



A Parallel Linear Algebra Server for Matlab-like Environments

Greg Morrow

and

Robert van de Geijn

Department of Computer Sciences

The University of Texas at Austin

Austin, Texas 78712

{morrow,rvdg}@cs.utexas.edu

An Extended Abstract Submitted to SC98

1 Introduction

Mathematical software packages such as Mathematica, Matlab, HiQ and others allow scientists and engineers to perform complex analysis and modeling in their familiar workstation environment. However, these systems are limited in the size of problem that can be solved, because the memory and CPU power of the workstation are limited. Obviously, the benefit of the interactive software is lost if the problem takes too long to run. With the advent of inexpensive “beowulf”-type parallel machines [4], and the proliferation of parallel computers in general, it is natural to wonder about combining the user-friendliness and interactivity of the commercially-available mathematical packages with the computing power of the parallel machines.

We have implemented a system, which we call PLAPACK Server Interface (“PSI”), that allows a user of one of the supported mathematical packages to export computationally intense problems to a parallel computer running PLAPACK. The interface consists of a set of functions that the user calls from within the mathematical package. These functions allow creating, filling, querying, manipulating, and freeing matrix, vector, and scalar objects on the parallel computer. Both memory and CPU power scale linearly with the number of processing elements in the parallel machine. Thus, PSI allows the interactive software packages to break the bonds of the workstation to solve ever larger and more complex problems.

PSI is not the first attempt to exploit parallelism from within interactive mathematical software packages. MultiMatlab [6] (from the Cornell Theory Center) and the MATLAB toolbox [5] (from University of Rostock, Germany) are extensions of the Matlab interpreter that essentially run on each node of the parallel machine. A similar product for Mathematica [10] is available from Mathconsult in Switzerland. Compiler-based systems such as FALCON [9] (from University of Illinois), Otter [8] (from Oregon State University) and CONLAB [7] (from University of Umea, Sweden) start with Matlab script files and use compiler technology to create explicit message-passing codes, which then execute essentially independently of the script’s original interactive software platform. This list is not exhaustive, but should give the idea that there are many approaches to this problem. Our approach is most similar to the MultiMATLAB and MATLAB Toolbox approaches, with one important difference. In PSI, the third-party software (MATLAB, Mathematica, etc.) runs on a workstation, while the parallel computation is performed on a completely separate massively parallel machine.¹ All parallel communication is handled from within PLAPACK.

This paper is organized as follows. Section 2 gives a brief overview of PLAPACK and of the interactive mathematical packages. Section 3 discusses some implementation details of PSI. Section 4 shows what PSI

¹PSI also runs on “beowulf” networks of workstations, in which case the third-party software runs on one node, and PLAPACK runs on the remaining nodes.

Permission to make digital/hard copy of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication and its date appear, and notice is given that copying is by permission of ACM, Inc. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

looks like from a user's point of view. Section 5 details some measurements of PSI's performance on a parallel system. Finally, section 6 gives some concluding remarks.

2 Overview of PLAPACK and the interactive packages

This section gives a brief description of PLAPACK and of the interactive packages that PSI connects to PLAPACK.

2.1 PLAPACK

PLAPACK (Parallel Linear Algebra Package) is an object-oriented system for doing dense linear algebra on parallel computers [3, 1, 2]. It is written in C, and uses the Message Passing Interface (MPI) for communication. It is distinguished by the fact that the programmer is not exposed to error-prone index computations. Instead, the concept of a "view" into a matrix or vector is used to allow for index-free programming, even of highly complex algorithms. PLAPACK's high-level abstraction and user-friendliness do not come at the expense of high performance. The PLAPACK complex LU factorization algorithm, for example, achieves over 390 MFLOPS per PE on 16 PE's of the Cray T3E-600 (300 MHz).

2.2 X-lab

Interactive mathematics packages such as Matlab (from the Mathworks, www.mathworks.com), Mathematica (from Wolfram Research, www.wolfram.com), HiQ (from National Instruments, www.natinst.com), and others allow their users access to sophisticated mathematics in an interactive, workstation environment. The packages typically include functionality for linear algebra, curve-fitting, differential equations, signal processing and sophisticated graphics, as well as many other areas of functionality.

Because PSI can connect with any of these products, and in an attempt to be even-handed, throughout the text we will refer to the interactive package as "X-lab." This is intended to refer to any of the above products.

3 Implementation

This section briefly describes the implementation of PSI. We begin by discussing the basic mechanism of communication in PSI. Then we describe the "third-party" part of the program, i.e. the part of the PSI software associated with a particular platform (Matlab, Mathematica, etc.). Next, we detail the PLAPACK side of the interface. Finally, we give some remarks about software layering in PSI.

3.1 Overview of implementation

Figure 1 shows a diagram of the PSI implementation. The third-party software ("X-lab") communicates via its inherent interface to a set of C routines. These routines then communicate via TCP with a PLAPACK server. The server either runs on other nodes of the same machine or on a completely separate parallel machine. The PLAPACK server then uses the PLAPACK application interface and MPI routines to communicate with the other PLAPACK nodes. For sending results back to the third party software, the process is reversed.

3.2 The "X-lab" side

Each piece of third-party software that is a candidate for a PLAPACK interface must be able to call user-supplied C code, and must also be able to pass data back and forth to that C code. (In practice, if user C code is callable within a package, then there is always a way to pass data back and forth.)

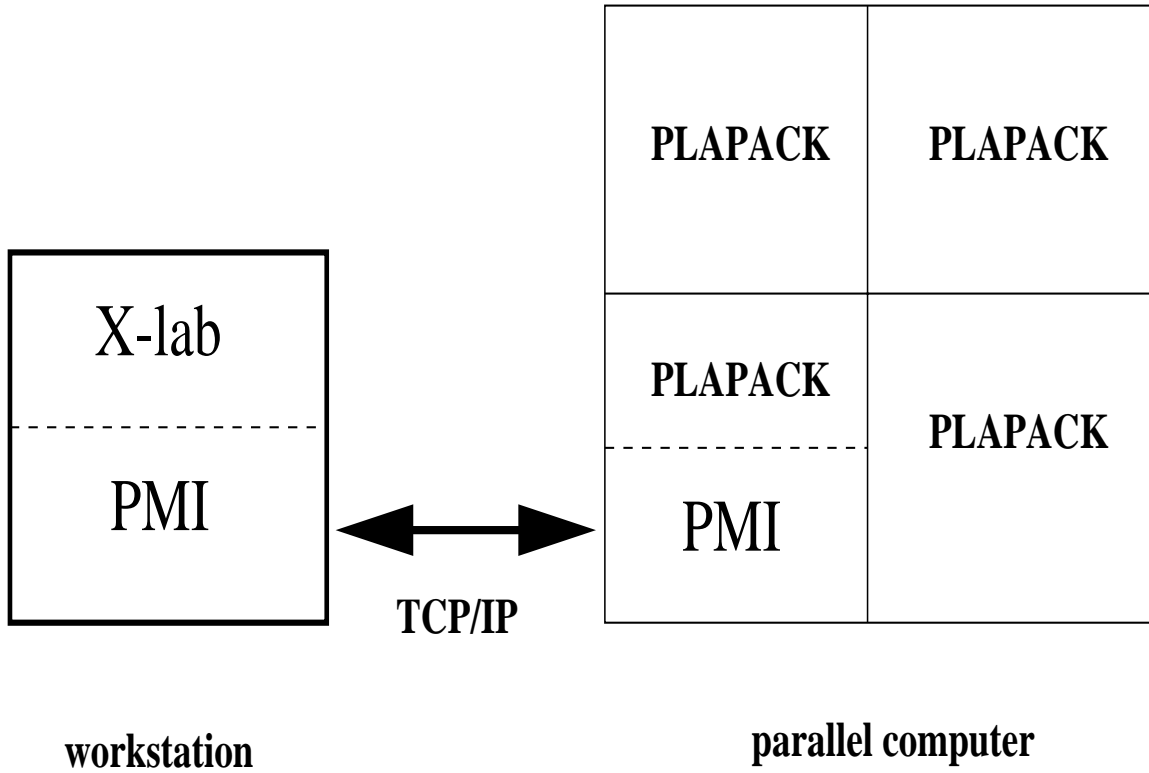


Figure 1: Diagram of the PSI machine layout.

Given that the third-party program possesses the features described above, the following outline shows how it processes PSI commands.

1. The user calls a PSI function from within X-lab
2. X-lab invokes a call to the PSI code, passing whatever parameters are necessary for this particular function
3. The PSI code writes a header to the command buffer, followed by data items to be transferred, where applicable
4. The PSI code sends the command to the PLAPACK server
5. The PSI code waits for the return message from the PLAPACK server
6. The PSI code performs any post-processing required, and either returns the relevant data to X-lab, or, if no return data is required, returns an integer whose value signifies the success or failure of the requested operation

3.3 PLAPACK side

The PLAPACK side of PSI is a parallel application that plays the role of a compute server: it sets up the PLAPACK environment, then waits for commands from the client (which is the third-party side of PSI.) Within the parallel application, one processing element (PE) plays a special role, called the master PE, and the rest of the PE's are slaves. The following is an outline of how the PLAPACK side processes commands.

1. The master PE reads a command from the TCP socket into a local command buffer
2. The master PE reads the command information from the buffer's header area, and broadcasts any relevant information to the slave PE's
3. In the case of a request to place data into a parallel object or to retrieve data from a parallel object, the master PE uses the PLAPACK application interface to transfer the data to or from the intended PE's
4. The PLAPACK PE's perform whatever parallel computation is required by the command
5. The master PE writes any return information, error messages, and an error return code to the header of the command buffer
6. In the cases where numerical data (e.g. matrix elements) need to be transferred back to the third-party software, the data are written by the master PE into the data area of the command buffer
7. The command is sent back to the client

3.4 Software layering

PSI is intended to be a general-purpose interface. That is, it should be able to plug into a variety of third-party packages, and should be able to run on a variety of parallel machines. Portability of the parallel program is easy : since PLAPACK itself is highly portable (requiring only C and MPI), this part of PSI's portability is automatic.

Because of the quirks and idiosyncrasies of the third-party interface specifications, there are certain functions that must be reimplemented for each intended third party platform. We have attempted to layer our software in such a way that these non-portable parts of the code are isolated and small.

Figure 2 shows a diagram of the PSI layering. There is a set of "core" functions that contain the workings of PSI. For each client platform, there is a thin wrapper to these functions that translates data from the

client's format to PSI's format. This layer also takes care of setting up any constants that need to be declared within the interactive environment, and is responsible for implementing a "printf"-like function to report errors and warnings.

In addition to the PSI core, there are a set of communications primitives that are used by both the client and the server. These communications functions are essentially platform-independent, requiring only a Berkeley Sockets implementation.

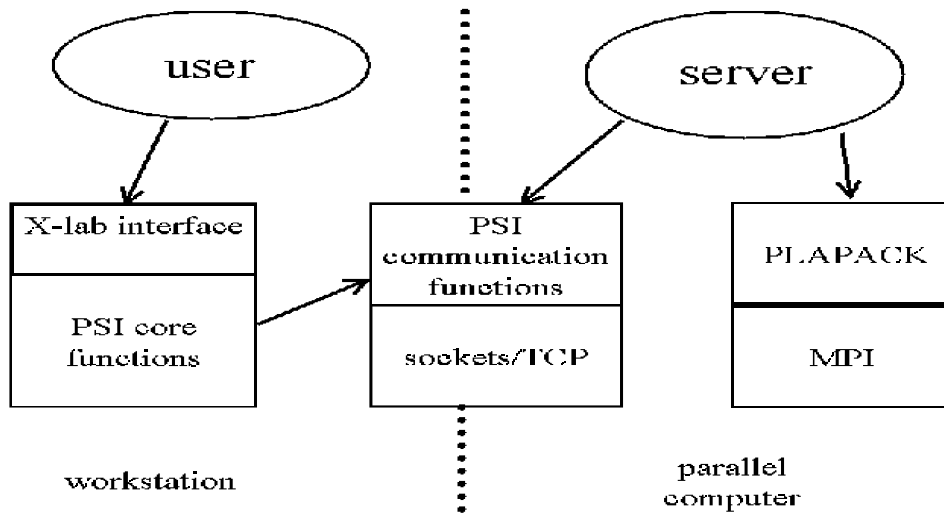


Figure 2: Software layering in PSI.

4 Using PSI

This section describes PSI from a user's point of view. What does she do to initiate a session? What does she see from within her third party mathematical software? What does a sample application look like?

4.1 The parallel application

The PLAPACK side of PSI is just like any other parallel MPI executable. Hence, whatever steps are necessary to launch a parallel program on your computer must be followed. We assume here that the "mpirun" facility is available for the parallel machine.

Let us suppose that we wish to have 16 PE's involved in the parallel side of PSI. Then we would launch the PSI.plapack.x executable, and specify 16 processors on the command line. This would look something like the following

```
% mpirun -np 16 PSI_plapack.x
PLAPACK software interface waiting for connection
PLAPACK/Matlab interface from simoom.cs.utexas.edu on a 4 by 4 mesh
```

Once the parallel executable is running, the user is free to start PSI from within her mathematical software. (Actually, the order in which the two sides of PSI are initiated is immaterial. However, a PSIopen[] call from within X-lab will block until the parallel executable is started.)

4.2 Within the math-environment

From within a third party mathematical package (which, for concreteness, we will again refer to as X-lab), PSI is accessed through a set of commands, all prefaced with the letters “PSI.” These commands fall into three basic categories.

The first category consists of commands to initialize, finalize, and manipulate the environment. Examples of commands in this category are `PSIOpen[]`, `PSIClose[]`, and `PSIVerbose[]`².

The second class of commands in PSI perform parallel object manipulations. The purpose of these commands is to create, free, query, and fill parallel matrices, multivectors, and multiscalars. Examples of commands in this category are `PSICreateObject[]`, `PSIFreeObject[]`, `PSIAxpyToObject[]`, `PSIAxpyFromObject[]`. The latter two functions are used to put values into a parallel object and to get values from a parallel object, respectively.

The third class of commands in PSI cause some action to be taken on parallel objects that already exist within the PSI parallel application (i.e. objects that have already been created and filled with data values.) Examples of these commands are `PSILU[]` and `PSIGemm[]`, which perform LU factorization and matrix multiplication, respectively.

4.3 A sample application

This section presents a sample application. This program would be executed from within a Matlab session, and of course would require a copy of the PSI parallel application to be running as well. This example performs the following steps.

1. Open the PSI interface
2. Create parallel objects : a matrix, a multivector, and a multiscalar
3. Fill the matrix and vector with data values
4. Perform a general linear solve (LU factorization of the matrix, triangular solves with the vector as right hand side, LU pivots stored in the multiscalar)
5. Retrieve the data from the vector
6. Close the PSI interface

It is important to note that the calls to the Matlab “`rand()`” function would in a real code be replaced by a meaningful routine that computes a submatrix of the matrix in question, and returns it as a Matlab matrix. For example, in a boundary element code, one could use a Matlab M-file script to compute the interaction matrix between two elements. Looping over pairs of elements, each contribution is added to the global matrix as it is generated. Thus the task of matrix generation (where users are likely to want to use Matlab-centric technology) stays inside Matlab, while the simple number-crunching of factoring the matrix is shipped off to the parallel machine.

Figure 3 shows the above program from within the Matlab version of PSI.

5 Performance

This section gives some performance measurements for the PSI system. We show results for two platforms. The first system is composed of an SGI workstation connected to NPACI’s University of Texas T3E. The latter machine is a Cray T3E-600, which consists of 56 300 MHz processing elements, each with 128 Mb of

²The exact semantics of these calls is platform-dependent. The calls as shown in the text are Mathematica-style See Figure 3 for the semantics of the Matlab-style commands.

```

matrixSize = 1000;
axpySize = 100;
PSI('open', 't3e-utexas.npaci.edu');
%
% Create the parallel objects
%
A = PSI('create',PSImatrix, PSIdouble, matrixSize, matrixSize);
x = PSI('create',PSImvector, PSIdouble, matrixSize, 1);
y = PSI('create',PSImvector, PSIdouble, matrixSize, 1);
alpha = PSI('create',PSImscalar, PSIdouble, 1, 1);
beta = PSI('create',PSImscalar, PSIdouble, 1, 1);
piv = PSI('create',PSImscalar, PSIint, 1, matrixSize);
%
% Problem set-up : Fill A and x with values, then let y = A x.
% Also, set up scalars alpha and beta.
%
for i=1:axpySize:matrixSize-1,
    for j=1:axpySize:matrixSize-1,
        aTmp = rand(axpySize, axpySize);
        PSI('addto',A,axpySize, axpySize, i-1, j-1, aTmp);
    end
end
for i=1:axpySize:matrixSize,
    xTmp = rand(axpySize, 1);
    PSI('addto',x,axpySize, 1, i-1, 0, xTmp);
end
alphaIn = 1.0;
    PSI('addto',alpha,1, 1, 0, 0, alphaIn);
betaIn = 0.0;
    PSI('addto',beta,1, 1, 0, 0, betaIn);
%
% Perform a matrix/vector multiplication to set y properly
%
trans='N';
PSI('gemv', trans, alpha, A, x, beta, y);
%
% do the solve
%
PSI('gesv', A, piv, y);
%
% retrieve the vectors and find the norm of the difference
%
norm( PSI('getfrom', y, matrixSize, 1, 0, 0) - PSI('getfrom', x, matrixSize, 1, 0, 0))
PSI('close');

```

Figure 3: A PSI program

machine	latency (sec)	bandwidth (Mbyte/sec)
beowulf	0.009	2.7
SGI with T3E	0.03	0.5

Table 1: Latency and bandwidth of the PSI interface

memory. The second system is a Pentium II-based “beowulf” system that consists of 16 300 MHz Pentium II workstations, each with 256 Mb of memory, connected with a 100Mbit ethernet network.

We concentrate on properties of the interface itself, rather than on the properties of the parallel executable. (The parallel executable is simply a PLAPACK program in disguise, and performance numbers for PLAPACK are available in the literature and from the PLAPACK web page.) We do, however, show some speedup values to get an idea of overheads inherent in the interface.

The main performance metrics for PSI concern the speed of the communication connection. In particular, we measure the latency of the connection (the time required to get a zero-length message back and forth to the parallel executable) and the inverse bandwidth (the time per byte of data sent to or received from the parallel executable.) In addition to these measurements, we also provide a profile of the sample application described in the previous section.

5.1 Latency

The latency of the PSI TCP connection is measured by timing a round-trip message, where the command involved requires no processing on the parallel side. Table 1 shows some measured latencies. The latencies are clearly small enough to not preclude interactivity.

5.2 Bandwidth

The bandwidth of the PSI connection is measured by timing the action of putting data into a parallel object, and by timing the action of retrieving data from a parallel object. The speed of these operations is dependent upon the specifics of the parallel mesh, and the specifics of the network connection between the workstation and the parallel machine. We report typical values for the measured bandwidth in Table 1.

Notice that the beowulf connection is several times faster than the workstation to t3e connection. This illustrates the benefit of running the interactive software on one of the nodes of the parallel machine. The bandwidth of the workstation to t3e connection is typical of the particular network we used. For example, an ftp connection between the same machines runs at around 0.7 Mbyte/sec, compared to the PSI bandwidth of 0.5 Mbyte/sec.

5.3 Performance of the sample application

This section presents data concerning the performance of the sample application described in the previous section of this paper. First, in Figure 4, we show a profile of the application (generated from within Matlab, the environment where we ran the PSI program) This plot shows a bar graph, with the vertical axis representing time in seconds, and the horizontal axis representing matrix size. These numbers were generated with a four processor configuration. Second, in Figures 5 and 6 we present performance numbers for a particular kernel (LU factorization with row pivoting, followed by forward and backward substitution). The horizontal axis is the matrix size, and the vertical axis is total MFLOPS (millions of floating point operations per second). Figure 5 shows the performance of bare Matlab running on one node of the beowulf for comparison.

It is important to note that the performance numbers presented do not include the time to fill the matrices. For large problems and with a relatively slow network, the communication of the matrix elements from the X-lab workstation to the parallel machine can dominate the execution time. The beowulf performs well in

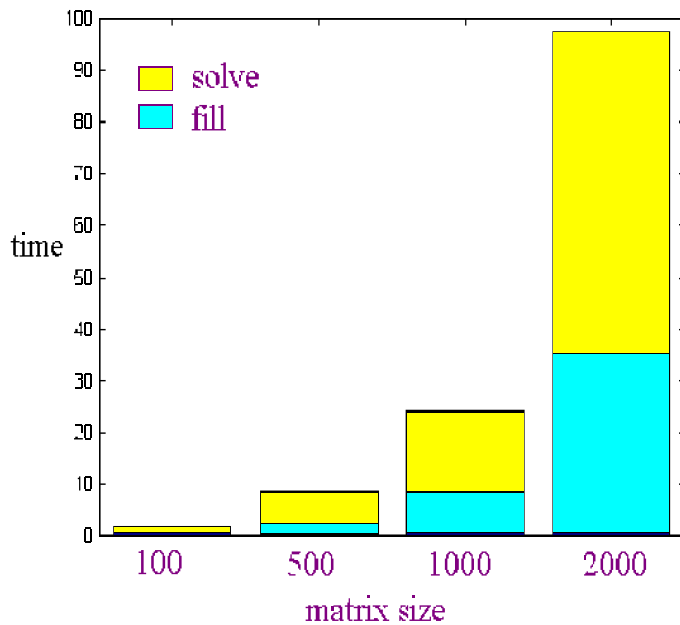


Figure 4: Profile of a PSI application. Vertical axis is time in seconds.

this sense, because its dedicated network has roughly 5 times the bandwidth of our campus network. This problem with the workstation to MPP connection would be mitigated by simply increasing the bandwidth of the network connection.

6 Conclusions and future work

This paper has described the PLAPACK Server Interface, which is a software connection allowing a user to plug a parallel computer into the back of their favorite interactive mathematical software. We have given some details of the implementation, use and performance of PSI. As yet, we have only realized a small subset of what can be done with this interface. First, the interface can be extended to support other software packages, subject only to the constraints referred to in the implementation section of this paper. Second, we intend to incorporate more of the unique features of PLAPACK (for example, the use of “views” into matrices and vectors) into PMI. Finally, we are interested in experimenting with “real applications.” That is, we wish to take an existing Matlab or Mathematica application code and parallelize it through PSI.

Please send any inquiries to plapack@cs.utexas.edu, or see our web site at www.cs.utexas.edu/users/plapack.

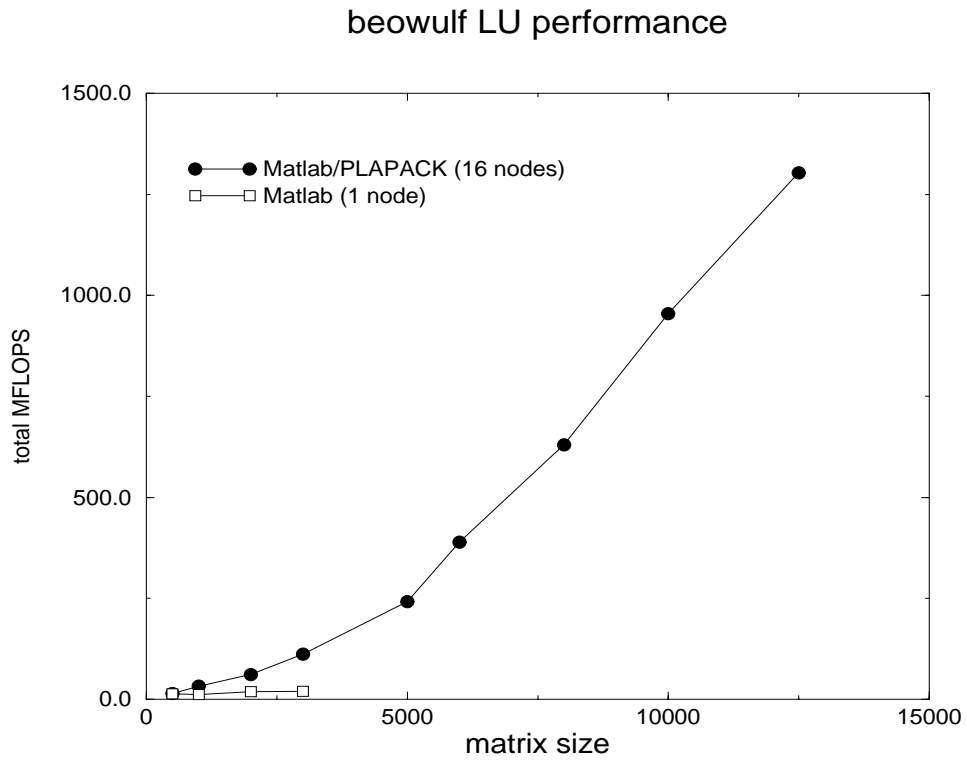


Figure 5: Total MFLOPS of the LU kernel : single node Matlab vs. Matlab with PSI on 16 processors

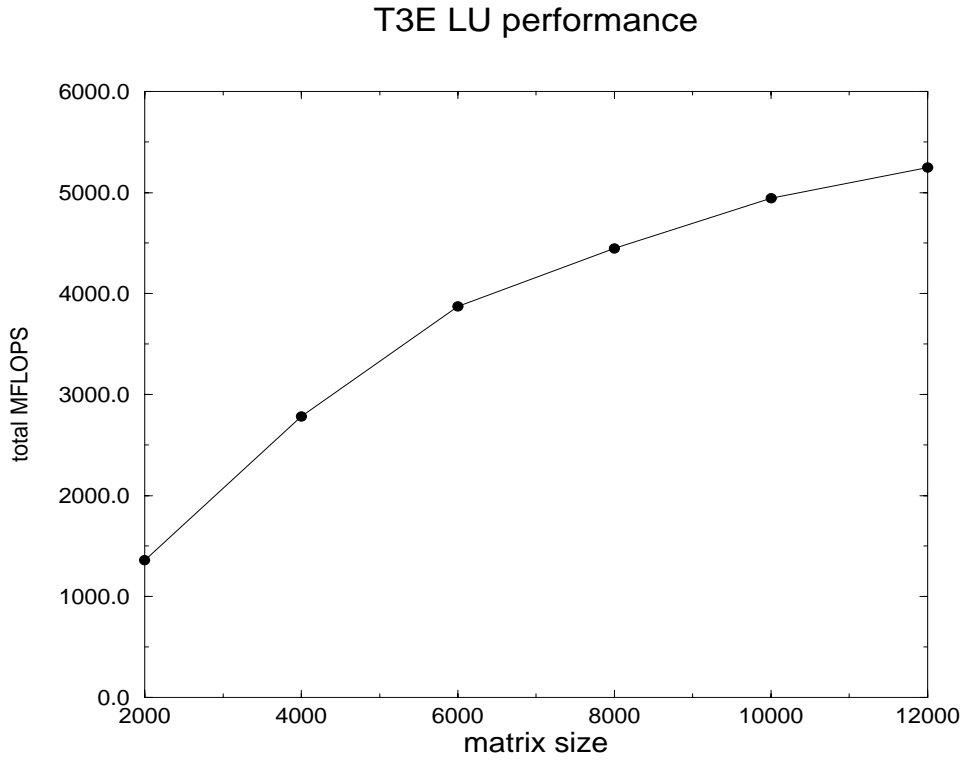


Figure 6: Total MFLOPS of the LU kernel : PSI between and SGI workstation and 16 processors of a T3E.

7 Acknowledgements

This work was sponsored in part by the Intel Research Council. The PLAPACK project was sponsored in part by the Parallel Research on Invariant Subspace Methods (PRISM) project (ARPA grant P-95006), the NASA High Performance Computing and Communications Program's Earth and Space Sciences Project (NRA Grants NAG5-2497 and NAG5-2511), and the Environmental Molecular Sciences construction project at Pacific Northwest National Laboratory (PNNL) (PNNL is a multiprogram national laboratory operated by Battelle Memorial Institute for the U.S. Department of Energy under Contract DE-AC06-76RLO 1830). We would like to acknowledge the use of computer facilities at the Texas Institute for Computational and Applied Mathematics in the University of Texas at Austin, and also at the University of Texas's T3E through the National Partnership for Advanced Computational Infrastructure.

References

- [1] P. Alpatov, G. Baker, C. Edwards, J. Gunnels, G. Morrow, J. Overfelt, R. van de Geijn, Y.-J. J. Wu, "PLAPACK: Parallel Linear Algebra Package," in **Proceedings of the SIAM Parallel Processing Conference**, 1997.
- [2] P. Alpatov, G. Baker, C. Edwards, J. Gunnels, G. Morrow, J. Overfelt, R. van de Geijn, Y.-J. J. Wu, "PLAPACK: Parallel Linear Algebra Package Design Overview," in **Proceedings of SC97**, 1997.
- [3] Robert van de Geijn, **Using PLAPACK: Parallel Linear Algebra Package**, The MIT Press, 1997.
- [4] D. J. Becker, T. Sterling, D. Savarese, J. E. Dorband, U. A. Ranawake, and C. V. Packer. BEOWULF: A parallel workstation for scientific computation. In **Proceedings of the 1995 International Conference on Parallel Processing (ICPP)**, pages 11-14, 1995.
- [5] Pawletta, S., Drewelow, W., Duenow, P., Pawlette, T., and Suesse, M. "A MATLAB toolbox for Distributed and Parallel Processing," in **Proceedings of the MATLAB Conference 95**, Cambridge, MA (1995).
- [6] A. E. Trefethen, V. S. Menon, C. C. Chang, G. J. Czajkowski, C. Myers, L. N. Trefethen, "MultiMATLAB: MATLAB on multiple processors," Technical Report 96-239, Cornell Theory Center, Ithaca, NY (1996).
- [7] P. Drakenberg, P. Jacobson, B. Kagstrom, "A CONLAB compiler for a Distributed Memory Multicomputer," in **Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computation, Volume 2** (1993), pp. 814-821.
- [8] M. J. Quinn, A. Malishevsky, N. Seelam, Y. Zhao, "Preliminary results from a parallel MATLAB compiler," in **Proceedings of the IEEE International Parallel Processing Symposium** (1998), pp. 81-87.
- [9] L. DeRose and D. Padua, "A MATLAB to Fortran 90 Translator and its effectiveness," in **Proceedings of the 10th ACM International Conference on Supercomputing** (May 1996).
- [10] R. A. Maeder, "Demonstration programs from keynote lectures given by R. Maeder at IMS'97 (Second International Mathematica Symposium), Rovaniemi, Finland, June 29 - July 4, 1997," available at <http://www.mathconsult.ch/math/stuff/ims.html>.