PMATLAB: PARALLEL MATLAB LIBRARY FOR SIGNAL PROCESSING APPLICATIONS[1]

Nadya T. Bliss, Jeremy Kepner, Hahn Kim, Albert Reuther
{nt, kepner, hgk, reuther}@ll.mit.edu
MIT Lincoln Laboratory

ABSTRACT

MATLAB® is one of the most commonly used languages for scientific computing with approximately one million users worldwide. At MIT Lincoln Laboratory, MATLAB is used by technical staff to develop sensor processing algorithms. MATLAB's popularity is based on availability of high-level abstractions leading to reduced code development time. Due to the compute intensive nature of scientific computing, these applications often require long running times and would benefit greatly from increased performance offered by parallel computing. pMatlab (*www.ll.mit.edu/pMatlab*) implements partitioned global address space (PGAS) support via standard operator overloading techniques. The core data structures in pMatlab are distributed arrays and maps, which simplify parallel programming by removing the need for explicit message passing. This paper presents the pMaltab design and results for the HPC Challenge benchmark suite. Additionally, two case studies of pMatlab use are described.

Index Terms — data processing, parallel languages, parallel programming, software

1. INTRODUCTION

MATLAB has emerged as one of the predominant languages for scientific and technical computing and is widely used at MIT Lincoln Laboratory for signal, image, and sensor processing. MATLAB's popularity is largely dependent on the expressiveness of the language and powerful graphics that allow visualization of multi-dimensional data sets. The users of MATLAB tend to be engineers and scientists, and high-level languages allow them to concentrate on their core competency and not implementations details. However, to fully test the validity of the algorithms, test runs on large data sets, with broader range of parameters are required. This often causes the codes to run for hours and even days and parallel capability without significant increase in programming complexity is beneficial. The pMatlab library provides this capability by implementing partitioned global address space (PGAS) support in MATLAB by introducing two core data structures: distributed arrays and maps. This paper describes the design of pMatlab and performance results of pMatlab implementations of the HPC Challenge benchmarks. Additionally, it highlights two pMatlab case studies at MIT Lincoln Laboratory.

The paper is organized as follows: Section 2 highlights related work; Section 3 discusses pMatlab design along with programming models. Section 4 presents benchmark results, while Section 5 discusses pMatlab use at the Laboratory. Finally, Section 6 summarizes and concludes the paper.

2. RELATED WORK

Parallel MATLAB has been an active area of research for a number of years and many different approaches have been developed. These different approaches can be roughly divided into three categories: message passing, client/server and PGAS (partitioned global address space).

The message passing approach [4, 10] requires the user to explicitly send messages within the code. These approaches often implement a variant of the Message Passing Interface (MPI) standard [16]. While MPI approaches are powerful, they significantly increase coding complexity. Nonetheless, a message passing functionality is the minimum requirement for parallel programming. Among the available MATLAB message passing implementations, MatlabMPI is currently the most popular implementation with thousands of users worldwide. More recently, the incorporation of MPI into The MathWorks' Distributed Computing Toolbox (DCT) [6] makes message passing available to a much broader range of users.

Client/server approaches [2, 15] use MATLAB as the user's front-end to a distributed library. For example, Star-P keeps

[1]This work is sponsored by the Department of the Air Force under Air Force contract FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the United States Government.

the distributed arrays on a parallel server, which calls the necessary routines from parallel libraries. These approaches often provide the best performance once the data are transferred to the server. However, these approaches are limited to those functions that have been specifically linked to a parallel library and require installation of the additional libraries.

pMatlab falls into the third category, the PGAS approach. Star-P and Falcon [9] also fall into this category. These approaches provide a mechanism for creating global arrays, which are distributed across multiple processors. Global arrays have a long history in other languages, for example Fortran [11, 18] and C [8], as well as in many C++ libraries such as POOMA [5], GA Toolkit [17], PVL and VSIPL++ [12]. The global array approach allows the user to view a distributed object as a single entity. This approach allows operations on the array as a whole or on local parts of the array.

pMatlab is a unique parallel MATLAB implementation for a number of reasons. pMatlab supports global arrays and allows combining global arrays with direct message passing for optimized performance. While pMatlab does use message passing in the library routines, a typical user does not have to explicitly incorporate messages into the code. pMatlab does not link in any external libraries, nor does it compile the language into an executable. Our library is implemented entirely in MATLAB, which significantly reduces the size of the library while providing support for distributions and redistributions of up to four-dimensional arrays distributed with any combination of block-cyclic distributions.

3. PMATLAB DESIGN AND IMPLEMENTATION

The pMatlab library is designed and implemented at MIT Lincoln Laboratory and builds upon concepts from the Parallel Vector Library (PVL) and Star-P, and uses MatlabMPI as the communication layer. Figure 1 illustrates the layered architecture of the parallel library. In the architecture, the pMatlab library implements distributed array constructs. In addition, a subset of functions, such as `plus`, `minus`, `fft`, `mtimes`, and all element-wise operations are implemented to operate on distributed arrays. If a user requires additional functionality, s/he has the flexibility of implementing specialized functions that are optimized for the required data sizes and distributions.

pMatlab uses standard operator overloading techniques. pMatlab map objects (see Section 3.1) can be passed to a MATLAB constructor, such as `rand`, or `zeros`. The constructors are overloaded and when a `map` object is passed into a constructor, the library creates a variable of type `dmat`, or a distributed array. pMatlab supports numerical arrays of up to four dimensions of different numerical data types and allows creation of distributed sparse matrices.

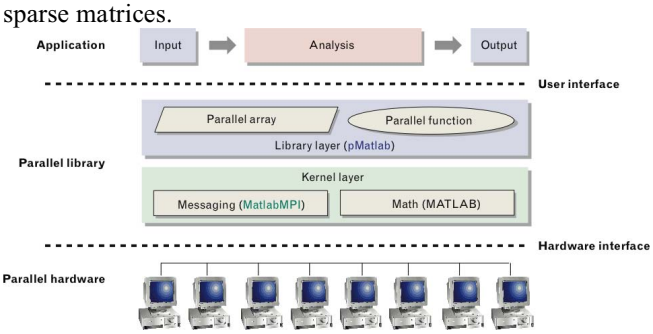

Figure 1. Layered architecture.

3.1. Maps

The concept of using maps to describe array distributions has a long history. The ideas for pMatlab maps are principally drawn from the High Performance Fortran (HPF) community [13, 20], MIT Lincoln Laboratory Space-Time Adaptive Processing Library (STAPL) [2], and Parallel Vector Library (PVL). A map for a numerical array defines how and where the array is distributed (Figure 2).

The pMatlab map construct is defined by three components: (1) grid description, (2) distribution description, and (3) processor list. The grid description together with the processor list describes where the data object is distributed, while the distribution describes how the object is distributed. pMatlab supports any combination of block-cyclic distributions up to four dimensions. Data overlap, required for some image processing applications, is also supported through the map interface. The addition of maps to the API represents the only major change to the general MATLAB syntax.

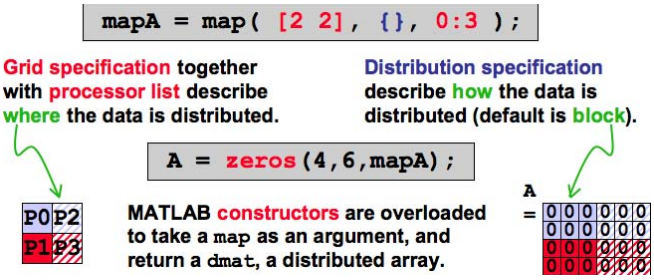

Figure 2: Anatomy of a map. A map is defined as an assignment of blocks of data to processing elements.

While maps introduce a new construct, they have significant advantages over both message passing approaches and predefined limited distribution approaches. Specifically, pMatlab maps are scalable and allow the user to separate the task of mapping the application from the task of writing the applications. Additionally, maps make it easy to specify different distributions for different algorithms. Finally, maps support pipelining via mapping of different computations on different subsets of processors.

3.2. Programming Models

pMatlab supports both pure global array and fragmented global array programming models (see Figure 3). Pure global arrays provide the highest level of abstraction and require minimum changes to the code.

It is impractical to provide optimized implementations of the approximately 8,000 built-in functions for every combination of array distributions. Instead, pMatlab also supports fragmented global array programming style. This style is less elegant but provides strict guarantees on performance. Here, distributed arrays are used as containers – the data is extracted from the distributed array, operated on, and then inserted back into the distributed array.

4. BENCHMARK RESULTS

This section focuses on pMatlab benchmark results. Performance is compared to serial MATLAB and C+MPI implementations. We have chosen to use the HPC Challenge Benchmark suite [14] for this comparison.

The four primary HPC Challenge benchmarks (STREAM, FFT, Top500 and RandomAccess) are implemented using pMatlab and run on a commodity cluster system [19]. Both the pMatlab and C+MPI reference implementation of the benchmarks are run on up to 128 processors. At each processor count the largest problem size is run that would fit in the main memory. The collected data measures the relative compute performance and memory overhead of pMatlab with respect to C+MPI. In addition, code sizes are compared.

In general, the pMatlab implementations can run problems that are typically ½ the size of C+MPI implementation problem size (Figure 4). This is mostly due to the need to create temporary arrays when using high-level expressions. The pMatlab performance ranges from being comparable to the C+MPI code (FFT and STREAM), to somewhat slower (Top500), to a lot slower (RandomAccess). In contrast, the pMatlab code is typically 3x to 40x smaller than the equivalent C+MPI code (Figure 5). For more details on pMatlab implementations of the benchmarks, see [1].

5. USER EXPERIENCES

The true measure of success for any technology is its effectiveness for real users. Table 1 highlights several projects that use pMatlab on the MIT Lincoln Laboratory interactive LLGrid system [19]. The projects are drawn from the approximately one hundred and fifty current users and are representative of the user base. The following two sub-sections discuss specific case studies.

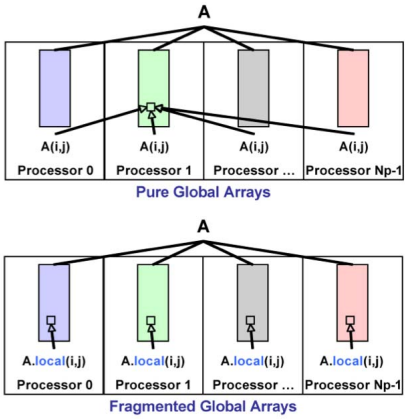

Figure 3. pMatlab programming models.

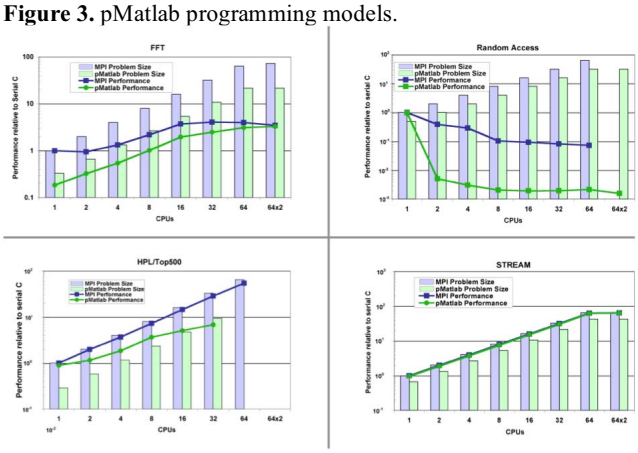

Figure 4. HPC Challenge Results.

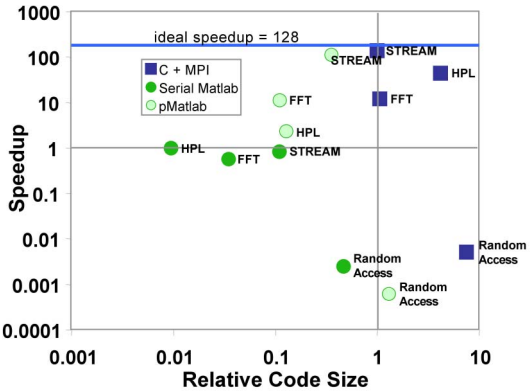

Figure 5. HPC Challenge speedup vs code size comparison.

Table 1. Selected pMatlab applications.

Code Description	Serial / Parallel Dev Time (hours)	Parallelization Enables More or Faster
Missile & Sensor Simulations	2000 / 8	Higher fidelity radar
First-principles LADAR	1300 / 1	Speckle image simulations
Analytic TOM Leakage	40 / 0.4	Parameter space studies
Hercules Metric TOM	900 / 0.75	Monte Carlos
Coherent laser propagation	40 / 1	Run time
Polynomial coefficient approx.	700 / 8	Faster training algorithm
Ground motion tracker	600 / 3	Faster & larger data sets
Automatic target recognition	650 / 40	Target classes & scenarios
Hyper-spectral Image Analysis	960 / 6	Larger datasets of images

5.1. Case Study 1: Terminal Doppler Weather Radar

The Terminal Doppler Weather Radar Data Quality Improvement program is developing signal-processing algorithms to mitigate range-velocity ambiguity [2]. For example, gust fronts that are moving radially with respect to the weather radar can be obscured by the inability of the radar signal processing algorithms to distinguish its Doppler velocity from weather that is not moving. The team needs to rapidly write, evaluate, and revise these algorithms. Running the algorithms on simulation data sets on a desktop workstation typically executed for eight to ten hours. The results of each simulation direct the parameter and algorithm choices for subsequent simulations, and they usually could only execute two of these simulations in a 24-hour period. After parallelizing the simulations, the team now runs the simulations on the desktop machines with eight to sixteen processors. These simulations now complete in 30 to 60 minutes, affording eight to ten engineering turns per day.

5.2. Case Study 2: Optical Synthetic Aperture Radar

Optical synthetic aperture radar (OSAR) is a method of generating images with laser radar that can resolve features smaller than real-aperture spot size. Developing OSAR algorithms requires simulating the return signals of a laser radar, which is computationally intensive. Initially, the parallel code distributed radar pulses across multiple processors, with a serial to parallel development time ratio of 100 to 1. Applying the OSAR simulation to new applications revealed that the number of pulses is often small containing many time bins. Due to pMatlab's map approach, modifying the code to distribute along time bins was trivial.

6. CONCLUSIONS

pMatlab combines the productivity inherent in the MATLAB programming language with PGAS, allowing MATLAB users to exploit distributed systems with only minor changes to the code. The implementation of the HPC Challenge benchmark suite using the pMatlab library allows for comparison with equivalent C+MPI codes. These results indicate that pMatlab can achieve comparable performance to C+MPI at usually one tenth the code size. Finally, implementation data collected from pMatlab applications at the Laboratory indicate that users are typically able to go from a serial code to a well-performing pMatlab code in about 3 hours while changing less than 1% of their code.

7. REFERENCES

[1] N.T. Bliss and J. Kepner, "pMatlab Parallel Matlab Library," To be published in the *Special Issue on High Productivity Programming Languages and Models, Int. Journal of High Performance Computing Applications.*

[2] J.Y.N. Cho, G. R. Elkin, and N.G. Parker, "Enhanced Radar Data Acquisition System and Signal Processing Algorithms for the Terminal Doppler Weather Radar," *Proc. AMS 32nd Conf. on Radar Meteorology, Albuquerque, NM*, 24-29 Oct 2005.

[3] R. Choy and A. Edelman, "Parallel MATLAB: Doing It Right," *Proc. of the IEEE* 93(2), pp. 331-341, 2005.

[4] Cornell Multitask Toolbox for MATLAB (CMTM), http://www.cs.cornell.edu/Info/People/lnt/multimatlab.html.

[5] J. C. Cummings et al, "Rapid Application Development and Enhanced Code Interoperability Using POOMA Framework," *Proc. SIAM Workshop on Object-Oriented Methods and Code Interoperability in Scientific and Engineering Computing (OO98)*, Yorktown Heights, NY, 21-23 Oct. 1998.

[6] L. Dean, S. Grad-Freilich, J. Kepner, A. Reuther, "Distributed and Parallel Computing with MATLAB," tutorial presented at *ACM/IEEE Conf. on Supercomputing*, Seattle, WA, 12-18 Nov. 2005.

[7] C.M. DeLuca, C.W. Heisey, R.A. Bond, and J.M. Daly, "A Portable Object-Based Parallel Library and Layered Framework for Real-Time Radar Signal Processing," *Proc. 1st Conf. Int. Scientific Computing in Object-Oriented Parallel Environments (ISCOPE '97)*, Marina del Rey, CA, pp.241-248, 8-11 Dec, 1997.

[8] T. El-Ghazawi, W. Carlson, T. Sterling, and K. Yelick, *UPC: Distributed Shared Memory Programming*, Wiley, Hoboken, NJ, May 2005.

[9] Falcon Project: Fast Array Language Computation, http://www.csrd.uiuc.edu/falcon/falcon/html.

[10] J. Kepner and S. Ahalt, "MatlabMPI," *Journal of Parallel and Distributed Computing* 64(8), pp. 997-1005, 2004.

[11] C.H. Koelbel, D.B. Loveman, R.S. Schreiber, G.L. Steel, Jr., and M.E. Zosel, *The High Performance Fortran Handbook*, MIT Press, Cambridge, MA, 1994.

[12] J. Lebak, J. Kepner, H. Hoffmann, and E. Rutledge, "Parallel VSIPL++: An Open Standard Software Library for High-Performance Parallel Signal Processing," *Proc. IEEE* 93(2), pp. 313-330, 2005.

[13] D.B. Loveman, "High Performance Fortran," *IEEE Parallel and Distributed Technology: Systems and Applications* 1(1), pp. 25-42, 1993.

[14] P. Luszczek, J.J. Dongarra, D. Koester, R. Rabenseifner, B. Lucas, J. Kepner, J. McCalpin, D. Bailey, and D. Takahashi, "Introduction to the HPC Challenge Benchmark Suite," Lawrence Berkley National Laboratory, Paper LBNL-57493, 25 Apr. 2005.

[15] G. Morrow and R. van de Geijn, "A Parallel Linear Algebra Server for Matlab-Like Environments," *Proc. ACM/IEEE Conf on Supercomputing*, Orlando, FL, 7-13 Nov. 1998.

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

[17] J. Nieplocha, R.J. Harrison, M.K. Kumar, B. Palmer, V. Tipparaju, and H. Trease, "Combining Shared and Disitrbuted Memory Models: Approach and Evolution of the Global Arrays Toolkit," *Workshop on Performance Optimization for High Level Languages and Libraries (POHLL-02), Int. Conf. on Supercomputing*, New York, NY, 22-26 June, 2002.

[18] R.W. Numrich and J. Reid, "Co-Array Fortran for Parallel Programming," *ACM SIGPLAN Fortran Forum* 17(2), pp.1-31, 1998.

[19] A. Reuther et. al., "LLGrid: Enabling On-Demand Grid Computing with gridMatlab and pMatlab," *Proc. of High Performance Embedded Computing Workshop (HPEC 2004)*, Lexington, MA, 28-30 September, 2004.

[20] M.E. Zosel, "High Performance Fortran: An Overview," *Compcon Spring '93, Digest of Papers*, San Francisco, CA, 22-26 Feb. 1993.