Parallel MATLAB: Doing it Right

Ron Choy , Alan Edelman

Computer Science AI Laboratory,

Massachusetts Institute of Technology, Cambridge, MA 02139

This project is supported in part by the Singapore-MIT Alliance

Abstract

MATLAB [20] is one of the most widely used mathematical computing environments in technical computing. It is an interactive environment that provides high performance computational routines and an easy-to-use, C-like scripting language. It has started out as an interactive interface to EISPACK [31] and LINPACK [13], and has remained a serial program. In 1995, Cleve Moler of Mathworks argued that there was no market at the time for a parallel MATLAB [26]. But times have changed and we are seeing increasing interest in developing a parallel MATLAB, from both academic and commercial sectors. In a recent survey, [10] 27 parallel MATLAB projects have been identified.

In this paper we will expand upon that survey and discuss the approaches the projects have taken to parallelize MATLAB. Also we will describe innovative features in some of the parallel MATLAB projects. Then we will conclude with an idea of a 'right' parallel MATLAB.Finally we will give an example of what we think is a 'right' parallel MATLAB: MATLAB*P [11]

I. MATLAB

MATLAB [20] is one of the most widely used tools in scientific and technical computing. It started in the 1970s as an interactive interface to EISPACK [31] and LINPACK [13], a set of eigenvalue and linear system solution routines. It has since grown to a feature rich product utilizing modern numerical libraries such as ATLAS [36] and FFTW [16], 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. Also it provides hooks to the Java programming language, making integration with compiled programs easy.

MATLAB gained popularity because of its user-friendliness. It has seen widespread use in classrooms as a teaching tool. Its strong graphical capabilities makes it a good data analysis tool. Also researchers have been known to build very complex systems using MATLAB scripts and toolboxes.

II. Why there should be a parallel MATLAB

Because of its roots in serial numerical libraries, MATLAB has always been a serial program. In 1995, Cleve Moler of Mathworks wrote an article *Why there isn't a parallel MATLAB* [26], stating Mathworks' intention not to develop a parallel MATLAB at that time. His arguments could be summarized as follows:

1) Memory model

   Distributed memory was the dominant model for parallel computers, and for linear algebra applications, scatter/gather of the matrix took too long to make parallel computation worthwhile.

2) Granularity

   For typical use, MATLAB spends most of its time in the parser, interpreter and graphics routines, where any parallelism is difficult to find. Also, to handle embarrassingly parallel applications, which only requires a collection of results at the end, MATLAB would require fundamental changes in its architecture.

3) Business situation

   There were not enough customers with parallel computers to support the development.

It has been eight years since the article was written, and we have seen tremendous changes in the computing world. These changes have invalidated the arguments that there should not be a parallel MATLAB.

1) Memory model

   As modern scientific and engineering problems grow in complexity, the computation time and memory requirements skyrocket. The increase in processor speed and the amount of memory that can fit in a single machine could not catch up with the pace of computation requirements. Very often current scientific problems simply do not fit into the memory of a single machine, making parallel computation a necessity. Combining with improvements in interconnect technologies, parallel computation has become a worthwhile endeavor.

2) Granularity

   Over the past eight years simple parallel MATLAB projects, some consisting of only 2 m-files, have shown that multiple MATLAB instances running on a parallel computer could be used to solve embarrassingly parallel problems, without any change to MATLAB itself. Also, increase in problem sizes and processor speed have reduced the portion of time spent in non-computation related routines.

3) Business situation

   The past few years have seen the proliferation of Beowulf clusters. Beowulf clusters are parallel computers made from commodity off-the-shelf (COTS) hardware. They often consist of workstations connected together with ethernet or other common, non-proprietary interconnect. Researchers prefer Beowulf clusters over traditional supercomputers because Beowulfs are quite easy to setup and maintain, and cheap enough so that a researcher can get his or her 'personal supercomputer'. However, often researchers in science wanting to use a parallel computer to solve problems are not experts in parallel programming. The dominant way of parallel programming, MPI [15], is too low-level and too error prone. MATLAB is well known for its user-friendliness. There is a huge potential market for a MATLAB that could be used to program parallel computers.

III. MATLAB, MAPLE, AND MATHEMATICA

Besides MATLAB, there are two other very popular technical computing environments. First is Maple ®, developed by Maplesoft, which is well known for its excellent symbolic calculation capabilities. Then there is Mathematica ®, developed by Wolfram Research, which features a procedural programming language.

We are interested in looking at parallel MATLAB mainly because:

1) MATLAB is popular

Compared to Maple and Mathematica, we feel a parallel MATLAB would reach a wider audience. MATLAB has seen extensive use in classrooms at MIT and worldwide. To get a rough idea of the popularity of the three software packages, we did a search on CiteSeer [8] for the number of citations and Google for the number of page hits:

	CiteSeer	Google
MATLAB	3117	1720000
Maple	1733	Common word
Mathematica	1773	1530000

TABLE I: CiteSeer and Google search for 3 popular math packages, measured on November 15, 2003

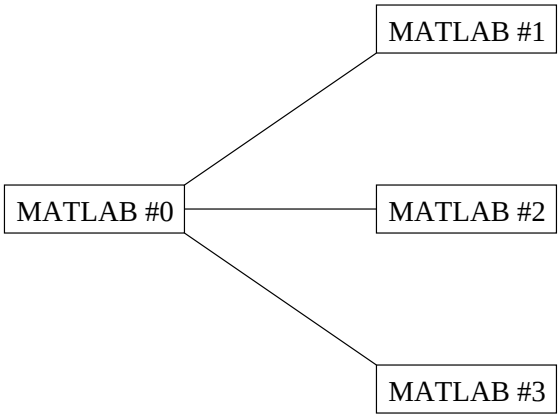
2) MATLAB is user friendly

From our experience of using the three packages, we feel that MATLAB has the best user interface. Its C/HPF like scripting language is the most intuitive among the three for numerical parallel computing applications.

IV. Parallel MATLAB Survey

The popularity of MATLAB and the fact that it could only utilize one processor sparked a lot of interest in creating a parallel MATLAB. We have done a survey [10] and found through extensive web searching 27 parallel MATLAB projects. These projects vary in their scope: some are one-man projects that provide basic embarrassingly parallel capabilities to MATLAB; some are university or government lab research projects; while some are commercial projects that enables the user of MATLAB in product development. Also their approaches to making MATLAB parallel are different: some compile MATLAB scripts into parallel native code; some provide a parallel backend to MATLAB, using MATLAB as a graphical frontend; and some others coordinate multiple MATLAB processes to work in parallel. These projects also vary widely in their status: some are now defunct and exist only in Google web cache, while some are entering their second or third revision.

A. *Embarrassingly Parallel*

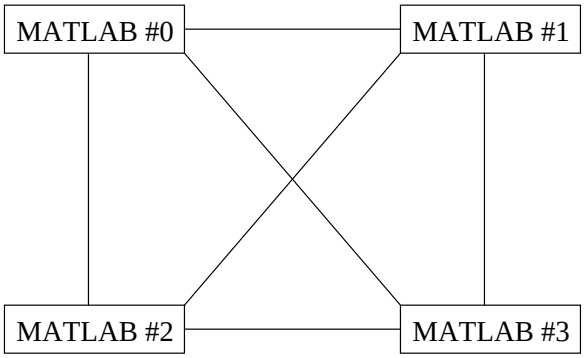


Software that make use of this approach: Multi [24], Paralize [1], PLab [23], Parmatlab [3]

This approach makes use of multiple MATLAB processes running on different machines or a single machine with multiple processors. However no coordination between the MATLAB processes is provided.. Instead, a parent process passes off data to the child processes. Then all processes work on its local data, and return the result to the parent process.

Under this model, the type of functions that can be parallelized is limited. For example, a for-loop with any data dependency across iteration would be impossible to parallelize under this model. However, this model has the advantage of simplicity. In the software we found that utilize this approach, usually no change in existing code is needed to parallelize the code, if the code is parallelizable using this simple approach. From our experience, this approach is sufficient for a lot of real world applications. For example, Seti@Home belongs to this category.

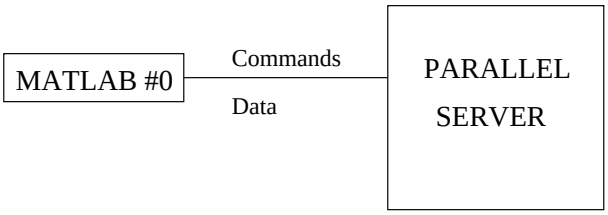
B. Message Passing



Software that make use of this approach: MultiMATLAB [34], CMTM [37], DPToolbox [28], MPITB/PVMTB [5], MATmarks [2], PMI [25], PTToolbox[18], MatlabMPI [22], pMatlab [21]

This approach provides message passing routines between MATLAB processes. This enables users to write their own parallel programs in MATLAB in a fashion similar to writing parallel programs with a compiled language using MPI. In fact, for some of the projects [34] [37] [5] [22], the routines provided are wrappers for MPI routines. This approach has the advantage of flexibility: users are theoretically able to build any parallel system in MATLAB that they can build in compiled languages with MPI. This approach is a superset of the embarrassingly parallel approach.

C. Backend Support



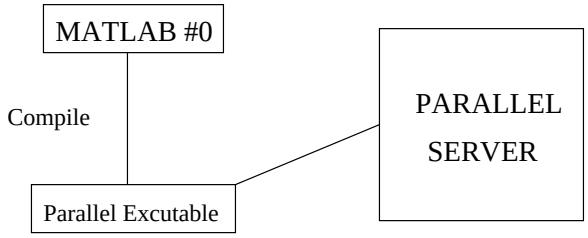
Software that make use of this approach: NetSolve [4], DLab [27], Matpar [32], PLAPACK [35], PARAMAT [33], MATLAB*P [11]

This approach uses MATLAB as a front end for a parallel computation engine. Computation is done on the engine, usually making use of high performance numerical computation libraries like ScaLAPACK

[7]. For some approaches e.g. [4] [27], the data resides in MATLAB and are passed to the engine and back. And for some approaches e.g. [11], the data reside on the server and are passed back to MATLAB only upon request. The latter approach has the advantage that there is less data traffic. This is important for performance when data sets are large.

The advantage of this approach is that it only requires one MATLAB session (therefore only one license), and it usually does not require the end user to have knowledge of parallel programming.

D. MATLAB Compilers



Software that make use of this approach: Otter [30], RTExpress [19], ParAL [29], FALCON [12], CONLAB [14], MATCH [6], Menhir [9]

These softwares compile MATLAB scripts into an executable, sometimes translating the scripts into a compiled language as an intermediate step. Some softwares e.g. [30] [14] links the compiled code with parallel numerical libraries, while some softwares e.g. [12] generates code that is already parallel.

This approach has the advantage that the compiled code runs without the overhead incurred by MATLAB. Also the compiled code can be linked with code written in C or FORTRAN, so MATLAB can be used to develop part of a system instead of the whole system. In this approach, MATLAB is used as a development platform instead of a computing environment. This allows the produced parallel program to run on platforms which does not support MATLAB (e.g. SGI).

E. The Parallel MATLABs

MultiMATLAB [34]	Cornell University
CMTM [37]	Cornell University
DP-Toolbox [28]	University of Rostock, Germany
MPITB/PVMTB [5]	University of Granada, Spain
MATmarks [2]	UIUC
PMI [25]	Lucent Technologies
PT Toolbox [18]	Wake Forest University
MatlabMPI [22]	MIT Lincoln Lab
pMatlab [21]	MIT Lincoln Lab

TABLE II: Message Passing

Matlab Parallelization Toolkit [17]	Linkopings Universitet, Sweden
MULTI Toolbox [24]	Purdue University
Paralize [1]	Chalmers University of Technology, Sweden
PLab [23]	Technical University of Denmark
Parmatlab [3]	Northeastern University

TABLE III: Embarrassingly Parallel

Netsolve [4]	University of Tennessee, Knoxville
DLab [27]	UIUC
Matpar [32]	Jet Propulsion Lab
PLAPACK [35]	University of Texas at Austin
Paramat [33]	Alpha Data Parallel Systems
MATLAB*P [11]	MIT

TABLE IV: Backend Support

Otter [30]	Oregon State University
RTExpress [19]	Integrated Sensors Inc.
ParAL [29]	University of Sydney, Australia
FALCON [12]	UIUC
CONLAB [14]	University of Umea, Sweden
MATCH [6]	Accelchip Inc.
Menhir [9]	IRISA, France

TABLE V: MATLAB Compiler

V. Parallel MATLAB Features

In evaluating the 27 parallel MATLAB projects, we found that some contain features that are innovative in the parallel MATLAB world:

A. File System Based Communication

The earlier parallel MATLABs like MultiMATLAB [34] used TCP/IP based communication. TCP/IP implementations and routine calls are often system dependent. Thus, a TCP/IP based parallel MATLAB would not work cross platform.

Realizing that problem, some parallel MATLAB projects began to utilize a file system based communication system. Machines in a Beowulf cluster often share a common file system, e.g. NFS, and it can be exploited for communication. One of the earliest parallel MATLAB projects to make use of this is Paralize [1](1998). Sends and receives are handles through writing to and reading from files, and synchronization is done through checking for the existence of certain lock files. MatlabMPI [22] implements basic MPI functions in MATLAB using cross-mounted directories. Bandwidth comparable to C-based MPI is reported, although the latency is inferior.

B. Pure m-file Implementation

Simple parallel MATLAB approaches like the embarrassingly parallel approach or MPI could be implemented with pure MATLAB m-files. One of the earliest parallel MATLAB to be implemented this way is Paralize [1] (1998), a parallel MATLAB using the embarrassingly approach and implementing dynamic load balancing in only 120 lines of MATLAB code. Pure m-file implementation has the advantage of being portable to any platform on which MATLAB runs. Also, it makes installation simple and user-friendly - no compilation is needed.

C. Parallelism through Polymorphism

Parallel MATLABs often introduce new commands into MATLAB. For example, a *pfor* function for parallel for-loops, or a set of MPI-like functions for message passing operations. The problem with this approach is that it does not reduce the complexity of writing a parallel program. MPI is not the level general users want to or should want to program at.

Since the number of functions in MATLAB is finite, it is sufficient to parallelize the functions in MATLAB through overloading, and make the users call the parallel version instead. This is introduced in MATLAB*P [11] seamlessly in the concept of *Parallelism through Polymorphism*. MATLAB*P is a parallel MATLAB using the backend support approach, attaching a parallel backend server to MATLAB. This will be explained further in a later section.

D. Lazy Evaluation

Backend support parallel MATLABs utilize a server to perform the computation. While the server is busy computing, the frontend MATLAB often idles, wasting precious cycles. To remedy this problem, DLab [27] introduces a concept called *lazy evaluation*. When a command is sent to the server, the MATLAB program does not block and wait for the result. Instead, the MATLAB program only blocks when it tries to make a server call again. This way, additional computations could be done in the time between the server call and the next, which is wasteful in other backend support parallel MATLABs.

VI. What does the user want in a parallel MATLAB?

A. The MATLAB experience

We could reflect on the experience of MATLAB to understand what a user would want in a parallel MATLAB.

MATLAB started out as an interactive interface to EISPACK and LINPACK, a set of eigenvalue and linear system solution routines. EISPACK and LINPACK were written in FORTRAN, so normally a

user would have to write program in FORTRAN to use the routines. That way the user will have to take care of the memory allocation, indexing, and studying the (quirky) EISPACK/LINPACK syntaxes. Furthermore, there is no way to visualize the result without writing your own code.

But MATLAB took care of all of this. User calls create matrices and operate on them using simple, intuitive calls. Visualization could also be done in the same framework. Furthermore, the scripting language in MATLAB contains features from C and HPF, making it easier to use than FORTRAN and more suitable for matrix computation than C.

All this convenience came at a cost of performance. The graphical user interface, parser and interpreter takes away processor cycles which could be used for computation. Also the interpreted scripts could not match the performance of compiled C or FORTRAN code. Yet still, MATLAB is a huge success. We can see that what the users really want is ease-of-use, not peak performance. Users prefer a system that is easy to use and has good performance, over a system with peak performance but is hard to use and clumsy. The gain in performance in the latter system is easily offset by the additional time needed to program it.

B. Modes of Parallel Computation

Over the years we have seen many parallel programs from various application areas: e.g. signal processing, graphics, artificial intelligence, computational mathematics, and climate modeling. From the code we have seen, we divide parallel computation in general into four categories:

1) A lot of small problems that require no communication

   Also known as embarrassingly parallel problems. The problem size is small enough so that it will fit into the memory of one machine, but there are a lot of them so parallel computation is needed. No communication is required between the parallel threads of computation, except for an initial scatter - to distribute the computation and a final gather - to collect the results. An example of this

type of computation would be animation rendering. This is the largest class of problems.

2) A lot of small problems that require some communication

   Just like the first case, the problem size is small enough to fit into the memory of one machine. However, in this category it is necessary to communicate between the parallel threads during the computation.

3) Large problems

   This class of problems has a problem size that would not fit into the memory of a single machine. Example: the high performance Linpack (HPL) benchmark

4) A mixture of the three

   In this class of problems, the data is sometimes processed individually in embarrassingly parallel mode, while sometimes it is treated as a global data structure. We have seen an example of this in climate modeling.

A good parallel MATLAB should address at least one of these areas well.

C. The 'Right' Parallel MATLAB

When building the 'Right' parallel MATLAB targeting the widest possible audience (and thus would be most commercially viable), we should take the above arguments into account. First, the interface in the parallel MATLAB should be easy to use, does not differ much from ordinary MATLAB, and does not require learning on the users' part. Secondly, it should allow the four modes of parallel computation described above.

VII. MATLAB*P

We present MATLAB*P [11] as an example of what could be a 'Right' parallel MATLAB. MATLAB*P 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 *embarrassingly parallel* approach allow simplistic, yet useful, division of work into multiple MATLAB sessions. The *message passing* approach, which is a superset of the embarrassingly parallel approach, allows finer control between the MATLAB sessions.

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 frontend 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.

The 'transparency' comes from the use of polymorphism in MATLAB. This will be explained in the next section.

VIII. Features of MATLAB*P

A. Parallelism through Polymophism - *p

The key to the parallelism lies in the *p variable. It is an object of *dlayout* class in MATLAB. By overloading MATLAB functions for the class *dlayout*, we were able to create a parallel MATLAB that has exactly the same interface as MATLAB.

Through the use of the *p variable, matrices that are distributed on the server could be created. For example,

```
X = randn(8192*p,8192);
```

The above creates a row distributed, 8192 x 8192 normally distributed random matrix on the server. X is a handle to the distributed matrix, identified by MATLAB as a *ddense* class object. By overloading randn and many other built-in functions in MATLAB, we are able to tie in the parallel support transparent

to the user. This is called *parallelism through polymorphism*. Note that the syntax is exactly the same as ordinary MATLAB except for the additional *p variable.

```
e = eig(X);
```

The command computes the eigenvalues of X by calling the appropriate ScaLAPACK routines, and store 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.

```
E = pp2matlab(e);
```

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 of them).

The use of the *p variable along with overloaded MATLAB routines enable existing MATLAB scripts to be reused. For example,

```
function H = hilb(n)


J = 1:n;
J = J(ones(n,1),:);
I = J’;
E = ones(n,n);
H = E./(I+J-1);
```

The above is the built-in MATLAB routine to construct a Hilbert matrix (obtained through *type hilb*). Because the operators in the routine (colon, ones, subsasgn, transpose, rdivide, +, -) are overloaded to work with *p, typing

```
H = hilb(16384*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 MATLAB*P without any modification.

B. 'MultiMATLAB/MultiOctave mode'

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.

In conjunction with Parry Husbands, we developed a prototype implementation of a MultiMATLAB[34]-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.

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.

Examples of the usage of the PPEngine system:

```
>> % Example 1
>> a = 1:100*p;
```

```
>> b = mm('chi2rnd',a);
```

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.

```
>> % Example 2
>> a = rand(100,100*p);
>> b = rand(100,100*p);
>> c = mm('plus',a,b);
```

This example creates two column distributed matrices of size 100x100, adds them, and puts the result in another matrix. This is the slow way of doing the equivalent of:

```
>> a = rand(100,100*p);
>> b = rand(100,100*p);
>> c = a+b;
```

The 'MultiOctave' mode works exactly the same as 'MultiMATLAB' mode, only using Octave, a freely available MATLAB-like scientific computing software, for the computation.

```
>> % Example 3
>> a = randn(4,4*p);
>> b = mm('sin',a)
>> c = mo('sin',a)
>> norm(b-c)
```

```
ans =

    6.7820e-07
```

The above interesting example shows that Octave and MATLAB uses a different algorithm for the sine function. Octave serves the purpose of an embarrassingly parallel backend very well, because although its graphical capabilities is not as good as MATLAB, it's numerical parts is up to par with MATLAB. As a backend engine, we are mostly concerned with numerical performance.

```
>> % Example 4
>> a = (0:(np-1)*p)/np;
>> b = a + (1/np);
>> [mypi, fcnt] = mm('quadl','4./(1+x.^2)',a,b);
>> disp('Pi calculated from quadl in mm mode')
>> pi_from_quadl=sum(mypi)
>> disp('Number of function evaluation on each processor')
>> fcnt(:)

Pi calculated from quadl in mm mode
pi_from_quadl =

    3.1416
```

```
Number of function evaluation on each processor

ans =


    18

    18

    18

    18
```

The above example illustrates how *np*, the variable that returns the number of processes running on the backend server, can be used in a script to write adaptive code. When the above example is run on 4 processes, a is 0:0.25:0.75, and b is 0.25:0.25:1. In the 'MultiMATLAB' call each slave MATLAB will compute the adaptive Lobatto quadrature of

$$\frac{4}{1+x^2}$$

in the intervals (0,0.25), (0.25,0.50), (0.50,0.75), (0.75,1.0) respectively. The result from each slave MATLAB is summed to form *pi*.

C. Visualization Package

This visualization package was written by Bruning, Holloway and Sulejmanpasic, under supervision of Ron Choy, as a term project for the class 6.338/18.337 - Applied Parallel Computing at MIT. It has since then been merged into the main MATLAB*P source.

This package adds *spy*, *surf*, and *mesh* routines to MATLAB*P. This enable visualization of very large matrices. The rendering is done in parallel, using the Mesa OpenGL library.

Figure 1, 2, 3 shows the routines in action.

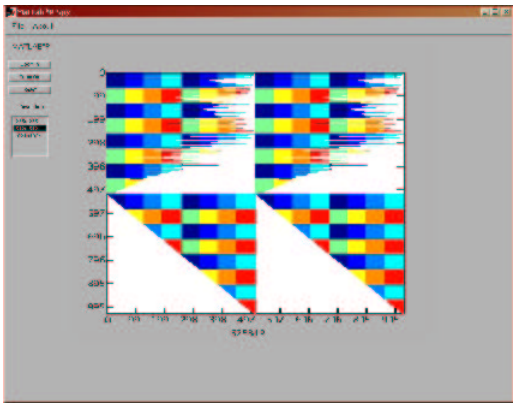


Fig. 1. ppspy on a distributed 1024x1024 matrix on eight nodes

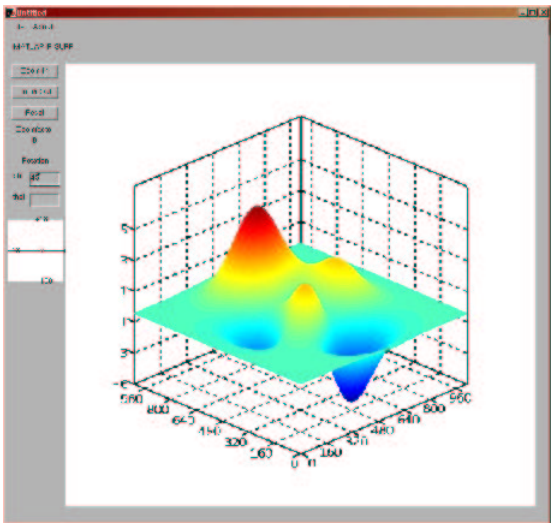


Fig. 2. ppsurf on the distributed 1024x1024 'peaks' matrix

All three routines allow zooming into a portion of the matrix. Also, ppsurf and ppmesh allow changing of the camera angle, just as supported in MATLAB.

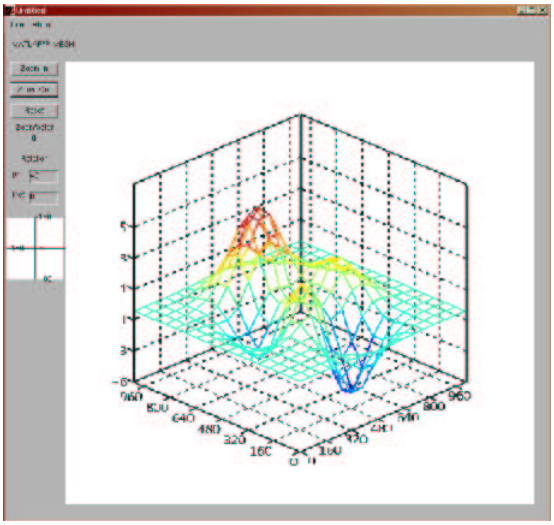


Fig. 3. ppmesh on the distributed 1024x1024 'peaks' matrix

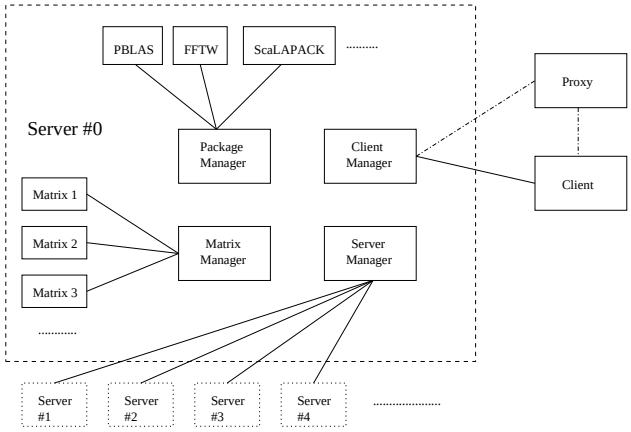


Fig. 4. Structure of MATLAB*P 2.0

*D. Structure of MATLAB*P 2.0 system*

MATLAB*P 2.0 is written in C++ using extensive use of templates. It interfaces with MATLAB using the MEX (MATLAB Extension) interface.

The server itself is divided in four self-contained parts:

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 communications between server processes. It mainly controls broadcasting of commands and arguments from head process to the slave processes, and collection of results and error codes. Also it provides rank and size information to the processes.

3) Package Manager

   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.

4) Matrix Manager

   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.

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.

Similar extensions can be made to Client Connection Manager as well. The basic Client Connection Manager uses TCP socket. An interesting replacement would be to make a Client Connection Manager that act as an interface to a language like C++ or Java.

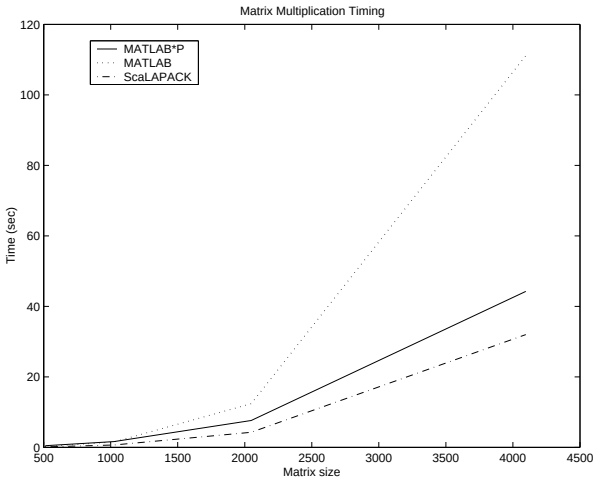


Fig. 5. Matrix multiplication timing results

IX. Benchmarks

We compare the performance of MATLAB*P, MATLAB, and ScaLAPACK on a Beowulf cluster running Linux.

A. Test Platform

- Beowulf cluster with 9 nodes (2 nodes are used in the tests).
- Dual-processor nodes, each with two 1.533GHz Athlon MP processors.
- 1GB DDR RAM on each node. No swapping occurred during benchmarks.
- Fast ethernet (100Mbps/sec) interconnect. Intel Etherfast 410T switch.
- Linux 2.4.18-4smp
- MATLAB 6.1.0 R12.1

B. Timing Results

See graphs of matrix multiplication timing results and linear system solve timing results.

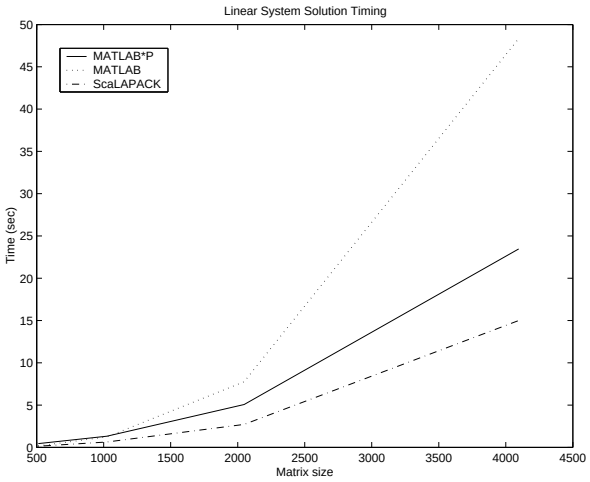


Fig. 6. Linear system solve timing results

C. Analysis of Performance

*1) MATLAB*P and MATLAB:* From the results, MATLAB*P on 4 processors begins to outperform MATLAB on single processor when the problem size is 2048 and upward. This shows that for smaller problems, one should use plain MATLAB instead of MATLAB*P. When the problem size is large, MATLAB*P offers two advantages:

- Better performance
- Distributed memory, enabling larger problems to fit in memory.

And all these come at close to zero effort on the user's part.

*2) MATLAB*P and ScaLAPACK:* Comparing the timing results of MATLAB*P and ScaLAPACK, we see that ScaLAPACK is always faster than MATLAB*P, although the gap narrows at larger problem size. This should be obvious from the fact that MATLAB*P uses ScaLAPACK for matrix multiplication and linear system solution, and MATLAB*P incurs overhead.

The difference in the timing results come from both overhead incurred by MATLAB*P and the design of the benchmark itself:

- Timing for MATLAB*P is done inside MATLAB, using tic/toc on the MATLAB call. The MATLAB

call includes memory allocation and matrix copying on the server side. Timing for ScaLAPACK is done in C++ using clock(), and only the actual computation routine is timed.

- There is a messaging overhead from the MATLAB client to the server. As the MATLAB call yields multiple calls to the server, this messaging overhead is multiplied.
- In linear system solution, ScaLAPACK overwrites the input matrix. In MATLAB*P, in order to mimic standard MATLAB behaviour, the input matrix is copied into another matrix which is used in the ScaLAPACK call. This incurred additional overhead.

X. Conclusion

MATLAB*P shows how a parallel MATLAB system can address different needs of parallel computing. Backend calls to ScaLAPACK handles large problems in parallel. The 'MultiMATLAB/MultiOctave' mode takes care of problems that are embarrassingly parallel in nature, and collect the result in a distributed matrix so that global operations can be performed on the result.

The system has been used to solve dense linear system of size 100000x100000 and 2D FFT of size 64000x64000 with success. It has been used for biomedical imaging and climate modeling applications, as well as a teaching tool in MIT.

References

[1] Thomas Abrahamsson. Paralize:. *ftp://ftp.mathworks.com/pub/contrib/v5/tools/paralize/*, 1998.

[2] George Almasi, Calin Cascaval, and Dvaid A. Padua. Matmarks: A shared memory environment for matlab programming. 1999.

[3] Lucio Andrade. Parmatlab. *ftp://ftp.mathworks.com/pub/contrib/v5/tools/parmatlab/*, 2001.

[4] D. Arnold, S. Agrawal, S. Blackford, J. Dongarra, M. Miller, K. Sagi, Z. Shi, and S. Vadhiyar. Users' Guide to NetSolve V1.4. Computer Science Dept. Technical Report CS-01-467, University of Tennessee, Knoxville, TN, July 2001.

[5] Javier Fernndez Baldomero. Mpi/pvm toolbox for matlab. *http://atc.ugr.es/javier-bin/mpitb*, 2000.

[6] P. Banerjee. A matlab compiler for distributed, hetergeneous, reconfigurable computing systems. *IEEE Symposium on FPGAs for Custom Computing Machines, 2000.*, 2000.

[7] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997.

[8] Kurt Bollacker, Steve Lawrence, and C. Lee Giles. A system for automatic personalized tracking of scientific literature on the web. In *Digital Libraries 99 - The Fourth ACM Conference on Digital Libraries*, pages 105–113, New York, 1999. ACM Press.

[9] Stephane Chauveau and Francois Bodin. Menhir: An environment for high performance matlab. In *Languages, Compilers, and Run-Time Systems for Scalable Computers*, pages 27–40, 1998.

[10] R. Choy. Parallel matlab survey. *http://theory.lcs.mit.edu/~cly/survey.html*, 2001.

[11] R. Choy. Matlab*p 2.0: Interactive supercomputing made practical. *Sc. M. Thesis, Massachusetts Institute of Technology*, 2002.

[12] L. DeRose, K. Gallivan, E. Gallopoulos, B. Marsolf, and D. Padua. Falcon: A matlab interactive restructing compiler. Technical report, 1995.

[13] J.J. Dongarra, J.R.Bunch, C.B.Moler, and G.W.Stewart. *LINPACK User's Guide*. SIAM, Philadelphia, 1979.

[14] Peter Drakenberg, Peter Jacobson, and Bo Kågström. A CONLAB compiler for a distributed memory multicomputer. *The Sixth SIAM Conference on Parallel Processing for Scientific Computation, Volume 2*, pages 814–821, 1993.

[15] Message Passing Interface Forum. MPI: A message-passing interface standard. Technical Report UT-CS-94-230, 1994.

[16] M. Frigo and S. Johnson. Fftw: An adaptive software architecture for the fft. *Proc. Intl. Conf. of Acoustics, Speech and Signal Processing, 1998, v. 3, p. 1381*, 1998.

[17] Einar Heiberg. Matlab parallelization toolkit
. *http://hem.passagen.se/einar_heiberg/history.html*, 2001.

[18] J. Hollingsworth, K. Liu, and P. Pauca. Parallel toolbox for matlab pt v. 1.00: Manual and reference pages. 1996.

[19] Integrated Sensors Inc. Rtexpress. *http://www.rtexpress.com/*.

[20] Mathworks Inc. *MATLAB 6 User's Guide*. 2001.

[21] J. Kepner and N. Travinin. Parallel matlab: The next generation. *7th High Performance Embedded Computing Workshop (HPEC 2003)*, 2003.

[22] Jeremy Kepner. Parallel programming with matlabmpi. *Accepted by High Performance Embedded Computing (HPEC 2001) Workshop*, 2001.

[23] Ulrik Kjems. Plab: reference page. *http://bond.imm.dtu.dk/plab/*, 2000.

[24] Tom Krauss. Multi - a multiple matlab process simulation engine. 2000.

[25] Daniel D. Lee. Pmi toolbox. *ftp://ftp.mathworks.com/pub/contrib/v5/tools/PMI*, 1999.

[26] C. Moler. Why there isn't a parallel matlab. *Cleve's corner, Mathworks Newsletter*, 1995.

[27] B.R. Norris. An environment for interactive parallel numerical computing. Technical Report 2123, Urbana, Illinois, 1999.

[28] Sven Pawletta, Andreas Westphal, Thorsten Pawletta, Wolfgang Drewelow, and Peter Duenow. Distributed and parallel application toolbox (dp toolbox) for use with matlab(r) version 1.4. 1999.

[29] M. Philippsen. Automatic alignment of array data and processes to reduce communication time on dmpps. *Proceedings of the Fifth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 1995.

[30] M. Quinn, A. Malishevsky, and N. Seelam. Otter: Bridging the gap between matlab and scalapack. *7th IEEE International Symposium on High Performance Distributed Computing*, 1998.

[31] B.T. Smith, J.M. Boyle, J.J. Dongarra, B.S. Garbow, Y. Ilebe, V.C. Kelma, and C.B. Moler. *Matrix Eigensystem Routines - EISPACK Guide*. Springer-Verlag, 2nd edition, 1976.

[32] Paul L. Springer. Matpar: Parallel extensions for matlab. *Proceedings of PDPTA*, 1998.

[33] Alpha Data Parallel Systems. Paramat. 1999.

[34] A.E. Trefethen, V.S. Menon, C.C. Chang, G.J. Czajkowski, C. Myers, and L.N. Trefethen. Multimatlab: Matlab on multiple processors. Technical report, 1996.

[35] Robert A. van de Geijn. *Using PLAPACK: Parallel Linear Algebra Package*. MIT Press, Cambridge, MA, USA, 1997.

[36] R. Clint Whaley and Jack J. Dongarra. Automatically tuned linear algebra software. Technical Report UT-CS-97-366, 1997.

[37] J. Zollweg. Cornell multitask toolbox for matlab
. *http://www.tc.cornell.edu/Services/Software/CMTM/*, 2001.