MatlabMPI Improves Matlab Performance By 300x

Jeremy Kepner

The true costs of high performance computing are currently dominated by software. Addressing these costs requires shifting to high productivity languages such as Matlab. MatlabMPI is a Matlab implementation of the Message Passing Interface (MPI) standard and allows any Matlab program to exploit multiple processors. The performance has been tested on both shared and distributed memory parallel computers (Sun, SGI, HP, IBM, and Linux). A test image filtering application using MatlabMPI achieved a speedup of ~300 using 304 CPUs and ~15% of the theoretical peak (450 Gigaflops) on an IBM SP at the Maui High Performance Computing Center. In addition, this entire parallel benchmark application was implemented in 70 software-lines-of-code (SLOC) yielding 0.85 Gigaflops/SLOC or 4.4 CPUs/SLOC. The MatlabMPI software will be available for downloading.

Research Objectives: Matlab[1] is the dominant programming language for implementing numerical computations and is widely used for algorithm development, simulation, data reduction, testing, and system evaluation. The popularity of Matlab is driven by the high productivity that is achieved by users because one line of Matlab code can typically replace ten lines of C or Fortran code. Many Matlab programs can benefit from faster execution on a parallel computer and there have been many previous attempts to provide an efficient mechanism for running Matlab programs on parallel computers (see Reference 4 for a complete list of these efforts).

The Message Passing Interface[2] is the de facto standard for implementing programs on multiple processors. MatlabMPI[5] consists of a set of Matlab scripts that implements a subset of MPI and allows any Matlab program to be run on a parallel computer. The key innovation of MatlabMPI is that it implements the widely used MPI "look and feel" on top of standard Matlab file I/O, resulting in a "pure" Matlab implementation that is exceedingly small (~250 lines of code). Thus, MatlabMPI will run on any combination of computers that Matlab supports.

Results: MatlabMPI has been run on Sun, HP, IBM, SGI, and Linux platforms. These results indicate that for large messages (~1 MByte), MatlabMPI is able to match the performance of MPI[5] written in C. In addition, MatlabMPI performance scales well to multiple processors (see Figure 1). To further test the scalability of MatlabMPI, a simple image filtering application was used based on the key computations used in many DoD sensor processing applications (e.g., wide area Synthetic Aperture Radar). The image processing application was implemented with a constant load per processor (1024 x 1024 image per processor) on a large shared/distributed memory system (the IBM SP at the Maui High Performance Computing Center). In this test, the application achieved a speedup of ~300 on 304 CPUs, as well as achieving ~15% of the theoretical peak (450 Gigaflops) of the system (see Figure 2).

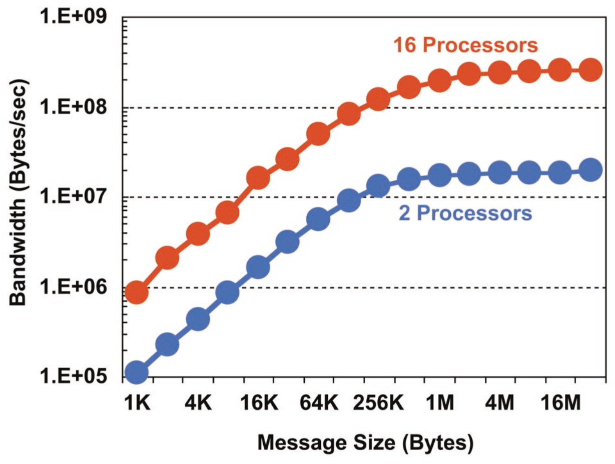

Figure 1. *Bandwidth on a Linux Cluster. Send/receive benchmark run on an eight node (16 cpu) Linux cluster connected with Gigabit ethernet.*

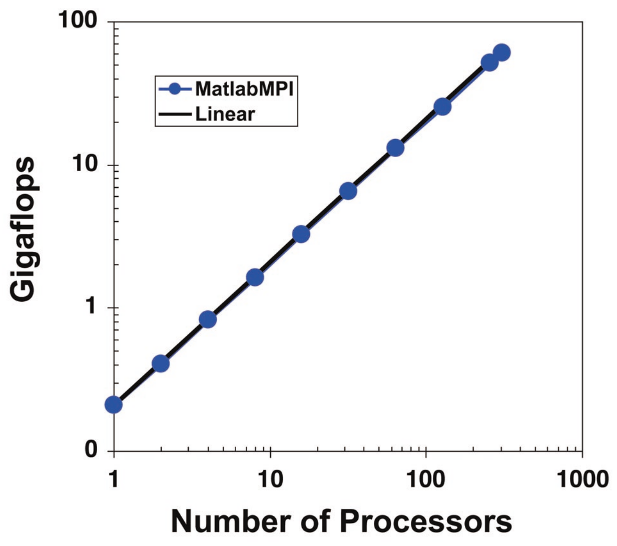

Figure 2. *Shared/Distributed Parallel Speedup. Measured performance on the IBM SP of a parallel image filtering application.*

Note: *This work is sponsored by the High Performance Computing Modernization Office, under Air Force Contract F19628-00-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the author and are not necessarily endorsed by the Department of Defense.*

The ultimate goal of running Matlab on parallel computers is to increase programmer productivity and decrease the large software cost of using HPC systems. Figure 3 plots the software cost (measured in Software Lines of Code or SLOCs) as a function of the maximum achieved performance (measured in units of single processor peak) for the same image filtering application implemented using several different libraries and languages (VSIPL, MPI, OpenMP; using C++, C, and Matlab[6]). These data show that higher level languages require fewer lines to implement the same level of functionality. Obtaining increased peak performance (i.e., exploiting more parallelism) requires more lines of code. MatlabMPI is unique in that it achieves a high-peak performance using a small number of lines of code.

Two useful metrics we have developed for measuring software productivity on high performance parallel systems are Gigaflops/SLOC and CPUs/SLOC. The test application does extremely well in both of these measures, achieving 0.85 Gigaflops/SLOC and 4.4 CPUs/SLOC.

Conclusions: MatlabMPI provides the highest productivity parallel computing environment available. However, because it is a point-to-point messaging library, a significant amount of code must be added to any application in order to do basic parallel operations. In the test application presented here, the number of lines of Matlab code increased from 35 to 70. While a 70-line parallel program is extremely small, it represents a significant increase over the single processor case.

Future Work: Our future work will aim at creating higher-level objects (e.g., distributed matrices) that will eliminate this parallel coding overhead. The resulting "Parallel Matlab Toolbox" will be built on top of the MatlabMPI communication layer, and will allow a user to achieve good parallel performance without increasing the number of lines of code.

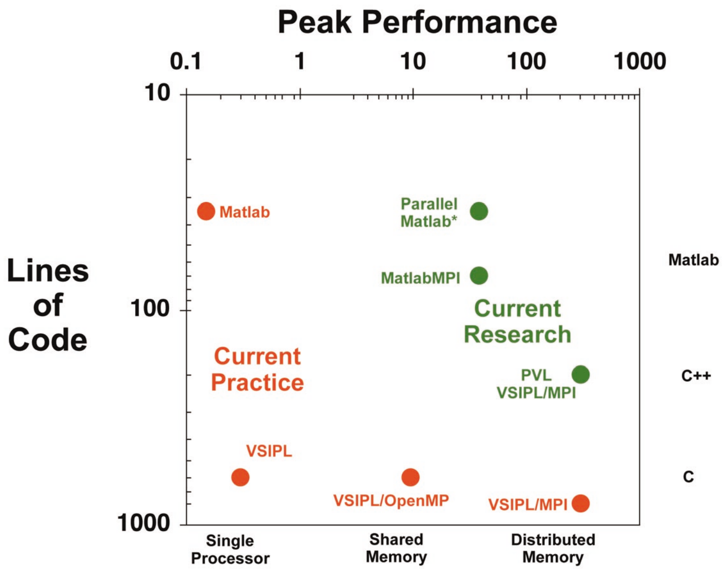

Figure 3. *Productivity vs Performance. Lines of code as a function of maximum achieved performance (measured in units of single processor theoretical peak) for different implementations of the same image filtering application.*

References:
1) Matlab, The MathWorks, Inc., http://www.mathworks.com/products/matlab/.
2) Message Passing Interface (MPI), http://www.mpi-forum.org/.
3) MATLAB*P, A. Edelman, MIT, http://www-math.mit.edu/~edelman/.
4) Parallel Matlab Survey, R. Choy, MIT, http://supertech.lcs.mit.edu/~cly/survey.html.
5) J. Kepner, Parallel Programming with MatlabMPI, 2002, High Performance Embedded Computing (HPEC) Workshop, MIT Lincoln Laboratory, Lexington, MA http://arXiv.org/abs/astro-ph/0107406.
6) J. Kepner, A Multi-Threaded Fast Convolver for Dynamically Parallel Image Filtering, 2002, accepted J*ournal of Parallel and Distributed Computing.*

Author and Contact: Jeremy Kepner
Organization: MIT Lincoln Laboratory, Lexington, MA, 02420
Resources: IBM SP at MHPCC
Sponsorship: DoD High Performance Computing Modernization Program (HPCMP)