Parallel Matlab: The Next Generation

Jeremy Kepner* (kepner@ll.mit.edu) and Nadya Travinin (nt@ll.mit.edu)
MIT Lincoln Laboratory, Lexington, MA 02420

Abstract

The true costs of high performance computing are currently dominated by software. Addressing these costs requires shifting to high productivity languages such as Matlab. The development of MatlabMPI (www.ll.mit.edu/MatlabMPI) was an important first step that has brought parallel messaging capabilities to the Matlab environment, and is now widely used in the community. The ultimate goal is to move beyond basic messaging (and its inherent programming complexity) towards higher level parallel data structures and functions. The pMatlab Parallel Toolbox provides these capabilities, and allows any Matlab user to parallelize their program by simply changing a few characters in their program. The performance has been tested on both shared and distributed memory parallel computers (e.g. Sun, SGI, HP, IBM, Linux and MacOSX) on a variety of applications.

1 Introduction

MATLAB®[1] is the dominant interpreted 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, but achieving this goal has been a significant challenge (see [2] for a reveiw). MatlabMPI [3, 4, 5] has brought parallel messaging capabilities to hundreds of Matlab users and is being installed in several HPC centers.

The ultimate goal is to move beyond basic messaging (and its inherent programming complexity) towards higher level parallel data structures and functions. pMatlab achieves this by combinng operator overloading (first demonstrated in Matlab*P) with parallel maps (first demonstrated in Lincoln's Parallel Vector Library - PVL) to provide implicit data parallelism and task parallelism. In addition, pMatlab is built on top of MatlabMPI and is a "pure" Matlab implementation which runs anywhere Matlab runs, and on any heterogeneous combination of computers. pMatlab allows a Matlab user to parallelize their program by changing a few lines. For example, the following program is a parallel implementation of a classic "corner turn" type of calculation commonly used in signal processing

```
pMATLAB_Init; Ncpus=comm_vars.comm_size;     % Initialize
mapX = map([1 Ncpus/2],{},[1:Ncpus/2])       % Map X
mapY = map([Ncpus/2 1],{},[Ncpus/2+1:Ncpus]) % Map Y
X = complex(rand(N,M,mapX),rand(N,M,mapX));  % Create X
Y = complex(zeros(N,M,mapY);                 % Create Y
coefs = ...              % Local matrix of coefs.
weights = ...            % Local matrix of weights.
Y(:,:) = conv2(coefs,X); % Parallel filter + corner turn.
Y(:,:) = weights*Y;      % Parallel matrix multiply.
pMATLAB_Finalize; exit;  % Finalize pMATLAB and exit.
```

The above example illustrates several powerful features of pMatlab: independence of computation and parallel mapping, "automatic" parallel computation, and data redistribution via operator overloading.

2 Performance Results

The vast majority of potential Matlab applications are "embarrassingly" parallel and require minimal performance out of the communication capabilities in pMatlab. These applications exploit coarse grain parallelism and communicate rarely. Figure 1 shows the speedup

*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 United States Government

[1]MATLAB is a registered trademark of The Mathworks, Inc.

obtained on a typical parallel clutter simulation. Nevertheless, measuring the communication performance is useful for determining which applications are most suitable for pMatlab. pMatlab has been run on several platforms. It has been benchmarked and compared to the performance of the underlying MatlabMPI upon which it is built. These results indicate that the overhead of pMatlab is minimal (see Figure 2), the primary difference is in the latency: 70 milliseconds for pMatlab compared to 35 millieseconds for MatlabMPI. Both pMatlab and MatlabMPI match the performance of native C MPI [1] for very large messages.

These results indicate that it is possible to write effective parallel programs in Matlab with minimal modifications to the serial Matlab code. In addition, these capabilities can be provided in a library that is written entirely in Matlab. Ultimately, it is our goal to establish a unified interface for parallel Matlab that a broad community supports. We are actively collaborating with Ohio State, UC Santa Barbara and the MIT Laboratory for Computer Science to provide a single Unified Parallel Matlab interface that is supported by multiple underlying implementations (e.g. pMatlab and Matlab*P).

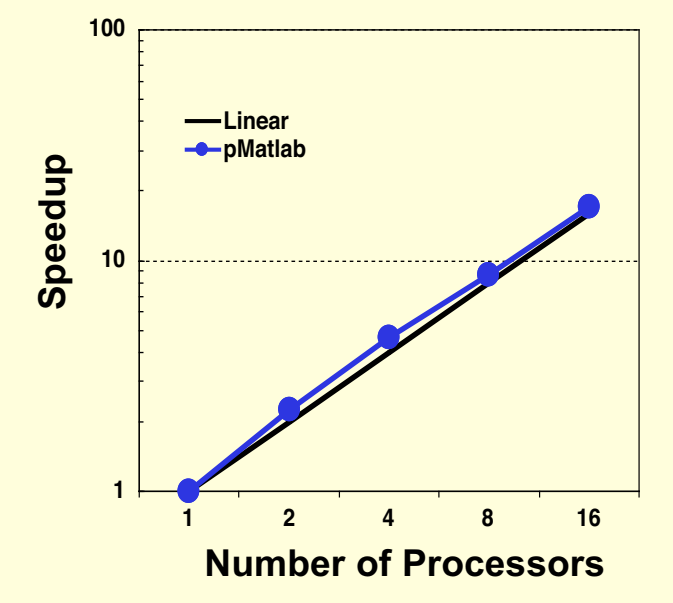

Figure 1: **Clutter Simulation Speedup.** Parallel performance speedup of a radar clutter simulation on a cluster of workstations.

References

[1] Message Passing Interface (MPI), http://www.mpi-forum.org/

[2] R. Choy, Parallel matlab survey, www.mit.edu/∼cly/survey.html

[3] J. Kepner, Parallel Programming with MatlabMPI, HPEC 2001 Workshop

[4] J. Kepner, 300x Matlab, HPEC 2002 Workshop

[5] J. Kepner and S. Ahalt MatlabMPI, submitted to the Journal of Parallel and Distributed Computing, www.arxiv.org/abs/astro-ph/0305090

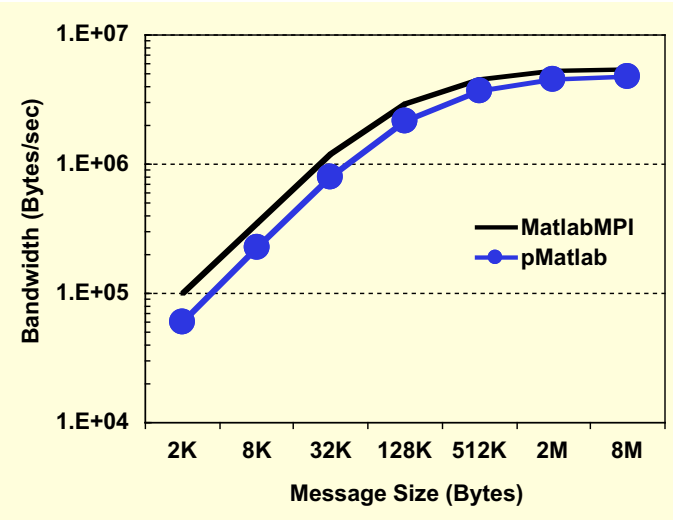

Figure 2: **pMatlab vs. MatlabMPI Bandwidth.** Communication performance on a "Ping Pong" benchmark as a function of message size on a Linux cluster. pMatlab equals underlying MatlabMPI performance at large message sizes. Primary difference is latency (70 vs. 35 milliseconds).