Enhancements to MatlabMPI: Easier Compilation, Collective Communication, and Profiling

Judy Gardiner, John Nehrbass, Juan Carlos Chaves, Brian Guilfoos, Stanley Ahalt, Ashok Krishnamurthy, Jose Unpingco, Alan Chalker, and Siddharth Samsi
Ohio Supercomputer Center, Columbus, OH
{judithg, nehrbass, jchaves, guilfoos, ahalt, ashok, unpingo, alanc, samsi}@osc.edu

Abstract

This paper provides a brief overview of several enhancements made to the MatlabMPI suite. MatlabMPI is a pure MATLAB code implementation of the core parts of the MPI specifications. The enhancements provide a more attractive option for HPCMP users to design parallel MATLAB code. Intelligent compiler configuration tools have also been delivered to further isolate MatlabMPI users from the complexities of the UNIX environments on the various HPCMP systems. Users are now able to install and use MatlabMPI with less difficulty, greater flexibility, and increased portability. Collective communication functions were added to MatlabMPI to expand functionality beyond the core implementation. Profiling capabilities, producing TAU (Tuning and Analysis Utility) trace files, are now offered to support parallel code optimization. All of these enhancements have been tested and documented on a variety of HPCMP systems. All material, including commented example code to demonstrate the usefulness of MatlabMPI, is available by contacting the authors.

1. Introduction

There is an increasing recognition that High-Level Languages, and in particular scripting languages such as The MathWorks' MATLAB, provide enormous productivity gains in developing technical and scientific code. MATLAB is widely used in academia and industry (almost 1 million users by some estimates) and has emerged as an important software code used by many Department of Defense (DoD) scientists and engineers. The attraction of MATLAB is that it combines an easy-to-use scientific programming language with a common environment for prototyping, coding, and visualization. A wide variety of add-on toolboxes addressing a number of specialized application areas, along with a very large and active open source user community, make MATLAB the software platform of choice for many IMT, SIP, and CEA users. The data analysis and visualization features offer excellent postprocessing capabilities and thus extend the user base to include users from the CCM, CWO, and CFD communities.

The popularity of MATLAB among DoD users, combined with growing memory and computational requirements, established the need for MATLAB versions able to leverage the power of high performance computers (HPCs). In a typical scenario, a user develops a prototype MATLAB algorithm on a limited computational resource, such as a single processor PC or workstation. MATLAB is chosen as the implementation language over other traditional languages because 1) it is substantially easier to collaborate with non computer science experts, allowing nonexperts to make changes and experiment; 2) it is interpretive and interactive; 3) it has excellent debugging capabilities; 4) it has integrated help with built-in example code; and 5) and the time to implementation is typically several orders of magnitude quicker.

The next step is to modify the prototypical parameters to solve real problems. This typically increases memory requirements beyond what can be accessed on the single processor PC or workstation, or it results in long-running simulations that reduce productivity or render the simulation impractical. The traditional approach to handle these large problems has been to translate the MATLAB code into C or FORTRAN, parallelize the resulting C/FORTRAN code using MPI or OpenMP, and then execute it on a HPC.

But this is an expensive, error-prone and time-consuming activity. Moreover, it is very difficult to propagate changes in the MATLAB code to the corresponding C code, especially since a single line of MATLAB code may correspond to many lines of C code. In practice this step is often outsourced to external contractors and the connection between the algorithm design team and the HPC code developers is lost.

The need for new methodologies that allow parallel development directly within the MATLAB environment is a widely recognized, and thus many software solutions are available that attempt to integrate MATLAB with HPC systems.[1] MatlabMPI[2] is a software technique developed by Dr. Jeremy Kepner at MIT Lincoln Labs (MIT-LL) that is one of the leading technologies for writing parallel MATLAB code. MatlabMPI allows Message Passing Interface (MPI)[3] communication calls to be embedded in MATLAB by implementing part of the MPI specification in pure MATLAB code. Thus MatlabMPI works anywhere MATLAB works. MatlabMPI is open source, freely available, and modifiable. This paper illustrates yet another example of how enhancements are continually being added to the suite to bring it closer to meeting the entire MPI specification.

MatlabMPI has grown quickly in popularity among DoD users, and numerous requests for additional features have been gathered. The OSC PET team delivered three specific enhancements to MatlabMPI that make it easier for DoD users to do parallel computations within MATLAB. The enhancements are intelligent compiler configuration tools, MPI collective communication calls, and a profiling version of MatlabMPI. The remainder of this paper addresses each of these enhancements. Efficiency issues are addressed where relevant.

In common computer science language, when a code or script is submitted to the queuing system, it is said to be running in "batch mode" and each instance of the corresponding submission is called a "job". This is not consistent with the definition that The MathWorks uses and often leads to confusion. For clarity, throughout the rest of this paper, "batch mode" shall refer to code or script submitted to a HPC queuing system and the particular submission shall be called a "batch job" or simply a "job".

2. Intelligent Compiler Configuration Tools for MatlabMPI

The first extension to MatlabMPI is a set of intelligent compiler configuration tools. The tools serve two purposes. First, they make it easier for users to install and use MatlabMPI on the HPCMP systems, isolating them from the peculiarities of the UNIX environments and increasing code portability. Second, they reduce the impact of the subtle differences between running MATLAB interactively and running it by submitting it to a queuing system such as LSF. When MatlabMPI is used in the "standard" interpretive mode, environment variables, paths, and access to the MATLAB license manager are all required and necessary elements for successful code execution. Batch jobs often require slight reconfigurations, and the changes are often not obvious to users. With the compiler tools, many of the user frustrations are eliminated and the results from jobs run in batch are identical to interactive jobs.

2.1. MatlabMPI Compiler Tools

MatlabMPI was first released (Version 0.95) in 2002 and was originally designed to work optimally on the MIT computer systems. DoD HPCMP users must make changes to this source code in order to run it on other HPC systems. Particularly difficult issues occur when running batch jobs. First note that a batch job may start several hours to days after being submitted, with no guarantee that resources at run time are identical to the resources available at the time of submission. The queuing system identifies resource requirements through job control parameters and thus waits for requirements to become available before running a job. Some centers allow job control parameters to check license availability as well as memory and processor resource availability, ensuring that runtime requirements are met. Unfortunately, the centers do not currently offer a common set of control parameters, so the responsibility to investigate and adjust code falls on each user, limiting code portability.

To solve this problem, the OSC PET team developed a version of MatlabMPI that integrates with the MATLAB compiler toolbox and generates executable code and scripts that eliminate known interoperability problems. The resulting executable code and scripts are launched in parallel on both batch and interactive systems alike and allow users to run MatlabMPI code with significantly more processors than previously possible. The Ohio Supercomputer Center (OSC) team refers to this technology as the MatlabMPI compiler technology. The first version of this technology, incorporated in the Version 1.2 of MatlabMPI, is identified as MPI_cc and was released to the public on 27 February 2004.

The MathWorks constantly strives to improve its product and consequently made significant advancements in Release 14 of the MATLAB software. These changes altered the way the compiler toolbox (Version 4.1) worked, and consequently these changes required modifications to the MPI_cc technology. Users found it exceedingly difficult to modify the MatlabMPI source to enable the MPI_cc components as the changes required advanced knowledge of the encryption technology embedded within the updated compiler toolbox. Users requested that the next version of MatlabMPI provide additional intelligence and return the promise of source code that continues to be easy to both understand and modify.

This release provides such resources to the user community. The team submitted these features along with other modifications and improvements to MIT-LL for peer review. After acceptance it is anticipated that a third release of MatlabMPI will be available from the MIT-LL website.[2] In the interim, this version is available to the public through contacting the authors and eventually will be posted on the OKC (https://okc.erdc.hpc.mil/index.jsp) under the SIP portal.

2.2. MatlabMPI Compiler Install and Use

This section summarizes a few key points on how to install and use this toolbox. The full documentation should be consulted before beginning installation.

Installation is trivial, requiring no make files, external libraries, or compilation. Specific configuration instructions, such as pointing to a particular installed version of MATLAB or selecting *ssh* versus *rsh* for communication between processors, are handled exactly the same as in previous versions and documented on the MIT-LL web pages.

Several HPCMP centers have created global installs of MatlabMPI. It is strongly suggested that anyone using this toolbox remove the global install from their MATLAB path and instead use the code included with this distribution. The path must be set in each MatlabMPI code developed.

This toolbox also solves the problem of executing special commands (unique to a given system) prior to running the MatlabMPI code. These commands can include setting environment variables and library paths. The solution involves a file called *setup.sh*, which contains all the lines needed to run the code. The solution for many of the HPCMP centers is included in the source distribution.

The MatlabMPI compiler technology mimics the functionality of make files. The *MPI_cc* command is still used to compile code as in previous releases; however the new version is intelligent enough to compile code only if changes have occurred since the last time it was compiled. This saves time when several jobs are submitted to the queue.

2.3. NSF Fixes

The MatlabMPI code works best for communications consisting of large messages and for codes that have a large compute to communication ratios. There are, however, times when many smaller messages must be sent. On some systems, file latency and network issues have caused NFS failures, which in turn cause the MatlabMPI codes to fail. This version provides updated versions of *MPI_Recv* and *MPI_Send* that greatly help to reduce NFS failures.

2.4. Known Issues

The new MATLAB compiler technology works much differently than the previous version. In short, it takes all the file dependencies and encrypts them. At run time, the compiled version reads in the encrypted files and decrypts them. This information is then passed to the *eval( )* function. If the code dynamically changes a path variable, then the executable code could point to an unencrypted source file. This will cause errors. At present, the project team has not found a workaround to this problem, but we are working on it. In the interim, make sure that all paths are defined prior to using the compiler.

3. MatlabMPI Collective Communication Functions

Another enhancement expands the MPI calls available in MatlabMPI to include collective communication calls. Although the functionality of collective communication calls (e.g., broadcast, allreduce) may be implemented using point-to-point communication calls, doing so is tedious for the user. Previously, only the broadcast (*MPI_Bcast*) function was available. The additional functions are implemented in pure MATLAB and preserve the “run anywhere” functionality of MatlabMPI.

3.1. Background

MatlabMPI currently implements the basic six functions that are the core of the MPI point-to-point communications standard. MPI also specifies a set of collective communication functions. Collective communication is defined as communication that involves a group of processes. In MatlabMPI the process group typically includes all the processes that are part of the task.

Collective communication functions specified in the MPI standard include barrier synchronization, broadcast from one process to all others, data gather from all processes to one process, data scatter from one process to all processes, complete data exchange, global reduction operations (such as sum, max, min, and others), and others. The MPI standard strictly defines the behavior of its functions with regard to communication deadlocks, race conditions, and interference between function calls.

3.2. Technical Overview

The OSC team now offers a subset of the MPI collective communication calls for MatlabMPI. In many cases the basic functionality is implemented while less essential options are omitted. Our goals remain to:

1. Implement the functions that are most useful for DoD MATLAB applications.
2. Retain the "look and feel" of MatlabMPI.
3. Achieve reasonable efficiency.

The new functions are simple to use, but the user assumes the burden of ensuring that certain concurrency problems do not occur, including communication deadlocks and race conditions. This is most easily done by using a different "tag" value for each communication operation.

The new MatlabMPI collective communication functions are tuned for efficiency as much as is practical. The algorithms and implementation used are described below.

3.3. Implementation

This section describes the functions that were implemented and the algorithms that were used.

MPI_Barrier – Barrier synchronization. Each process blocks until all processes in the communication group have entered the call. The algorithm used is the one used in MPICH[4], cited there as the dissemination algorithm of Hensgen, Finkel, and Manbet. [5]

MPI_Reduce – Global reduction operation, performed over all processes in the group. Operations supported are global sum, global product, global maximum, and global minimum. The process designated as root receives the result. The reduction is performed using a binomial tree algorithm with the root process as root of the tree.

MPI_Allreduce – Global reduction operation with the result returned to all processes in the group. This function is implemented as a call to *MPI_Reduce* followed by *MPI_Bcast*.

MPI_Gather – Gathers data from all processes to the root. All processes send their data to the root. The root concatenates the data in process rank order.

MPI_Bcast – Broadcast from one process to all others. This is a modification of the distributed version of *MPI_Bcast* that fixes a bug and makes it conform more closely to the MPI standard.

3.4. Utilization

The MatlabMPI collective communication functions consist of four user-callable functions and two support functions. The functions and their calling sequences are defined in the documentation included with the code, which is available from the authors. All code is compatible with MATLAB versions 6.5 and 7.1.

4. Profiling Version of MatlabMPI

A profiling version of MatlabMPI allows performance evaluation of MatlabMPI programs. The profiler generates data in the TAU trace file format. TAU trace files are converted into other formats for analysis by a variety of programs, including Vampir.

The profiling version of MatlabMPI has a "pure MATLAB" runtime component plus a combined MATLAB/C postprocessing program. The runtime portion includes wrappers for the MatlabMPI functions to be profiled (*MPI_Send*, *MPI_Recv*, *MPI_Bcast*, *MPI_Barrier*) plus functions to record trace data in *.mat* files. Functions are provided to allow tracing of user functions as well. The code is designed to be extremely easy to install and use. No modifications are made to the actual MatlabMPI code, so upgrades are easy and customizations maintained.

The postprocessing program converts the trace data in the *.mat* files to TAU trace files. It is a MATLAB script that calls C/C++ functions from the TAU trace file writer library. This portion of the profiler is more cumbersome to install and use, but it does not have to run on the system that the data was generated on. It is serial code that can be run on any Linux/Unix system that has MATLAB installed. The TAU libraries are freely available.

MATLAB's builtin profile command was not utilized, as proposed by Kim and Reuther[6] because it is not TAU compatible. While the "history" option records trace data, it is limited to 10,000 events, which limits usability.

5. Conclusion

With the enhancements developed by the OSC PET team, MatlabMPI is a useful tool for HPCMP users who need to run large or memory intense applications. All of the enhancements have been tested on a variety of HPCMP systems to demonstrate their usefulness.

Intelligent compiler configuration tools isolate users from the complexities of the UNIX environments on the various HPCMP systems and provide significant enhancement to functionality. Users install and use MatlabMPI with less difficulty and greater flexibility.

The addition of collective communication functions to MatlabMPI makes certain common operations less tedious to implement. Specifically, the global reduction operations of sum, product, maximum, and minimum are now available to the MatlabMPI programmer, along with barrier synchronization and gather from all to one.

Performance evaluation tools, including a profiling version of MatlabMPI, enable users to profile and improve their applications. These same tools support studies that compare the performance of different parallel MATLAB implementations.

All codes developed for this project are available from the authors and will become available on the OKC (https://okc.erdc.hpc.mil/index.jsp).

Acknowledgements

This publication was made possible through support provided by DoD HPCMP PET activities through Mississippi State University under contract. The opinions expressed herein are those of the author(s) and do not necessarily reflect the views of the DoD or Mississippi State University.

References

1. Choy, R., "The Parallel Matlab Survey." http://supertech.lcs.mit.edu/~cly/survey.html, 2002.

2. MatlabMPI Web Site at http://www.ll.mit.edu/MatlabMPI.

3. Message Passing Interface Forum, "*MPI: A Message-Passing Interface Standard*." http://www.mpi-forum.org/.

4. MPICH – A Portable Implementation of MPI, http://www-unix.mcs.anl.gov/mpi/mpich1/.

5. Hensgen, D., R. Finkel, and U. Manbet, "Two Algorithms for Barrier Synchronization." *International Journal of Parallel Programming*, vol. 17, no. 1, pp.1–17, 1988.

6. Kim, H. and A. Reuther, "Profiling pMatlab and MatlabMPI Applications Using the MATLAB 7 Profiler." http://www.ll.mit.edu/pMatlab/files/Profiling_pMatlab_MatlabMPI.pdf .