Procedia Computer Science
Volume 80, 2016, Pages 1461–1472
ICCS 2016. The International Conference on Computational Science

HPCmatlab: A Framework for Fast Prototyping of Parallel Applications in Matlab

Xinchen Guo[*], Mukul Dave and Mohamed Sayeed[†]
ASU Research Computing, Arizona State University, Tempe, Arizona, U.S.
Xinchen.Guo@asu.edu, Mukul.Dave@asu.edu, msayeed@asu.edu

Abstract
The HPCmatlab framework has been developed for Distributed Memory Programming in Matlab/Octave using the Message Passing Interface (MPI). The communication routines in the MPI library are implemented using MEX wrappers. Point-to-point, collective as well as one-sided communication is supported. Benchmarking results show better performance than the Mathworks Distributed Computing Server. HPCmatlab has been used to successfully parallelize and speed up Matlab applications developed for scientific computing. The application results show good scalability, while preserving the ease of programmability. HPCmatlab also enables shared memory programming using Pthreads and Parallel I/O using the ADIOS package.

Keywords: Parallel Programming, Message Passing Interface, Matlab, MEX Functions, Parallel I/O

1 Introduction

Matlab is a ubiquitous numerical computing tool used by scientists, engineers and others in a wide range of applications from computational physics to social sciences. The increasing complexity and size of these simulations requires additional computational power - not available on a desktop - through the use of parallel computing. Matlab/Octave is primarily used in sequential mode. However, Matlab provides built-in parallelization support through Parallel Computing Toolbox (PCT) or Distributed Computing Server (DCS) [1] [2]. The parallel toolboxes are seldom used and have many limitations. These are discussed in detail in section 2.7.

The HPCmatlab/HPCoctave framework has been developed to provide users the flexibility to run applications developed in Matlab/Octave as parallel programs for distributed or shared memory platforms, while still having access to the convenient built-in functions of Matlab and preserving the ease of programmability. It uses the standard Application Programming Interfaces (API), namely MPI and POSIX threads (Pthreads) through MEX wrappers.

[*] Current Affiliation: Purdue University, West Lafayette, Indiana, U.S.
[†] Corresponding Author

doi:10.1016/j.procs.2016.05.467

MPI is considered a de-facto standard for explicit message passing parallel programing paradigm for distributed memory systems [3]. It has specific, predefined set of routines for point-to-point, collective and Remote Direct Memory Access (RDMA) communication between different processes. It is very popular due to its high performance, scalability and portability. Through HPCmatlab, even Matlab users can use these routines for distributed memory parallel computing. The developed framework employing MEX wrappers maintains almost exactly the semantics of the MPI routines in the C programming language. Thus, making it easier for users to later use MPI with other programming languages. The framework provides all basic routines for performing point to point, collective and one-sided communication.

POSIX threads provide a standard interface for thread based parallelism. They are often used for getting optimum performance for parallel applications running on shared memory platforms such as a node in a large HPC cluster. In HPCmatlab, Pthreads are used to parallelize Matlab functions after translating them to C function. Using this module requires relatively more skill and effort and is meant for users willing to indulge in low-level programming.

In many applications file I/O can be a significant source of performance bottleneck. Our framework supports and uses Adaptable IO System (ADIOS), an easy-to-use package for fast, scalable and portable parallel IO [4]. ADIOS provides a simple, transparent, and flexible mechanism to define how to read, write or process data in simulations.

The implementation of HPCmatlab demonstrates how a simple interface between an interpreted programming language and an existing library can provide such effective functionality for the users to run their programs on a distributed memory platform. It stresses the significance and usefulness of the mere availability of such an interface - here, MEX functions – and the need for more flexibility in that.

This paper is organized as follows. Section 2 gives an overview of the approaches used and packages developed over the years for parallel programming using Matlab. Section 3 describes in detail, how the HPCmatlab MPI framework is implemented. Section 4 shows how it can be used. Section 5 gives a top-level explanation of the Pthreads based shared memory programming solution mentioned above. Section 6 includes the benchmarking results. Section 7 reports the scalability and speedup observed while using HPCmatlab to parallelize two serial Matlab programs from different application domains. These are being used for active research. Finally, Section 8 presents our conclusions.

2 Related Work

Matlab has become a very popular high-level programming language due to its ease of programming. On the other hand, parallel computing has become a necessity due to higher computational and memory requirements of numerical simulations. This has been a motivation for the development of several packages which allow users to exploit parallel computation through Matlab using one of the different approaches which will be described in this section. These expect different levels of knowledge in multiprocessing on the user's side. More or less, the aim is to retain the Integrated Development Environment (IDE) experience along with the programmability of Matlab and providing the maximum functionality in terms of parallel computing at the same time.

2.1 Embarrassingly Parallel Applications

This class of problems has the advantage of simplicity in that, the individual processes do not need to communicate with each other apart from the scatter and gather operations before and after the computation. An example of projects making use of this approach is the PLab [5]. However, the scope of this type of problems is quite limited and many of the large computations in practical applications require processes to frequently communicate with each other.

2.2 Message Passing Interface

This is an extensively used approach for implementing parallelism involving multiple processes. The main reason for this is, it gives the user complete control over the data being exchanged and the instructions being executed on each process. Message Passing primitives are used to send or receive data between different processes. An example of a project using this approach is MatlabMPI [6]. For message passing, it exploits Matlab's built in file I/O capabilities which allows any Matlab variable to be written and read by Matlab running on any machine. But it offers only a small subset of the general MPI routines. Also, using file I/O for communication will be very slow, requires a shared file system and is also a serious performance bottleneck in real applications. HPCmatlab supports all basic point-to-point, collective and one-sided communication routines and hence, can prove to be very useful for users having knowledge of standard MPI semantics and who need explicit control over parallelism.

2.3 Compiler Based Approach

Here, Matlab is used at the front-end while the script is translated into or compiled using a lower level language. While this demands efforts on the tool developer's side, it can be a very effective way to achieve parallelism because the user is not expected to have advanced knowledge of parallel programming semantics. For example, the solution by Tadonki and Caruana [7] to achieve multicore parallelism uses the Pthreads library for cooperation between Matlab and C. Each thread executes its associated Matlab instruction using a call to the Matlab engine.

The Pthreads based implementation in the HPCmatlab framework also has the above stated advantages while it expects the user to have a basic knowledge of the parallel programming concepts involved and the syntax provided by the framework. However, it does not have to encounter the overhead associated with using Matlab engines.

2.4 Global Address Space Models

This approach uses distributed arrays which can be viewed as a single entity, i.e. as a global array. Message Passing is used implicitly for interchange of data between the different parts. The pMATLAB [8] and GAMMA toolboxes [9] make use of "Global Arrays" model. These implement parallelism through polymorphism and overloading of the existing built in functions [10].

2.5 Back-end Support

The Star-P project [11] integrates all of the above approaches to create a parallel scientific computing environment in Matlab. It is an example of the back-end support approach. Data exists on the parallel server as distributed matrices and exists on the Matlab front-end only as a handle. Operations on the matrix are relayed to the server transparently, which then calls an appropriate routine from a parallel numerical library and the results stay on the server until explicitly requested. Polymorphism in Matlab is used to achieve the transparency.

2.6 Mathworks Products

In addition to the examples mentioned above, Mathworks Parallel Computing Toolbox [1] also includes features wherein the user can use built in functions to achieve parallelism, with an emphasis on implicit multithreading and higher level abstractions [12]. Explicit parallel programming features like message passing are also provided. Moreover, Matlab's Distributed Computing Server [2] includes a built in job scheduler and facilitates running the computationally intensive programs developed using the PCT on computer clusters, clouds and grids.

2.7 Limitations

- Using the above mentioned Mathworks products requires a separate license. Academic licenses for Matlab, often do not include the license for DCS. Researchers may not be willing to buy the license for one time use or for a single project. Also, many HPC clusters/centers do not have the license.
- To use the Message Passing features (of DCS), the user needs to know the semantics which are quite different than the MPI standard.
- The user is limited to just one compute node with PCT, limiting the performance to 12 or 16 cores of a dual processor node.
- Support for heterogeneous compute nodes having accelerator devices such as a GPU (Graphics Processing Unit) or the many-core Intel Xeon Phi cards is very limited and rudimentary. Only a small subset of functions are currently enabled to run on a GPU and no support exists for the Xeon Phi's [13].

On the other hand HPCmatlab uses the standard C MPI routine semantics and does not require the user to acquire a PCT or DCS license.

The Pthreads based framework of HPCmatlab can be compared to parfor loop feature in the toolbox. Both provide a higher level abstraction to the user. But through HPCmatlab, user can use Pthreads to achieve multithread parallelism on a customized function and not just loop iterations.

Currently, work is under progress to include support for accelerator devices within the framework.

3 Implementation of MPI Functions

The objective is to use an MPI standard implementation like MVAPICH2 [14] and enable calling the MPI routines from within Matlab. MEX (Matlab EXecutable) functions are used to provide this interface between Matlab and MPI library. This has its own set of challenges which are addressed below.

3.1 MEX Wrapper Based Approach

Since Matlab is a proprietary product, MEX-function is the best approach to develop plugins to add low level functionality to it. With `mex` command, a C or FORTRAN program can be compiled into a library that can be called from Matlab. This approach provides full access to low level C programming. Its execution speed is fast compared to Matlab script, Matlab being an interpreted language with dynamic typing.

The calls to MPI library functions are made from within MEX functions and the MEX function name is kept same as the corresponding MPI function. Arguments to be passed to the MPI function are received in MEX function in the same order. This preserves the syntax of MPI functions with some minor changes. They can simply use the MPI routines as if they are directly calling them, whereas the MEX functions are actually providing the interface.

3.2 Passing of Arguments for MPI

HPCmatlab uses the C Language syntax of MPI. This requires pointers to be passed to MPI functions. Matlab as such does not support pointers directly. However, MEX functions provide the functionality to access pointers to data passed to it. This allows us to pass pointers to MPI functions from MEX functions. This also results in a minor change in syntax for HPCmatlab functions as compared to C Language. Users do not have to specify the ampersand (&) symbol for passing pointers to data being communicated. They can simply mention name of the array and MEX function will be able to access the pointer. This can also be looked upon as a simplification of the syntax.

The same functionality also allows HPCmatlab to assign values to input arguments in case of functions like `MPI_Comm_rank()` where it is required to do so (i.e. returning rank of the current process to an input variable). Access to the corresponding pointers allows HPCmatlab to change value of input variable and have it available for user on the main Matlab process.

3.3 MPI Objects

MPI standard implementations use special MPI objects or handles to represent datatypes, operations, attributes and several other features. It also uses structures like `MPI_Status` to store the status of communication calls. These require special attention. They have to be initialized in Matlab before HPCmatlab MPI functions (MEX) are called from Matlab programs. Users can call a simple Matlab script which is part of HPCmatlab at the start of their program which manages all this at the back-end. Subsequently, MPI handles can be passed in the same way as in C language from the Matlab programs. HPCmatlab provides standard naming of basic MPI handles like `MPI_CHAR, MPI_INT, MPI_DOUBLE, MPI_SUM, MPI_MAX, MPI_COMM_WORLD`, etc.

4 Using HPCmatlab

This section demonstrates how using HPCmatlab is as intuitive as using MPI library to parallelize a C program, while it does not affect rest of the Matlab script.

4.1 Comparison of Syntax

Table 1 lists the syntax of some basic MPI functions from Matlab's PCT/DCS, HPCmatlab and in C. HPCmatlab conforms to MPI standard's C syntax. Hence, it becomes simpler for users to convert their prototype Matlab applications to a low-level programming language if they wish to do so. Also, it would encourage other Matlab users to use parallel programming and familiarize themselves with distributed computing.

Table 1: Comparison of Syntax of Basic MPI Functions

MATLAB	HPCmatlab	C
`N/A`	`err = MPI_Init(0,0)`	`err = MPI_Init(&argc,&argv)`
`labindex`	`err = MPI_Comm_rank (MPI_COMM_WORLD,rank)`	`err = MPI_Comm_rank (MPI_COMM_WORLD,&rank)`
`numlabs`	`err = MPI_Comm_size (MPI_COMM_WORLD,commsize)`	`err = MPI_Comm_size (MPI_COMM_WORLD,&commsize)`
`labSend(data, dest,tag)`	`err = MPI_Send (data,ncount,MPI_DOUBLE, dest,tag,MPI_COMM_WORLD)`	`err = MPI_Send (&data,ncount,MPI_DOUBLE, dest,tag,MPI_COMM_WORLD)`
`data= labReceive (source,tag)`	`err = MPI_Recv (data,ncount,MPI_DOUBLE, source,tag, MPI_COMM_WORLD,status)`	`err = MPI_Recv (&data,ncount,MPI_DOUBLE, source,tag, MPI_COMM_WORLD,&status)`
`N/A`	`err = MPI_Finalize()`	`err = MPI_Finalize()`

4.2 Example script

Here is a simple example of Matlab script using HPCmatlab MPI Functions for a send and receive between 2 MPI processes. To run such an application, required number of MPI processes are launched using `srun/mpirun/mpiexec` and the Matlab script is run on them.

```
GetMPIEnvironment;     % Set up HPCmatlab MPI Environment
status=MPI_Status;     % Define a 'struct' for storing status of MPI_Recv
commsize=0;            % Initialize variable to store size of communicator
rank=0;                % Initialize variable to store rank of the process
err=MPI_Init(0,0);     % Initialize MPI
err=MPI_Comm_size(MPI_COMM_WORLD,commsize);     % Get size
err=MPI_Comm_rank(MPI_COMM_WORLD,rank);         % Get rank
disp(['I am rank ' num2str(rank) ' out of ' num2str(commsize)]); %Display

N=5;
tag=123;
% We will send data from rank '0' to rank '1'
if rank==0
    data=1:N;
    err=MPI_Send(data,N,MPI_DOUBLE,1,tag,MPI_COMM_WORLD);
end

if rank==1
    % Initialize buffer (array) on rank 1 which will receive data
    recv_buff=zeros(1,N);
    disp('Before communication:');
    recv_buff
    % Now receive N 'double' type elements from rank '0'
    err=MPI_Recv(recv_buff,N,MPI_DOUBLE,0,tag,MPI_COMM_WORLD,status);
    disp('After communication:');
    recv_buff
end

err=MPI_Finalize;       % Terminates HPCmatlab MPI Environment
exit;
```

The output for the above script looks like this:

```
I am rank 0 out of 2
I am rank 1 out of 2
Before communication:
recv_buff =
     0     0     0     0     0
After communication:
recv_buff =
     1     2     3     4     5
```

By providing access to low level MPI communication functions, HPCmatlab makes it possible for the user to run heterogeneous programs under one communicator, i.e. communicate between Matlab and other standalone programs through MPI functions. E.g. a C program can run on a part of the MPI processes launched using `mpirun` and Matlab script on others. They can then communicate under the same communicator as HPCmatlab functions are used by the Matlab script. An example where such an arrangement can be useful is real-time visualization in Matlab of values computed in C program. Users are only required to understand the distributed memory programming model and MPI communication functions to use the framework without worrying about compiling and system configurations.

HPCmatlab enables the user to take further advantage of the easiness of Matlab to prototype more complex MPI programs. Both non-blocking and one-sided (Remote Memory Access) functions are supported in HPCmatlab.

4.3 Parallel I/O

Based on MPI, HPCmatlab utilizes ADIOS package [4] from Oak Ridge National Laboratory to provide parallel file I/O for Matlab users. HPCmatlab provides an interface similar to Matlab I/O functions. In addition to the benefit of high performance, data from all processes are stored in one file as a global array. Although Matlab provides load and save functions for parallel sessions, data is stored in multiple files with the number of files being same as the number of processes. This largely limits the flexibility of using arbitrary number of processes for different runs on the same data set. The script below shows an example use of the save function in HPCmatlab.

```
saveadios('test.bp', …                  % filename
        'testdata', …                   % variable name
        num2str(elenum*commsize), …     % size of global array
        num2str(elenum*commrank), …     % offset for current process
        num2str(elenum));               % size of local array
```

5 Pthreads based Shared Memory Model

HPCmatlab framework provides Pthreads based solution for shared memory programming paradigm in addition to MPI based distributed memory model and parallel file I/O.

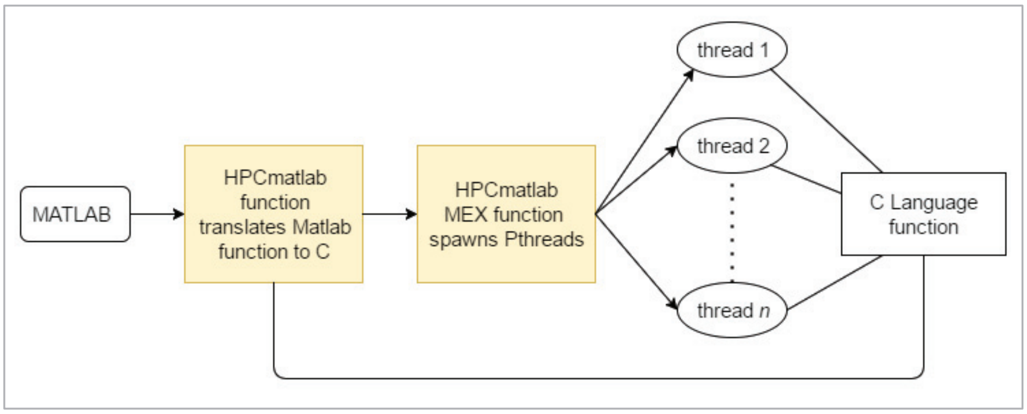
MATLAB
HPCmatlab function translates Matlab function to C
HPCmatlab MEX function spawns Pthreads
thread 1
thread 2
thread n
C Language function

Figure 1: Flow diagram of Pthreads based solution

Matlab Coder toolbox is used to translate Matlab script to C code, which is automatically integrated into a Pthreads enabled MEX-function. The flowchart explaining how this works is shown in Figure 1. The boxes which are highlighted represent functions that are provided as part of the framework. Users just have to ensure that the Matlab function is suitable for translation and then call the respective functions using appropriate input variables.

Here is how the process works:

- Matlab function is translated to C language function by Matlab Coder.
- Pthreads are spawned from MEX function using `pthread_create`.
- The translated C function is called and executed from each thread.

6 Benchmarking

Gordon cluster at the San Diego Supercomputing Center is used as the primary system to benchmark the HPCmatlab framework as well as for collecting real application performance data shown in the next section. Gordon cluster contains two 8-core Intel Xeon E5 Sandy Bridge CPUs in each node. The memory capacity of each node is 64GB (DDR3-1333). Those nodes are connected through 4x QDR (40 GB/s) network. The version of Matlab on Gordon cluster is R2014b.

In all benchmarking cases, Matlab's message passing solution implemented using DCS turns out to be slower. The overhead of Matlab's native parallel functions may be from repeated runtime error checks. Real application results show very good scalability using HPCmatlab. They are not compared with results using DCS as it does not support some of the specific collective communication routines (e.g., gather data without using distributed arrays) used for parallelization. Hence, parallelizing the programs becomes much difficult and comparing both the performance results is not justified.

6.1 Point to Point Communication

`MPI_Send` and `MPI_Recv` functions are used to benchmark the performance of point to point communication. In this benchmark, the maximum hop is limited to 1, which means two nodes are located within the same switch. Figure 2 shows that C, HPCmatlab, and MDCS have the same performance when the message size is small, which means that they have similar latency. With increasing size of message, the performance advantage of HPCmatlab appears. Its performance approaches that of C program.

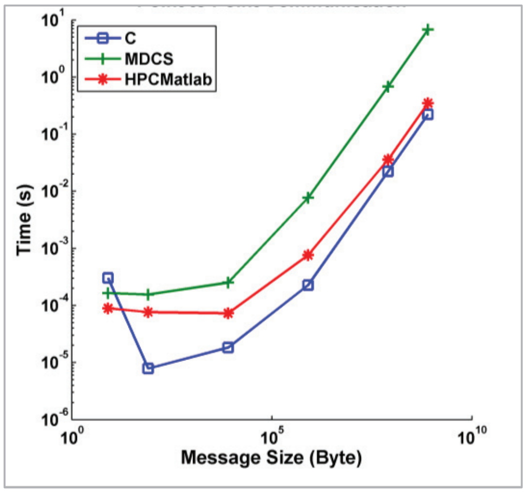

Figure 2: Point to Point Communication Benchmark

6.2 Collective Communication

For collective communication, MPI_Allgather is used. Figure 3 shows strong scaling results. HPCmatlab shows performance comparable to C for larger message sizes.

The scaling performance of HPCmatlab is also examined. A message of 1GB data is divided into local chunks by the number of processes (strong scaling). The time cost of MPI_Allgather to gather all local chunks is recorded. Figure 4 verifies that HPCmatlab provides performance similar to C program.

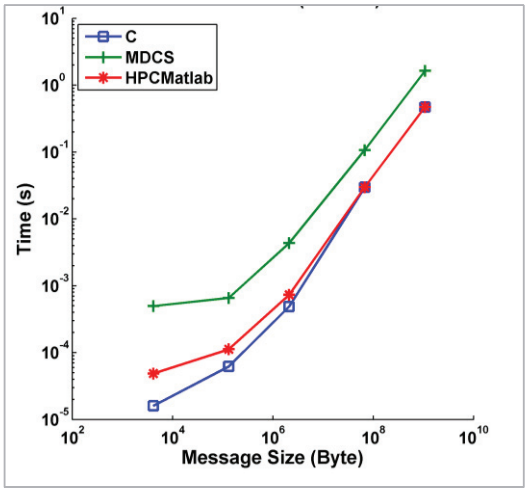

Figure 3: Collective Communication Benchmark (Package Size)

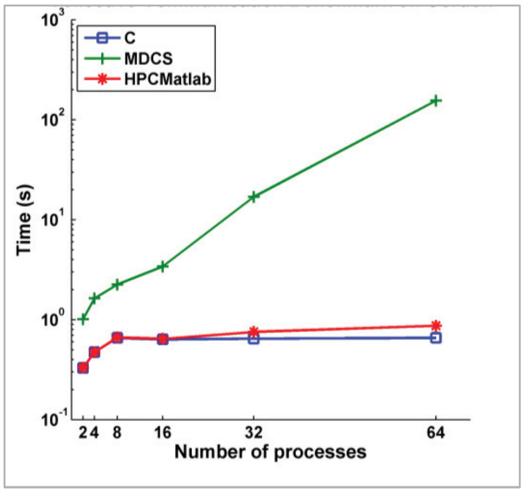

Figure 4: Strong Scaling Performance (1 GB data)

6.3 One-sided Communication

Because Matlab does not provide one-sided communication functions, we only compare the performance of HPCmatlab with C program. MVAPICH benchmarks [15] from Ohio State University is adapted in this test. `MPI_Win_create` is used for the creation of windows, while `MPI_Win_fence` is used for synchronization. Similar to point to point communication test, 2 nodes are used with maximum hop equal to one. Results are as seen in Figure 5.

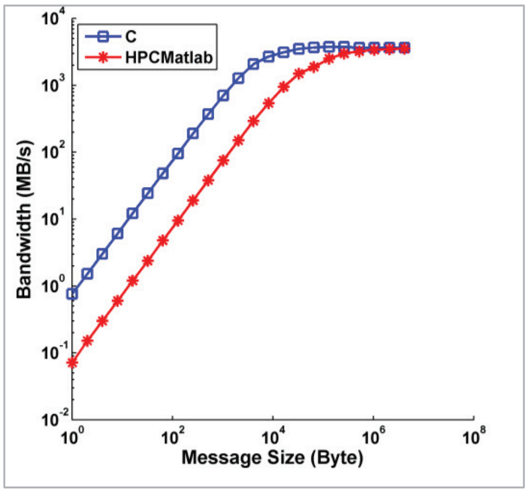

Figure 5: One-sided Communication Benchmark

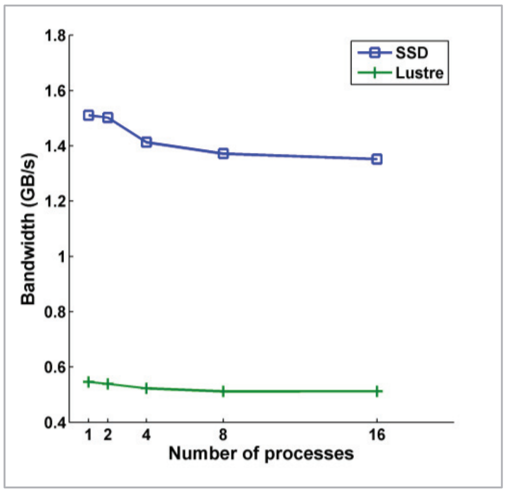

Figure 6: Save Bandwidth Scaling (4 GB data)

When the message size is very small, C program shows much better performance, which is caused by the overhead of MEX-functions. When the size of message increases to 512 KB, HPCmatlab shows performance similar to C program. Remote Memory Access is however an advanced feature being provided by HPCmatlab which is otherwise not possible with Matlab.

6.4 Parallel I/O

Figure 6 shows the result of scaling of HPCmatlab save bandwidth on Gordon SSD drive and Lustre file system within one compute node. A total of 4GB data is written. It is clear that this ADIOS based save function scales very well to saturate the overall bandwidth.

Matlab does not provide the functionality to save data from multiple workers into one global array file. One has to save data from each worker into a separate file, which imposes a limitation on flexibility.

7 Real Applications

It is imperative that the framework be tested on real applications to truly gauge its effectiveness in speeding up Matlab programs. Only two representative cases are presented here where HPCmatlab was used for parallelizing Matlab programs which simulate real application scenarios under study by the respective researchers at ASU.

7.1 Estimation of Labor Market Model Parameters

This code is part of an Earnings Risk project. It uses the guesses for labor market model parameters to simulate earnings and employment data. It is the most time-intensive piece of the larger estimation program.

Simulation is carried out for 40 different cases which are independent of each other. The simulation for each case consists of iterations over time. At the end of each iteration, all of the data is to be gathered in a matrix based upon which, some variable values for next iteration are calculated. Hence, maximum amount of parallelism that could be exploited was 40 processes. The problem size is fixed and so strong scaling is used to observe the scalability. The speedup plot is as shown in Figure 7. Speedups are calculated with reference to run time for 2 processes. The resulting parallel program turned out to be fairly scalable up to 40 processes.

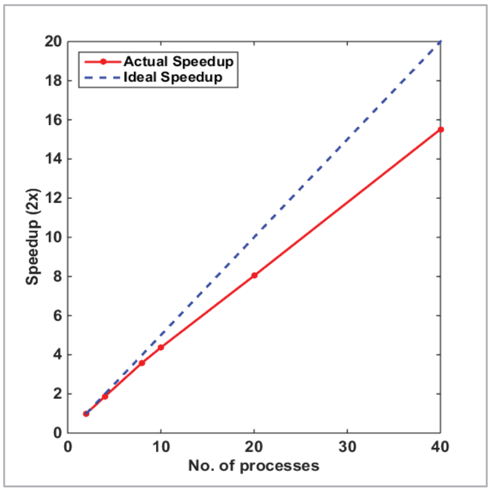

Figure 7: Speedup for Earnings Risk Code

7.2 Computation of Lyapunov Exponents

Lyapunov exponent is the measure of chaoticity of a dynamical system. The aim of this project is to be able to predict epileptic seizures by the study of electroencephalogram (EEG) recorded from human brain. Seizure happens when the brain enters into an ordered state from being chaotic and hence, can be predicted by analyzing the Lyapunov exponents over time. If this can be done in real time, electrical stimulus can be passed into the brain in order to reduce the possibility of seizure. Extensive offline investigations are first required to optimize the model parameters and decide the level and location of electrical stimulation.

In this code, Lyapunov exponents are calculated over segments of 10 seconds of data. These segments overlap with an offset of 2 seconds between two consecutive segments. Such computation is to be done for 10 channels of EEG. A single EEG recording can be as long as 178 hours (1 week). Hence, the need for parallel programming is apparent, especially if the ultimate objective is to analyze data in real time. To enforce loop based parallelism over the 10 EEG channels, parfor was already being used. But this limits the computation to just one node. HPCmatlab was used to divide the work segments among MPI processes. Each MPI process calculates the Lyapunov exponent for its own part of data segments in the EEG data file and at the end, values from all processes are gathered in one single array on the root process. This has resulted in a hybrid programming model where data segments were distributed on different nodes via MPI and parfor was used on each node to parallelize the computation on 10 different channels.

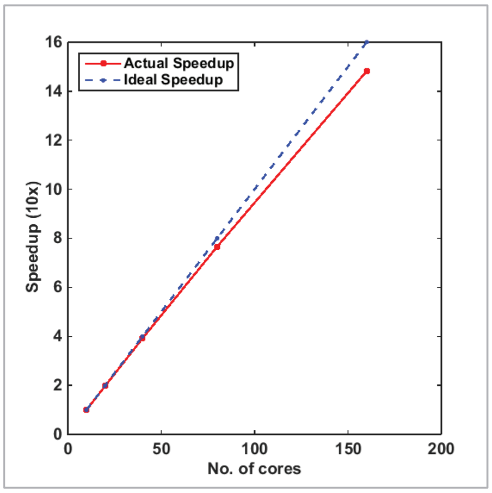

Figure 8: Speedup for Lyapunov Exponent Computation code

As the problem size is fixed, a strong scaling curve is used to observe the scalability due to MPI. For scalability analysis, Lyapunov exponents are computed for 10 hours of EEG data. Figure 8 shows excellent scalability up to 160 cores (16 MPI processes with 10 parfor workers each). Speedup is calculated with reference to run time for a simple program sans MPI using 10 parfor workers. The solution using parfor can be implemented without the use of HPCmatlab and hence, speedup from that should not be a factor in observing the scalability due to HPCmatlab.

8 Conclusion

HPCmatlab shows better performance than the parallel computing toolbox and the distributed computing server from Mathworks, for both MPI communication routines as well as Pthreads module. Moreover, it is comparable to the performance of C programs, but providing the programmability of Matlab at the same time. The file I/O using ADIOS package enables the user to have a global view of arrays along with the speed of parallel I/O with parallel file system and hardware support.

The time user has to spend for parallelizing applications using HPCmatlab MPI functions is minimal. With a very basic understanding of how message passing works, applications can be parallelized by adding a few lines of code to the serial script. These features can motivate users to learn the basics of parallel programming, say for example, MPI routines. They can use the framework for fast prototyping of parallel applications and then potentially go ahead and use other programming languages to parallelize their applications using MPI which is the de-facto programming model for distributed memory programming.

HPCmatlab MPI module is currently installed for general use on the HPC clusters at ASU. Future course of action includes adding the Parallel I/O capabilities to the install for users. More MPI routines will be added in subsequent versions of the framework. The framework is also compatible with Octave.

The use of HPCmatlab in parallelizing serial Matlab codes for researchers at ASU has shown promising results. Efforts will be made to make the framework more robust and user friendly and encourage more number of users to take up parallel programming on Matlab - potentially using HPCmatlab. A comprehensive user guide is available to users wanting to learn HPCmatlab. Application developers interested in using the framework may contact us.

9 Acknowledgements

This work used the XSEDE environment, which is supported by NSF grant number ACI-1053575. We are thankful to XSEDE and San Diego Supercomputing Center for their services.

Our sincere thanks to Eleanor Dillon at ASU for deciding to use HPCmatlab as a parallelizing framework with her application - 'Simulating the Earnings Risk Estimation'. We are grateful to Konstantinos Tsakalis and Ashfaque Shafique at ASU for putting faith in HPCmatlab to parallelize their Lyapunov Exponent code. Their initiative helped us gauge the framework's efficacy and usability.

References

[1] "Parallel Computing Toolbox," [Online]. Available: http://www.mathworks.com/products/parallel-computing/.

[2] "MATLAB Distributed Computing Server," [Online]. Available: http://www.mathworks.com/products/distriben/.

[3] "MPI Standard Documents," [Online]. Available: http://www.mpi-forum.org/docs/docs.html.

[4] J. F. Lofstead, S. Klasky, K. Schwan, N. Podhorszki and C. Jin, "Flexible IO and Integration for Scientific Codes through tha Adaptable IO System (ADIOS)," in *Proceedings of the 6th International Workshop on Challenges of Large Applications in Distributed Environments*, 2008.

[5] U. Kjems, "PLab reference page," 2000. [Online]. Available: http://bond.imm.dtu.dk/plab.

[6] J. Kepner and S. Ahalt, "MatlabMPI," *Journal of Parallel and Distributed Computing,* pp. 997-1005, 2004.

[7] C. Tadonki and P. L. Caruana, "Seamless Multicore Parallelism in MATLAB," *Parallel and Disributed Computing and Networks,* 2014.

[8] N. T. Bliss and J. Kepner, "pMATLAB Parallel MATLAB Library," *International Journal of High Performance Computing Applications,* pp. 336-359, 2007.

[9] R. Panuganti, M. M. Baskaran, A. Krishnamurthy, J. Nieplocha, A. Rountev and P. Sadayappan, "An Efficient Distributed Shared Memory Toolbox for MATLAB," 2007.

[10] R. Choy and A. Edelman, "Parallel MATLAB: Doing it right," in *Proceedings of the IEEE,* 2005.

[11] R. Choy, A. Edelman, J. R. Gilbert, V. Shah and D. Cheng, "Star-P: High Productivity Parallel Computing," Massachusetts Inst of Tech, Cambridge, 2004.

[12] G. Sharma and J. Martin, "MATLAB: A Language for Parallel Computing," *International Journal of Parallel Programming,* pp. 3-36, 2009.

[13] "MATLAB - GPU Support," [Online]. Available: http://www.mathworks.com/discovery/matlab-gpu.html.

[14] "MVAPICH: MPI over InfiniBand," [Online]. Available: http://mvapich.cse.ohio-state.edu/.

[15] "MVAPICH: MPI over InfiniBand," [Online]. Available: http://mvapich.cse.ohio-state.edu/benchmarks/.