MATLAB*P 2.0: Interactive Supercomputing Made Practical

by

Long Yin Choy,
B.S., Computer Science,
Cornell University, 2000

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2002

Author . . . .
Department of Electrical Engineering and Computer Science
February 5, 2003

Certified by . . . .
Alan Edelman
Professor of Applied Mathematics
Thesis Supervisor

Accepted by . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Students

MATLAB*P 2.0: Interactive Supercomputing Made Practical

by

Long Yin Choy,

B.S., Computer Science,

Cornell University, 2000

Submitted to the Department of Electrical Engineering and Computer Science
on February 5, 2003, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

MATLAB [21] is one of the most widely used mathematical computing environment in technical computing. It is an interactive environment that provides high performance computational routines and an easy-to-use, C-like scripting language. Mathworks, the company that develops MATLAB, currently does not provide a version of MATLAB that can utilize parallel computing [27]. This has led to academic and commercial efforts outside Mathworks to build a parallel MATLAB, using a variety of approaches. In a survey [9], which will be expanded in this thesis, 26 parallel MATLAB projects utilizing four different approaches have been identified.

MATLAB*P [19] is one of the 26 projects. It makes use of the backend support approach. This approach provides parallelism to MATLAB programs by relaying MATLAB commands to a parallel backend. The main difference between MATLAB*P and other projects that make use of the same approach is in its focus. MATLAB*P aims to provide a user-friendly supercomputing environment in which parallelism is achieved transparently through the use of objected oriented programming features in MATLAB. One advantage of this approach is that existing scripts can be run in parallel with no or minimal modifications.

MATLAB*P 1.0 was developed as a proof of concept to the above idea. It suffered from deficiencies which limited its usefulness. The parallel numerical libraries used by MATLAB*P 1.0 is not portable. Secondly, it has no user documentation, and the code is organized in such a way that it is very hard to extend. Furthermore, it lacks administration tools, making installation and configuration a daunting task.

This thesis describes the development of MATLAB*P 2.0, which is a complete rewrite of MATLAB*P. Its goal is to solve the above problems and bring new features into MATLAB*P, making it a truly practical interactive supercomputing environment. Furthermore, although this project starts out as a parallel MATLAB, it is almost trivial to make the system work with other frontend such as Maple or Mathematica.

Thesis Supervisor: Alan Edelman
Title: Professor of Applied Mathematics

Acknowledgments

First of all I would like to thank my advisor Alan Edelman for his support and guidance. Without his insights there would not be a MATLAB*P in the first place, and again without his insights there would not be this project to improve MATLAB*P. Alan has firmly believed in this project from start. He and I share the vision that one day MATLAB*P would become the premier tool in parallel scientific computing. I am grateful to him for giving me the chance to make it happen.

I owe to Parry Husbands for laying the foundation to the MATLAB*P project by writing the first version. Most of the code has been completely rewritten in the new version, but a lot of the original ideas remain. Also he was the force behind the PPEngine extension which allows MultiMATLAB and MatlabMPI like functionalities in MATLAB*P.

Also I would like to thank everyone at Mathworks for making MATLAB such a wonderful product. I have benefited from MATLAB as an undergraduate, using it for simple calculations and data visualization (as well as my math homework). I feel excited at the opportunity to add on to a software that I have learned to love.

Some of the students in 6.338/18.337 Applied Parallel Computing has tried out and gave valuable comments on MATLAB*P. I would especially like to thank Oskar Bruning, Jack Holloway and Adnan Sulejmanpasic for developing the visualization package for MATLAB*P as their term project.

Last, but not least, I would like to thank my family and all my friends for suppporting me, and making my time at MIT such an enjoyable one. Also I would like to thank the MIT Table Tennis Club and Boston Table Tennis Club for providing the getaway that I often needed when working on the thesis. Finally, I want to give special thanks to Prof. Charles Van Loan, who introduced me into the world of scientific computing and made all this possible.

Contents

List of Figures

List of Tables

Chapter 1

Background

MATLAB [21] is one of the most widely used tools in scientific and technical computing. It started in 1970s as an interactive interface to EISPACK [33] and LINPACK [12], a set of eigenvalue and linear system solution routines. It has since grown to a feature rich product utilizing modern numerical libraries like ATLAS [38] 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. It is a very useful tool to a wide range of 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.

Because of its roots in serial numerical libraries, MATLAB has always been a serial program. However, as modern engineering and simulation problems become more and more complex, space and computing time requirements for solutions skyrocketed, and a serial MATLAB is no longer able to handle them. And compared to similar softwares like Maple and Mathematica, MATLAB is known to have a larger number of users. For example, at MIT, MATLAB is the dominant mathematical computing software used by the student body.

The above factors sparked a lot of interest in creating a parallel MATLAB. In a recent survey [9], 26 different parallel MATLAB projects have been identified. 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 products that enables the use of MATLAB in product development. Also their approaches to making MATLAB parallel are different: some compile MATLAB script 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.

Chapter 2

Parallel MATLAB Study

In a survey by Choy [9], 26 parallel Matlab projects have been identified. The goal here is to expand the survey into a systematic study of parallel MATLAB software, in the hope of identifying the best approach to such software.

The 26 parallel MATLAB projects can be divided into four categories, each of them utilizing a different approach:

2.1 Parallel MATLAB Approaches

2.1.1 Message Passing in MATLAB

Software that make use of this approach: MatlabMPI [22], MultiMATLAB [36], CMTM [39], DPToolbox [29], MPITB/PVMTB [5], MATmarks [2], PTToolbox[18]

Making use of multiple MATLAB instances running on a cluster of machines or a machine with multiple processors, this approach provides message passing routines between the MATLAB processes. The enables users to write their own parallel programs in MATLAB in a fashion similar to writing parallel programs with a compiled language using MPI. This approach has the advantage of flexibility: users in theory are able to build any parallel software system in MATLAB the way that they build the system in compiled languages with MPI.

2.1.2 Embarrassingly Parallel

Software that make use of this approach: Matlab Parallelization Toolkit [17], Multi [24], Paralize [1], PMI [25], PLab [23], Parmatlab [3]

This approach also make use of multiple MATLAB processes. The main difference is that the approach does not provide coordination between the MATLAB processes. Instead, a parent process pass off data to the child processes. Then all processes work on its local data, and return 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 much of 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.

Notice that this approach provides functionalities that are a subset of message passing.

2.1.3 Backend Support

Software that make use of this approach: NetSolve [4], DLab [28], Matpar [34], PLAPACK [37], PARAMAT [35], MATLAB*P [19]

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 software e.g. [4] [28], the data resides in MATLAB and are passed to the engine and back. And for other software e.g. [19], 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.

2.1.4 Compilation

Software that make use of this approach: Otter [32], RTExpress [20], ParAL [31], FALCON [10], CONLAB [13], MATCH [6], Menhir [8]

These software compile MATLAB scripts into an executable, sometimes translating the scripts into a compiled language as an intermediate step. Some software e.g. [32] [13] links the compiled code with parallel numerical libraries, while some software e.g. [10] 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.

The following tables summarizes the software and their approaches:

MultiMATLAB [36]	Cornell University
CMTM [39]	Cornell University
DP-Toolbox [29]	University of Rostock, Germany
MPITB/PVMTB [5]	University of Granada, Spain
MATmarks [2]	UIUC
PT Toolbox [18]	Wake Forest University
MatlabMPI [22]	MIT Lincoln Lab

Table 2.1: Message Passing

Matlab Parallelization Toolkit [17]	Linkopings Universitet, Sweden
MULTI Toolbox [24]	Purdue University
Paralize [1]	Chalmers University of Technology, Sweden
PMI [25]	Lucent Technologies
PLab [23]	Technical University of Denmark
Parmatlab [3]	Northeastern University

Table 2.2: Embarrassingly Parallel

Netsolve [4]	University of Tennessee, Knoxville
DLab [28]	UIUC
Matpar [34]	Jet Propulsion Lab
PLAPACK [37]	University of Texas at Austin
Paramat [35]	Alpha Data Parallel Systems
MATLAB*P [19]	MIT

Table 2.3: Backend Support

Otter [32]	Oregon State University
RTExpress [20]	Integrated Sensors Inc.
ParAL [31]	University of Sydney, Australia
FALCON [10]	UIUC
CONLAB [13]	University of Umea, Sweden
MATCH [6]	Accelchip Inc.
Menhir [8]	IRISA, France

Table 2.4: MATLAB Compiler

2.2 Comparison between the approaches

One good way to contrast the approaches is to look at what each approach will enable to user to do.

The message passing approach, used by MultiMATLAB, which is one of the earliest parallel MATLAB software, basically recreates inside MATLAB the basic MPI routines. By starting up multiple MATLAB instances, it simulates a standard MPI program where multiple instance of the same program is running concurrently. This is the most low level and flexible: the user has fine control over the communications, and in theory can build very complicated systems using these low level constructs. However, this means that the approach does not help users that do not know MPI or do not want to write MPI programs, which can be an error-prone process.

On the other hand, backend support tries to hide the nuts and bolts of the parallelism away from the users. The backend server provide a set of parallel routines which can be called from MATLAB. Data is passed back and forth from MATLAB to the backend server. This approach uses MATLAB as a frontend; all parallel computations are done elsewhere. It has the advantage that the users do not need to know much about parallel computing: they just need to learn a set of new commands. However this approach is also the most inflexible. If the user needs functionality not provided by the server, MPI code has to be written to extend the server.

Single program multiple data (SPMD) operations are the strong point of software making use of the embarrassingly parallel approach. Using multiple MATLAB instances, this approach allows the user to run the same MATLAB script on different data in parallel. Also some software (e.g. Matlab Parallelization Tooklit) in this category provides constructs that simulates a parallel for-loop. This approach is no different from manually starting up a lot of MATLAB instances and running the same script on each - however it makes the process much easier. Although it sounds quite limiting, a lot of real applications (for example in image processing) can make use of this approach effectively.

The MATLAB compiler approach is very different from the three other approaches.

The MATLAB script to be parallelized is first translated to an intermediate language (e.g. C or FORTRAN), then linked with parallel numerical libraries like ScaLAPACK [7]. The actual MATLAB program is not required here, as the compiler will take MATLAB scripts on the command line. Usually a hook (through the MATLAB MEX interface) is provided to communicated between the compiled program and an MATLAB instance, but it is not necessary. This way MATLAB scripts, whose syntax is very suitable for matrix manipulations, can be used to build systems that are deployed in environments with no MATLAB.

2.3 What is the best approach?

From the previous section, we can see that the four approaches cater to different audiences. Message passing is appealing to system builders, while backend support is more suitable for end-users that want to avoid writing parallel programs. Embarrassing parallel approach is especially strong in SPMD scenarios, and the compiler approach is good for specialized systems like embedded systems.

A general purpose parallel MATLAB software should appeal to as many users as possible, in order to acquire the necessary user base that will ensure its long term survival. This means that the software should provide as many features as possible - a synthesis between the four approaches.

Parallel MATLAB software that combines two approaches are already available. All of them combines message passing with embarrassingly parallel capabilities, since they are quite similar. Examples of such software are MultiMATLAB, Matlab Parallelization Toolkit, and MatlabMPI. No software has successfully combined more than two approaches, and this is part of what MATLAB*P 2.0 addresses.

2.4 Status of parallel MATLAB projects

Actively supported	Available	Inactive
CMTM	MultiMATLAB	MATmarks
DP-Toolbox	MULTI Toolbox	PT Toolbox
MPITB/PVMTB	Paralize	DLab
MatlabMPI	PMI	Otter
PLab	Parmatlab	ParAL
Matlab Parallelization Toolkit	Matpar	FALCON
Netsolve	PLAPACK	Menhir
MATLAB*P	Paramat	
RTExpress	CONLAB	
MATCH		

Table 2.5: Status of various projects

Explanation:

- Actively supported - a recently updated webpage (within 1 year). Source code or binary available for download or purchase.
- Available - a dormant (not updated for 1 year) webpage. Source code or binary available for download or purchase.
- Inactive - a dormant (not updated for 1 year) or missing webpage. Source code or binary not available. Some of these projects only exists as citations in other papers.

Chapter 3

MATLAB*P

MATLAB*P [19] is a parallel MATLAB project started by P. Husbands, as part of his Ph.D. thesis, with help from C. Isbell and supervision under A. Edelman. It is a parallel MATLAB in the 'backend support' category. Note that although this project starts out as a parallel MATLAB project, the backend support paradigm is applicable to any other frontend software, e.g. Mathematica or Maple. In fact, it is almost trivial to build the necessary client-side software to make the server compatible with Maple or Mathematica.

The main idea of MATLAB*P is to build a user friendly supercomputing environment. This is different from MatPar [34] or NetSolve [4], which aims to create a general computational infrastructure. By making use of the operator overloading features in MATLAB, MATLAB*P integrates the parallelism seamlessly into MATLAB.

3.1 Structure of the system

The system uses a client-server architecture. The essential pieces of the system are the server, the client, and the packages.

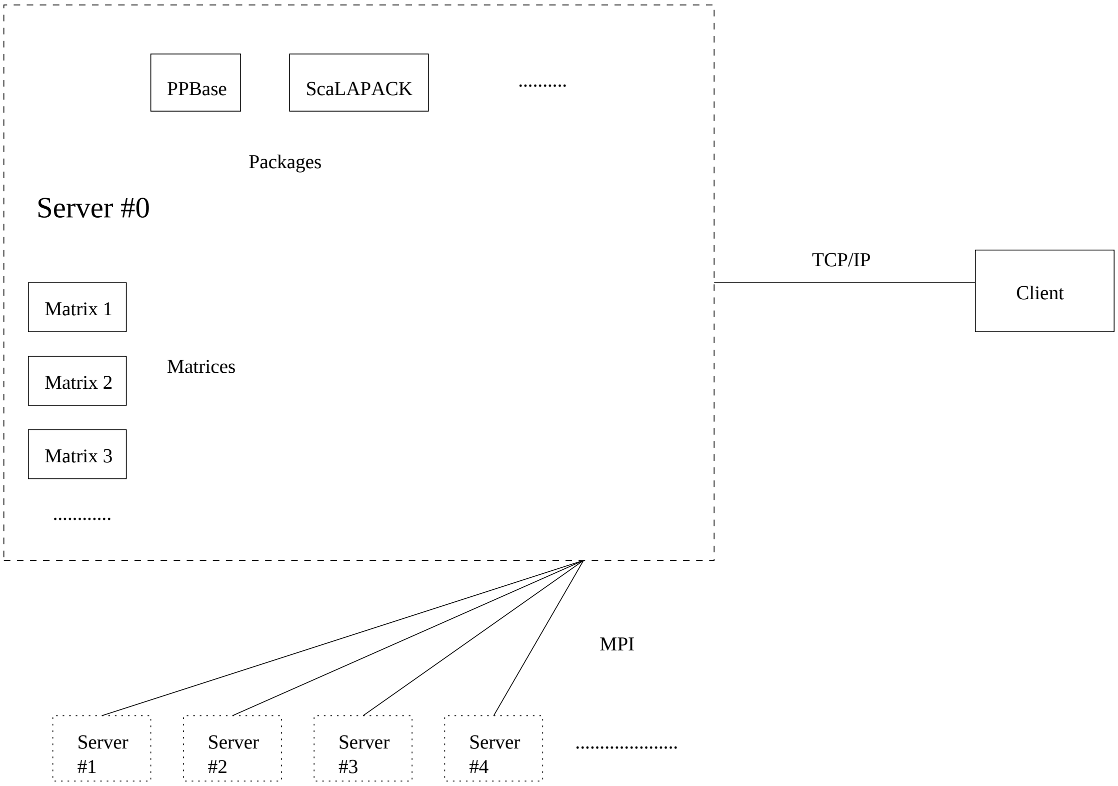

Figure 3-1: Structure of MATLAB*P 1.0

3.1.1 Server

The server is responsible for storing and managing the data, maintaining the packages, and making calls to the packages to perform the actual computations. The server is a distributed application, meaning a copy of the server resides on each node of the parallel computer. Coordination between the copies is through MPI.

3.1.2 Client

The client is responsible for communication between MATLAB and the server. The client consist of MATLAB scripts which defines a new matrix class for the server matrices, and a C program using the MEX interface which is responsible for packing and unpacking messages to and from the server. The client communicates with the server through TCP socket.

3.1.3 Packages

Packages are where the actual computations are performed. They are interfaces to routines in parallel libraries like ScaLAPACK [7]. Besides making the appropriate function calls, packages are also responsible for converting the matrices into formats appropriate for the underlying libraries. Users can write their own packages which can be dynamically loaded by the server.

3.2 MATLAB*P 1.0

MATLAB*P 1.0 was developed by P. Husbands as a proof of concept. It provides the following functionalities in the backend:

1. Support for real distributed matrices

2. Support for single and double precision arithmetic

3. Custom code for PBLAS-like [30] routines

4. ScaLAPACK routines

5. eigs and svds routines from PARPACK [26]

6. FFT routines from Sun S3L library

3.3 Using the system from MATLAB

3.3.1 *p

The key to the system lies in the *p variable. It is an object of *layout* class in 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);
```

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.

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

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.

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

returns the result to MATLAB.

3.3.2 Existing scripts reuse

```
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. 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.

3.4 Problems with MATLAB*P 1.0

MATLAB*P 1.0 was written as a proof of concept in a Ph.D. thesis. Because of this reason, it lacks some important features and suffer from some deficiencies which limit its usefulness to a general audience:

- The first, and the most important deficiency, is the lack of documentation regarding installation and usage. This discourages potential users from trying out the software, because installation and configuration of MATLAB*P is not intuitive.

- The backend is dependent on code that is not platform-independent. For example, the backend uses the Sun S3L library for its FFT routine, which only works on Sun machines.

- Lacks support for general matrix distribution type. The only distribution types supported for dense matrices are row and column distribution. Two dimensional block cyclic distribution, which generally leads to significant higher performance, is not supported.

- No complex number support.

Part of the goal of MATLAB*P 2.0 is to address these deficiencies.

Chapter 4

MATLAB*P 2.0

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 studied the existing parallel MATLAB projects, some of which are defunct and some of which are prospering, and figure out what makes a parallel MATLAB project successful. Our conclusion is as follows:

1. Clear, adequate documentation

   Clear documentation for installing, configuring and using the software should be available. This is especially important for systems like MATLAB*P or MatlabMPI, for which the installation procedure is complex. Without good documentation, users will be unwilling to try out the software, or will easily get stuck and give up during the installation.

2. Support from the authors

   Support from the authors can help users solve problems that are not documented, and also maintain the long term integrity of the project by patching the software and updating the documentation. Also support from the authors will give users confidence in using the software, as problems they will feel that problems they encounter in the software will not get them stuck, and the software is more likely to be alive in the long run.

3. Success stories

   Since there are so many choices for freely available parallel MATLAB software, users will tend to try out the ones that people have already obtained success with in real-world applications. Thus obtaining and documentating such success stories is necessary to attract new users.

4. User base achieving a critical mass

   A sufficient number of users need to be amass to create enough discussion to keep the project alive. This is important because discussion in the form of for example a mailing list will give new users quick access to help. Also it is a very good way to get suggestion for extensions and bug reports.

5. A good set of features

   Finally, in order to attract users with a variety of applications on hand, the software should provide a broad set of features.

We use the above as guideline in developing and maintaining MATLAB*P 2.0.

4.1 Development of MATLAB*P 2.0

MATLAB*P 2.0 is a complete rewrite of MATLAB*P in C++. The code is organized in a way to ensure easy extension of the software.

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 commucation between server processes. It mainly controls broadcasting of commands and arguments from head

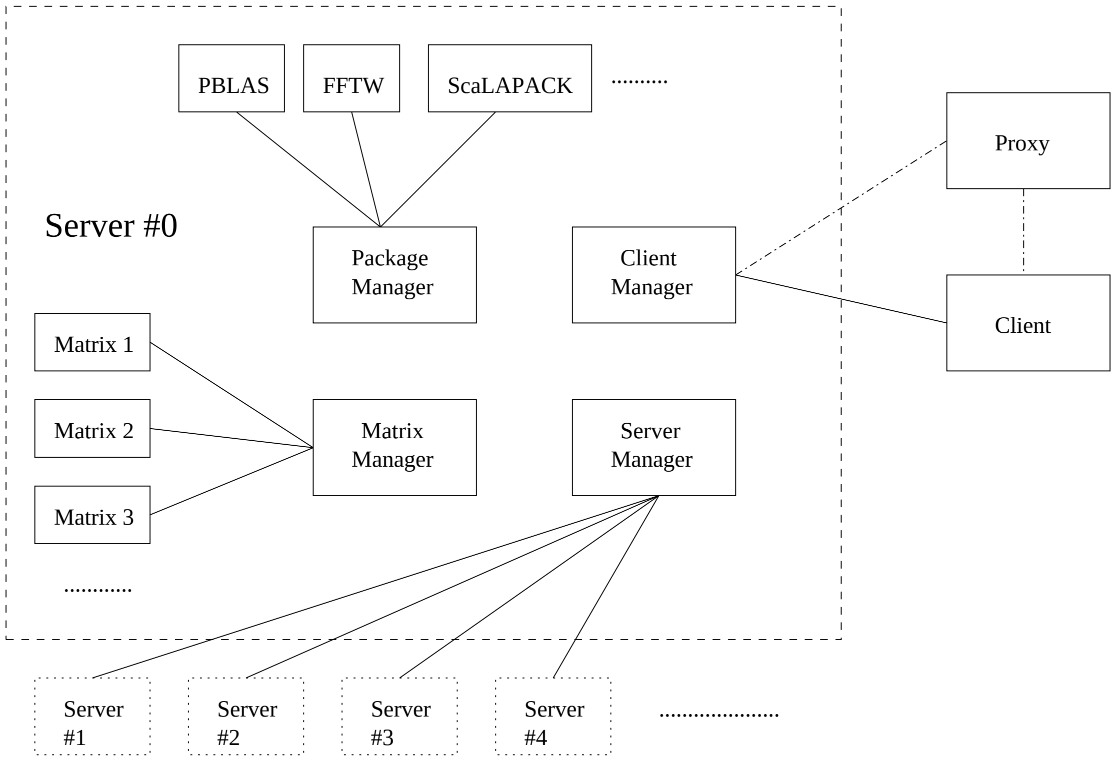

Figure 4-1: Structure of MATLAB*P 2.0

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.

4.2 New features and improvements in MATLAB*P 2.0

4.2.1 Two-dimensional Block Cyclic Distribution

MATLAB*P 1.0 supports row and column distribution of dense matrices. These distributions assign a block on contiguous row/column of a matrix to successive processes. Each process receives only one block of the matrix. These distribution suffers from load balancing issues.

A good example that illustrate the load balancing problem is Gaussian elimiination. Recall the Gaussian elimiation algorithm solves a square linear system by 'zeroing out' the subdiagonal entries.

```
for i = 1:n
  for j = i+1:n
    A(j,i:n) = A(j,i:n) - A(i,i:n)*A(j,i)/A(i,i);
  end
end
```

The algorithm proceeds column by column from left to right. For column distribution, as the algorithm go past the columns stored in process i, process 0, . . . , i will be idle for the rest of the computation. Row distribution suffers from a similar deficiency.

Two-dimensional block cyclic distribution avoid this problem by 'dealing out' blocks of the matrix to the processes in a cyclic fashion. This is illustrated in figure 4-2. The figure shows the two-dimensional block cyclic distribution for four processes in a 2x2 grid. This way all processes remains busy throughout the computation, thus achieving load balancing. Furthermore, this approach carries the additional advantages of allowing BLAS level-2 and BLAS level-3 operations on local blocks. A

1	2	1	2	1	2	1	2
3	4	3	4	3	4	3	4
1	2	1	2	1	2	1	2
3	4	3	4	3	4	3	4
1	2	1	2	1	2	1	2
3	4	3	4	3	4	3	4
1	2	1	2	1	2	1	2
3	4	3	4	3	4	3	4

Figure 4-2: Two-dimensional block cyclic distribution

detailed discussion of the advantages of two dimensional block cyclic distribution can be found in Dongarra, Van De Geijn and Walker [11].

4.2.2 Complex Number Support

MATLAB*P 1.0 only supports real matrices. In the new version complex number support is added. This is particularly important for FFT routines and eigenvalue routines.

Adding this support is the main reason why it is necessary to rewrite the code from scratch. Interaction between real and complex matrices, and hooks for complex number routine calls to numerical libraries are extremely error prone if they were to be added on top of MATLAB*P 1.0.

4.2.3 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.

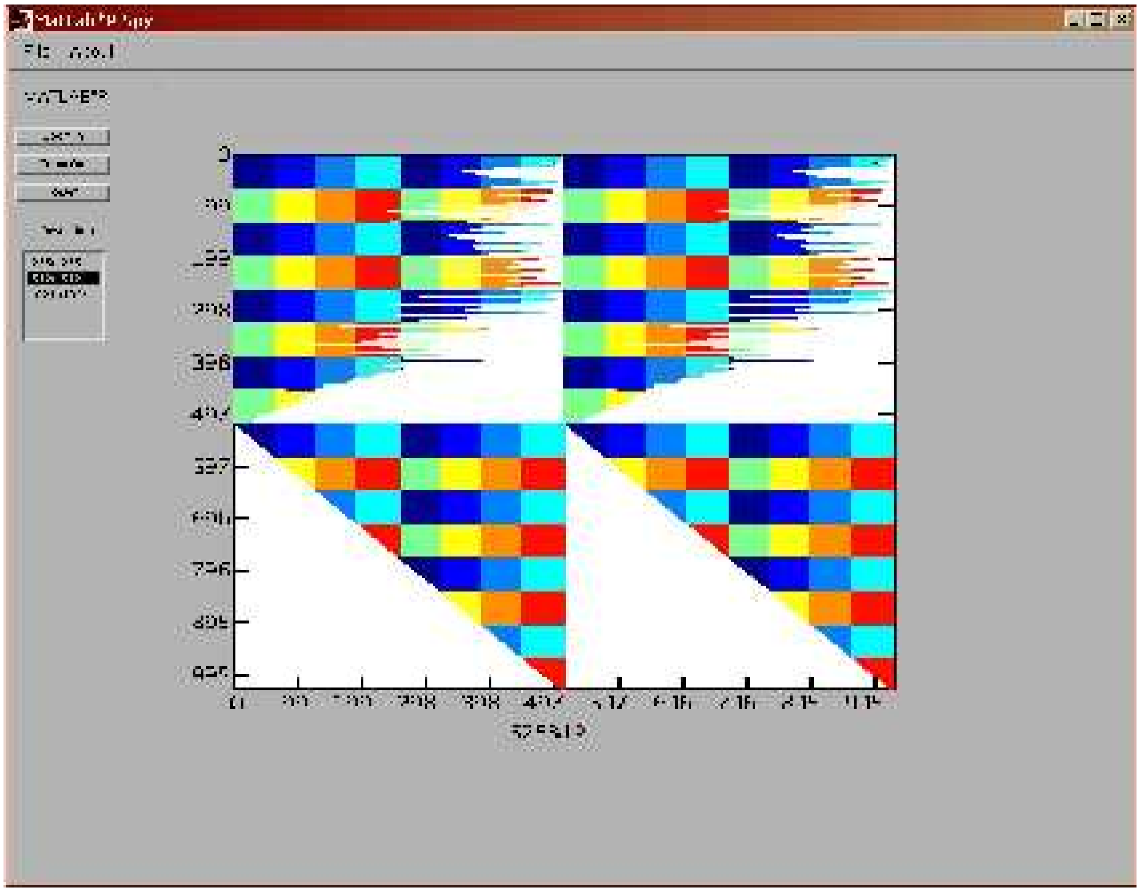

Figure 4-3: ppspy on 1024x1024 matrix on eight nodes

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 4-3, 4-4, 4-5 shows the routines in action.

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.

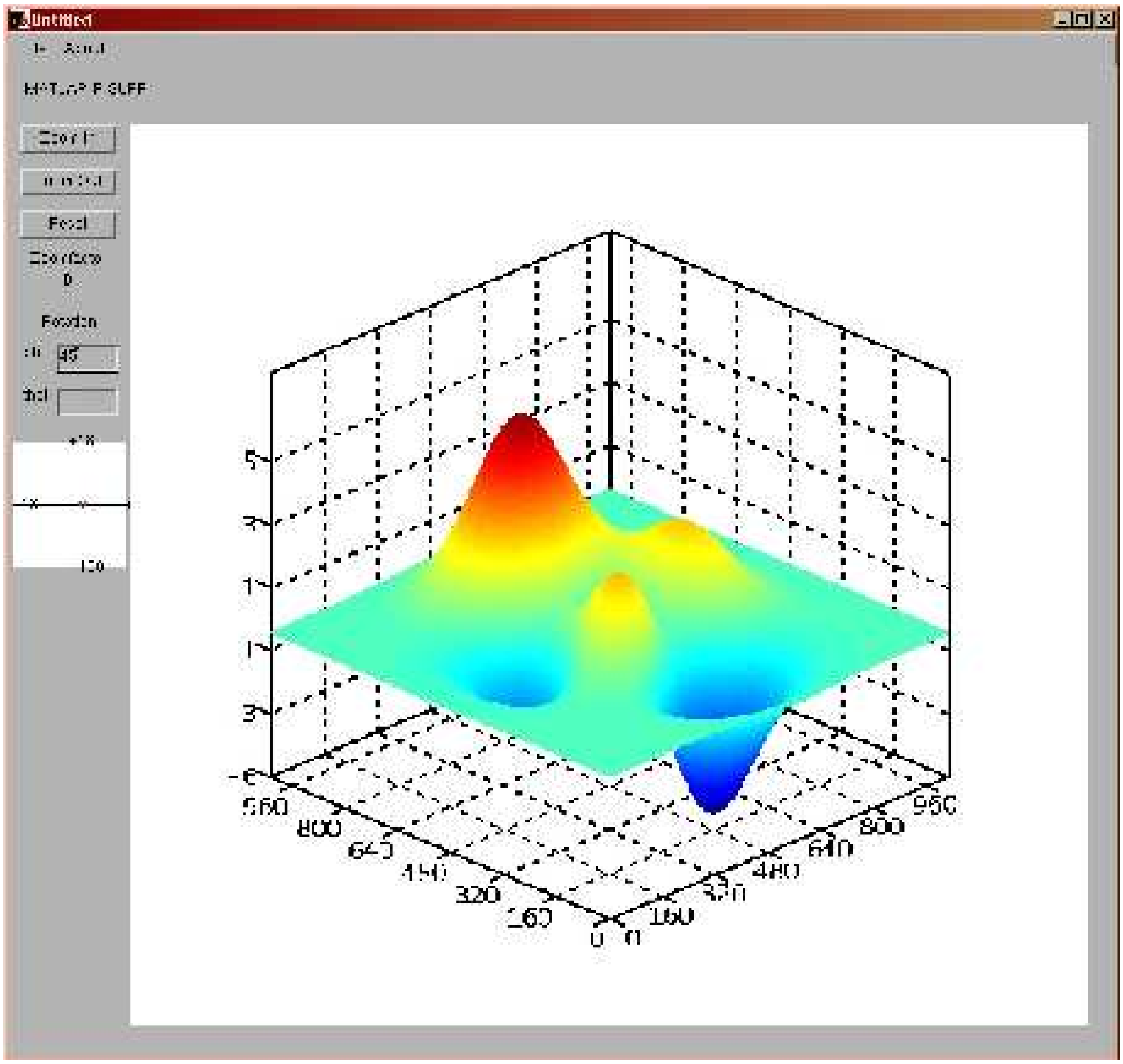

Figure 4-4: ppsurf on the 1024x1024 'peaks' matrix

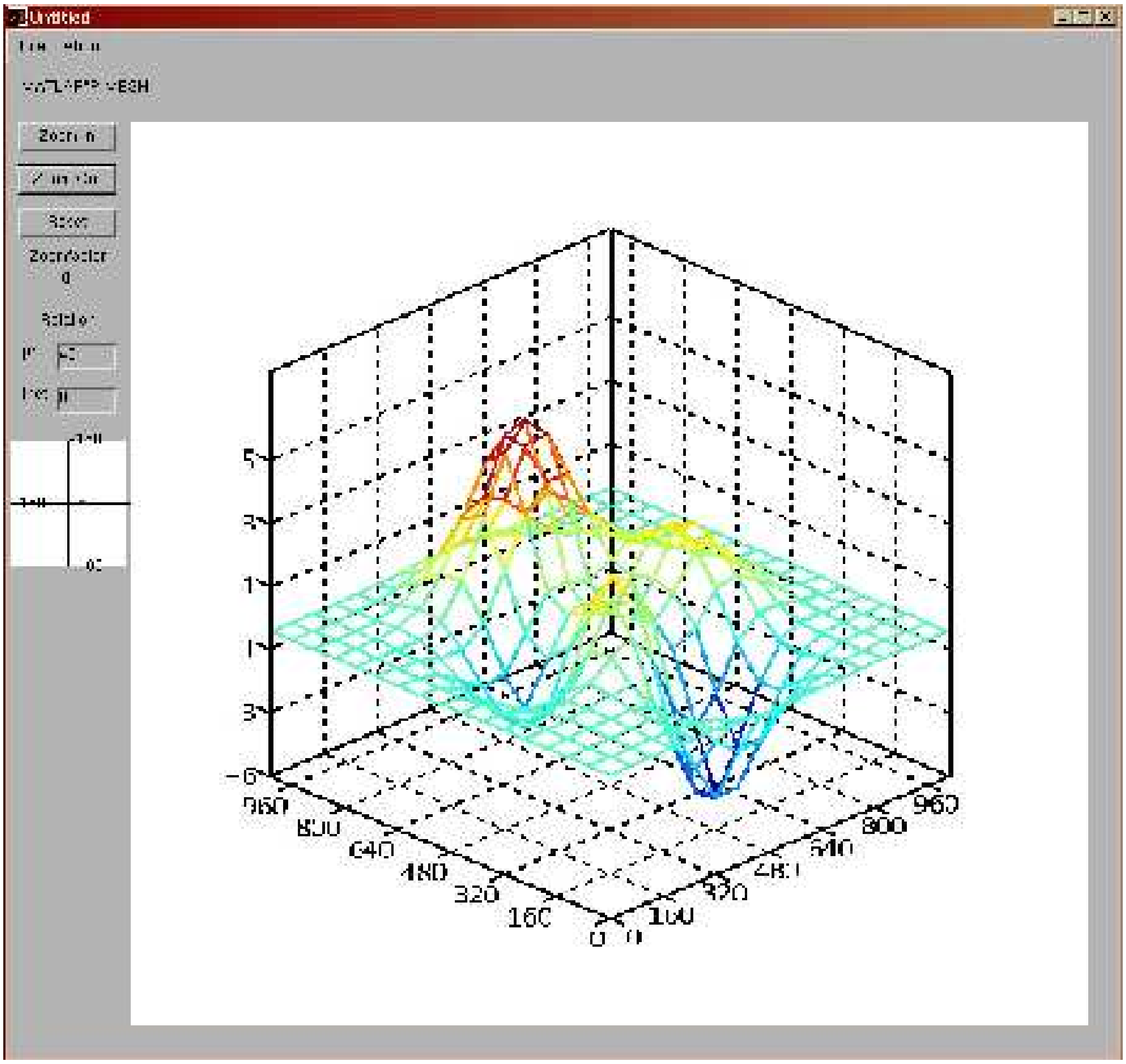

Figure 4-5: ppmesh on the 1024x1024 'peaks' matrix

4.2.4 Data Parallel/Embarassingly Parallel Operations

One of the goals of the project is to make the software to be useful to as wide as audience as possible. In order to achieve this, we found that it would be fruitful to combine other parallel MATLAB approaches into MATLAB*P.

In conjuction 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.

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;
```

Besides providing embarrassingly parallel capabilities like in the above examples, we are also interested in integrating MPI-like functionalities into the system. We do so by making use of the MatlabMPI [22] package. This package has the advantage of being written in pure MATLAB code, so integrating simply means making calls to this package through PPEngine.

```
>> % Example 3
>> piapprox = mmmpi('cpi',100);
```

This example calls cpi.m with 100. This calls a MATLAB version of the popular pi-finding routine distributed with MPICH, cpi.c. All the MPI calls are simulated within MATLAB through MatlabMPI.

4.3 Routines provided

Basic operations		Resulting distribution
a + b	Binary addition	Same as a
a - b	Binary subtraction	Same as a
-a	Unary minus	Unchanged
+a	Unary plus	Unchanged
a .* b	Element-wise multiplication	Same as a
a * b	Matrix multiplication	Same as a
a ./ b	Right element-wise division	Same as a
a .\ b	Left element-wise division	Same as b
a / b	Matrix right division	Same as a
a \ b	Matrix left division	Same as b
a:d:b , a:b	Colon operator	Row
a'	Complex conjugate transpose	Unchanged
a.'	Matrix transpose	Unchanged
a(s1,s2,...,sn)	Subscripted reference	Local
a(s1,...,sn) = b	Subscripted assignment	Unchanged

Table 4.1: Overloaded operators in MATLAB*P 2.0

abs	Absolute value	Unchanged
round	Rounding	Unchanged
ceil	Ceiling	Unchanged
floor	Floor	Unchanged
sqrt	Element-wise square root	Unchanged
sin	Sine	Unchanged
sinh	Hyperbolic Sine	Unchanged
asin	Arc sine	Unchanged
cos	Cosine	Unchanged
cosh	Hyperbolic Cosine	Unchanged
acos	Arc cosine	Unchanged
tan	Tangent	Unchanged
tanh	Hyperbolic tangent	Unchanged
atan	Arc tangent	Unchanged

Table 4.2: Basic operations in MATLAB*P 2.0

Relational operations		Resulting distribution
a < b	Less than	Same as a
a > b	More than	Same as a
a <= b	Less than or equal to	Same as a
a >= b	More than or equal to	Same as a
a == b	Equality	Same as a
a & b	Logical AND	Same as a
$a \mid b$	Logical OR	Same as a
∼a	Logical NOT	Same as a

Table 4.3: Relational operations in MATLAB*P 2.0

Computational routines		Resulting distribution
fft(A)	Fast Fourier transform (1D)	Same as A
fft2(A)	Fast Fourier transform (2D)	Same as A
eig(A)	Eigenvalues and eigenvectors	Column
eigs(A,k)	A few (k) eigenvalues/eigenvectors	Column
svd(A)	Singular value decomposition	Column
svds(A,k)	A few (k) singular values/vectors	Column
qr(A)	QR factorization	Same as A
lu(A)	LU factorization	Same as A
chol(A)	Cholesky factorization	Same as A
schur(A)	Schur decomposition	Same as A
hess(A)	Hessenberg form	Same as A
inv(A)	Matrix inverse	Same as A
tril(A)	Lower triangular part	Same as A
triu(A)	Upper triangular part	Same as A
real(A)	Real part of a matrix	Same as A
imag(A)	Imaginary part of a matrix	Same as A

Table 4.4: Matrix operations in MATLAB*P 2.0

Chapter 5

Benchmarks

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

5.1 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

5.2 Timing Results

5.3 Interpretation

5.3.1 MATLAB*P and MATLAB

From the results, MATLAB*P on 4 processors begin to outperform MATLAB on single processor when the problem size is 2048 and upward. This shows that for

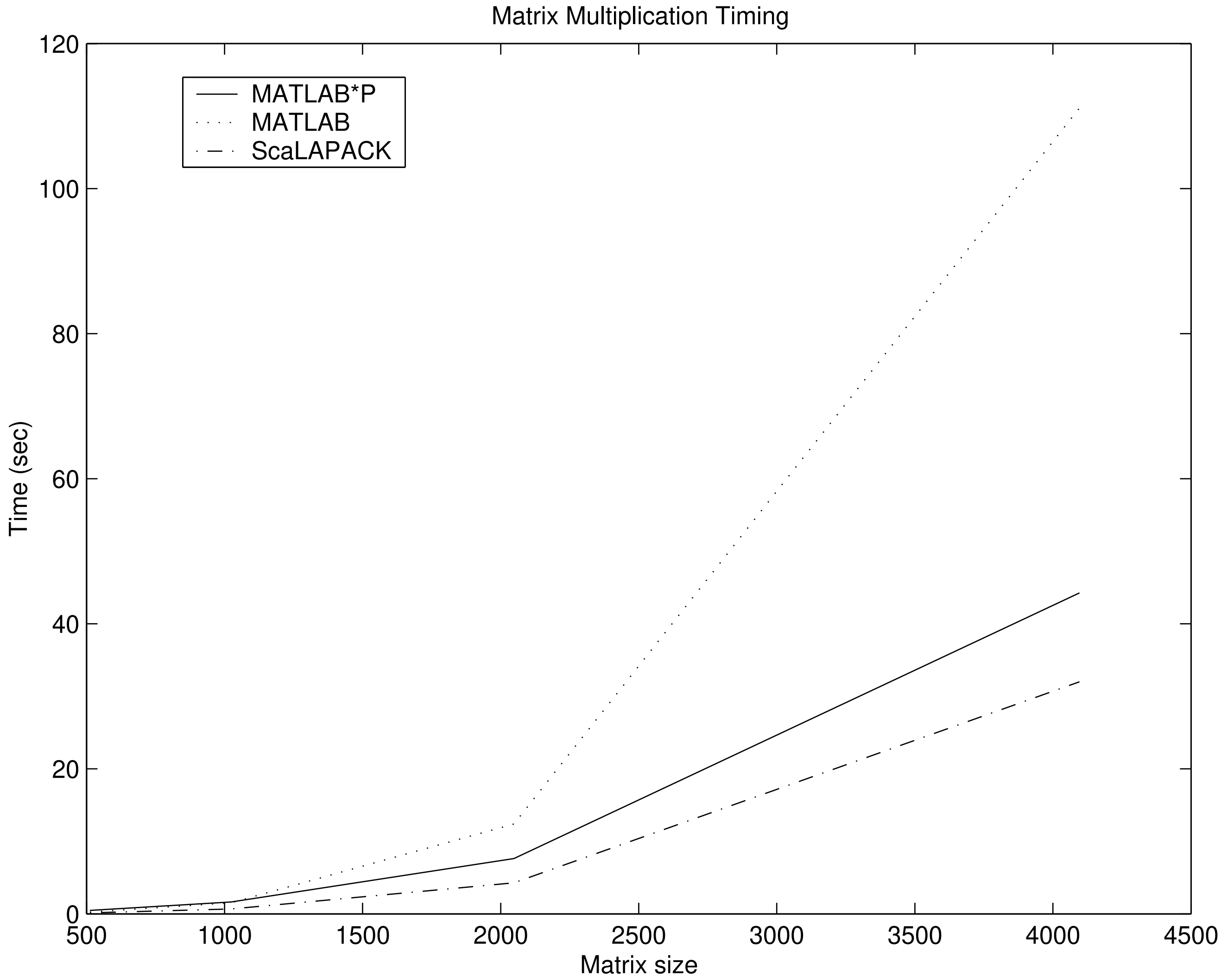

Figure 5-1: Matrix multiplication timing results

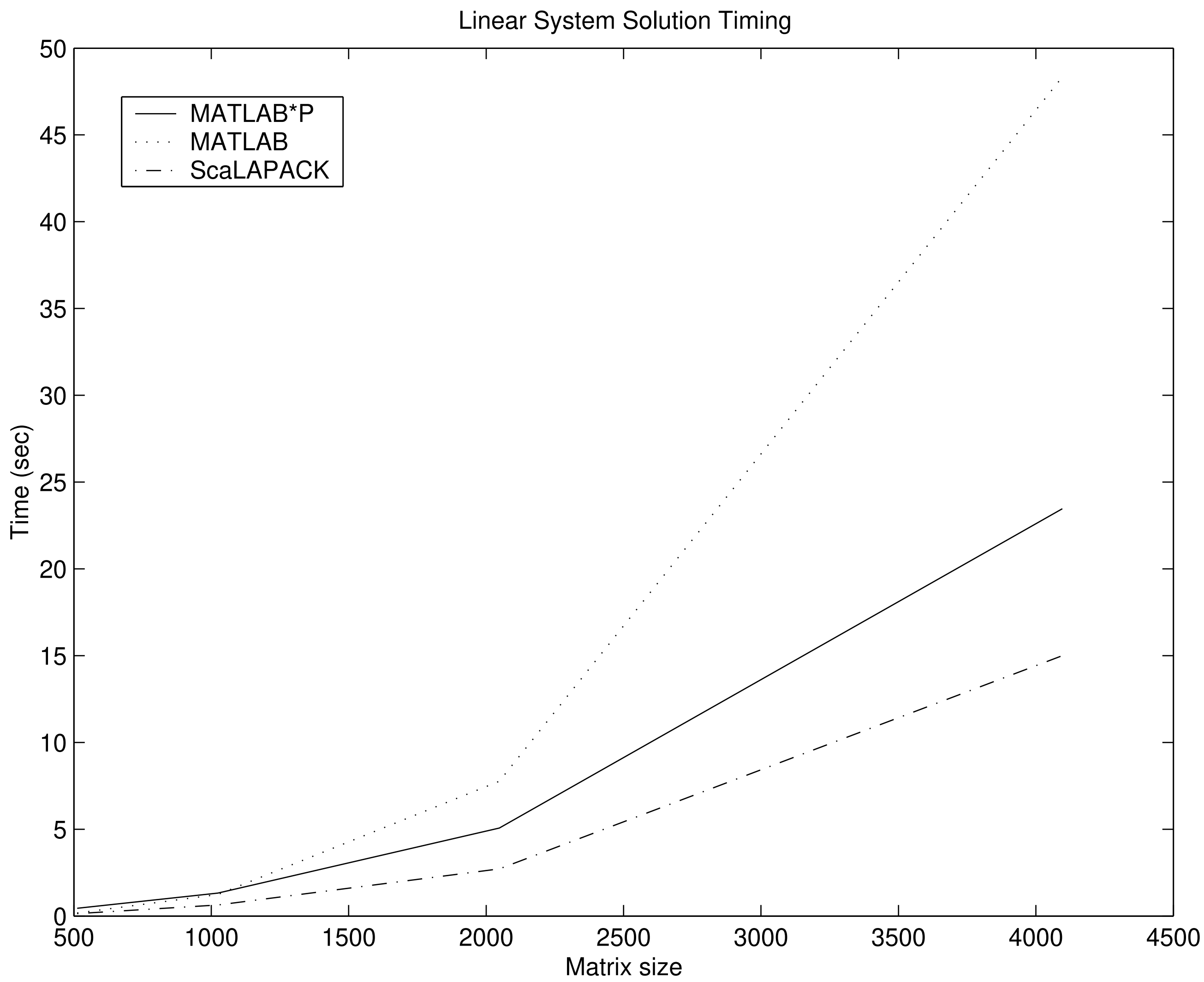

Figure 5-2: Linear system solve timing results

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, once the system is installed and configured.

5.3.2 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.

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.

Chapter 6

Future work

6.1 Sparse Matrix Routines

Because of time constraints, sparse matrix routines are not included in the release of MATLAB*P 2.0. If there is sufficient interest they will be added.

6.2 MATLAB*P on Grids of Beowulfs

Recently there has been a surge on interest in Beowulf-class supercomputers. Beowulfs are clusters of machines made from commodity-off-the-shelf (COTS) hardware. Example of COTS hardware includes 80x86 processors, DDR memory, and IDE harddrives.

The advantage of Beowulfs over traditional supercomputers like SGI Origin or IBM SP2 is that Beowulfs are much cheaper to build and maintain. For example, a Beowulf with 18 processors (1.533GHz) and 9GB RAM costs around $15000 at time of writing. On the other hand, a Sun Enterprise 250 Server, with 2 processors and 2GB RAM, costs around $16000. As Beowulfs generally run open source operating systems like Linux, the software cost is also cut substantially.

Along with the popularity of Beowulfs, there has also been rising interest in the concept of computational grids [15]. Its focus is on large-scale resource sharing, namely, resource sharing among geographically distributed individuals and institutions. The resources shared could be applications, databases, archival storage devices, visualization devices, scientific instruments, or CPU cycles. One of the most popular software that implements the Grid architecture is the Globus Toolkit [14].

We are interested in combining the above two very popular ideas, to see how MATLAB*P would run on a Grid of Beowulfs. In a recent parallel computing class co-offered at MIT and at the National University of Singapore (NUS), a group of students at NUS have taken up a project that attempted to make MATLAB*P run on a Grid joinig a Beowulf cluster at MIT and another Beowulf cluster at NUS. Both the clusters are Linux-based, and the Globus Toolkit was used. Also a student at MIT attempted to run MATLAB*P on a Grid of two MIT Beowulfs, also using the Globus Toolkit.

Both of these projects failed. Both of them failed at the point of setting up a working MPICH-G2 on the clusters. In order to have MATLAB*P run on the grid, it is necessary to build it upon a special, Grid-aware version of MPI. All the Beowulf clusters we used have private networks, meaning only designated 'front end' nodes are exposed to the outside world, while the other nodes are all hidden and form its own network. This caused MPICH-G2 to fail, because the MPI processes started up by MPICH-G2 were located in different private networks and were unable to contact one another.

This is very unfortunate, because we believe Grids of Beowulf clusters is a very powerful and cost-effective architecture. When we turned to the Globus discussion mailing list for help, we were told that this could be achieved by making MPICH-G2 aware of local MPI implementations in each Beowulf cluster and making use of them, but documentations on this setup is scarce and we were unable to proceed with this approach.

6.3 Parallel MATLAB Toolbox

One problem with the current state of MATLAB*P 2.0 is the difficulty in setting up the softare. MATLAB*P 2.0 depends on a number of established libraries for communication and computational routines: BLACS, MPI, PBLAS, ScaLAPACK, PARPACK, FFTW and SPRNG. Unless the system MATLAB*P 2.0 is running on already has these libraries up and running, installing MATLAB*P 2.0 would often be a painful process of tracking down bugs in the installation of libraries totally unfamiliar to the user.

A solution to this problem is to port the entire code into MATLAB scripts. This is best demonstrated by MatlabMPI, which implements basic MPI routines entirely in MATLAB scripts. There are several advantages to this approach.

1. Portability: First of all, the software would run where MATLAB runs. And because MATLAB is a commercial product by Mathworks, it is tested extensively to ensure its portability. By building the software upon MATLAB, we would be able to take advantage of this extensive testing.

2. Easy to install: Installing a system made up of MATLAB scripts would require no compilation. Just set the MATLAB path correctly and the system should be ready.

MatlabMPI is a good example that illustrates the strong points of this approach. Because MATLAB*P is MPI-based, there have been discussions of porting MATLAB*P to a purely MATLAB script code base, using MatlabMPI as the underlying communication layer.

If this were to be pursued, the resulting MATLAB*P would consist of a group of MATLAB processes working together. In addition to developing the equivalent of the current MATLAB*P code in MATLAB scripts, we have identified the following additional functionalities that have to be implemented:

1. Supporting libraries: One of the strong points of MATLAB*P is the leverage of established, high-performance parallel numerical libraries. In moving towards a purely MATLAB code base, since these libraries do not exist in MATLAB, those functions they provide will have to be implemented by ourselves. This means that all or part of ScaLAPACK, PARPACK, and FFTW have to be

implemented in MATLAB, debugged and maintained. Also as MATLAB scripts are interpreted instead of compiled, they do not get compiler optimization which optimizes memory access patterns and cache usage. Thus the resulting code would be slower.

2. MPI routines: MatlabMPI supports only basic MPI routines like Send, Recv, Size and Rank. As MATLAB*P utilizes more advanced features like Reduce and Scatter/Gather, they have to be implemented.

These alone would increase the size of the code base dramatically and make code maintenance harder than before, despite the move to MATLAB scripts, which is supposed to be easier to write and understand.

Yet still, it is interesting to investigate whether this approach would yield a better MATLAB*P. The first steps of the investigation would be to write routines to distribute data. The following list of routines should be implemented:

- A = zeros(m*p,n), to generate a row-distributed matrix which is consisted of MATLAB matrices.

- A = zeros(m,n*p), to generate a column-distributed matrix which is consisted of MATLAB matrices.

- A = zeros(m*p,n*p), to generate a 2D block cyclic distributed matrix which is consisted of MATLAB matrices.

- B = pp2local(A), which collects the distributed matrix into a local matrix

Also some simple computation routine to test the distribution would be helpful. A very good choice would be matrix-matrix multiplication.

Appendix A

Configuration and Installation of MATLAB*P 2.0

A.1 Configuration

A.1.1 Organization of the source tree

matlabpv2

client docs include matlab packages proxy server

Contents of the various directories:

- client - source code for the MATLAB*P client.
- docs - documentations, which include this guide and documentations to the packages routines.
- include - header files, used by both the client and the server.
- matlab - MATLAB scripts used by the client.
- packages - source code for the packages used by server.

- proxy - source code for the proxy. The proxy is needed when a firewall exists between the client and the server.
- server - source code for the MATLAB*P server.

A.1.2 PPMake.inc

All the configuration of MATLAB*P 2.0 takes place in the file PPMake.inc.

1. CC - the MPI C++ compiler
2. CCC - the regular C++ compiler, used for compiling the client.
3. OPTFLAG - optimization flag for the C++ compiler
4. OPTIONS - option flags.
   - -DGCC - GCC compiler is used
   - -DPPSTATIC - link packages statically (only supported method for now)
   - -DSPRNG - use SPRNG for parallel random number generation
   - -DSCALAPACKUNDERSCORE - specifies that routines in ScaLAPACK has one trailing underscore
   - -DBLACSUNDERSCORE - specifies that routines in BLACS has one trailing underscore
   - -DBLASUNDERSCORE - specifies that routines in BLAS has one trailing underscore
5. CFLAG - additional flags to the C++ compiler specified by the user
6. INTCFLAG - used internally
7. SHARED - C++ flags for generating shared libraries. Currently not supported.
8. PACKAGES, STATICLINKPACKAGES - these have to be modified in pairs. Base package, PBLAS and ScaLAPACK are required.

Entries added to PACKAGES	Entries added to STATICLINKPACKAGES	
../packages/PPBase.o	-DPPBASE	base package
../packages/PPPBlas.o	-DPPPBLAS	interface to PBLAS
../packages/PPFFTW.o	-DFFTW	interface to FFTW
../packages/PPScalapack.o	-DPPSCALAPACK	interface to ScaLAPACK
../packages/PPEngine.o	-DPPENGINE	data parallel interface
../packages/PPVisual.o ../packages/tdrender.o ../packages/surfrender.o ../packages/meshrender.o ../packages/spyrender.o ../packages/render.o	-DPPVISUAL	visualization toolkit

9. MATH - location of system math library

10. G2C - location of g2c (or f2c) library, only if needed

11. MATLABINC, MATLABLIB - location of MATLAB external interface includes and libraries, needed by PPENGINE package

12. BLAS - location of BLAS library

13. SCALAPACKPATH, PBLAS, REDIST, SCALAPACK, TOOLS - location of various components of scalapack. SCALAPACKPATH, which points to teh base directory of the ScaLAPACK installation, is required only for ScaLAPACK 1.6. PBLAS, REDIST, SCALAPACK and TOOLS are the locations of the 4 libraries.

14. BLACS - location of BLACS libraries and necessary interfaces

15. MPI - location of MPI libraries, usually not needed if CC is mpiCC

16. SPRNG - location of SPRNG random number generator library and include directory

17. LAPACK - location of LAPACK library

18. FFTW, FFTWINC - location of FFTW library and includes

19. GL - location of OpenGL libraries (needed by PPVisual package)

A.2 Installation

MATLAB*P 2.0 makes use of Makefile for compilation and installation. In the root directory of the installation, the Makefile has four targets:

- server_s - staticly linked server. It is the only supported mode currently.
- client - MATLAB*P client. NOTE that this target has to be modified to suit the architecture of your system.
- clean - cleans all .o files created
- realclean - cleans all .o and executables created.

To install, modify PPMake.inc as in section A.1.2, then do

```
make server_s
```

on the machine on which the server is to be installed. On the client machine,

```
make client
```

Note that the client and the server can be installed on the same machine.

Appendix B

Running MATLAB*P 2.0

B.1 Starting up the server

There are three different procedures for starting up the server, for different system setups:

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.

For clarity, all steps will be spelt out in the procedures, even though some steps are the same in all three procedures.

B.1.1 Assumptions

All the procedures assume the following:

- Server machine IP: 10.1.1.1
- Server executable: /home2/joeuser/matlabxp
- Proxy machine IP: 123.123.1.2 (external) / 10.1.1.254 (internal)
- Proxy executable: /home2/joeuser/ppproxy
- Client machine IP: 192.168.1.1
- Client matlab scripts installation: /home/joeuser/matlab/

B.1.2 Procedure A

1. Start up server by issuing the command:

   ```
   mpirun -np 4 /home2/joeuser/matlabxp
   ```

   or its equivalent in your MPI installation. As a reminder for general MPI practice, the executable *matlabxp* should be placed in a directory accessible from all the nodes to be used in the run, otherwise the run is likely to fail.

   You should see something like the following:

   ```
   Server - Server started
   PackageManager - loaded package: PPBase
   PackageManager - loaded package: PPPBlas
   PackageManager - loaded package: PPFFTW
   PackageManager - loaded package: PPScalapack
   PackageManager - loaded package: PPEngine
   PackageManager - loaded package: PPVisual
   Server - Initialized packages
   Server - Waiting for client connection, IP: 10.1.1.1 port: 33421
   ```

2. On the client machine, set up the environment variables to identify the server. In bash, do the following:

```
export PPHOST=10.1.1.1
export PPPORT=33421
```

In tcsh/csh, do the following:

```
setenv PPHOST 10.1.1.1
setenv PPPORT 33421
```

NOTE: If the client machine is an IRIX system, due to a bug in its gethostbyname() routine, you will need to provide its own IP address:

```
export PPMYIP=192.168.1.1
or
setenv PPMYIP 192.168.1.1
```

3. On the client machine, change directory to the matlab scripts installation:

   ```
   cd /home/joeuser/matlab
   ```

4. Start up MATLAB.

   ```
   matlab
   ```

5. Check the connection by issuing the command:

   ```
   ppping
   ```

   inside MATLAB. The command should return the number of processes used by the server.

B.1.3 Procedure B

1. Start up server by issuing the command:

   ```
   mpirun -np 4 /home2/joeuser/matlabxp
   ```

   or its equivalent in your MPI installation. As a reminder for general MPI practice, the executable *matlabxp* should be placed in a directory accessible from all the nodes to be used in the run, otherwise the run is likely to fail.

   You should see something like the following:

   ```
   Server - Server started
   PackageManager - loaded package: PPBase
   PackageManager - loaded package: PPPBlas
   PackageManager - loaded package: PPFFTW
   PackageManager - loaded package: PPScalapack
   PackageManager - loaded package: PPEngine
   PackageManager - loaded package: PPVisual
   Server - Initialized packages
   Server - Waiting for client connection, IP: 10.1.1.1 port: 33421
   ```

2. A proxy is required because it is assumed that the client machine 192.168.1.1 cannot connect to 10.1.1.1 directly. It must go through the frontend or proxy machine 123.123.1.2.

   On the proxy machine, start up the proxy with the server IP and port:

   ```
   ppproxy 10.1.1.1 33421
   ```

   The output would be similar to the following:

   ```
   Connecting to server at 10.1.1.1 port 33426
   Server connection established
   Waiting for client connection, IP: 123.123.1.2 port: 27549
   ```

NOTE: On a frontend machine where it has both an IP address on the private network and an IP address on the external network, the IP address printed out on the third line depends on the entries in /etc/hosts. You should always use the external IP for client connection regardless of this output.

3. On the client machine, set up the environment variables to connect to the proxy. In bash, do the following:

   ```
   export PPHOST=123.123.1.2
   export PPPORT=27549
   ```

   In tcsh/csh, do the following:

   ```
   setenv PPHOST 123.123.1.2
   setenv PPPORT 27549
   ```

   NOTE: If the client machine is an IRIX system, due to a bug in its gethostbyname() routine, you will need to provide its own IP address:

   ```
   export PPMYIP=192.168.1.1
   or
   setenv PPMYIP 192.168.1.1
   ```

4. On the client machine, change directory to the matlab scripts installation:

   ```
   cd /home/joeuser/matlab
   ```

5. Start up MATLAB.

   ```
   matlab
   ```

6. Check the connection by issuing the command:

   ```
   ppping
   ```

   inside MATLAB. The command should return the number of processes used by the server.

B.1.4 Procedure C

1. Start up server by issuing the command:

   ```
   mpirun -np 4 /home2/joeuser/matlabxp
   ```

   or its equivalent in your MPI installation. As a reminder for general MPI practice, the executable *matlabxp* should be placed in a directory accessible from all the nodes to be used in the run, otherwise the run is likely to fail.

   You should see something like the following:

   ```
   Server - Server started
   PackageManager - loaded package: PPBase
   PackageManager - loaded package: PPPBlas
   PackageManager - loaded package: PPFFTW
   PackageManager - loaded package: PPScalapack
   PackageManager - loaded package: PPEngine
   PackageManager - loaded package: PPVisual
   Server - Initialized packages
   Server - Waiting for client connection, IP: 10.1.1.1 port: 33421
   ```

2. A proxy is required because it is assumed that the client machine 192.168.1.1 cannot connect to 10.1.1.1 directly. It must go through the frontend or proxy machine 123.123.1.2. Also, to bypass the firewall, SSH tunneling would be needed. We will need to establish two SSH tunnels, one from proxy to client, and one from client to proxy.

   On the proxy machine, start up the proxy with the server IP and port. Also needed are the IP address and port of the SSH tunnel which forwards to the client. For simplicity, we assume that the SSH tunnel is established on the proxy machine, although this is not necessary. The port can be chosen arbitrarily above 1024.

```
ppproxy 10.1.1.1 33421 123.123.1.2 16798
```

The output would be similar to the following:

```
Connecting to server at 10.1.1.1 port 33426
Server connection established
Waiting for client connection, IP: 123.123.1.2 port: 27549
```

NOTE: On a frontend machine where it has both an IP address on the private network and an IP address on the external network, the IP address printed out on the third line depends on the entries in /etc/hosts. You should always use the external IP for client connection regardless of this output.

3. On the client machine, set up the SSH tunnel to connect to the proxy. The listening port can be chosen arbitrarily above 1024. This can be done by:

   ```
   ssh -L 5498:123.123.1.2:27549 123.123.1.2
   ```

4. Set up the environment variables to connect to the proxy. Note that as you are now using SSH tunneling, you should connect to your local listening port. In bash, do the following:

   ```
   export PPHOST=127.0.0.1
   export PPPORT=5498
   ```

   In tcsh/csh, do the following:

   ```
   setenv PPHOST 127.0.0.1
   setenv PPPORT 5498
   ```

   NOTE: If the client machine is an IRIX system, due to a bug in its gethostbyname() routine, you will need to provide its own IP address:

```
export PPMYIP=192.168.1.1
or
setenv PPMYIP 192.168.1.1
```

5. On the client machine, change directory to the matlab scripts installation:

```
cd /home/joeuser/matlab
```

6. Start up MATLAB.

```
matlab
```

7. Complete the SSH tunnel by attempting to connect to the proxy. This can be done by issuing in MATLAB the command:

```
ppping
```

On the proxy machine you will out similar to the following:

```
Client connection established, receiving client IP and port
Client found at 192.168.1.1 port 13472
Establish your SSH tunnel from 123.123.1.2:16798 to 192.168.1.1:13472 now.
ssh -L 16798:192.168.1.1:13472 192.168.1.1
Press ENTER when ready
```

8. On the proxy machine, issue the command on the fourth line of the output:

```
ssh -L 16798:192.168.1.1:13472 192.168.1.1
```

then press ENTER to continue the proxy program.

9. The SSH tunneling is complete. You should see the number of server processes as the output in MATLAB.

Appendix C

Using MATLAB*P 2.0

Most MATLAB commands have been overloaded to take advantage of the parallel backend. Besides that, some additional commands have been added for managing the parallel matrices. In this document we focus on the commands that do not originate from MATLAB.

C.1 Basic commands

- ppping, ppprocs - returns the number of processes in the server.
- ppclear(A) - remove the matrix A from the server.
- ppgcall - remove all matrices from the server.
- ppquit - disconnect from the server and cause the server to terminate.
- B = pp2matlab(A) - retrieve matrix A from the server and store it in local matrix B.
- A = matlab2pp(B) - transfter local matrix B to the server and store the handle to the server matrix in A.

C.2 Matrix creation routines

Four routines are typically used to create distributed matrices. Also they could be created as a result of operations on distribution matrices. The four routines are:

- zeros - creates a matrix of zeros
- ones - creates a matrix of ones
- rand - creates a matrix of uniformly distributed random numbers.
- randn - creates a matrix of normally distributed random numbers.

The distribution type of the server matrix depends on the placement of the *p variable. Using *ones* as an example,

- ones(2048*p,1024) - 2048x1024, row-distributed matrix of ones.
- ones(8192,400*p) - 8192x400, column-distributed matrix of ones.
- ones(16384*p,16384*p) - 16384x16384, 2D block cyclic-distributed matrix of ones.

C.3 Visualization routines

The visualization package is written by Bruning, Holloway and Sulejmanpasic, under supervision of Ron Choy. It provides the equivalent of three MATLAB visualization commands for distributed matrices, running in parallel.

- ppspy - equivalent of MATLAB spy
- ppsurf - equivalent of MATLAB surf
- ppmesh - equivalent of MATLAB mesh

NOTE: because of a technical issue involved, the refresh button on the figure window has to be pressed at least once in order for the image to be shown.

C.4 Data-parallel routines

The PPEngine package, written in conjuction with Parry Husbands, provides MultiMATLAB and MatlabMPI-like routines, essentially combining the three approaches. In order to make use of this package, MATLAB has to be installed or accessible from all the nodes on which the server is run.

The two commands in this system are:

- result = mm(fname,arg1,arg2,...) - calls Matlab function fname on all the nodes and put the result in a distributed matrix. At least one argument should be distributed and all the distributed arrays must be the same size. The result is the same size as the arrays. In other words the function must return a single double. Also, for now only scalar and string arguments to the function are supported.
- result = mmmpi(fname,arg1,arg2,...) - calls MatlabMPI function fname on all the nodes and return the result. Currently only scalars are supported for arguments.

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;
```

```
>> % Example 3
>> piapprox = mmmpi('cpi',100);
```

This example calls cpi.m with 100. This calls a MATLAB version of the popular pi-finding routine distributed with MPICH, cpi.c. All the MPI calls are simulated within MATLAB through MatlabMPI.

Bibliography

[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] 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.

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

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

[11] J.J. Dongarra, R. Van De Geijn, and D.W. Walker. A look at scalable dense linear algebra libraries. *Proceedings of the Scalable High-Performance Computing Conference 1992*, pages 372–379.

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

[13] 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.

[14] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.

[15] Ian Foster. The anatomy of the Grid: Enabling scalable virtual organizations. *Lecture Notes in Computer Science*, 2150, 2001.

[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] P. Husbands and C. Isbell. Matlab*p: A tool for interactive supercomputing. *The Ninth SIAM Conference on Parallel Processing for Scientific Computing*, 1999.

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

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

[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] Kristi Maschhoff and Danny Sorensen. A portable implementation of arpack for distributed memory parallel archituectures. *Preliminary proceedings, Copper Mountain Conference on Iterative Methods*, 1996.

[27] C. Moler. Why there isn't a parallel matlab . *http://www.mathworks.com/company/newsletter/pdf/spr95cleve.pdf*, 1995.

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

[29] 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.

[30] Parallel basic linear algebra subprograms . *http://www.netlib.org/scalapack/html/pblas_qref.html.*

[31] 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.

[32] 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.

[33] 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.

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

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

[36] 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.

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

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

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