MPI over Scripting Languages: Usability and Performance Tradeoffs

Craig Shue, Joshua Hursey, and Arun Chauhan

Department of Computer Science, Indiana University
{cshue,jjhursey,achauhan}@cs.indiana.edu

Abstract. We present a comparative study of two popular implementations that make the MPI available on MATLAB—MatlabMPI and MPI-TB. We evaluate their performance through micro-benchmarks on a high-performance Linux cluster and compare those to their corresponding implementations on Octave[1] as well as to the LAM-MPI library accessed through a C API. We have discovered that there are significant performance advantages to using an implementation of the MPI that utilizes highly tuned libraries built for high-speed interconnects, such as the Myrinet. However, a price must be paid in terms of higher installation and setup times and a more complicated API.
We conclude that even though there are advantages to using the MPI within a high-level scripting language, such as MATLAB or Octave, there are important philosophical differences between the programming models of scripting languages and a relatively low-level communication library interface, such as the MPI. This points to the need for a more sophisticated long-term support for parallel programming from the language compiler and runtime system.

1 Introduction

Productivity in high-end application development is increasingly considered a critical issue [1]. A very effective way to improve programming productivity is to enable users to write in high-level languages, which may be domain-specific. To be usable these high-level languages should afford the abstractions that the users need. MATLAB® is one such language and its popularity among scientists and engineers is an evidence of the attraction of high-level languages for scientific computing.

Unfortunately, these language systems suffer from unacceptably high performance overheads. One way to ameliorate the performance problem is to parallelize the applications. High-performance distributed-memory clusters are among the most popular architectures for parallel machines today and the Message Passing Interface (MPI) is, perhaps, the most popular message passing standard

[1] Octave is an open source version of MATLAB. MatlabMPI has originally been released only for MATLAB. With apologies to Jeremy Kepner, we call our port of it to Octave, "OctaveMPI."

® MATLAB is a registered trademark of MathWorks Inc.

that is used to write programs for such machines. MatlabMPI and MPI-TB are two open source efforts that make the MPI available to MATLAB (and Octave) programmers [2–4].

These MPI implementations also serve to enable the language environments of MATLAB or Octave for prototype development of high-performance parallel libraries. Parallel algorithms can often be widely different from single-processor algorithms. The availability of an interface into a message passing library can aid the development and testing of such parallel algorithms before these are eventually translated, automatically or manually, into a lower-level language. Such a lowering of the language-level can serve as a pragmatic way to improve the scalar performance of parallel algorithms while allowing developers to carefully craft the communication optimizations for their algorithms. This may be particularly important for library developers who want to retain a finer control over their optimizations than an automatically parallelizing system would afford.

In this paper we report the results of a comparative performance analysis of MatlabMPI and MPI-TB using a suite of micro-benchmarks. The purpose of this analysis was to characterize these two implementations of the MPI, which may help in estimating the expected performance of a parallel MATLAB or Octave program on a modern distributed-memory parallel machine. A secondary objective was to informally judge the ease of use of these implementations and to evaluate how well each integrated with the high-level scripting language model of MATLAB or Octave. To the best of our knowledge no such study of a direct comparison of the two popular MPI implementations for MATLAB has ever been published[2].

For the remainder of this paper, unless otherwise stated, the phrase "MATLAB program" should be construed as a program that may be executed on either the MATLAB or the Octave language system.

2 MPI on MATLAB and Octave

2.1 MPI

The Message Passing Interface (MPI) is a standardized library specification for distributed memory programming [6]. The standard comes in two primary sections: MPI-1 and MPI-2. The MPI-1 specification defines much of the basic functionality including point-to-point, and collective operations. The MPI-2 specification builds upon the MPI-1 specification adding features such as dynamic process creation, and one-sided communication. There are many implementations of the MPI standard which are widely available, the most popular of which include MPICH and LAM-MPI.

[2] We are aware of a preliminary study undertaken at the Ohio-State University to compare MatlabMPI and a home-grown version of a MATLAB API to a subset of MPICH [5]. However, we are not aware of any further work on that study or publication of those results.

MPI has become the most widely used library for programming in distributed memory environments. The abstraction from many of the complexities of parallel hardware allow for portability of code between different machines, interconnection networks, and MPI implementations without changing the original source code. The encapsulation of common parallel algorithms such as reduction, barrier, and broadcast allow application programmers the ability to wield powerful collective operations in single function calls. The MPI standard supports language bindings for C, Fortran, and C++. There have also been proposals for Java, and MATLAB language bindings.

2.2 MatlabMPI

MatlabMPI, created by Jeremy Kepner of the Lincoln Laboratory at the Massachusetts Institute of Technology, provides a set of MATLAB scripts which implement several MPI primitives [2].

The scripts implement a subset of MPI-1 operations by utilizing a shared NFS mount for data distribution and by issuing remote commands through secure shell (SSH). The coverage includes functions to initialize, run, send, receive, probe, broadcast, finalize, and abort as well as functions to discover the process's rank and the number of processes running. Notably, however, the functionality includes neither the asynchronous communication primitive nor any commonly used collective operations, such as `Alltoall` or `Reduce`, with the exception of `Bcast`. However, `Probe` is supported.

2.3 OctaveMPI

Octave is a GNU-licensed integrated environment similar to MATLAB developed chiefly by John Eaton at University of Wisconsin. A stated goal of Octave is to be compatible with MATLAB.

OctaveMPI is the name given to our port of MatlabMPI to Octave. Even though Octave is largely compatible with MATLAB differences remain, primarily in the availability of library operations (Octave supports much fewer than MATLAB). There was only a moderate amount of effort involved in the porting owing mainly to these differences in the library coverage.

2.4 MPI Toolbox for MATLAB and Octave

MPI Toolbox for MATLAB, created by Javier Fernández Baldomero from the Department of Computer Architecture and Technology at the University of Granada, provides an interface to more advanced MPI-1 and MPI-2 primitives [3, 4].

The MPI Toolbox, called MPI-TB, is built upon LAM-MPI, and provides MATLAB function wrappers around much of the MPI implementation providing the MATLAB user access to a much broader subset of MPI primitives than MatlabMPI. MPI-TB has the ability to take advantage of different interconnection networks, limited only by the capabilities of the underlying MPI implementation.

A version of the toolbox is also available for Octave.

3 Experimental Setup

3.1 Hardware and Software Platform

We chose a distributed-memory cluster to run our experiments, not only because it is a popular configuration for high-performance parallel machines, but also because the MPI was designed primarily with such machines in mind. Even though there are MPI implementations that are optimized for shared-memory, there are often other more attractive programming alternatives available for shared-memory machines when a choice is possible.

LAM-MPI was selected as the MPI implementation to use because MPI-TB is designed to work with LAM-MPI. While there is nothing in MPI-TB that inherently depends on a particular implementation of the MPI we chose to use the one that works with least amount of configuration effort.

MatlabMPI uses the Network File System (NFS) for communication. However, to achieve reasonable performance the NFS parameters must be tweaked from their default values.

Table 1 summarizes the hardware and software platform used for the experiments. Each experiment was run at least 30 times and we report the means, as well as the ranges in the observed data, wherever relevant.

Table 1. Details of the experimental platform

Platform	*Details*
Machine	cluster of 8 nodes, 4 CPUs per node (only one used), Intel Xeon 2.5 GHz, 2 GB RAM, 0.5 MB cache
Operating System	Linux kernel 2.6.11 (gentoo) with SMP support
MATLAB	version 7.0.1 (R14), invoked with Java disabled
Octave	version 2.1.71
MatlabMPI	version 1.2
MPI-TB	unknown version (only one release so far)
LAM-MPI	version 7.1.1, compiled with GCC version 3.3, using the TCP RPI
NFS	ext3 filesystem locally, mounted remotely with the options rw, sync, acdirmin=0.1, hard, intr, rsize=8192, wsize=8192, nfsvers=2, udp
Interconnect	Gigabit Ethernet (GigE)

3.2 Baseline

Our goal is to compare the performance of MatlabMPI and MPI-TB. However, in order to give these results more perspective, we used C with MPI as a baseline. This provides a practical upper-bound on the performance of the communication library.

3.3 Micro-benchmarks

We created micro-benchmarks in order to test the performance of basic elements of the various approaches. These benchmarks focus on the communication aspects of the systems; computational micro-benchmarks are irrelevant since both MatlabMPI and the MPI-TB use MATLAB's core for computation.

Latency is a key element of communication. We computed this by using a simple ping-pong test, and measuring the round-trip time. Dividing that value in half determines the one-way latency between two processors.

For many applications where latency hiding approaches are applicable, bandwidth becomes the factor that determines the parallel efficiency. Bandwidth was estimated by measuring the point-to-point round trip time for a piece of data and dividing the amount of data sent by half the round-trip time. We varied the message sizes to measure the variation of achievable bandwidth with data size.

Additionally, we timed how quickly a node could broadcast a value to the other nodes in the system. Broadcasts facilitate the sharing of information, which many algorithms rely upon. Therefore, it is valuable to see the performance of the primitive.

Unfortunately, MatlabMPI lacks support for other popular collective operations, such as `Alltoall` and `Reduce`, therefore no comparative study could be carried out for these operations.

All the above tests were conducted over Gigabit Ethernet (GigE) interface. Even though our cluster has Myrinet, Infiniband, and Quadrics interconnects, the software systems relevant to this paper have very limited or non-existent support for these interfaces. Our hardware setup does not allow NFS mounting over any of these interfaces, which precludes us from using these proprietary interfaces for MatlabMPI. MPI-TB does not support Myrinet and LAM-MPI has very limited support for Inifiniband and Quadrics.

4 Results and Analysis

4.1 Bandwidth

Given the well studied and widely known latency hiding techniques, bandwidth can play a critical role in performance of parallel programs. The benchmark used for bandwidth estimation was inspired by Jeremy Kepner's `speedtest` that ships with MatlabMPI.

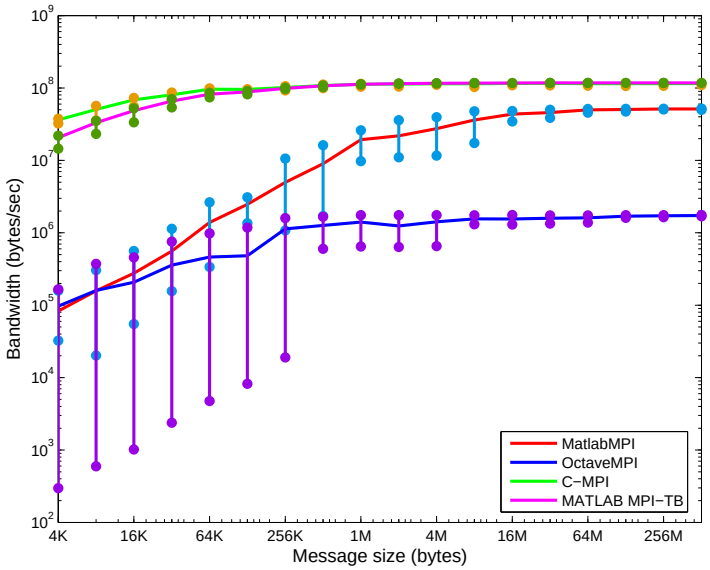

Fig. 1. Bandwidth on Gigabit Ethernet

Figure 1 shows the bandwidth achieved on sizes ranging from 4K bytes to a little over 256M bytes over Gigabit Ethernet. The vertical bars at each sample point indicate the spread of observed values across several experimental runs.

MPI used from within a C program clearly achieves the maximum bandwidth. Except for some differences for small message sizes, MPI-TB running over MATLAB closely follows the bandwidth performance of C. This is not very surprising since it is the same MPI that is being used underneath the MATLAB API for communication. The MATLAB library call overheads cause MPI-TB to be a little slower up to message sizes of about 64K bytes. There is little variation in achieved bandwidth across runs, which is reflected in almost complete absence of vertical bars for both these cases. Also, the highest bandwidth is 10^8 bytes per second (i.e., about 800 Mbps), which is very close the maximum bandwidth that could be achieved on Gigabit Ethernet.

MatlabMPI uses a file system-based communication mechanism and can be expected to take a performance hit due to the the high NFS overheads. However, for large message sizes of greater than 16M bytes it achieves bandwidths that fall in the same ball-park as C-based MPI or MPI-TB over MATLAB. This corroborates the findings of earlier studies [7]. Interestingly, and somewhat surprisingly, the Octave port of MatlabMPI, which we call OctaveMPI, is only able to achieve bandwidth that is more than an order of magnitude lower than that achieved by

MatlabMPI. It is not clear why there is such a huge performance difference. One possibility is that Octave's interaction with the file system might be unoptimized.

Another interesting observation is that while implementations utilizing LAM-MPI show very consistent bandwidth across message sizes, communication implemented using NFS shows wild variations for smaller message sizes. Since there was no other network traffic when these experiments were conducted these variations are likely to be an artifact of operating system-level scheduling triggered by the system calls that interface with the NFS.

4.2 Latency

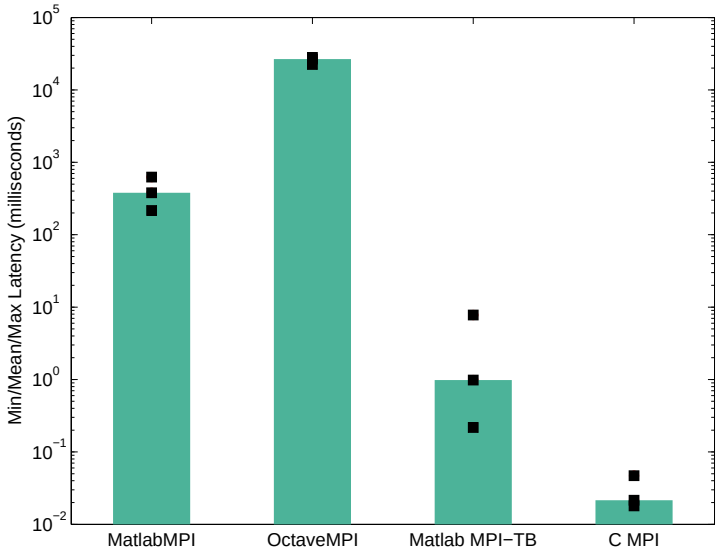

Fig. 2. Latencies on Gigabit Ethernet

Latency measurements would give us an idea of how an implementation of the communication layer can affect applications that require frequent small communications. Unsurprisingly, file system-based communication has much higher latencies than that using a communication interface. Figure 2 shows the latencies measured for the four APIs to MPI.

Once again, MatlabMPI has a poorer performance over Octave than MATLAB. The MATLAB library overheads show up very clearly here in the form of much higher latencies for MPI-TB. Notice the log scale for the vertical axis. The small

squares superimposed on the bars represent maximum, mean, and minimum latencies observed across the experimental runs.

4.3 Broadcast

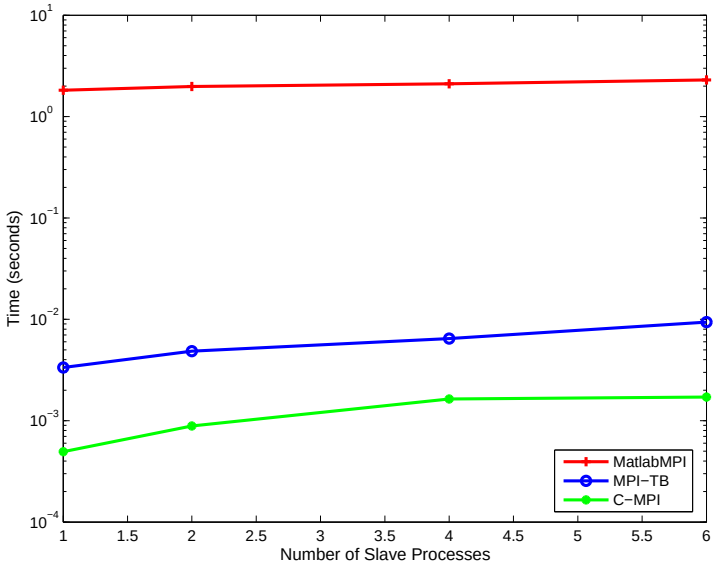

Fig. 3. Broadcast Scaling on Gigabit Ethernet

As of version 1.2 of MatlabMPI broadcast is the only supported collective operation. Figure 3 shows the scaling of the broadcast operation using various MPI APIs. Clearly, the broadcast operation scales well up to the small number of processors that we tested. These results are for small messages of about 1K bytes to bring out the worst case differences.

4.4 Analysis

Based on measuring bandwidths and latencies MPI-TB appears to be the clear winner in terms of the performance delivered within a high-level scripting language. Unfortunately, it involves a difficult learning curve to set up and use[3].

[3] We have not been able to set up the Octave version of MPI-TB on our system due to incompatibilities between rapidly changing Octave and MPI-TB versions. It would be interesting to obtain performance measurements on the Octave version of MPI-TB as well.

There are benefits to using an existing MPI implementation, such as the LAM-MPI. All the existing platform-specific and interface-specific optimizations become available, such as optimizations specific to shared memory, Myrinet, Infiniband, etc.

In the balance of performance and ease of setup and use, MatlabMPI is designed to favor the latter. It is remarkably small and easy to install. However, it does require careful tuning of the NFS mount parameters to achieve a reasonable performance—the package documentation has suggested parameters for specific platforms.

Table 2 summarizes the differences between MatlabMPI and MPI-TB based on some of the usability and performance parameters.

Table 2. Usability and performance tradeoffs

	MatlabMPI	*MPI-TB*
Setup	Easy	Moderately difficult
Using in applications	Moderately easy	Moderately difficult
Learning curve	Easy	Difficult
Latency	Poor	Good
Bandwidth	Poor to Very Good	Excellent
MPI coverage	Limited	Excellent

We emphasize the need for an easy to use communication library because a major attraction of using scripting languages is their easy of use. It is arguable if making a straightforward C-like API to MPI available within MATLAB will appeal to a broad class of users. Indeed, it is also debatable if MPI is the right level of parallel abstraction for scripting languages. Alternative, completely transparent, approaches have been proposed, which are likely to appeal to a large class of users, if these approaches are able to meet the user's performance requirements. Examples include MATLAB*P and the Distributed Computing Toolbox by MathWorks [8, 9].

If MATLAB is used as a development environment by parallel library developers, the availability of full MPI within MATLAB (or Octave) can be a great aid in developing the algorithms. This also paves the path to leverage scalar compilation technologies for high-level scripting languages [10]. Such compilation technologies could be used by library developers to lower their algorithms to a C-like language while retaining substantial control over communication.

4.5 Mandelbrot

We wrote a benchmark to compute Mandelbrot sets to test the hypothesis that an application requiring infrequent large communication will perform similarly

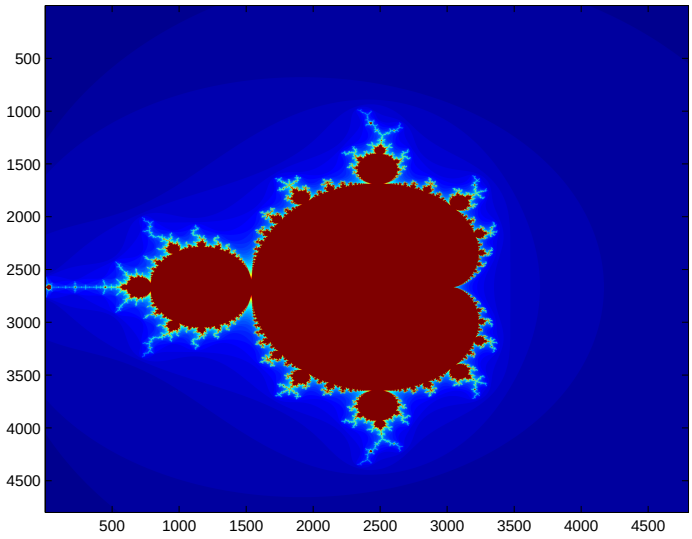

Fig. 4. Visualization of the sets computed by the Mandelbrot benchmark

on MatlabMPI as well as MPI-TB. This hypothesis is based on the observation that the bandwidth performance of MatlabMPI approaches that of other implementations for large message sizes. Computing Mandelbrot sets can be completely, "embarrassingly", parallelized. The only communication is at the end of the program when all the computed data is communicated to a central processor.

The benchmark was implemented following the description in [11]. Figure 4 shows the plot of the Mandelbrot sets as computed by our benchmark. The sets are computed over a square matrix of size 4800×4800, except for OctaveMPI that takes an inordinately long time to compute the sets over a large matrix, so the matrix size is reduced to 800×800. The matrix is divided into column blocks and the work farmed out equally to all processors. At the end of the computation all processors communicate the individually computed blocks to processor 0, which records the total time after receiving data from all the other processors.

Figure 5 shows the scaling of the Mandelbrot benchmark by plotting its parallel efficiency against the number of processors. Even though MatlabMPI has an unexpectedly low efficiency on two processors, its efficiency follows that of MPI-TB closely on both MATLAB and Octave. This validates our hypothesis and our observed bandwidth statistics. In all cases the drop in parallel efficiency is caused by the collective communication step at the end of the computation.

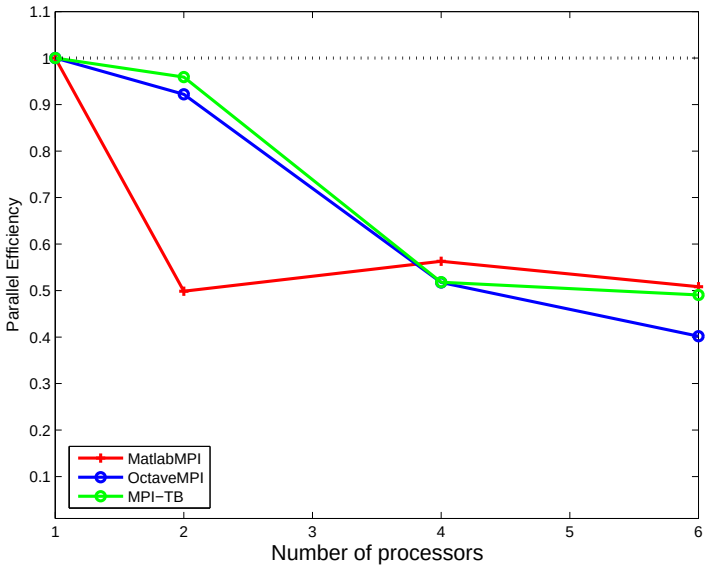

Fig. 5. Scaling of Mandelbrot

5 Related Work

There has been a large amount of interest recently in developing high-level systems for programmer's productivity in high-performance computing, evidenced by DARPA's High Productivity Computing Systems initiative [1]. Collaborative groups headed by IBM, Sun, and Cray are each in the process of developing new high-level languages for parallel computation.

There has also been work at enhancing, or developing, scripting languages for parallel computation. Some of the past work in this direction includes Scilab, Otter, pMATLAB, MultiMATLAB, etc. [12–15]. A comprehensive survey of several of these and related projects can be found in [9].

Several of these efforts have aimed at making the use of parallelism as transparent to the end-user as possible. The MPI interfaces discussed in this paper go to the other extreme of exposing the entire parallel communication library to the user. Some of the efforts currently underway fall somewhere in between—for example, providing an interface to a shared-memory programming model on distributed memory clusters [5, 16].

An alternative approach to providing parallel environment within MATLAB is to treat MATLAB as a sequential development environment and have advanced parallelizing compilers lower the language level as well as parallelize the programs for maximal performance. A large amount of knowledge base exists for automatic parallelization, in general. After some of the past effort in parallelizing compilers

for MATLAB, mentioned earlier, there has been a renewed interest lately in this direction. Clearly, it is a difficult problem to solve, but with potentially high gains.

6 Conclusions and Future Work

In this paper we have studied two popular libraries that provide the MPI interface to MATLAB (and Octave)—MatlabMPI and MPI-TB. These two libraries take two different approaches to implementing the Message Passing Interface. While MPI-TB builds upon the existing MPI implementations, MatlabMPI uses a filesystem-based communication mechanism. Even though there are clear performance advantages to using existing highly tuned MPI implementations, a simple filesystem-based system offers higher usability and easier learning curve. In cases where its performance is acceptable the simpler system may be more approachable to the general users. On the other hand, expert library developers can greatly benefit from the availability of full-fledged MPI on MATLAB and Octave.

There is clearly a need for a parallel system that appeals to the end-users of high-level languages and also delivers the performance. Several tradeoff points are possible. This paper has studied and attempted to characterize two such points.

Several future extensions to this work are possible:

- Comparing other parallel execution strategies: e.g., relying on parallel libraries such as ScaLAPACK instead of providing MPI access directly from within MATLAB [17]. MATLAB*P takes a similar approach.
- Using a wider variety of macro-benchmarks and interconnection networks, such as Myrinet and Infiniband.
- Studying other communication mechanisms, such as Global Arrays [16]. MPI may be too low-level for most people who like using MATLAB. Such users may be willing to give up some amount of parallel efficiency if that allows an easier to use shared-memory-like programming model. It would be interesting to study what performance penalties, if any, are involved.

7 Acknowledgments

We thank Jeremy Kepner for his inputs on tuning NFS parameters and his feedback on our initial results for MatlabMPI. Rob Henderson and Jon Burgoyne helped us set up the Thor cluster to the point where we could run the tests meaningfully. The work was partly supported by a grant from the DoE ASC and the Ohio Supercomputing Center.

References

1. DARPA High Productivity Computing Systems Program: (http://www.darpa.mil/ipto/programs/hpcs/)

2. MatlabMPI: (http://www.ll.mit.edu/MatlabMPI/)
3. MPI-TB under MATLAB: (http://atc.ugr.es/javier-bin/mpitb_eng)
4. MPI-TB under Octave: (http://atc.ugr.es/javier-bin/mpitb)
5. Baskaran, M., Panuganti, R., Sadayappan, P.: A comparison of MatlabMPI and MexMPI. Personal communication (2005)
6. Message Passing Interface Forum: (http://www.mpi-forum.org/)
7. Kepner, J., Ahalt, S.: MatlabMPI. Journal of Parallel and Distributed Computing **8** (2004) 997–1005
8. Mathworks, Inc.: (http://www.mathworks.com/)
9. Choy, R., Edelman, A.: Parallel MATLAB: Doing it right. IEEE Proceedings: Special Issue on Program Generation, Optimization, and Platform Adaptation **93** (2005) 331–341
10. Kennedy, K., Broom, B., Chauhan, A., Fowler, R., Garvin, J., Koelbel, C., McCosh, C., Mellor-Crummey, J.: Telescoping languages: A system for automatic generation of domain languages. IEEE Proceedings: Special Issue on Program Generation, Optimization, and Platform Adaptation **93** (2005) 387–408
11. Wilkinson, B., Allen, M.: Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers. Prentice Hall, New Jersey, 07458 (1999)
12. Desprez, F., Fleury, E., Grigori, L.: Scilab//: User interactive applications and high performances. In: Information Systems, Analysis and Synthesis. (1999) http://www.ens-lyon.fr/~desprez/FILES/RESEARCH/SOFT/SCILAB.
13. Quinn, M.J., Malishevsky, A., Seelam, N.: Otter: Bridging the gap between MATLAB and ScaLAPACK. In: Proceedings of IEEE International Symposium on High Performance Distributed Computing. (1998)
14. Kepner, J., Travinin, N.: Parallel MATLAB: The next generation. In: 7th High Performance Embedded Computing Workshop (HPEC 2003). (2003)
15. Menon, V., Trefethen, A.E.: MultiMATLAB: Integrating MATLAB with high-performance parallel computing. In: Proceedings of the ACM / IEEE SC Conference on High Performance Networking and Computing. (1997)
16. Nieplocha, J., Palmer, B., Tipparaju, V., Krishnan, M., Trease, H.: Advances, applications and performance of the global arrays shared memory programming toolkit. (To appear in the International Journal of High Performance Computer Applications.)
17. Blackford, L.S., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., Whaley, R.C.: ScaLAPACK User's Guide. (1997)