claim element. <i>See, e.g.,</i> Choy 2 at 334. Accordingly, it would have been obvious to modify MultiMatlab in view of the teachings of PVM and MPI publications, such as those referenced in these contentions, as well as references and systems implementing aspects of PVM or MPI. As another example, because MultiMatlab is directed to cluster</p>

Asserted Claim	Prior Art Disclosure
	<p>computing, it would have been obvious to combine with one or more of the prior art references or systems authored by the alleged inventors of the patents in suit or their collaborators, including patents, applications, articles and code by Messrs. Dager, Tannenbaum and Decyk. These references and code, including those related to the Appleseed cluster, Pooch, MacMPI and related projects, are likewise directed to cluster computing using an existing message passing library and a cluster management tool. Moreover, these prior art publications teach peer-to-peer communications between cluster nodes, and use of these cluster computing tools with CAS kernels. Given the similarity of their teaching, it would have been obvious to combine one of more of these systems or publications with MultiMatlab.</p> <p>In addition, this claim is rendered obvious by Intel Paragon Maple in view of Cramer, Choy, FFTW, Howard, and/or Chu for the reasons stated in the Cover Pleading, section IV.</p>
4.pre. The computer cluster of claim 1,	The MultiMatlab References disclose this claim.
4.a. wherein each of the nodes comprises one or more cluster node modules.	<p>The MultiMatlab References disclose “wherein each of the nodes comprises one or more cluster node modules.”</p> <p>MultiMatlab implemented a mechanism for node-to-node communication, including peer-to-peer messaging of the results of mathematical expression evaluation, using the MPI messaging protocol. Implementations could use other protocols, such as PVM and BLACS. The “cluster node module” components comprised one or more interface modules, parallel server modules, and communication libraries, e.g., MPI libraries, and enabled the peer-to-peer architecture. The interface modules comprised one or more of the distributed server / client modules and could further comprise specialized computational libraries, such as BLAS and/or FFTW. These modules and libraries were stored on disk and/or in memory on each of the cluster nodes. Each of the “cluster node modules” enabled communication with other cluster node modules, e.g., through MPI or another message protocol, and/or through a distributed client / server architecture. Moreover, each of the cluster node modules communicated with its local kernel, and communicated with remote kernels by sending communications, e.g., MPI messages, to other cluster node modules, which then passed the instructions to their associated kernels.</p>

Asserted Claim	Prior Art Disclosure
	<p>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.</p> <p>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.</p> <p>Trefethen 1997 at 3.</p>

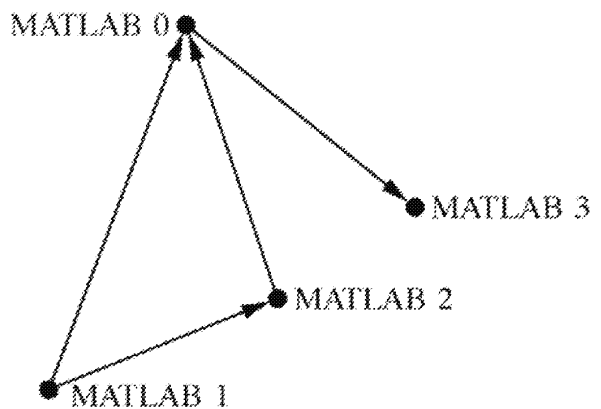
Asserted Claim	Prior Art Disclosure
	<div><pre>graph TD; MATLAB[MATLAB] <--> SeqMEX[Sequential MEX Routine]; MATLAB <--> ParMEX[Parallel MEX Routine]; MATLAB <--> IM[MultiMATLAB Interface Module]; SeqMEX <--> IM; ParMEX <--> IM; IM <--> IT[Indirection Table]; IT <--> CL[Communication Layer]; CL <--> Network[Network];</pre><p>Trefethen 1997 at 4.</p></div>

Asserted Claim	Prior Art Disclosure														
	<table><tr><th colspan="2">SPMD Communication</th></tr><tr><td>Send(pid,data)</td><td>Send data from one process to another.</td></tr><tr><td>Recv(pid)</td><td>Receive data sent from another process.</td></tr><tr><td>Barrier</td><td>Synchronize processes.</td></tr><tr><td>Bcast(pid,data)</td><td>Broadcasts data from processor pid to all processes.</td></tr><tr><td>Sum(data)</td><td>Adds data across all processors to form a global sum.</td></tr><tr><td>Collect(data)</td><td>Collects local data segments into a single global one.</td></tr></table> <p>Table 1: Selected commands in the MultiMATLAB system</p> <p>Trefethen 1997 at 9.</p> <pre>if ID==0 % first process: send a = 1 Send(ID+1,a) elseif ID == Nproc-1 % last process: receive and double a = 2*Recv else % middle processes: receive, double, and send a = 2*Recv Send(ID+1,a) end;</pre> <p>Process 0 creates the variable <code>a</code> with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of <code>a</code>, and sends it along to process 2; and so on. If there are six processors the command <code>Eval('cycle')</code> produces the output</p> <pre>a = 1 a = 2 a = 4 a = 8 a = 16 a = 32</pre> <p>The processes run asynchronously, but since each <code>Send</code> command is only executed after the corresponding <code>Recv</code> has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning.</p> <p>Trefethen 1996 at 7.</p>	SPMD Communication		Send(pid,data)	Send data from one process to another.	Recv(pid)	Receive data sent from another process.	Barrier	Synchronize processes.	Bcast(pid,data)	Broadcasts data from processor pid to all processes.	Sum(data)	Adds data across all processors to form a global sum.	Collect(data)	Collects local data segments into a single global one.
SPMD Communication															
Send(pid,data)	Send data from one process to another.														
Recv(pid)	Receive data sent from another process.														
Barrier	Synchronize processes.														
Bcast(pid,data)	Broadcasts data from processor pid to all processes.														
Sum(data)	Adds data across all processors to form a global sum.														
Collect(data)	Collects local data segments into a single global one.														

Asserted Claim	Prior Art Disclosure
	<div><pre>graph TD User[User] <--> IP[Interactive Processor] subgraph IP_Box [] direction TB IP_Label[Interactive Processor] MP1[MATLAB Process] end subgraph P1_Box [] direction TB P1_Label[Processor] MP2[MATLAB Process] end subgraph P2_Box [] direction TB P2_Label[Processor] MP3[MATLAB Process] end subgraph P3_Box [] direction TB P3_Label[Processor] MP4[MATLAB Process] end IP_Box --- IN[Interconnecting Network] P1_Box --- IN P2_Box --- IN P3_Box --- IN</pre></div> <p>Figure 1: MultiMATLAB Architecture</p> <p>Trefethen 1997 at 3.</p> <p><u>Additional Evidence of Obviousness</u></p>

Asserted Claim	Prior Art Disclosure
	<div data-bbox="711 315 1511 386"><h2>Structure of the system (2)</h2></div> <ul data-bbox="614 467 1606 917" style="list-style-type: none">• Client (C++)<ul style="list-style-type: none">– Attaches to MATLAB through MEX interface– Relays commands to server and process returns• Server (C++/MPI)<ul style="list-style-type: none">– Manipulates/processes data (matrices)– Performs the actual parallel computation by calling the appropriate library routines <p data-bbox="582 958 735 987">Choy 3 at 36.</p>

Asserted Claim	Prior Art Disclosure
	<div><p>The diagram illustrates the architecture of MATLAB*P 2.0. A central dashed box represents 'Server #0'. Inside this box, at the top, are three boxes labeled 'PBLAS', 'FFTW', and 'ScaLAPACK', each with a line connecting to a 'Package Manager' box below them. To the left of the 'Package Manager' are three boxes labeled 'Matrix 1', 'Matrix 2', and 'Matrix 3', each with a line connecting to a 'Matrix Manager' box. To the right of the 'Package Manager' is a 'Client Manager' box. Below the 'Matrix Manager' is a 'Server Manager' box. Outside the 'Server #0' box, to the right, are a 'Proxy' box and a 'Client' box, connected by a vertical line. The 'Client Manager' box has lines connecting to both the 'Proxy' and 'Client' boxes. Below the 'Server #0' box are four boxes labeled 'Server #1', 'Server #2', 'Server #3', and 'Server #4'. Lines connect the 'Server Manager' box to each of these four server boxes. Ellipses (...) are used to indicate that there can be more than one instance of PBLAS, FFTW, ScaLAPACK, Matrix, and Server.</p></div> <p>Fig. 1. Structure of MATLAB*P 2.0</p> <p>Choy 1 at 2.</p>

Asserted Claim	Prior Art Disclosure
	 <p>The diagram illustrates message passing between four MATLAB nodes: MATLAB 0, MATLAB 1, MATLAB 2, and MATLAB 3. MATLAB 0 is at the top, MATLAB 1 is at the bottom left, MATLAB 2 is at the bottom right, and MATLAB 3 is to the right of MATLAB 2. Arrows indicate the direction of message passing: from MATLAB 1 to MATLAB 0, from MATLAB 0 to MATLAB 2, from MATLAB 2 to MATLAB 3, and from MATLAB 2 back to MATLAB 0.</p> <p>Fig. 2. Message Passing.</p> <p>Choy 2 at 333.</p>

Asserted Claim	Prior Art Disclosure
	<p data-bbox="592 310 1034 337"><i>B. MultiMATLAB/MultiOctave Mode</i></p> <p data-bbox="592 355 1293 521">One of the goals of the project is to make the software to be useful to as wide an audience as possible. In order to achieve this, we found that it would be fruitful to combine other parallel MATLAB approaches into MATLAB * P, to provide a unified parallel MATLAB framework.</p> <p data-bbox="592 527 1293 727">In conjunction with P. Husbands, we developed a prototype implementation of a MultiMATLAB [35]-like, distributed MATLAB package in MATLAB * P, which we call the PPEngine. With this package and associated m-files, we can run multiple MATLAB processes on the back end and evaluate MATLAB functions in parallel on dense matrices.</p> <p data-bbox="592 734 1293 899">The system works by starting up MATLAB engine instances on each node through calls to the MATLAB engine interface. From that point on, MATLAB commands can be relayed to the MATLAB engine. The command is do this is <i>mm</i>.</p> <p data-bbox="592 906 743 933">Choy 2 at 336</p>

Asserted Claim	Prior Art Disclosure
	<p>Server software 105 may operate in any suitable way to receive commands from the client and perform operations in parallel, as the invention is not limited in this respect. FIG. 2 is but one example of a suitable configuration of server software 105 for at least some embodiments. In FIG. 2, parallel server 201 includes a plurality of processors, 203a, 203b, . . . , 203n, and a memory 213. Each of processors 203a-203n includes four software modules: client managers 205a, 205b, . . . , 205n; server managers 207a, 207b, . . . , 207n; library managers 209a, 209b, . . . , 209n; and data managers 211a, 211b, . . . , 211n. Memory 213 of parallel server 201 stores a plurality of software libraries 215a, 215b, 215c, . . . , 215n that include routines for performing various operations.</p> <p>Client managers 205 interface with clients, and provides routines for reading commands and arguments from a client and sending results and other data back to the client. Server managers 207 handle communications between server processes executing on different processors of parallel server 201. That is, server managers 207 manage the transmission of data between the processors of parallel server 201 and collect results and error codes from the processors. Library managers</p> <p>Husbands at 10:46-67.</p> <p>To the extent ACS contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of MultiMatlab and the knowledge of a POSITA, it would have been obvious to combine it with one or more of the other references in these invalidity contentions. For example, MultiMatlab and Matlab*P publications reference MPI and PVM-based parallel processing systems as comparable cluster computing systems, motivating a POSITA to look to such systems for modifications relating to this claim element. <i>See, e.g., Choy 2 at 334.</i> Accordingly, it would have been obvious to modify MultiMatlab in view of the teachings of PVM and MPI publications, such as those referenced in these contentions, as well as references and systems</p>

Asserted Claim	Prior Art Disclosure
	<p>implementing aspects of PVM or MPI. As another example, because MultiMatlab is directed to cluster computing, it would have been obvious to combine with one or more of the prior art references or systems authored by the alleged inventors of the patents in suit or their collaborators, including patents, applications, articles and code by Messrs. Dager, Tannenbaum and Decyk. These references and code, including those related to the Appleseed cluster, Pooch, MacMPI and related projects, are likewise directed to cluster computing using an existing message passing library and a cluster management tool. Moreover, these prior art publications teach peer-to-peer communications between cluster nodes, and use of these cluster computing tools with CAS kernels. Given the similarity of their teaching, it would have been obvious to combine one of more of these systems or publications with MultiMatlab.</p> <p>In addition, this claim is rendered obvious by Intel Paragon Maple in view of Cramer, Choy, FFTW, Howard, and/or Chu for the reasons stated in the Cover Pleading, section IV.</p>
5. The computer cluster of claim 1,	The MultiMatlab References disclose this claim.
wherein each single-node kernel is stored in a non-transitory computer-readable medium and configured to accept and execute a request.	The MultiMatlab References disclose “wherein each single-node kernel is stored in a non-transitory computer-readable medium and configured to accept and execute a request.” <i>See</i> 1.g.
6. The computer cluster of claim 5,	The MultiMatlab References disclose this claim.
wherein each of the nodes comprises one or more cluster node modules,	The MultiMatlab References disclose “wherein each of the nodes comprises one or more cluster node modules, wherein each of the cluster node modules comprises instructions stored in a non-transitory computer-readable medium, and wherein the instructions, when executed by the hardware processor, cause the cluster node module to communicate with the single-node kernel and with one or more other cluster node modules. <i>See</i> claim 4.

Asserted Claim	Prior Art Disclosure
<p>wherein each of the cluster node modules comprises instructions stored in a non-transitory computer-readable medium, and wherein the instructions, when executed by the hardware processor, cause the cluster node module to communicate with the single-node kernel and with one or more other cluster node modules.</p>	
<p>7.pre. The computer cluster of claim 6,</p>	<p>The MultiMatlab References disclose this claim.</p>
<p>7.a. wherein the plurality of cluster node modules act as a cluster.</p>	<p>The MultiMatlab References disclose “wherein the plurality of cluster node modules act as a cluster.”</p> <p>The plurality of cluster node modules act as a cluster by performing functions enabling cluster communications and scheduling of parallel workloads. <i>See</i> claims 1, 4.</p>
<p>8.pre. The computer cluster of claim 6,</p>	<p>The MultiMatlab References disclose this claim.</p>
<p>8.a. wherein the plurality of cluster</p>	<p>The MultiMatlab References disclose “wherein the plurality of cluster node modules communicate with one another to act as a cluster.”</p>

Asserted Claim	Prior Art Disclosure
node modules communicate with one another to act as a cluster.	The cluster node modules communicate with one another to act as a cluster, including the communication of MPI messages and/or remote function calls for evaluating mathematical expressions on the cluster in parallel. <i>See</i> claims 1, 4.
9.pre. The computer cluster of claim 8,	The MultiMatlab References disclose this claim.
9.a. wherein the computer cluster includes the user interface.	The MultiMatlab References disclose “wherein the computer cluster includes the user interface.” <i>See</i> claim 1.
10.pre. The computer cluster of claim 9,	The MultiMatlab References disclose this claim.
10.a. wherein each cluster node module accepts instructions from the user interface and interprets one or more of the instructions.	<p>The MultiMatlab References disclose “wherein each cluster node module accepts instructions from the user interface and interprets one or more of the instructions.”</p> <p>MultiMatlab cluster node module components on each node, accept and interpret the user instructions, e.g., parallel MultiMatlab scripts, and others. For example, the cluster node modules may interpret these user instructions as commands to send or receive MPI messages and/or make remote function calls relating to parallel MATLAB computations. The root cluster node module may accept instructions directly from the user interface, while remote modules accept them when the root process communicates to the remote nodes. <i>See</i> claims 1, 4.</p> <p>As a further example, the cluster node modules accept MPI message instructions and MATLAB instructions from the user interface and interpret them in order to execute the instructions:</p>

Asserted Claim	Prior Art Disclosure														
	<table><tr><th colspan="2">SPMD Communication</th></tr><tr><td>Send(pid,data)</td><td>Send data from one process to another.</td></tr><tr><td>Recv(pid)</td><td>Receive data sent from another process.</td></tr><tr><td>Barrier</td><td>Synchronize processes.</td></tr><tr><td>Bcast(pid,data)</td><td>Broadcasts data from processor pid to all processes.</td></tr><tr><td>Sum(data)</td><td>Adds data across all processors to form a global sum.</td></tr><tr><td>Collect(data)</td><td>Collects local data segments into a single global one.</td></tr></table> <p>Table 1: Selected commands in the MultiMATLAB system</p> <p>Trefethen 1997 at 9.</p> <pre>if ID==0 % first process: send a = 1 Send(ID+1,a) elseif ID == Nproc-1 % last process: receive and double a = 2*Recv else % middle processes: receive, double, and send a = 2*Recv Send(ID+1,a) end;</pre> <p>Process 0 creates the variable <code>a</code> with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of <code>a</code>, and sends it along to process 2; and so on. If there are six processors the command <code>Eval('cycle')</code> produces the output</p> <pre>a = 1 a = 2 a = 4 a = 8 a = 16 a = 32</pre> <p>The processes run asynchronously, but since each <code>Send</code> command is only executed after the corresponding <code>Recv</code> has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning.</p>	SPMD Communication		Send(pid,data)	Send data from one process to another.	Recv(pid)	Receive data sent from another process.	Barrier	Synchronize processes.	Bcast(pid,data)	Broadcasts data from processor pid to all processes.	Sum(data)	Adds data across all processors to form a global sum.	Collect(data)	Collects local data segments into a single global one.
SPMD Communication															
Send(pid,data)	Send data from one process to another.														
Recv(pid)	Receive data sent from another process.														
Barrier	Synchronize processes.														
Bcast(pid,data)	Broadcasts data from processor pid to all processes.														
Sum(data)	Adds data across all processors to form a global sum.														
Collect(data)	Collects local data segments into a single global one.														

Asserted Claim	Prior Art Disclosure
	<p>Trefethen 1996 at 7.</p> <p><u>Additional Evidence of Obviousness</u></p> <p>The following additional example illustrates an FFT calculation on a Matlab *P cluster, including the acceptance and interpretation of user instructions, communication of instructions, computation and return of results:</p> <div data-bbox="579 557 1461 1243" data-label="Code-Block"> <pre> Example 3 % FFT2 in 4 lines • A = randn(10000,10000*p); • B = mm('fft',A); • C = B.'; • D = mm('fft',C); • F = D.'; </pre> </div> <p>Choy 3 at 47.</p> <p>The following additional example further illustrates MPI message passing between Matlab*P cluster nodes:</p>

Asserted Claim	Prior Art Disclosure
	<div><pre>function result = simpletest(X) import mpi.*; MPI.init; if MPI.COMM_WORLD.rank == 0; data = [4 2]; MPI.COMM_WORLD.send(data, 2, 1, 13); result = 9; elseif MPI.COMM_WORLD.rank == 1 got = MPI.COMM_WORLD.recv(2, 0, 13); result = got(1) + got(2); end MPI.fnlz;</pre></div> <p>Choy 3 at 49.</p> <p>To the extent ACS contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of MultiMatlab and the knowledge of a POSITA, it would have been obvious to combine it with one or more of the other references in these invalidity contentions. For example, MultiMatlab and Matlab*P publications reference MPI and PVM-based parallel processing systems as comparable cluster computing systems, motivating a POSITA to look to such systems for modifications relating to this claim element. <i>See, e.g.,</i> Choy 2 at 334. Accordingly, it would have been obvious to modify MultiMatlab in view of the teachings of PVM and MPI publications, such as those referenced in these contentions, as well as references and systems</p>

Asserted Claim	Prior Art Disclosure
	<p>implementing aspects of PVM or MPI. As another example, because MultiMatlab is directed to cluster computing, it would have been obvious to combine with one or more of the prior art references or systems authored by the alleged inventors of the patents in suit or their collaborators, including patents, applications, articles and code by Messrs. Dager, Tannenbaum and Decyk. These references and code, including those related to the Appleseed cluster, Pooch, MacMPI and related projects, are likewise directed to cluster computing using an existing message passing library and a cluster management tool. Moreover, these prior art publications teach peer-to-peer communications between cluster nodes, and use of these cluster computing tools with CAS kernels. Given the similarity of their teaching, it would have been obvious to combine one of more of these systems or publications with MultiMatlab.</p> <p>In addition, this claim is rendered obvious by Intel Paragon Maple in view of Cramer, Choy, FFTW, Howard, and/or Chu for the reasons stated in the Cover Pleading, section IV.</p>
18.pre. The computer cluster of claim 1,	The MultiMatlab References disclose this claim.
18.a. wherein one or more of the nodes are configured to communicate at least some of the user instructions.	<p>The MultiMatlab References disclose “wherein one or more of the nodes are configured to communicate at least some of the user instructions.”</p> <p>MultiMatlab cluster nodes, accept and interpret the user instructions, e.g., parallel MultiMatlab scripts, and others. For example, the cluster node modules may interpret these user instructions as commands to send or receive MPI messages and/or make remote function calls relating to parallel MATLAB computations. The root node may communicate user instructions to other nodes in the form of MPI messages and/or remote function calls. The receiving nodes pass the user instructions to their corresponding kernels for evaluation. <i>See</i> claims 1, 4.</p> <p>As a further example, the cluster node modules accept MPI message instructions and MATLAB instructions from the user interface and communicate them to other nodes:</p>

Asserted Claim	Prior Art Disclosure														
	<table border="1"> <thead> <tr> <th colspan="2" data-bbox="592 303 879 336">SPMD Communication</th></tr> </thead> <tbody> <tr> <td data-bbox="592 336 879 368">Send(pid,data)</td><td data-bbox="879 336 1757 368">Send data from one process to another.</td></tr> <tr> <td data-bbox="592 368 879 401">Recv(pid)</td><td data-bbox="879 368 1757 401">Receive data sent from another process.</td></tr> <tr> <td data-bbox="592 401 879 433">Barrier</td><td data-bbox="879 401 1757 433">Synchronize processes.</td></tr> <tr> <td data-bbox="592 433 879 466">Bcast(pid,data)</td><td data-bbox="879 433 1757 466">Broadcasts data from processor pid to all processes.</td></tr> <tr> <td data-bbox="592 466 879 498">Sum(data)</td><td data-bbox="879 466 1757 498">Adds data across all processors to form a global sum.</td></tr> <tr> <td data-bbox="592 498 879 531">Collect(data)</td><td data-bbox="879 498 1757 531">Collects local data segments into a single global one.</td></tr> </tbody> </table> <p data-bbox="793 592 1507 625">Table 1: Selected commands in the MultiMATLAB system</p> <p data-bbox="579 660 812 693">Trefethen 1997 at 9.</p> <pre data-bbox="644 742 1554 946"> if ID==0 % first process: send a = 1 Send(ID+1,a) elseif ID == Nproc-1 % last process: receive and double a = 2*Recv else % middle processes: receive, double, and send a = 2*Recv Send(ID+1,a) end; </pre> <p data-bbox="586 971 1660 1057">Process 0 creates the variable <i>a</i> with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of <i>a</i>, and sends it along to process 2; and so on. If there are six processors the command <code>Eval('cycle')</code> produces the output</p> <pre data-bbox="644 1084 733 1219"> a = 1 a = 2 a = 4 a = 8 a = 16 a = 32 </pre> <p data-bbox="586 1250 1588 1336">The processes run asynchronously, but since each <code>Send</code> command is only executed after the corresponding <code>Recv</code> has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning.</p>	SPMD Communication		Send(pid,data)	Send data from one process to another.	Recv(pid)	Receive data sent from another process.	Barrier	Synchronize processes.	Bcast(pid,data)	Broadcasts data from processor pid to all processes.	Sum(data)	Adds data across all processors to form a global sum.	Collect(data)	Collects local data segments into a single global one.
SPMD Communication															
Send(pid,data)	Send data from one process to another.														
Recv(pid)	Receive data sent from another process.														
Barrier	Synchronize processes.														
Bcast(pid,data)	Broadcasts data from processor pid to all processes.														
Sum(data)	Adds data across all processors to form a global sum.														
Collect(data)	Collects local data segments into a single global one.														

Asserted Claim	Prior Art Disclosure
	<p>Trefethen 1996 at 7.</p> <p><u>Additional Evidence of Obviousness</u></p> <p>The following additional example illustrates an FFT calculation on a Matlab *P cluster, including the acceptance and interpretation of user instructions, communication of instructions to other nodes, computation and return of results:</p> <div><h3>Example 3</h3><p>% FFT2 in 4 lines</p><ul style="list-style-type: none">• A = randn(10000,10000*p);• B = mm('fft',A);• C = B.:';• D = mm('fft',C);• F = D.:';</div> <p>Choy 3 at 47.</p> <p>The following additional example further illustrates MPI message passing between Matlab*P cluster nodes:</p>

Asserted Claim	Prior Art Disclosure
	<pre data-bbox="582 329 1507 1063"> function result = simpletest(X) import mpi.*; MPI.init; if MPI.COMM_WORLD.rank == 0; data = [4 2]; MPI.COMM_WORLD.send(data, 2, 1, 13); result = 9; elseif MPI.COMM_WORLD.rank == 1 got = MPI.COMM_WORLD.recv(2, 0, 13); result = got(1) + got(2); end MPI.fnlz; </pre> <p data-bbox="582 1068 735 1101">Choy 3 at 49.</p> <p data-bbox="582 1133 1858 1354">To the extent ACS contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of MultiMatlab and the knowledge of a POSITA, it would have been obvious to combine it with one or more of the other references in these invalidity contentions. For example, MultiMatlab and Matlab*P publications reference MPI and PVM-based parallel processing systems as comparable cluster computing systems, motivating a POSITA to look to such systems for modifications relating to this claim element. <i>See, e.g.,</i> Choy 2 at 334. Accordingly, it would have been obvious to modify MultiMatlab in view of the teachings of PVM and MPI publications, such as those referenced in these contentions, as well as references and systems</p>

Asserted Claim	Prior Art Disclosure
	<p>implementing aspects of PVM or MPI. As another example, because MultiMatlab is directed to cluster computing, it would have been obvious to combine with one or more of the prior art references or systems authored by the alleged inventors of the patents in suit or their collaborators, including patents, applications, articles and code by Messrs. Dager, Tannenbaum and Decyk. These references and code, including those related to the Appleseed cluster, Pooch, MacMPI and related projects, are likewise directed to cluster computing using an existing message passing library and a cluster management tool. Moreover, these prior art publications teach peer-to-peer communications between cluster nodes, and use of these cluster computing tools with CAS kernels. Given the similarity of their teaching, it would have been obvious to combine one of more of these systems or publications with MultiMatlab.</p> <p>In addition, this claim is rendered obvious by Intel Paragon Maple in view of Cramer, Choy, FFTW, Howard, and/or Chu for the reasons stated in the Cover Pleading, section IV.</p>
19.pre. The computer cluster of claim 18,	The MultiMatlab References disclose this claim
19.a. wherein one or more of the nodes are configured to communicate at least some of the user instructions to one or more single-node kernels.	The MultiMatlab References disclose “wherein one or more of the nodes are configured to communicate at least some of the user instructions to one or more single-node kernels.” <i>See</i> claims 1, 18.
20.pre. The computer cluster of claim 1,	The MultiMatlab References disclose this claim.
20.a. wherein one or more of the	The MultiMatlab References disclose “wherein one or more of the nodes are configured to accept user instructions via one or more of the nodes.” <i>See</i> claims 1, 18.

Asserted Claim	Prior Art Disclosure
nodes are configured to accept user instructions via one or more of the nodes.	
21.pre. The computer cluster of claim 20,	The MultiMatlab References disclose this claim.
21.a. wherein one or more of the nodes are configured to communicate at least some of the user instructions using the mechanism for the nodes to communicate with each other.	The MultiMatlab References disclose “wherein one or more of the nodes are configured to communicate at least some of the user instructions using the mechanism for the nodes to communicate with each other.” <i>See</i> claims 1, 18.
22.pre. The computer cluster of claim 1,	The MultiMatlab References disclose this claim.
22.a. wherein one or more of the nodes are configured to transmit at least some of the user instructions that	The MultiMatlab References disclose “wherein one or more of the nodes are configured to transmit at least some of the user instructions that originate from a user interface.” <i>See</i> claims 1, 18.

Asserted Claim	Prior Art Disclosure
originate from a user interface.	
24.pre. The computer cluster of claim 1,	The MultiMatlab References disclose this claim.
24.a. wherein one or more of the nodes are configured to accept user instructions before communicating at least some of the user instructions to one or more single-node kernels.	The MultiMatlab References disclose “wherein one or more of the nodes are configured to accept user instructions before communicating at least some of the user instructions to one or more single-node kernels.” <i>See</i> claims 1, 18.
25.pre. The computer cluster of claim 1,	The MultiMatlab References disclose this claim.
25.a. wherein the plurality of nodes are configured to communicate with one another to interpret and translate commands for execution by a plurality of single-node kernels.	<p>The MultiMatlab References disclose “wherein the plurality of nodes are configured to communicate with one another to interpret and translate commands for execution by a plurality of single-node kernels.”</p> <p>The MultiMatlab cluster nodes may communicate commands to other nodes in the form of MPI messages and/or remote function calls. The receiving nodes pass the commands to their corresponding kernels for evaluation. In order to solve parallelized computations, the nodes communicate, and each node interprets and translates commands to other nodes in the form of MPI messages and/or remote function calls. <i>See</i> claims 1, 4.</p> <p>As a further example, the cluster node modules accept MPI message instructions and MATLAB instructions from the user interface and communicate them to other nodes, with each node translating and interpreting the commands:</p>

Asserted Claim	Prior Art Disclosure														
	<table><tr><th colspan="2">SPMD Communication</th></tr><tr><td>Send(pid,data)</td><td>Send data from one process to another.</td></tr><tr><td>Recv(pid)</td><td>Receive data sent from another process.</td></tr><tr><td>Barrier</td><td>Synchronize processes.</td></tr><tr><td>Bcast(pid,data)</td><td>Broadcasts data from processor pid to all processes.</td></tr><tr><td>Sum(data)</td><td>Adds data across all processors to form a global sum.</td></tr><tr><td>Collect(data)</td><td>Collects local data segments into a single global one.</td></tr></table> <p>Table 1: Selected commands in the MultiMATLAB system</p> <p>Trefethen 1997 at 9.</p>	SPMD Communication		Send(pid,data)	Send data from one process to another.	Recv(pid)	Receive data sent from another process.	Barrier	Synchronize processes.	Bcast(pid,data)	Broadcasts data from processor pid to all processes.	Sum(data)	Adds data across all processors to form a global sum.	Collect(data)	Collects local data segments into a single global one.
SPMD Communication															
Send(pid,data)	Send data from one process to another.														
Recv(pid)	Receive data sent from another process.														
Barrier	Synchronize processes.														
Bcast(pid,data)	Broadcasts data from processor pid to all processes.														
Sum(data)	Adds data across all processors to form a global sum.														
Collect(data)	Collects local data segments into a single global one.														

Asserted Claim	Prior Art Disclosure
	<pre> if ID==0 % first process: send a = 1 Send(ID+1,a) elseif ID == Nproc-1 % last process: receive and double a = 2*Recv else % middle processes: receive, double, and send a = 2*Recv Send(ID+1,a) end; </pre> <p>Process 0 creates the variable <i>a</i> with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of <i>a</i>, and sends it along to process 2; and so on. If there are six processors the command <code>Eval('cycle')</code> produces the output</p> <pre> a = 1 a = 2 a = 4 a = 8 a = 16 a = 32 </pre> <p>The processes run asynchronously, but since each <code>send</code> command is only executed after the corresponding <code>recv</code> has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning.</p> <p>Trefethen 1996 at 7.</p> <p><u>Additional Evidence of Obviousness</u></p> <p>The following additional example illustrates an FFT calculation on a Matlab *P cluster, including the translation and interpretation of user instructions, communication of instructions to other nodes, computation and return of results:</p>

Asserted Claim	Prior Art Disclosure
	<p style="text-align: center;">Example 3</p> <p><code>% FFT2 in 4 lines</code></p> <ul style="list-style-type: none"> • <code>A = randn(10000,10000*p);</code> • <code>B = mm('fft',A);</code> • <code>C = B.{'</code> • <code>D = mm('fft',C);</code> • <code>F = D.{'</code> <p>Choy 3 at 47.</p> <p>The following additional example further illustrates MPI message passing between Matlab*P cluster nodes, which includes interpretation and translation of high level code into low level code and/or interpretation and translation from one data format to another:</p>

Asserted Claim	Prior Art Disclosure
	<pre data-bbox="631 316 1425 987">function result = simpletest(X) import mpi.*; MPI.init; if MPI.COMM_WORLD.rank == 0; data = [4 2]; MPI.COMM_WORLD.send(data, 2, 1, 13); result = 9; elseif MPI.COMM_WORLD.rank == 1 got = MPI.COMM_WORLD.recv(2, 0, 13); result = got(1) + got(2); end MPI.fnlz;</pre> <p data-bbox="582 1036 735 1065">Choy 3 at 49.</p> <p data-bbox="582 1101 1841 1195">The following example further illustrates the interpretation and translation of commands, with Matlab *P commands communicated to the cluster nodes as server calls, and the commands interpreted and translated for execution by the relevant libraries, e.g., ScalPACK or FFTW:</p>

Asserted Claim	Prior Art Disclosure
	<p>We present MATLAB*P [15] as an example of what could be a “right” parallel MATLAB. MATLAB * P is a parallel MATLAB using the back-end support approach, aimed at widespread circulation among a general audience. In order to achieve this, we took ideas from the approaches used by other software found in the survey. For example, the <i>embarrassingly parallel</i> approach allow simple, yet useful, division of work into multiple MATLAB sessions. The <i>message passing</i> approach, which is a superset of the embarrassingly parallel approach, allows finer control between the MATLAB sessions.</p> <p>The main idea in MATLAB*P is that data exist on the parallel server as distributed matrices. Any operations on a distributed matrix (which exists in the MATLAB front end only as a handle) will be relayed to the server transparently. The server calls the appropriate routine from a parallel numerical library (e.g. ScaLAPACK, FFTW, . . .) and the results stay on the server until explicitly requested.</p> <p>Choy 2 at 336.</p>

Asserted Claim	Prior Art Disclosure
	<pre>e = eig(X);</pre> <p>The command computes the eigenvalues of X by calling the appropriate ScaLAPACK routines and stores the result in a matrix e, which resides on the server. The result is not returned to the client unless explicitly requested, to reduce data traffic. Again the syntax is the same as ordinary MATLAB</p> <pre>E = pp2matlab(e);</pre> <p>This command returns the result to MATLAB. This is one of the few commands in MATLAB * P not found in ordinary MATLAB (matlab2pp is another).</p> <p>The use of the p variable along with overloaded MATLAB routines enable existing MATLAB scripts to be reused. For example</p> <pre>function H = h1lb(n) J = 1 : n; J = J(ones(n,1),:); I = J'; E = ones(n,n); H = E./(I + J - 1);</pre> <p>The above is the built-in MATLAB routine to construct a Hilbert matrix (obtained through <i>type h1lb</i>). Because the operators in the routine (colon, ones, subsasgn, transpose, rdivide, +, -) are overloaded to work with p, typing</p> <pre>H = h1lb(16384 * p)</pre> <p>would create a 16384-by-16384 Hilbert matrix on the server. By exploiting MATLAB's object-oriented features in this way, many existing scripts would run in parallel under</p>

Asserted Claim	Prior Art Disclosure
	Choy 2 at 336.
26.pre. A computer cluster comprising:	To the extent the preamble is limiting, the MultiMatlab References disclose it. <i>See</i> 1.pre.
26.a. a plurality of nodes, wherein one or more of the nodes are configured to receive:	The MultiMatlab References disclose “a plurality of nodes, wherein one or more of the nodes are configured to receive.” <i>See</i> 1.a, 1.b.
26.b. a command to start a cluster initialization process for the computer cluster,	The MultiMatlab References disclose “a command to start a cluster initialization process for the computer cluster.” <i>See</i> 1.b.
26.c. wherein the cluster initialization process comprises establishing communication among two or more of the nodes; and	The MultiMatlab References disclose “wherein the cluster initialization process comprises establishing communication among two or more of the nodes.” MultiMatlab initiation process loads and initiates the communication components on each node, e.g., MPI libraries and/or parallel server libraries, thereby establishing communications between the cluster nodes. <i>See</i> 1.b.
26.d. an instruction from a user interface or a script; and	The MultiMatlab References disclose “an instruction from a user interface or a script.” <i>See</i> 26.c.
26.e. a mechanism for the nodes to communicate results of mathematical expression	The MultiMatlab References disclose “a mechanism for the nodes to communicate results of mathematical expression evaluation with each other using asynchronous calls.” MultiMatlab implemented a mechanism for node-to-node communication, including peer-to-peer messaging of the results of mathematical expression evaluation, using the MPI messaging protocol and/or remote function calls. <i>See</i> 1.d.

Asserted Claim	Prior Art Disclosure
evaluation with each other using asynchronous calls;	<p>This mechanism comprised asynchronous calls, e.g., MPI and/or remote function calls, including calls to send, receive and otherwise process computation results. The MultiMatlabcluster kernels and/or compute libraries performed continuous computations, while the nodes communicated with one another using the communication mechanism:</p> <pre> 1 1 1 1 1 1 2 1 1 1 1 1 3 1 1 1 1 1 4 1 1 1 1 1 5 1 1 1 1 1 6 1 1 1 1 1 7 1 1 1 1 1 8 1 1 1 1 1 9 1 1 1 1 1 10 1 1 1 1 1 11 1 1 1 1 1 12 1 1 1 1 1 13 1 1 1 1 1 14 1 1 1 1 1 15 1 1 1 1 1 16 1 1 1 1 1 17 1 1 1 1 1 18 1 1 1 1 1 19 1 1 1 1 1 20 1 1 1 1 1 21 1 1 1 1 1 22 1 1 1 1 1 23 1 1 1 1 1 24 1 1 1 1 1 25 1 1 1 1 1 26 1 1 1 1 1 27 1 1 1 1 1 28 1 1 1 1 1 29 1 1 1 1 1 30 1 1 1 1 1 31 1 1 1 1 1 32 1 1 1 1 1 33 1 1 1 1 1 34 1 1 1 1 1 35 1 1 1 1 1 36 1 1 1 1 1 37 1 1 1 1 1 38 1 1 1 1 1 39 1 1 1 1 1 40 1 1 1 1 1 41 1 1 1 1 1 42 1 1 1 1 1 43 1 1 1 1 1 44 1 1 1 1 1 45 1 1 1 1 1 46 1 1 1 1 1 47 1 1 1 1 1 48 1 1 1 1 1 49 1 1 1 1 1 50 1 1 1 1 1 51 1 1 1 1 1 52 1 1 1 1 1 53 1 1 1 1 1 54 1 1 1 1 1 55 1 1 1 1 1 56 1 1 1 1 1 57 1 1 1 1 1 58 1 1 1 1 1 59 1 1 1 1 1 60 1 1 1 1 1 61 1 1 1 1 1 62 1 1 1 1 1 63 1 1 1 1 1 64 1 1 1 1 1 65 1 1 1 1 1 66 1 1 1 1 1 67 1 1 1 1 1 68 1 1 1 1 1 69 1 1 1 1 1 70 1 1 1 1 1 71 1 1 1 1 1 72 1 1 1 1 1 73 1 1 1 1 1 74 1 1 1 1 1 75 1 1 1 1 1 76 1 1 1 1 1 77 1 1 1 1 1 78 1 1 1 1 1 79 1 1 1 1 1 80 1 1 1 1 1 81 1 1 1 1 1 82 1 1 1 1 1 83 1 1 1 1 1 84 1 1 1 1 1 85 1 1 1 1 1 86 1 1 1 1 1 87 1 1 1 1 1 88 1 1 1 1 1 89 1 1 1 1 1 90 1 1 1 1 1 91 1 1 1 1 1 92 1 1 1 1 1 93 1 1 1 1 1 94 1 1 1 1 1 95 1 1 1 1 1 96 1 1 1 1 1 97 1 1 1 1 1 98 1 1 1 1 1 99 1 1 1 1 1 100 1 1 1 1 1 </pre> <pre> if ID==0 % first process: send a = 1 Send(ID+1,a) elseif ID == Nproc-1 % last process: receive and double a = 2*Recv else % middle processes: receive, double, and send a = 2*Recv Send(ID+1,a) end; </pre> <p>Process 0 creates the variable <i>a</i> with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of <i>a</i>, and sends it along to process 2; and so on. If there are six processors the command <code>Eval('cycle')</code> produces the output</p> <pre> a = 1 a = 2 a = 4 a = 8 a = 16 a = 32 </pre> <p>The processes run asynchronously, but since each <code>send</code> command is only executed after the corresponding <code>recv</code> has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning.</p> <p>Trefethen 1996 at 7.</p> <p>As a further example, MultiMatlab implemented MPI calls, including calls that the '768 preferred embodiment identifies as used for "asynchronous" communication of computation results, such as <code>MPI_Isend</code>, <code>MPI_IRecv</code>, <code>MPI_IWait</code> and others. '768, 13:42-65.</p>

Asserted Claim	Prior Art Disclosure
	<p>2.3. Send, Recv, Probe, Barrier</p> <p>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</p> <pre> x = [pi pi^2]; Send(3,x) Eval(3, 'Recv') </pre> <p>passes a message containing a 2-vector from the master process to process 3, leading to the output</p> <pre> 3.1416 9.8696 </pre> <p>An optional argument can be added in <code>Recv</code> to specify the source. Another optional argument may be added in both <code>Send</code> and <code>Recv</code> to specify a message tag so as to ensure that sends and receives are properly matched and to aid in error checking. The command</p> <pre> Probe, </pre> <p>run on any process, again with optional source process number and message tag, returns 1 (true) if a message has arrived from the indicated source with the indicated tag, otherwise 0 (false).</p> <p>Trefethen 1996 at 6.</p> <p>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.</p> <p>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.</p> <p>Trefethen 1996 at 11.</p>

Asserted Claim	Prior Art Disclosure
	<p data-bbox="584 334 1021 362"><u>Additional Evidence of Obviousness</u></p> <p data-bbox="584 402 1535 873">This organization offers great advantages. First of all, debugging is made easier because bugs are localized and thus are much easier to track down. Also, this compartmentized approach allows easier extension of the server. For example, the basic Server Connection Manager makes use of MPI (Message Passing Interface) as the means of communication between server processes. However, one could write a Server Connection Manager that uses PVM (Parallel Virtual Machine) instead. As long as the new version implements all the public functions in the class, no change is needed in any other part of the code.</p> <p data-bbox="584 883 720 911">Choy 1 at 2.</p>

Asserted Claim	Prior Art Disclosure
	<pre>function result = simpletest(X) import mpi.*; MPIinit; if MPI.COMM_WORLD.rank == 0; data = [4 2]; MPI.COMM_WORLD.send(data, 2, 1, 13); result = 9; elseif MPI.COMM_WORLD.rank == 1 got = MPI.COMM_WORLD.recv(2, 0, 13); result = got(1) + got(2); end MPI.fnlz;</pre> <p>Choy 3 at 49.</p> <p>In addition, because, as demonstrated above, the processes on each Matlab *P node operate asynchronously, it would have been obvious to implement asynchronous calls, such as MPI, PVM or remote function calls to share computation results between the nodes. Such calls were known in the art, and included in standard communication libraries, such as MPI and PVM. MPI, for example, included the calls identified in the '768 preferred embodiment as comprising "asynchronous" calls, and disclosed advantages of such calls:</p>

Asserted Claim	Prior Art Disclosure
	<p>3.7 Nonblocking communication</p> <p>One can improve performance on many systems by overlapping communication and computation. This is especially true on systems where communication can be executed autonomously by an intelligent communication controller. Light-weight threads are one mechanism for achieving such overlap. An alternative mechanism that often leads to better performance is to use nonblocking communication. A nonblocking send start call initiates the send operation, but does not complete it. The send start call will return before the message was copied out of the send buffer. A separate send complete call is needed to complete the communication, i.e., to verify that the data has been copied out of the send buffer. 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. Similarly, a nonblocking receive start call initiates the receive operation, but does not complete it. The call will return before a message is stored into the receive buffer. A separate receive complete call is needed to complete the receive operation and verify that the data has been received into the receive buffer. With suitable hardware, the transfer</p> <p>MPI Specification at 36.</p>

Asserted Claim	Prior Art Disclosure																					
	<p>MPI.ISEND(buf, count, datatype, dest, tag, comm, request)</p> <table><tr><td>IN</td><td>buf</td><td>initial address of send buffer (choice)</td></tr><tr><td>IN</td><td>count</td><td>number of elements in send buffer (integer)</td></tr><tr><td>IN</td><td>datatype</td><td>datatype of each send buffer element (handle)</td></tr><tr><td>IN</td><td>dest</td><td>rank of destination (integer)</td></tr><tr><td>IN</td><td>tag</td><td>message tag (integer)</td></tr><tr><td>IN</td><td>comm</td><td>communicator (handle)</td></tr><tr><td>OUT</td><td>request</td><td>communication request (handle)</td></tr></table> <p>int MPI_Isend(void* buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *request)</p> <p>MPI.ISEND(BUF, COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR)</p> <p><type> BUF(*)</p> <p>INTEGER COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR</p> <p>Start a standard mode, nonblocking send.</p> <p>MPI Specification at 38.</p>	IN	buf	initial address of send buffer (choice)	IN	count	number of elements in send buffer (integer)	IN	datatype	datatype of each send buffer element (handle)	IN	dest	rank of destination (integer)	IN	tag	message tag (integer)	IN	comm	communicator (handle)	OUT	request	communication request (handle)
IN	buf	initial address of send buffer (choice)																				
IN	count	number of elements in send buffer (integer)																				
IN	datatype	datatype of each send buffer element (handle)																				
IN	dest	rank of destination (integer)																				
IN	tag	message tag (integer)																				
IN	comm	communicator (handle)																				
OUT	request	communication request (handle)																				

Asserted Claim	Prior Art Disclosure																					
	<p>MPI_RECV (buf, count, datatype, source, tag, comm, request)</p> <table><tr><td>OUT</td><td>buf</td><td>initial address of receive buffer (choice)</td></tr><tr><td>IN</td><td>count</td><td>number of elements in receive buffer (integer)</td></tr><tr><td>IN</td><td>datatype</td><td>datatype of each receive buffer element (handle)</td></tr><tr><td>IN</td><td>source</td><td>rank of source (integer)</td></tr><tr><td>IN</td><td>tag</td><td>message tag (integer)</td></tr><tr><td>IN</td><td>comm</td><td>communicator (handle)</td></tr><tr><td>OUT</td><td>request</td><td>communication request (handle)</td></tr></table> <p>int MPI_Irecv(void* buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Request *request)</p> <p>MPI_RECV(BUF, COUNT, DATATYPE, SOURCE, TAG, COMM, REQUEST, IERRDR)</p> <p><type> BUF(*)</p> <p>INTEGER COUNT, DATATYPE, SOURCE, TAG, COMM, REQUEST, IERRDR</p> <p>Start a nonblocking receive.</p> <p>MPI Specification at 40.</p>	OUT	buf	initial address of receive buffer (choice)	IN	count	number of elements in receive buffer (integer)	IN	datatype	datatype of each receive buffer element (handle)	IN	source	rank of source (integer)	IN	tag	message tag (integer)	IN	comm	communicator (handle)	OUT	request	communication request (handle)
OUT	buf	initial address of receive buffer (choice)																				
IN	count	number of elements in receive buffer (integer)																				
IN	datatype	datatype of each receive buffer element (handle)																				
IN	source	rank of source (integer)																				
IN	tag	message tag (integer)																				
IN	comm	communicator (handle)																				
OUT	request	communication request (handle)																				

Asserted Claim	Prior Art Disclosure						
	<p>MPI_WAIT(request, status)</p> <table><tr><td>INOUT</td><td>request</td><td>request (handle)</td></tr><tr><td>OUT</td><td>status</td><td>status object (Status)</td></tr></table> <p>int MPI_Wait(MPI_Request *request, MPI_Status *status)</p> <p>MPI_WAIT(REQUEST, STATUS, IERROR)</p> <p>INTEGER REQUEST, STATUS(MPI_STATUS_SIZE), IERROR</p> <p>A call to MPI_WAIT returns when the operation identified by request is complete. If the communication object associated with this request was created by a nonblocking send or receive call, then the object is deallocated by the call to MPI_WAIT and the request handle is set to MPI_REQUEST_NULL. MPI_WAIT is a non-local operation.</p> <p>MPI Specification at 41.</p> <p>The PVM library likewise supported similar calls. The following is an example of a PVM asynchronous call <code>pvm_send</code>:</p> <p>The <code>pvm_send</code> routine is asynchronous. Computation on the sending processor resumes as soon as the message is safely on its way to the receiving processor. This is in contrast to synchronous communication, during which computation on the sending processor halts until the matching receive is executed by the receiving processor.</p> <p>Geist at 252.</p> <p>To the extent ACS contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of MultiMatlab and the knowledge of a POSITA, it would have been obvious to combine it with one or more of the other references in these invalidity contentions. For example, MultiMatlab and Matlab*P publications reference MPI and PVM-based parallel processing systems as comparable cluster computing systems, motivating a POSITA to look to such systems for modifications relating to this claim element. <i>See, e.g., Choy 2 at 334.</i> Accordingly, it would have been obvious to modify MultiMatlab in view of the teachings of PVM and MPI publications, such as those referenced in these contentions, as well as references and systems</p>	INOUT	request	request (handle)	OUT	status	status object (Status)
INOUT	request	request (handle)					
OUT	status	status object (Status)					

Asserted Claim	Prior Art Disclosure
	<p>implementing aspects of PVM or MPI. As another example, because MultiMatlab is directed to cluster computing, it would have been obvious to combine with one or more of the prior art references or systems authored by the alleged inventors of the patents in suit or their collaborators, including patents, applications, articles and code by Messrs. Dager, Tannenbaum and Decyk. These references and code, including those related to the Appleseed cluster, Pooch, MacMPI and related projects, are likewise directed to cluster computing using an existing message passing library and a cluster management tool. Moreover, these prior art publications teach peer-to-peer communications between cluster nodes, and use of these cluster computing tools with CAS kernels. Given the similarity of their teaching, it would have been obvious to combine one of more of these systems or publications with MultiMatlab.</p> <p>In addition, this claim is rendered obvious by Intel Paragon Maple in view of Cramer, Choy, FFTW, Howard, and/or Chu for the reasons stated in the Cover Pleading, section IV.</p>
26.f. 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;	The MultiMatlab References disclose “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.” <i>See</i> 1.c.
26.g. wherein the plurality of nodes comprises:	The MultiMatlab References disclose “wherein the plurality of nodes comprises” for the reasons stated below:
26.h. a first node	The MultiMatlab References disclose “a first node comprising a first hardware processor configured to access

Asserted Claim	Prior Art Disclosure
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,	a first memory comprising program code for a user interface and program code for a first single-node kernel.” <i>See</i> 1.c., 1.f.
26.i. 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	The MultiMatlab References disclose “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.” <i>See</i> 1.g.
26.j. a second node comprising a second hardware processor with a plurality of processing cores,	The MultiMatlab References disclose “a second node comprising a second hardware processor with a plurality of processing cores.” <i>See</i> 1.h.
26.k. wherein the second node is configured to receive calls from the first node, execute at least a	The MultiMatlab References disclose “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.” <i>See</i> 1.i.

Asserted Claim	Prior Art Disclosure
first mathematical expression evaluation, and communicate a result of mathematical expression evaluation to a third node;	
26.l. wherein the third node comprises a third hardware processor with a plurality of processing cores,	The MultiMatlab References disclose “wherein the third node comprises a third hardware processor with a plurality of processing cores.” <i>See</i> 1.j.
26.m. 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	The MultiMatlab References disclose “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.” <i>See</i> 1.k.

Asserted Claim	Prior Art Disclosure
expression evaluation to the first node;	
26.n. wherein the first node is configured to return the result of the second mathematical expression evaluation to the user interface or the script;	The MultiMatlab References disclose “wherein the first node is configured to return the result of the second mathematical expression evaluation to the user interface or the script.” <i>See</i> 1.1.
26.o. wherein one or more of the nodes are configured to:	The MultiMatlab References disclose “wherein one or more of the nodes are configured to” for the reasons stated below:
26.p. accept user instructions;	The MultiMatlab References disclose “accept user instructions.” <i>See</i> 1.n.
26.q. after accepting user instructions, communicate at least some of the user instructions using the mechanism for the nodes to communicate with each other; and	The MultiMatlab References disclose “after accepting user instructions, communicate at least some of the user instructions using the mechanism for the nodes to communicate with each other.” <i>See</i> 1.o.
26.r. after communicating at least some of the	The MultiMatlab References disclose “after communicating at least some of the user instructions using the mechanism, communicate at least some of the user instructions to one or more single-node kernels.” <i>See</i> 1.p.

Asserted Claim	Prior Art Disclosure
user instructions using the mechanism, communicate at least some of the user instructions to one or more single-node kernels.	
27.pre. The computer cluster of claim 26, wherein the asynchronous calls comprise a first command to create a first packet containing:	<p>The MultiMatlab References disclose “The computer cluster of claim 26, wherein the asynchronous calls comprise a first command to create a first packet containing.” <i>See</i> claims 1, 26.</p> <p>As established for claim 26, the MultiMatlab References disclose a number of asynchronous calls, including calls to send and receive payloads comprising expressions, such as computation results:</p>

Asserted Claim	Prior Art Disclosure
	<pre> if ID==0 % first process: send a = 1 Send(ID+1,a) elseif ID == Nproc-1 % last process: receive and double a = 2*Recv else % middle processes: receive, double, and send a = 2*Recv Send(ID+1,a) end; </pre> <p>Process 0 creates the variable <code>a</code> with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of <code>a</code>, and sends it along to process 2; and so on. If there are six processors the command <code>Eval('cycle')</code> produces the output</p> <pre> a = 1 a = 2 a = 4 a = 8 a = 16 a = 32 </pre> <p>The processes run asynchronously, but since each <code>send</code> command is only executed after the corresponding <code>Recv</code> has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning.</p> <p>Trefethen 1996 at 7.</p>

Asserted Claim	Prior Art Disclosure
	<p>2.3. Send, Recv, Probe, Barrier</p> <p>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</p> <pre> x = [pi pi^2]; Send(3,x) Eval(3, 'Recv') </pre> <p>passes a message containing a 2-vector from the master process to process 3, leading to the output</p> <pre> 3.1416 9.8696 </pre> <p>An optional argument can be added in <code>Recv</code> to specify the source. Another optional argument may be added in both <code>Send</code> and <code>Recv</code> to specify a message tag so as to ensure that sends and receives are properly matched and to aid in error checking. The command</p> <pre> Probe, </pre> <p>run on any process, again with optional source process number and message tag, returns 1 (true) if a message has arrived from the indicated source with the indicated tag, otherwise 0 (false).</p> <p>Trefethen 1996 at 6.</p> <p>MultiMatlab implemented MPI message passing that allowed any node to pass computation results to any other node, as further illustrated by the following code:</p>

Asserted Claim	Prior Art Disclosure
	<pre> if ID==0 % first process: send a = 1 Send(ID+1,a) elseif ID == Nproc-1 % last process: receive and double a = 0*Recv else % middle processes: receive, double, and send a = 2*Recv Send(ID+1,a) end; </pre> <p>Process 0 creates the variable <i>a</i> with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of <i>a</i>, and sends it along to process 2; and so on. If there are six processors the command <code>Eval('cycle')</code> produces the output</p> <pre> a = 1 a = 2 a = 4 a = 8 a = 16 a = 32 </pre> <p>The processes run asynchronously, but since each <code>Send</code> command is only executed after the corresponding <code>Recv</code> has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning.</p> <p>Trefethen 1996 at 7.</p> <p>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 <i>A</i> 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 <i>rho</i> and <i>alpha</i> and the matrix-vector multiply needed to compute <i>w_{local}</i>. The dot products only require communication of a single value over all processes. On the other hand, the matrix-vector multiplication needed for <i>w_{local}</i> requires the global vector <i>p</i>, and, thus, more expensive communication.</p>

Asserted Claim	Prior Art Disclosure
	<p>Trefethen 1997 at 11, 13.</p> <pre> r_local = b_local; rho = Sum(r_local'*r_local); k = 0; while((k < kmax)) k = k+1; if (k == 1) p_local = r_local; else beta = rho/oldrho; p_local = r_local + beta*p_local; end p = Gather(p_local); w_local = A_local * p; alpha = rho/(Sum(p_local'*w_local)); x_local = x_local + alpha * p_local; r_local = r_local - alpha * w_local; oldrho = rho; rho = Sum(r_local'*r_local); end </pre> <p>b. Message Passing MATLAB</p> <p>Trefethen 1997 at 11.</p>

Asserted Claim	Prior Art Disclosure
	<pre>function result = simpletest(X) import mpi.*; MPIinit; if MPI.COMM_WORLD.rank == 0; data = [4 2]; MPI.COMM_WORLD.send(data, 2, 1, 13); result = 9; elseif MPI.COMM_WORLD.rank == 1 got = MPI.COMM_WORLD.recv(2, 0, 13); result = got(1) + got(2); end MPI.fnlz;</pre> <p>Choy 3 at 49.</p> <p>The following additional examples show asynchronous calls sending and receiving payloads comprising computation results and other data and instructions:</p>

Asserted Claim	Prior Art Disclosure
	<p>We present MATLAB *P [15] as an example of what could be a “right” parallel MATLAB. MATLAB * P is a parallel MATLAB using the back-end support approach, aimed at widespread circulation among a general audience. In order to achieve this, we took ideas from the approaches used by other software found in the survey. For example, the <i>embarrassingly parallel</i> approach allow simple, yet useful, division of work into multiple MATLAB sessions. The <i>message passing</i> approach, which is a superset of the embarrassingly parallel approach, allows finer control between the MATLAB sessions.</p> <p>The main idea in MATLAB *P is that data exist on the parallel server as distributed matrices. Any operations on a distributed matrix (which exists in the MATLAB front end only as a handle) will be relayed to the server transparently. The server calls the appropriate routine from a parallel numerical library (e.g. ScaLAPACK, FFTW, . . .) and the results stay on the server until explicitly requested.</p> <p>Choy 2 at 336.</p>

Asserted Claim	Prior Art Disclosure
	<p>Server software 105 may operate in any suitable way to receive commands from the client and perform operations in parallel, as the invention is not limited in this respect. FIG. 2 is but one example of a suitable configuration of server software 105 for at least some embodiments. In FIG. 2, parallel server 201 includes a plurality of processors, 203a, 203b, . . . , 203n, and a memory 213. Each of processors 203a-203n includes four software modules: client managers 205a, 205b, . . . , 205n; server managers 207a, 207b, . . . , 207n; library managers 209a, 209b, . . . , 209n; and data managers 211a, 211b, . . . , 211n. Memory 213 of parallel server 201 stores a plurality of software libraries 215a, 215b, 215c, . . . , 215n that include routines for performing various operations.</p> <p>Client managers 205 interface with clients, and provides routines for reading commands and arguments from a client and sending results and other data back to the client. Server managers 207 handle communications between server processes executing on different processors of parallel server 201. That is, server managers 207 manage the transmission of data between the processors of parallel server 201 and collect results and error codes from the processors. Library managers</p> <p>Husbands at 10:46-67.</p> <p>The following additional example illustrates an FFT calculation on a Matlab *P cluster, resulting in communications of intermediate results between the nodes in the cluster:</p>

Asserted Claim	Prior Art Disclosure
	<div data-bbox="991 332 1347 414" data-label="Section-Header"> <h2>Example 3</h2> </div> <div data-bbox="636 487 1347 950" data-label="Text"> <pre>% FFT2 in 4 lines • A = randn(10000,10000*p); • B = mm('fft',A); • C = B.>'; • D = mm('fft',C); • F = D.>';</pre> </div> <div data-bbox="571 982 743 1015" data-label="Text"> <p>Choy 3 at 47.</p> </div> <div data-bbox="571 1047 1854 1339" data-label="Text"> <p>To the extent ACS contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of MultiMatlab and the knowledge of a POSITA, it would have been obvious to combine it with one or more of the other references in these invalidity contentions. For example, MultiMatlab and Matlab*P publications reference MPI and PVM-based parallel processing systems as comparable cluster computing systems, motivating a POSITA to look to such systems for modifications relating to this claim element. <i>See, e.g.,</i> Choy 2 at 334. Accordingly, it would have been obvious to modify MultiMatlab in view of the teachings of PVM and MPI publications, such as those referenced in these contentions, as well as references and systems implementing aspects of PVM or MPI. As another example, because MultiMatlab is directed to cluster computing, it would have been obvious to combine with one or more of the prior art references or systems</p> </div>

Asserted Claim	Prior Art Disclosure
	<p>authored by the alleged inventors of the patents in suit or their collaborators, including patents, applications, articles and code by Messrs. Dager, Tannenbaum and Decyk. These references and code, including those related to the Appleseed cluster, Pooch, MacMPI and related projects, are likewise directed to cluster computing using an existing message passing library and a cluster management tool. Moreover, these prior art publications teach peer-to-peer communications between cluster nodes, and use of these cluster computing tools with CAS kernels. Given the similarity of their teaching, it would have been obvious to combine one of more of these systems or publications with MultiMatlab.</p> <p>In addition, this claim is rendered obvious by Intel Paragon Maple in view of Cramer, Choy, FFTW, Howard, and/or Chu for the reasons stated in the Cover Pleading, section IV.</p>
27.a. an expression to be sent as payload; and	The MultiMatlab References disclose “an expression to be sent as payload.” <i>See</i> 27[pre].
27.b. a target node where the expression should be sent;	<p>The MultiMatlab References disclose “a target node where the expression should be sent.”</p> <p>As established for claim 26, MultiMatlab implemented message passing protocols such as MPI and PVM, and/or remote call procedures that allowed a node to pass computation results in a message to another node. <i>See</i> claims 1, 26, 27[pre]. The calls implementing such protocols, for example, send and receive calls, comprised a target node identifier.</p> <p>For example, send and receive calls (e.g., the calls below) included a process ID specifying the target node:</p>

Asserted Claim	Prior Art Disclosure
	<p>2.2. Put, Get</p> <p>So far, we have not communicated between processes except to send screen output to the master process. Of course, a nontrivial MultiMATLAB system depends on such communication.</p> <p>One form of communication we have implemented is puts and gets, executable solely by the master MATLAB process. For example, the command</p> <pre>Put(1:4, 'A');</pre> <p>sends the matrix A from the master process 0 to processes 1 through 4; an optional argument permits the name of A to be changed at the destination. The command</p> <pre>Get(3, 'B');</pre> <p>gets the matrix B back from process 3 to the master.</p> <p>2.3. Send, Recv, Probe, Barrier</p> <p>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</p> <pre>x = [pi pi^2]; Send(3,x) Eval(3, 'Recv')</pre> <p>passes a message containing a 2-vector from the master process to process 3, leading to the output</p> <pre>3.1416 9.8696</pre> <p>An optional argument can be added in <code>Recv</code> to specify the source. Another optional argument may be added in both <code>Send</code> and <code>Recv</code> to specify a message tag so as to ensure that sends and receives are properly matched and to aid in error checking. The command</p> <p>Trefethen 1996 at 8.</p> <p>tem 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</p>

Asserted Claim	Prior Art Disclosure
	<p>Trefethen 1997 at 4.</p> <p><u>Additional Evidence of Obviousness</u></p> <pre> function result = simpletest(X) import mpi.*; MPI.init; if MPI.COMM_WORLD.rank == 0; data = [4 2]; MPI.COMM_WORLD.send(data, 2, 1, 13); result = 9; elseif MPI.COMM_WORLD.rank == 1 got = MPI.COMM_WORLD.recv(2, 0, 13); result = got(1) + got(2); end MPI.fnlz; </pre> <p>Choy 3 at 49.</p>

Asserted Claim	Prior Art Disclosure
	<p data-bbox="620 315 929 345">3.2.3 Message envelope</p> <p data-bbox="620 367 1711 464">In addition to the data part, messages carry information that can be used to distinguish messages and selectively receive them. This information consists of a fixed number of fields, which we collectively call the message envelope. These fields are</p> <div data-bbox="1078 483 1252 607" style="margin-left: 100px;"> <p>source</p> <p>destination</p> <p>tag</p> <p>communicator</p> </div> <p data-bbox="620 626 1711 688">The message source is implicitly determined by the identity of the message sender. The other fields are specified by arguments in the send operation.</p> <p data-bbox="676 695 1381 724">The message destination is specified by the dest argument.</p> <p data-bbox="579 764 858 794">MPI Specification at 19.</p> <p data-bbox="579 829 1862 1339">To the extent ACS contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of MultiMatlab and the knowledge of a POSITA, it would have been obvious to combine it with one or more of the other references in these invalidity contentions. For example, MultiMatlab and Matlab*P publications reference MPI and PVM-based parallel processing systems as comparable cluster computing systems, motivating a POSITA to look to such systems for modifications relating to this claim element. <i>See, e.g., Choy 2 at 334.</i> Accordingly, it would have been obvious to modify MultiMatlab in view of the teachings of PVM and MPI publications, such as those referenced in these contentions, as well as references and systems implementing aspects of PVM or MPI. As another example, because MultiMatlab is directed to cluster computing, it would have been obvious to combine with one or more of the prior art references or systems authored by the alleged inventors of the patents in suit or their collaborators, including patents, applications, articles and code by Messrs. Dager, Tannenbaum and Decyk. These references and code, including those related to the Appleseed cluster, Pooch, MacMPI and related projects, are likewise directed to cluster computing using an existing message passing library and a cluster management tool. Moreover, these prior art publications teach peer-to-peer communications between cluster nodes, and use of these cluster computing tools with CAS kernels. Given the similarity of their teaching, it would have been obvious to combine one of more of these systems or publications with MultiMatlab.</p>

Asserted Claim	Prior Art Disclosure
	In addition, this claim is rendered obvious by Intel Paragon Maple in view of Cramer, Choy, FFTW, Howard, and/or Chu for the reasons stated in the Cover Pleading, section IV.
27.c. wherein the first command is configured to be called from within a single-node kernel;	<p>The MultiMatlab References disclose “wherein the first command is configured to be called from within a single-node kernel.”</p> <p>MultiMatlab commands, such as the send and receive commands, can be entered by a user into a user interface of a CAS kernel, or be part of a CAS kernel language script running within the single-node kernel:</p>

Asserted Claim	Prior Art Disclosure
	<p>2.2. Put, Get</p> <p>So far, we have not communicated between processes except to send screen output to the master process. Of course, a nontrivial MultiMATLAB system depends on such communication.</p> <p>One form of communication we have implemented is puts and gets, executable solely by the master MATLAB process. For example, the command</p> <pre>Put(1:4, 'A');</pre> <p>sends the matrix A from the master process 0 to processes 1 through 4; an optional argument permits the name of A to be changed at the destination. The command</p> <pre>Get(3, 'B');</pre> <p>gets the matrix B back from process 3 to the master.</p> <p>2.3. Send, Recv, Probe, Barrier</p> <p>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</p> <pre>x = [pi pi^2]; Send(3,x) Eval(3, 'Recv');</pre> <p>passes a message containing a 2-vector from the master process to process 3, leading to the output</p> <pre>3.1416 9.8696</pre> <p>An optional argument can be added in <code>Recv</code> to specify the source. Another optional argument may be added in both <code>Send</code> and <code>Recv</code> to specify a message tag so as to ensure that sends and receives are properly matched and to aid in error checking. The command</p> <p>Trefethen 1996 at 8.</p>

Asserted Claim	Prior Art Disclosure
	<pre> if ID==0 % first process: send a = 1 Send(ID+1,a) elseif ID == Nproc-1 % last process: receive and double a = 2*Recv else % middle processes: receive, double, and send a = 2*Recv Send(ID+1,a) end; </pre> <p>Process 0 creates the variable <code>a</code> with value 1 and sends it to process 1. Process 1 receives the message, doubles the value of <code>a</code>, and sends it along to process 2; and so on. If there are six processors the command <code>Eval('cycle')</code> produces the output</p> <pre> a = 1 a = 2 a = 4 a = 8 a = 16 a = 32 </pre> <p>The processes run asynchronously, but since each <code>Send</code> command is only executed after the corresponding <code>Recv</code> has completed, the proper sequence of computations and final value 32 are guaranteed so long as all of the nodes are functioning.</p> <p>Trefethen 1996 at 7.</p> <p><u>Additional Evidence of Obviousness</u></p>

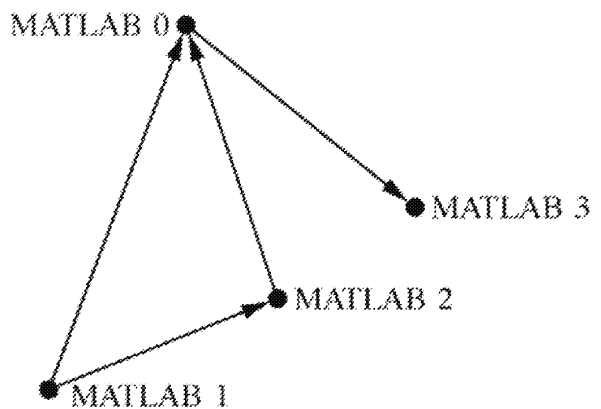
Asserted Claim	Prior Art Disclosure
	<pre>function result = simpletest(X) import mpi.*; MPIinit; if MPI.COMM_WORLD.rank == 0; data = [4 2]; MPI.COMM_WORLD.send(data, 2, 1, 13); result = 9; elseif MPI.COMM_WORLD.rank == 1 got = MPI.COMM_WORLD.recv(2, 0, 13); result = got(1) + got(2); end MPI.fnlz;</pre> <p>Choy 3 at 49.</p>

Asserted Claim	Prior Art Disclosure
	<p style="text-align: center;">Example 3</p> <p><code>% FFT2 in 4 lines</code></p> <ul style="list-style-type: none"> • <code>A = randn(10000,10000*p);</code> • <code>B = mm('fft',A);</code> • <code>C = B.{'</code> • <code>D = mm('fft',C);</code> • <code>F = D.{'</code> <p>Choy 3 at 47.</p> <p>To the extent ACS contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of MultiMatlab and the knowledge of a POSITA, it would have been obvious to combine it with one or more of the other references in these invalidity contentions. For example, MultiMatlab and Matlab*P publications reference MPI and PVM-based parallel processing systems as comparable cluster computing systems, motivating a POSITA to look to such systems for modifications relating to this claim element. <i>See, e.g.,</i> Choy 2 at 334. Accordingly, it would have been obvious to modify MultiMatlab in view of the teachings of PVM and MPI publications, such as those referenced in these contentions, as well as references and systems implementing aspects of PVM or MPI. As another example, because MultiMatlab is directed to cluster computing, it would have been obvious to combine with one or more of the prior art references or systems</p>

Asserted Claim	Prior Art Disclosure
	<p>authored by the alleged inventors of the patents in suit or their collaborators, including patents, applications, articles and code by Messrs. Dager, Tannenbaum and Decyk. These references and code, including those related to the Appleseed cluster, Pooch, MacMPI and related projects, are likewise directed to cluster computing using an existing message passing library and a cluster management tool. Moreover, these prior art publications teach peer-to-peer communications between cluster nodes, and use of these cluster computing tools with CAS kernels. Given the similarity of their teaching, it would have been obvious to combine one of more of these systems or publications with MultiMatlab.</p> <p>In addition, this claim is rendered obvious by Intel Paragon Maple in view of Cramer, Choy, FFTW, Howard, and/or Chu for the reasons stated in the Cover Pleading, section IV.</p>
<p>27.d. wherein the single-node kernel is configured to send the first packet to a local cluster node module; and</p>	<p>The MultiMatlab References disclose “wherein the single-node kernel is configured to send the first packet to a local cluster node module.”</p> <p>MultiMatlab commands, such as the send and receive commands, are sent to the other nodes via the communication interface components, one or both of which comprise the local “cluster node module:”</p> <p>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.</p> <p>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.</p> <p>Trefethen 1997 at 3.</p>

Asserted Claim	Prior Art Disclosure
	<div><p>The diagram illustrates the MultiMATLAB Architecture. At the top, a 'User' is connected via a bidirectional arrow to an 'Interactive Processor'. This 'Interactive Processor' contains a 'MATLAB Process'. Below it, an 'Interconnecting Network' connects to three 'Processor' blocks. Each 'Processor' block contains a 'MATLAB Process'. The 'Interactive Processor' and the three 'Processor' blocks are all connected to the 'Interconnecting Network'.</p><p>Figure 1: MultiMATLAB Architecture</p><p>Trefethen 1997 at 3.</p></div>

Asserted Claim	Prior Art Disclosure
	<div><pre>graph TD; MATLAB[MATLAB] <--> SeqMEX[Sequential MEX Routine]; MATLAB <--> ParMEX[Parallel MEX Routine]; SeqMEX <--> IM[MultiMATLAB Interface Module]; ParMEX <--> IM; IM <--> IT[Indirection Table]; IT <--> CL[Communication Layer]; CL <--> Network[Network];</pre></div> <p>Trefethen 1997 at 4.</p> <p><u>Additional Evidence of Obviousness</u></p>

Asserted Claim	Prior Art Disclosure
	 <p>The diagram shows four nodes labeled MATLAB 0, MATLAB 1, MATLAB 2, and MATLAB 3. MATLAB 0 is at the top, MATLAB 1 is at the bottom left, MATLAB 2 is at the bottom right, and MATLAB 3 is to the right of MATLAB 2. Arrows indicate message passing: MATLAB 1 points to MATLAB 0, MATLAB 0 points to MATLAB 2, MATLAB 2 points to MATLAB 3, and MATLAB 3 points back to MATLAB 0.</p> <p>Fig. 2. Message Passing.</p> <p>Choy 2 at 333.</p>

Asserted Claim	Prior Art Disclosure
	<p><i>B. MultiMATLAB/MultiOctave Mode</i></p> <p>One of the goals of the project is to make the software to be useful to as wide an audience as possible. In order to achieve this, we found that it would be fruitful to combine other parallel MATLAB approaches into MATLAB * P, to provide a unified parallel MATLAB framework.</p> <p>In conjunction with P. Husbands, we developed a prototype implementation of a MultiMATLAB [35]-like, distributed MATLAB package in MATLAB * P, which we call the PPEngine. With this package and associated m-files, we can run multiple MATLAB processes on the back end and evaluate MATLAB functions in parallel on dense matrices.</p> <p>The system works by starting up MATLAB engine instances on each node through calls to the MATLAB engine interface. From that point on, MATLAB commands can be relayed to the MATLAB engine. The command is do this is <i>mm</i>.</p> <p>Choy 2 at 336</p>

Asserted Claim	Prior Art Disclosure
	<div><p>The diagram illustrates the architecture of MATLAB*P 2.0. A central dashed box represents 'Server #0'. Inside this box, at the top, are three boxes labeled 'PBLAS', 'PFTW', and 'ScaLAPACK', each with a line connecting to a 'Package Manager' box below them. To the left of the 'Package Manager' are three boxes labeled 'Matrix 1', 'Matrix 2', and 'Matrix 3', each with a line connecting to a 'Matrix Manager' box. To the right of the 'Matrix Manager' is a 'Server Manager' box. Above the 'Server Manager' is a 'Client Manager' box. Below the 'Server Manager' are four boxes labeled 'Server #1', 'Server #2', 'Server #3', and 'Server #4'. Lines connect the 'Server Manager' to each of these four server boxes. To the right of the 'Server #0' box, outside the dashed box, are two boxes: 'Proxy' and 'Client'. A line connects the 'Client Manager' to the 'Proxy' box, and another line connects the 'Client Manager' to the 'Client' box. The 'Proxy' box is connected to the 'Client' box by a vertical line.</p></div> <p>Fig. 1. Structure of MATLAB*P 2.0</p> <p>Choy 1 at 2.</p>

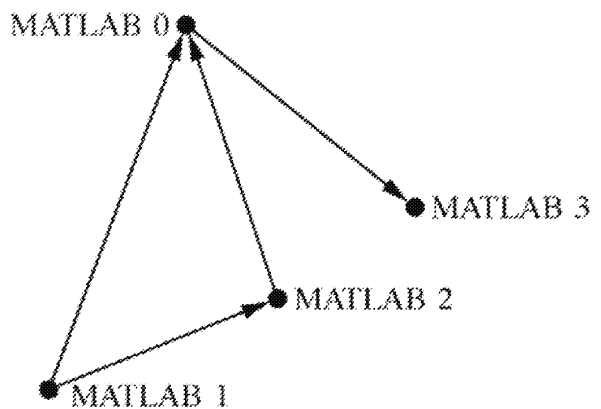
Asserted Claim	Prior Art Disclosure
	<p>Server software 105 may operate in any suitable way to receive commands from the client and perform operations in parallel, as the invention is not limited in this respect. FIG. 2 is but one example of a suitable configuration of server software 105 for at least some embodiments. In FIG. 2, parallel server 201 includes a plurality of processors, 203a, 203b, . . . , 203n, and a memory 213. Each of processors 203a-203n includes four software modules: client managers 205a, 205b, . . . , 205n; server managers 207a, 207b, . . . , 207n; library managers 209a, 209b, . . . , 209n; and data managers 211a, 211b, . . . , 211n. Memory 213 of parallel server 201 stores a plurality of software libraries 215a, 215b, 215c, . . . , 215n that include routines for performing various operations.</p> <p>Client managers 205 interface with clients, and provides routines for reading commands and arguments from a client and sending results and other data back to the client. Server managers 207 handle communications between server processes executing on different processors of parallel server 201. That is, server managers 207 manage the transmission of data between the processors of parallel server 201 and collect results and error codes from the processors. Library managers</p> <p>Husbands at 10:46-67.</p>

Asserted Claim	Prior Art Disclosure
	<p>In one embodiment, when server software 105 is initialized, one processor may be designated the head processor and all or some of the remaining processors may be designated slave processors that operate under the control of the head processor. The head processor may serve as the central processor and may receive the commands from the client. Thus, in one embodiment, only the client manager module 205 on the head processor is used. When the head processor receives a command from the client, the server manager 207 on the head processor may send messages to the “slave” processors to perform the desired operation. The library managers 209 on each of the processors may call the appropriate routine in libraries 215.</p> <p>The server managers 207 on the processors 203 may communicate with each other in any suitable way, as the invention is not limited in this respect. In one embodiment, the message passage interface (MPI) application programming interface (API) may be used. Alternatively, the parallel virtual machine (PVM) API or any other suitable form of communication may be used.</p> <p>Husbands at 11:10-30.</p> <p>To the extent ACS contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of MultiMatlab and the knowledge of a POSITA, it would have been obvious to combine it with one or more of the other references in these invalidity contentions. For example, MultiMatlab and Matlab*P publications reference MPI and PVM-based parallel processing systems as comparable cluster computing systems, motivating a POSITA to look to such systems for modifications relating to this claim element. <i>See, e.g.,</i> Choy 2 at 334. Accordingly, it would have been obvious to modify MultiMatlab in view of the teachings of PVM and MPI publications, such as those referenced in these contentions, as well as references and systems implementing aspects of PVM or MPI. As another example, because MultiMatlab is directed to cluster computing, it would have been obvious to combine with one or more of the prior art references or systems</p>

Asserted Claim	Prior Art Disclosure
	<p>authored by the alleged inventors of the patents in suit or their collaborators, including patents, applications, articles and code by Messrs. Dauger, Tannenbaum and Decyk. These references and code, including those related to the Appleseed cluster, Pooch, MacMPI and related projects, are likewise directed to cluster computing using an existing message passing library and a cluster management tool. Moreover, these prior art publications teach peer-to-peer communications between cluster nodes, and use of these cluster computing tools with CAS kernels. Given the similarity of their teaching, it would have been obvious to combine one of more of these systems or publications with MultiMatlab.</p> <p>In addition, this claim is rendered obvious by Intel Paragon Maple in view of Cramer, Choy, FFTW, Howard, and/or Chu for the reasons stated in the Cover Pleading, section IV.</p>
<p>27.e. wherein the local cluster node module is configured to forward the expression to the target node.</p>	<p>The MultiMatlab References disclose “wherein the local cluster node module is configured to forward the expression to the target node.”</p> <p>MultiMatlab cluster node module components forward the packet, including the expression to be evaluated, to one or more target nodes:</p> <p>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.</p> <p>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.</p> <p>Trefethen 1997 at 3.</p>

Asserted Claim	Prior Art Disclosure
	<div><p>The diagram illustrates the MultiMATLAB Architecture. At the top, a 'User' is connected via a bidirectional arrow to an 'Interactive Processor'. This 'Interactive Processor' contains a 'MATLAB Process'. Below it, an 'Interconnecting Network' connects to three 'Processor' blocks. Each 'Processor' block contains a 'MATLAB Process'. The network is represented by a horizontal line with vertical connectors to each processor block.</p></div> <p>Figure 1: MultiMATLAB Architecture</p> <p>Trefethen 1997 at 3.</p>

Asserted Claim	Prior Art Disclosure
	<div><pre>graph TD MATLAB[MATLAB] <--> SeqMEX[Sequential MEX Routine] MATLAB <--> ParMEX[Parallel MEX Routine] MATLAB <--> IntMod[MultiMATLAB Interface Module] SeqMEX <--> IntMod ParMEX <--> IntMod IntMod <--> IndTable[Indirection Table] IndTable <--> CommLayer[Communication Layer] CommLayer <--> Network[Network]</pre><p>Trefethen 1997 at 4.</p><p><u>Additional Evidence of Obviousness</u></p></div>

Asserted Claim	Prior Art Disclosure
	 <p>The diagram shows four nodes labeled MATLAB 0, MATLAB 1, MATLAB 2, and MATLAB 3. MATLAB 0 is at the top, MATLAB 1 is at the bottom left, MATLAB 2 is at the bottom right, and MATLAB 3 is to the right of MATLAB 2. Directed arrows indicate the flow of message passing: from MATLAB 1 to MATLAB 0, from MATLAB 0 to MATLAB 2, from MATLAB 2 to MATLAB 3, and from MATLAB 2 back to MATLAB 0.</p> <p>Fig. 2. Message Passing.</p> <p>Choy 2 at 333.</p>

Asserted Claim	Prior Art Disclosure
	<p><i>B. MultiMATLAB/MultiOctave Mode</i></p> <p>One of the goals of the project is to make the software to be useful to as wide an audience as possible. In order to achieve this, we found that it would be fruitful to combine other parallel MATLAB approaches into MATLAB * P, to provide a unified parallel MATLAB framework.</p> <p>In conjunction with P. Husbands, we developed a prototype implementation of a MultiMATLAB [35]-like, distributed MATLAB package in MATLAB * P, which we call the PPEngine. With this package and associated m-files, we can run multiple MATLAB processes on the back end and evaluate MATLAB functions in parallel on dense matrices.</p> <p>The system works by starting up MATLAB engine instances on each node through calls to the MATLAB engine interface. From that point on, MATLAB commands can be relayed to the MATLAB engine. The command is do this is <i>mm</i>.</p> <p>Choy 2 at 336</p>

Asserted Claim	Prior Art Disclosure
	<div><p>The diagram illustrates the architecture of MATLAB*P 2.0. It features a central 'Server #0' box containing several components: 'PBLAS', 'FFTW', and 'ScaLAPACK' at the top; 'Package Manager' and 'Client Manager' in the middle; 'Matrix 1', 'Matrix 2', and 'Matrix 3' on the left; 'Matrix Manager' and 'Server Manager' on the right. Below 'Server #0' are four boxes labeled 'Server #1', 'Server #2', 'Server #3', and 'Server #4'. Arrows indicate connections: from 'PBLAS', 'FFTW', and 'ScaLAPACK' to 'Package Manager'; from 'Package Manager' to 'Matrix Manager'; from 'Matrix 1', 'Matrix 2', and 'Matrix 3' to 'Matrix Manager'; from 'Client Manager' to 'Server Manager'; from 'Proxy' and 'Client' to 'Client Manager'; and from 'Server Manager' to each of the four lower servers. A 'Proxy' box is also connected to the 'Client Manager'.</p></div> <p>Fig. 1. Structure of MATLAB*P 2.0 Choy 1 at 2.</p>

Asserted Claim	Prior Art Disclosure
	<p>Server software 105 may operate in any suitable way to receive commands from the client and perform operations in parallel, as the invention is not limited in this respect. FIG. 2 is but one example of a suitable configuration of server software 105 for at least some embodiments. In FIG. 2, parallel server 201 includes a plurality of processors, 203a, 203b, . . . , 203n, and a memory 213. Each of processors 203a-203n includes four software modules: client managers 205a, 205b, . . . , 205n; server managers 207a, 207b, . . . , 207n; library managers 209a, 209b, . . . , 209n; and data managers 211a, 211b, . . . , 211n. Memory 213 of parallel server 201 stores a plurality of software libraries 215a, 215b, 215c, . . . , 215n that include routines for performing various operations.</p> <p>Client managers 205 interface with clients, and provides routines for reading commands and arguments from a client and sending results and other data back to the client. Server managers 207 handle communications between server processes executing on different processors of parallel server 201. That is, server managers 207 manage the transmission of data between the processors of parallel server 201 and collect results and error codes from the processors. Library managers</p> <p>Husbands at 10:46-67.</p>

Asserted Claim	Prior Art Disclosure
	<p data-bbox="599 310 1375 735">In one embodiment, when server software 105 is initialized, one processor may be designated the head processor and all or some of the remaining processors may be designated slave processors that operate under the control of the head processor. The head processor may serve as the central processor and may receive the commands from the client. Thus, in one embodiment, only the client manager module 205 on the head processor is used. When the head processor receives a command from the client, the server manager 207 on the head processor may send messages to the “slave” processors to perform the desired operation. The library managers 209 on each of the processors may call the appropriate routine in libraries 215.</p> <p data-bbox="599 748 1375 974">The server managers 207 on the processors 203 may communicate with each other in any suitable way, as the invention is not limited in this respect. In one embodiment, the message passage interface (MPI) application programming interface (API) may be used. Alternatively, the parallel virtual machine (PVM) API or any other suitable form of communication may be used.</p> <p data-bbox="599 987 840 1015">Husbands at 11:10-30.</p> <p data-bbox="599 1057 1854 1339">To the extent ACS contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of MultiMatlab and the knowledge of a POSITA, it would have been obvious to combine it with one or more of the other references in these invalidity contentions. For example, MultiMatlab and Matlab*P publications reference MPI and PVM-based parallel processing systems as comparable cluster computing systems, motivating a POSITA to look to such systems for modifications relating to this claim element. <i>See, e.g.,</i> Choy 2 at 334. Accordingly, it would have been obvious to modify MultiMatlab in view of the teachings of PVM and MPI publications, such as those referenced in these contentions, as well as references and systems implementing aspects of PVM or MPI. As another example, because MultiMatlab is directed to cluster computing, it would have been obvious to combine with one or more of the prior art references or systems</p>

Asserted Claim	Prior Art Disclosure
	<p>authored by the alleged inventors of the patents in suit or their collaborators, including patents, applications, articles and code by Messrs. Dager, Tannenbaum and Decyk. These references and code, including those related to the Appleseed cluster, Pooch, MacMPI and related projects, are likewise directed to cluster computing using an existing message passing library and a cluster management tool. Moreover, these prior art publications teach peer-to-peer communications between cluster nodes, and use of these cluster computing tools with CAS kernels. Given the similarity of their teaching, it would have been obvious to combine one of more of these systems or publications with MultiMatlab.</p> <p>In addition, this claim is rendered obvious by Intel Paragon Maple in view of Cramer, Choy, FFTW, Howard, and/or Chu for the reasons stated in the Cover Pleading, section IV.</p>
29.pre. A computer cluster comprising:	To the extent the preamble is limiting, the MultiMatlab References disclose “a computer cluster comprising.” <i>See</i> 26.pre.
29.a. a plurality of nodes, wherein one or more of the nodes are configured to receive:	The MultiMatlab References disclose “a plurality of nodes, wherein one or more of the nodes are configured to receive.” <i>See</i> 26.a.
29.b. a command to start a cluster initialization process for the computer cluster,	The MultiMatlab References disclose “a command to start a cluster initialization process for the computer cluster.” <i>See</i> 26.b.
29.c. wherein the cluster initialization process comprises establishing communication among two or more of the nodes; and	The MultiMatlab References disclose “wherein the cluster initialization process comprises establishing communication among two or more of the nodes.” <i>See</i> 26.c.

Asserted Claim	Prior Art Disclosure
29.d. an instruction from a user interface or a script; and	The MultiMatlab References disclose “an instruction from a user interface or a script.” <i>See</i> 26.d.
29.e. a mechanism for the nodes to communicate results of mathematical expression evaluation with each other;	The MultiMatlab References disclose “a mechanism for the nodes to communicate results of mathematical expression evaluation with each other.” <i>See</i> 26.e.
29.f. 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;	The MultiMatlab References disclose “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.” <i>See</i> 26.f.
29.g. wherein the plurality of nodes comprises:	The MultiMatlab References disclose “wherein the plurality of nodes comprises.” <i>See</i> 26.g.
29.h. a first node comprising a first	The MultiMatlab References disclose “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.”

Asserted Claim	Prior Art Disclosure
hardware processor configured to access a first memory comprising program code for a user interface and program code for a first single-node kernel,	<i>See</i> 26.h.
29.i. 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	The MultiMatlab References disclose “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.” <i>See</i> 26.i.
29.j. 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,	The MultiMatlab References disclose “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.” <i>See</i> 26.j, 26.k.
29.k. execute at least a first mathematical	The MultiMatlab References disclose “execute at least a first mathematical expression evaluation.” <i>See</i> 26.k.

Asserted Claim	Prior Art Disclosure
expression evaluation,	
29.l. and communicate a result of mathematical expression evaluation to a third node;	The MultiMatlab References disclose “and communicate a result of mathematical expression evaluation to a third node.” <i>See</i> 26.k.
29.m. wherein the third node comprises a third hardware processor with a plurality of processing cores,	The MultiMatlab References disclose “wherein the third node comprises a third hardware processor with a plurality of processing cores.” <i>See</i> 26.l.
29.n. 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	The MultiMatlab References disclose “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.” <i>See</i> 26.m.

Asserted Claim	Prior Art Disclosure
expression evaluation to the first node; and	
29.o. wherein the first node is configured to return the result of the second mathematical expression evaluation to the user interface or the script;	The MultiMatlab References disclose “wherein the first node is configured to return the result of the second mathematical expression evaluation to the user interface or the script.” <i>See</i> 26.n.
29.p. wherein one or more of the nodes are configured to:	The MultiMatlab References disclose “wherein one or more of the nodes are configured to.” <i>See</i> 26.o.
29.q. accept user instructions;	The MultiMatlab References disclose “accept user instructions.” <i>See</i> 26.p.
29.r. after accepting user instructions, communicate at least some of the user instructions using the mechanism for the nodes to communicate with each other; and	The MultiMatlab References disclose “after accepting user instructions, communicate at least some of the user instructions using the mechanism for the nodes to communicate with each other.” <i>See</i> 26.q.
29.s. after communicating at least some of the	The MultiMatlab References disclose “after communicating at least some of the user instructions using the mechanism, communicate at least some of the user instructions to one or more single-node kernels.” <i>See</i> 26.r.

Asserted Claim	Prior Art Disclosure
user instructions using the mechanism, communicate at least some of the user instructions to one or more single-node kernels.	
30.pre. The computer cluster of claim 4,	The MultiMatlab References disclose this claim.
30.a. wherein each of the plurality of nodes implements asynchronous calls that enable the single-node kernel to perform computation tasks while the cluster node modules are simultaneously communicating with one another.	The MultiMatlab References disclose “wherein each of the plurality of nodes implements asynchronous calls that enable the single-node kernel to perform computation tasks while the cluster node modules are simultaneously communicating with one another.” <i>See</i> claim 27.
31.pre. The computer cluster of claim 4,	The MultiMatlab References disclose this claim.
31.a. wherein intercommunication among the plurality	The MultiMatlab References disclose “wherein intercommunication among the plurality of single-node kernels during thread execution is enabled by the plurality of cluster node modules.” <i>See</i> 1.d.

Asserted Claim	Prior Art Disclosure
of single-node kernels during thread execution is enabled by the plurality of cluster node modules,	
31.b. and wherein the computer cluster is configured to permit exchange of information between nodes during the course of a parallel computation.	<p>The MultiMatlab References disclose “and wherein the computer cluster is configured to permit exchange of information between nodes during the course of a parallel computation.”</p> <p>MultiMatlab cluster operated so as to allow the kernels to compute while sending and receiving messages, e.g., MPI, PVM and/or remote function call communications between the nodes. <i>See</i> claims 1, 26.</p>
33.pre. The computer cluster of claim 1,	The MultiMatlab References disclose this claim.
33.a. wherein the plurality of nodes are configured to permit exchange of information between nodes during the course of parallel computation.	The MultiMatlab References disclose “wherein the plurality of nodes are configured to permit exchange of information between nodes during the course of parallel computation.” <i>See</i> claim 31.
34.pre. The computer cluster of claim 1,	The MultiMatlab References disclose this claim.

Asserted Claim	Prior Art Disclosure
34.a. wherein each of the plurality of nodes comprises instructions executable by the hardware processor and configured to implement asynchronous behavior, wherein the instructions comprise:	The MultiMatlab References disclose “wherein each of the plurality of nodes comprises instructions executable by the hardware processor and configured to implement asynchronous behavior, wherein the instructions comprise.” <i>See</i> claims 26, 27.
34.b. a first instruction to asynchronously send a payload to another node;	The MultiMatlab References disclose “a first instruction to asynchronously send a payload to another node.” The MultiMatlab References disclose asynchronous send instructions, e.g., MPI, PVM and/or remote function call instructions, that send data payloads, such as instructions or results, to other nodes. <i>See</i> claims 26, 27.
34.c. a second instruction to asynchronously receive a payload from another node; and	The MultiMatlab References disclose “a second instruction to asynchronously receive a payload from another node.” The MultiMatlab References disclose asynchronous receiver instructions, e.g., MPI, PVM and/or remote function call instructions, that receive data payloads, such as instructions or results, from other nodes. <i>See</i> claims 26, 27.
34.d. a third instruction to search for a payload matching a message specifier.	The MultiMatlab References disclose “a third instruction to search for a payload matching a message specifier.” The MultiMatlab References disclose the MPI messaging protocol, which includes instructions MPI_Test and/or MPI_Wait that the '768 preferred embodiment identifies as allegedly performing the claimed function. '768, 13:31-14:29. <i>See</i> claims 26, 27.

Asserted Claim	Prior Art Disclosure
	<p>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.</p> <p>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.</p> <p>Trefethen 1997 at 3.</p> <p><u>Additional Evidence of Obviousness</u></p> <p>3.7.3 Communication Completion</p> <p>The functions <code>MPLWAIT</code> and <code>MPLTEST</code> are used to complete a nonblocking communication. The completion of a send operation indicates that the sender is now free to update the locations in the send buffer (the send operation itself leaves the content of the send buffer unchanged). It does not indicate that the message has been received, rather, it may have been buffered by the communication subsystem. However, if a synchronous mode send was used, the completion of the send operation indicates that a matching receive was initiated, and that the message will eventually be received by this matching receive.</p> <p>The completion of a receive operation indicates that the receive buffer contains the received message, the receiver is now free to access it, and that the status object is set. It does not indicate that the matching send operation has completed (but indicates, of course, that the send was initiated).</p>

Asserted Claim	Prior Art Disclosure
	<p data-bbox="584 305 862 329">MPI Specification at 41.</p> <p data-bbox="584 378 1034 402"><i>B. MultiMATLAB/MultiOctave Mode</i></p> <p data-bbox="584 423 1293 586">One of the goals of the project is to make the software to be useful to as wide an audience as possible. In order to achieve this, we found that it would be fruitful to combine other parallel MATLAB approaches into MATLAB * P, to provide a unified parallel MATLAB framework.</p> <p data-bbox="584 594 1293 789">In conjunction with P. Husbands, we developed a prototype implementation of a MultiMATLAB [35]-like, distributed MATLAB package in MATLAB * P, which we call the PPEngine. With this package and associated m-files, we can run multiple MATLAB processes on the back end and evaluate MATLAB functions in parallel on dense matrices.</p> <p data-bbox="584 797 1293 959">The system works by starting up MATLAB engine instances on each node through calls to the MATLAB engine interface. From that point on, MATLAB commands can be relayed to the MATLAB engine. The command is do this is <i>mm</i>.</p> <p data-bbox="584 967 743 992">Choy 2 at 336</p> <p data-bbox="584 1032 1858 1351">To the extent ACS contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of MultiMatlab and the knowledge of a POSITA, it would have been obvious to combine it with one or more of the other references in these invalidity contentions. For example, MultiMatlab and Matlab*P publications reference MPI and PVM-based parallel processing systems as comparable cluster computing systems, motivating a POSITA to look to such systems for modifications relating to this claim element. <i>See, e.g.,</i> Choy 2 at 334. Accordingly, it would have been obvious to modify MultiMatlab in view of the teachings of PVM and MPI publications, such as those referenced in these contentions, as well as references and systems implementing aspects of PVM or MPI. As another example, because MultiMatlab is directed to cluster computing, it would have been obvious to combine with one or more of the prior art references or systems authored by the alleged inventors of the patents in suit or their collaborators, including patents, applications,</p>

Asserted Claim	Prior Art Disclosure
	<p>articles and code by Messrs. Dager, Tannenbaum and Decyk. These references and code, including those related to the Appleseed cluster, Pooch, MacMPI and related projects, are likewise directed to cluster computing using an existing message passing library and a cluster management tool. Moreover, these prior art publications teach peer-to-peer communications between cluster nodes, and use of these cluster computing tools with CAS kernels. Given the similarity of their teaching, it would have been obvious to combine one of more of these systems or publications with MultiMatlab.</p> <p>In addition, this claim is rendered obvious by Intel Paragon Maple in view of Cramer, Choy, FFTW, Howard, and/or Chu for the reasons stated in the Cover Pleading, section IV.</p>
35.pre. A computer cluster node for evaluating expressions in parallel with other computer cluster nodes, the computer cluster node comprising:	To the extent the preamble is limiting, the MultiMatlab References disclose “a computer cluster node for evaluating expressions in parallel with other computer cluster nodes, the computer cluster node comprising.” <i>See</i> claims 1, 26.
35.a. a hardware processor configured to access one or more non-transitory memory devices comprising program code for a single-node kernel that, when executed, causes the hardware processor to interpret user	The MultiMatlab References disclose “a hardware processor configured to access one or more non-transitory memory devices comprising program code for a single-node kernel that, when executed, 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.” <i>See</i> 1.h., 1.g., 1.h., 1.i.

Asserted Claim	Prior Art Disclosure
instructions, to evaluate mathematical expressions, and to produce results of mathematical expression evaluation, wherein the hardware processor comprises multiple processor cores;	
35.b. a user connection interface configured to receive a command to start a cluster initialization process for a computer cluster;	The MultiMatlab References disclose “a user connection interface configured to receive a command to start a cluster initialization process for a computer cluster.” <i>See</i> 1.b.
35.c. a mechanism to communicate results of evaluation with other computer cluster nodes using a peer-to-peer architecture; and	The MultiMatlab References disclose “a mechanism to communicate results of evaluation with other computer cluster nodes using a peer-to-peer architecture.” <i>See</i> 1.d.
35.d. program code that, when executed, is capable of causing the hardware processor	The MultiMatlab References disclose “program code that, when executed, is capable of causing the hardware processor to” for the reasons stated below.

Asserted Claim	Prior Art Disclosure
to:	
35.e. receive calls from a second node comprising a second hardware processor configured 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 interpret user instructions and distribute calls to at least one of a plurality of other nodes for execution;	The MultiMatlab References disclose “receive calls from a second node comprising a second hardware processor configured 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 interpret user instructions and distribute calls to at least one of a plurality of other nodes for execution.” <i>See</i> 1.h, 1.i, 1.f, 1.g.
35.f. execute, using the hardware processor, at least a first mathematical expression evaluation; and	The MultiMatlab References disclose “execute, using the hardware processor, at least a first mathematical expression evaluation.” <i>See</i> 1.h, 1.i.
35.g. communicate a result of the first mathematical expression	The MultiMatlab References disclose “communicate a result of the first mathematical expression evaluation to a third node comprising a third hardware processor with a plurality of processing cores.” <i>See</i> 1.i, 1.j.

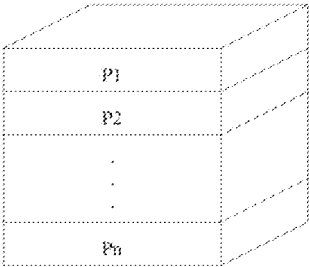
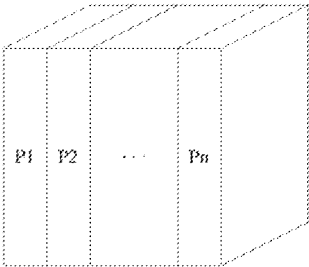
Asserted Claim	Prior Art Disclosure
evaluation to a third node comprising a third hardware processor with a plurality of processing cores,	
35.h. wherein the third node is configured to receive the result of mathematical expression evaluation from the computer cluster node, execute at least a second 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;	The MultiMatlab References disclose “wherein the third node is configured to receive the result of mathematical expression evaluation from the computer cluster node, execute at least a second 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.” <i>See</i> 1.k.
35.i. wherein the user connection interface is	The MultiMatlab References disclose “wherein the user connection interface is configured to return at least one result of mathematical expression evaluation to a user interface or a script.” <i>See</i> 1.l.

Asserted Claim	Prior Art Disclosure
configured to return at least one result of mathematical expression evaluation to a user interface or a script; and	
35.j. wherein the computer cluster node is configured to:	The MultiMatlab References disclose “wherein the computer cluster node is configured to” for the reasons stated below.
35.k. accept user instructions;	The MultiMatlab References disclose “accept user instructions.” <i>See</i> 1.n.
35.l. after accepting user instructions, communicate at least some of the user instructions using the mechanism for the nodes to communicate with each other; and	The MultiMatlab References disclose “after accepting user instructions, communicate at least some of the user instructions using the mechanism for the nodes to communicate with each other.” <i>See</i> 1.o.
35.m. after communicating at least some of the user instructions using the mechanism, communicate at least some of the user instructions to	The MultiMatlab References disclose “after communicating at least some of the user instructions using the mechanism, communicate at least some of the user instructions to the single-node kernel.” <i>See</i> 1.p.

Asserted Claim	Prior Art Disclosure
the single-node kernel.	
36.pre. The computer cluster node of claim 35,	The MultiMatlab References disclose this claim.
36.a. wherein the one or more non-transitory memory devices comprise program code for performing a parallel fast Fourier transform,	<p>The MultiMatlab References disclose “wherein the one or more non-transitory memory devices comprise program code for performing a parallel fast Fourier transform.”</p> <p>The MultiMatlab References disclose computational library code for performing FFT in parallel, e.g., the FFTW or additional FFT libraries:</p> <p>MATLAB® is a high-level language, and a problem-solving environment, for mathematical and scientific calculations. It originated in the late 1970s with an attempt by Cleve Moler to provide interactive access to the Fortran linear algebra software packages EISPACK and LINPACK. Soon a programming language emerged (programs conventionally have the extension <code>.m</code> and are called “m-files”) containing dozens of high-level commands such as <code>svd</code> (singular value decomposition), <code>fft</code> (fast Fourier transform), and <code>roots</code> (polynomial 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.</p> <p>Trefethen 1996 at 2.</p> <p>See also Trefethen 1997 at 7, 10.</p> <p><u>Additional Evidence of Obviousness</u></p>

Asserted Claim	Prior Art Disclosure
	<div>Example 3</div> <div>% FFT2 in 4 lines</div> <div><ul style="list-style-type: none">• A = randn(10000,10000*p);• B = mm('fft',A);• C = B.');• D = mm('fft',C);• F = D.');</div> <div>Choy 3 at 47</div>

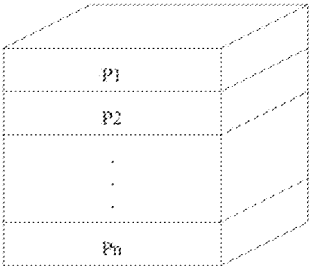
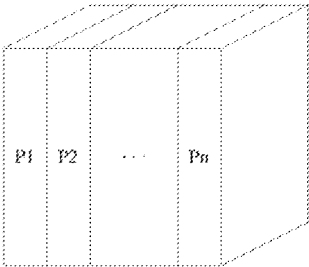
Asserted Claim	Prior Art Disclosure
	<p data-bbox="605 321 1554 639">The main idea in MATLAB *P is that data exist on the parallel server as distributed matrices. Any operations on a distributed matrix (which exists in the MATLAB front end only as a handle) will be relayed to the server transparently. The server calls the appropriate routine from a parallel numerical library (e.g. ScaLAPACK, FFTW, . . .) and the results stay on the server until explicitly requested.</p> <p data-bbox="605 691 750 719">Choy 2 at 336.</p> <p data-bbox="605 756 1858 911">Moreover, parallelization of FFT across a cluster was well-known in the art. Many references teach parallelizing FFTs in a cluster using MPI, PVM or similar protocols, and it would have been obvious to apply these teachings to Matlab *P. Matlab *P kernels implemented an FFT function, and it would have been obvious to combine Matlab *P with any of a number of references teaching FFT parallelization in an MPI or similar cluster.</p> <p data-bbox="605 948 1858 1008">One such reference is Cramer, which teaches parallel FFT implementation in a cluster using MPI or PVM, with each node receiving a data “slice” to operate upon.</p>

Asserted Claim	Prior Art Disclosure
	<div></div> <p>Figure 1: Processor representation of the 3D data space.</p> <div></div> <p>Figure 2: Processor representation of the 3D data space after 2D FFTs and communication step.</p> <p>Cramer at 2.</p> <p>FFTW teaches a similar data slicing approach, which would have been obvious to implement with Matlab *P for the reasons established above:</p>

Asserted Claim	Prior Art Disclosure
	<p>4.2.3 MPI Data Layout</p> <p>The transform data used by the MPI FFTW routines is <i>distributed</i>: a distinct portion of it resides with each process involved in the transform. This allows the transform to be parallelized, for example, over a cluster of workstations, each with its own separate memory, so that you can take advantage of the total memory of all the processors you are parallelizing over.</p> <p>In particular, the array is divided according to the rows (first dimension) of the data: each process gets a subset of the rows of the data. (This is sometimes called a “slab decomposition.”) One consequence of this is that you can’t take advantage of more processors</p> <p>FFTW at 43.</p> <p>The teachings of Cramer and/or FFTW could be implemented by a POSITA without undue experimentation by a set of send and FFT calls, causing a subset of data to be processed by each of the cluster nodes in parallel, such that the Fourier transform is evaluated across the cluster in parallel.</p> <p>Howard teaches a similar distributed approach to FFT parallelization that would be obvious to implement in the same manner. <i>See, e.g.</i>, Howard pars. 142-147.</p> <p>To the extent ACS contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of MultiMatlab and the knowledge of a POSITA, it would have been obvious to combine it with one or more of the other references in these invalidity contentions. For example, MultiMatlab and Matlab*P publications reference MPI and PVM-based parallel processing systems as comparable cluster computing systems, motivating a POSITA to look to such systems for modifications relating to this claim element. <i>See, e.g.</i>, Choy 2 at 334. Accordingly, it would have been obvious to modify MultiMatlab in view of the teachings of PVM and MPI publications, such as those referenced in these contentions, as well as references and systems implementing aspects of PVM or MPI. As another example, because MultiMatlab is directed to cluster computing, it would have been obvious to combine with one or more of the prior art references or systems authored by the alleged inventors of the patents in suit or their collaborators, including patents, applications, articles and code by Messrs. Dager, Tannenbaum and Decyk. These references and code, including those related to the Appleseed cluster, Pooch, MacMPI and related projects, are likewise directed to cluster computing using an existing message passing library and a cluster management tool. Moreover, these prior art publications teach peer-to-peer communications between cluster nodes, and use of these cluster computing tools</p>

Asserted Claim	Prior Art Disclosure
	<p>with CAS kernels. Given the similarity of their teaching, it would have been obvious to combine one of more of these systems or publications with MultiMatlab.</p> <p>In addition, this claim is rendered obvious by Intel Paragon Maple in view of Cramer, Choy, FFTW, Howard, and/or Chu for the reasons stated in the Cover Pleading, section IV.</p>
<p>36.b. wherein the program code causes the hardware processor to evaluate a command to perform a Fourier transform on an array comprising a first data portion that is stored on the computer cluster node and a second data portion that is not stored on the computer cluster node.</p>	<p>The MultiMatlab References disclose “wherein the program code causes the hardware processor to evaluate a command to perform a Fourier transform on an array comprising a first data portion that is stored on the computer cluster node and a second data portion that is not stored on the computer cluster node.”</p> <p>As established for element 36.a above, MultiMatlab implemented a parallelized FFT with each node operating on a portion of data stored locally, followed by at least a second transform on a portion of data sent from another node. Breaking up the FFT dataset into portions, with each portion stored and evaluated on a given cluster node was standard in the art of parallelizing FFT on a cluster. This solution was implemented by Cramer, FFTW and Howard, as just some examples.</p> <p>MATLAB® is a high-level language, and a problem-solving environment, for mathematical and scientific calculations. It originated in the late 1970s with an attempt by Cleve Moler to provide interactive access to the Fortran linear algebra software packages EISPACK and LINPACK. Soon a programming language emerged (programs conventionally have the extension <code>.m</code> and are called “m-files”) containing dozens of high-level commands such as <code>svd</code> (singular value decomposition), <code>fft</code> (fast Fourier transform), and <code>roots</code> (polynomial zero-finding). Graphical commands were built into the language, and a company called The MathWorks, Inc. was formed in 1984 by Moler and John Little, now based in Natick, Massachusetts.</p> <p>Trefethen 1996 at 2.</p>

Asserted Claim	Prior Art Disclosure
	<p>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.</p> <p>Trefethen 1997 at 10.</p> <p>See also Trefethen 1997 at 7.</p> <p><u>Additional Evidence of Obviousness</u></p> <p>Moreover, parallelization of FFT across a cluster was well-known in the art. Many references teach parallelizing FFTs in a cluster using MPI, PVM or similar protocols, and it would have been obvious to apply these teachings to Matlab *P. Matlab *P kernels implemented an FFT function, and it would have been obvious to combine Matlab *P with any of a number of references teaching FFT parallelization in an MPI or similar cluster.</p> <p>One such reference is Cramer, which teaches parallel FFT implementation in a cluster using MPI or PVM, with each node receiving a data portion to operate upon:</p>

Asserted Claim	Prior Art Disclosure
	<div></div> <p>Figure 1: Processor representation of the 3D data space.</p> <div></div> <p>Figure 2: Processor representation of the 3D data space after 2D FFTs and communication step.</p> <p>Cramer at 2.</p> <p>FFTW teaches a similar data slicing approach, which would have been obvious to implement with MultiMatlab and Matlab *P for the reasons established above:</p>

Asserted Claim	Prior Art Disclosure
	<p>4.2.3 MPI Data Layout</p> <p>The transform data used by the MPI FFTW routines is <i>distributed</i>: a distinct portion of it resides with each process involved in the transform. This allows the transform to be parallelized, for example, over a cluster of workstations, each with its own separate memory, so that you can take advantage of the total memory of all the processors you are parallelizing over.</p> <p>In particular, the array is divided according to the rows (first dimension) of the data: each process gets a subset of the rows of the data. (This is sometimes called a “slab decomposition.”) One consequence of this is that you can’t take advantage of more processors</p> <p>FFTW at 43.</p> <p>The teachings of Cramer and/or FFTW could be implemented by a POSITA without undue experimentation by a set of send and FFT calls, causing a subset of data to be processed by each of the cluster nodes in parallel, such that the Fourier transform is evaluated across the cluster in parallel.</p> <p>Howard teaches a similar distributed approach to FFT parallelization that would be obvious to implement in the same manner. <i>See, e.g.</i>, Howard pars. 142-147. Likewise, Agarwal teaches an approach to split the FFT data set such that each node performs FFT on a portion of the data, and would be obvious to combine with MultiMatlab and Matlab *P for the same reasons. <i>See</i> Agarwal at 36-38.</p>
37.pre. The computer cluster node of claim 35,	The MultiMatlab References disclose this claim.
37.a. wherein the computer cluster node is configured to permit exchange of information with other computer cluster nodes during the course of parallel	The MultiMatlab References disclose “wherein the computer cluster node is configured to permit exchange of information with other computer cluster nodes during the course of parallel computation.” <i>See</i> 31.b

Asserted Claim	Prior Art Disclosure
computation.	
39.pre. The computer cluster node of claim 35,	The MultiMatlab References disclose this claim.
39.a. wherein the hardware processor comprises a special purpose microprocessor.	<p>The MultiMatlab References disclose “wherein the hardware processor comprises a special purpose microprocessor.”</p> <p>To the extent processors disclosed by the MultiMatlab references do not including a hardware processor including a special purpose microprocessor, it would have been obvious to use a special purpose microprocessors, such as math co-processors, for performing evaluation of mathematical expressions using Matlab and MultiMatlab..</p>

Asserted Claim	Prior Art Disclosure
	<p>Scientific computing software applications are widely used in the science and engineering community as computing and modeling tools. As used herein, the term scientific computing software application refers to a software application capable of solving computational problems that are input by a user. Examples of commercially available scientific computing software applications include MATLAB™, produced and sold by The MathWorks, Natick, Mass., Maple™, produced and sold by Maplesoft, Waterloo, Ontario, Canada, and Mathematica™, produced and sold by Wolfram Research, Champaign, Ill. These software applications are useful in a variety of disciplines, including, for example, engineering, mathematics, physics, life sciences, chemistry, chemical engineering, social sciences, finance, economics, and others. It should be appreciated that while referred to as a scientific computing software application, scientific software applications are not limited to solving problems or performing calculations of a scientific nature, as any suitable type of problems or calculations may be processed by the scientific computing software application.</p> <p>Husbands at 3:37-56.</p>

Asserted Claim	Prior Art Disclosure
	<div></div> <p>PC Magazine 1 at 425.</p> <p>As another example, it would have been obvious to run MultiMatlab and Matlab *P nodes on computers with special purpose processors including DSPs, as such processors were known to be advantageous for certain types of computation, including FFT and matrix computations. These and other use cases could be enhanced with DSP processors, as was known in the art. For example, Melpignano teaches that computer clusters can have heterogeneous processors, including DSPs. <i>See, e.g.,</i> Melpignano, 7:29-31, 7:65-8:5; <i>see also, e.g.,</i> Pruvost, 4:14-22, 4:58-5:8. The '768 patent recites conventional processors, including a “special purpose processor” such as a DSP, confirming the obviousness of this limitation. <i>See, e.g.,</i> '768, 7:10-32.</p> <p>As another example, Nayak teaches the advantages of cluster computing on DSPs:</p>

Asserted Claim	Prior Art Disclosure
	<p>The DSP architecture has some custom hardware for multiply and accumulate type of operations, which make them ideal for signal and image processing applications in comparison to the general purpose processors. Hence, it is wise to implement computation intensive matrix operations on the DSP as they would take a longer time on general purpose processors.</p> <p>Nayak at 4.</p> <p>Nayak teaches a cluster with a combination of DSP and/or FPGA nodes, in addition to other processor nodes. It uses a Matlab kernel and MPI for peer-to-peer message passing, making it obvious to combine with MultiMatlab and Matlab *P:</p> <p>Nayak, Fig. 4.</p>

Asserted Claim	Prior Art Disclosure
	<p>It would have been obvious to combine MultiMatlab and Matlab *P with Nayak for implementing matrix and FFT computations, as expressly motivated by Nayak. Nayak provides disclosure of the software and hardware implementation to make the combination without undue experimentation.</p> <p>Nayak teaches the use of TI TMS320C40, each of which comprised at least two processing cores. Moreover, Nayak taught that DSP processors could be used together as a cluster node in a heterogeneous cluster. In addition, Nayak taught that FPGA processors could be used in a cluster.</p> <p>To the extent ACS contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of MultiMatlab and the knowledge of a POSITA, it would have been obvious to combine it with one or more of the other references in these invalidity contentions. For example, MultiMatlab and Matlab*P publications reference MPI and PVM-based parallel processing systems as comparable cluster computing systems, motivating a POSITA to look to such systems for modifications relating to this claim element. <i>See, e.g.,</i> Choy 2 at 334. Accordingly, it would have been obvious to modify MultiMatlab in view of the teachings of PVM and MPI publications, such as those referenced in these contentions, as well as references and systems implementing aspects of PVM or MPI. As another example, because MultiMatlab is directed to cluster computing, it would have been obvious to combine with one or more of the prior art references or systems authored by the alleged inventors of the patents in suit or their collaborators, including patents, applications, articles and code by Messrs. Dager, Tannenbaum and Decyk. These references and code, including those related to the Appleseed cluster, Pooch, MacMPI and related projects, are likewise directed to cluster computing using an existing message passing library and a cluster management tool. Moreover, these prior art publications teach peer-to-peer communications between cluster nodes, and use of these cluster computing tools with CAS kernels. Given the similarity of their teaching, it would have been obvious to combine one of more of these systems or publications with MultiMatlab.</p> <p>In addition, this claim is rendered obvious by Intel Paragon Maple in view of Cramer, Choy, FFTW, Howard, and/or Chu for the reasons stated in the Cover Pleading, section IV.</p>