Dynamic Load Balancing with MatlabMPI*

Ricolindo L. Cariño¹, Ioana Banicescu^{1,2}, and Wenzhong Gao³

¹ Center for Computational Sciences ERC, PO Box 9627

² Department of Computer Science and Engineering, PO Box 9637

³ Department of Electrical and Computer Engineering, PO Box 9571

Mississippi State University, Mississippi State MS 39762, USA

{rlc@erc, ioana@cse, wgao@ece}.msstate.edu

Abstract. A number of packages have recently been developed to enable the execution of MATLAB programs on parallel processors. For many applications, an integrated load balancing functionality is also necessary to realize the full benefits of parallelization. This paper describes a toolkit based on MatlabMPI to ease the parallelization and integration of load balancing into MATLAB applications that contain computationally-intensive loops with independent iterations. Modifications to existing code to incorporate the toolkit are minimal. Performance tests of two nontrivial MATLAB programs with the toolkit on a general-purpose Linux cluster indicate that significant speedups are achievable.

1 Introduction

MatlabMPI [1] is a set of MATLAB [2] scripts that implements a subset of the Message-Passing Interface (MPI) standard [3]. MatlabMPI allows multiple copies of a MATLAB program executing on parallel processors to communicate. MatlabMPI utilizes file I/O for communications and exploits the built-in I/O capabilities of MATLAB to read/write (receive/send) any MATLAB variable. While MatlabMPI enables parallelization of a MATLAB program, the program developer is still responsible for ensuring that the parallel version achieves high performance. Thus, additional MatlabMPI functionality is necessary to reduce the burden of developers in writing efficient parallel MATLAB code.

The MATLAB programs that benefit most from MatlabMPI are those that are embarrassingly parallel. These programs typically contain computationally-intensive parallel loops. The iterations of these loops can be executed independently and in any order on several processors without affecting the correctness of the computations. However, the parallel execution of iterations gives rise to a load balancing problem, especially if the iterations have nonuniform execution times which may not be known *a priori*, or if the processors are heterogeneous or are unpredictably loaded. To our knowledge, as of the time of writing this paper, there are no visible research efforts addressing the dynamic load balancing problem in MatlabMPI.

* This work was partially supported by the NSF grants #9984465, #0313274 and #0132618.

The contribution of this paper is a toolkit to address the potential load imbalance that arises while using MatlabMPI. The toolkit simplifies the parallelization *and* load balancing of MATLAB programs that contain loops with independent iterations, requiring only very minor revisions to the programs. Owing to the overhead of using file I/O for communications, the toolkit achieves significant speedups only when the loop iterations are more costly than file I/O. The toolkit implements a scheduler-worker load balancing strategy, where the scheduler decides the worker loads and also participates in executing iterations. The worker loads are determined by a user-selected dynamic loop scheduling technique. The toolkit has been successfully integrated into a number of nontrivial MATLAB applications that exhibit significant performance improvement when executed on a general-purpose Linux cluster. Thus, we believe the toolkit is useful for load balancing many other MATLAB applications, thereby increasing the productivity of users and improving the utilization of parallel computing resources.

The rest of this paper first reviews related work on parallel MATLAB in Section 2, and then describes our implementation of the load balancing toolkit based on MatlabMPI in Section 3. Section 4 describes the integration of the toolkit into actual MATLAB programs, and their resulting parallel performance on a general-purpose Linux cluster. Section 5 presents our conclusions and describes future work.

2 Parallel MATLAB

MATLAB is a high-level language and interactive environment for algorithm development, data visualization, data analysis, and numeric computation [2]. It is very popular and is widely used in scientific and technical computing. Arguments for not developing parallel MATLAB were offered in 1995 [4]; however, a recent survey [5] found 27 parallel MATLAB projects, some defunct and some implemented in commercial packages. The survey summarizes the approaches to making MATLAB parallel, including: compiling MATLAB programs into native parallel code; providing a parallel back end to a graphical MATLAB front end; and providing communication and coordination facilities for MATLAB processes. MatlabMPI [1] uses the last approach. It is a set of MATLAB scripts implementing a subset of MPI. With MatlabMPI, a MATLAB program may use the six basic functions (MPI_Init, MPI_Comm_size, MPI_Comm_rank, MPI_Send, MPI_Recv and MPI_Finalize) to carry out parallel processing. Additionally, MPI_Abort, MPI_Bcast and MPI_Probe are implemented for convenience.

In order to realize the full benefits of parallelization in MATLAB programs, load balancing is usually required especially if the parallel tasks have nonuniform computational requirements or if the processors are heterogeneous. *Dynamic* load balancing becomes a necessity when the nonuniformity of tasks or heterogeneity of processors are not known *a priori*. Two of the authors have previously designed a dynamic load balancing routine based on MPI for C or Fortran applications containing parallel loops [6]. The toolkit described in this paper is based on similar design principles.

3 A Dynamic Load Balancing Toolkit

We developed a toolkit based on MatlabMPI to reduce the burden of developing parallel MATLAB programs that require dynamic load balancing. In addition to reducing programming efforts, the toolkit improves the productivity of users through shorter program running times, achieved through parallelization and load balancing. And as a consequence of decreased running times, parallel computing resources are utilized more efficiently. This section briefly describes the usage of the toolkit, the MATLAB scripts, and the shell script to submit the parallel MATLAB job to a Linux cluster managed by the Portable Batch System (PBS) [7].

3.1 Toolkit Usage

The toolkit inherits the target programs of MatlabMPI. These targets are MATLAB programs that contain embarrassingly parallel loops. These loops must be of the form `for i=1:N {i-iteration} end` or the equivalent `while` form, where the iterations are independent and computationally-intensive. The integration of the toolkit into a target program is illustrated by Figure 1. Essentially, the original `for` loop is converted to a `while` loop where chunks of iterations can be executed concurrently on different processors. Since the `i-iteration` invokes CPU-intensive computations which may require tens or hundreds of MATLAB statements, the additional code to utilize the toolkit constitutes a small percentage of the total number of lines of code for the application.

<pre>% serial version ... for i=1:N {i-iteration} end ...</pre>		<pre>% parallel version MPI_Init; ... scheduler = 0; method = 4; dispLevel = 2; DLBinfo = DLB_StartLoop (MPI_COMM_WORLD, ... scheduler, 1, N, method, dispLevel); while (~DLB_Terminated(DLBinfo)) [iStart iSize DLBinfo] = DLB_StartChunk(DLBinfo); for i=iStart:(iStart+iSize-1) % was i=1:N {i-iteration} end DLBinfo = DLB_EndChunk (DLBinfo); end [iIters iTime] = DLB_EndLoop (DLBinfo); % (OPTIONAL) ... MPI_Finalize;</pre>
---	--	---

Fig. 1. Parallelization and dynamic load balancing of a MATLAB for-loop

Results of the `i-iterations`, if any, will be scattered among the processors in `MPI_COMM_WORLD`. The `scheduler` maintains information about all the

chunks of iterations in the `chunkMap` field of `DLBinfo`. The other processors keep the information about chunks they execute in their own `DLBinfo.chunkMap`. After loop execution, the workers can send results to the scheduler. An example of how this can be accomplished is provided in Section 4.

3.2 Descriptions of Functions

The toolkit consists of the following functions:

`function DLBinfo = DLB_StartLoop ( iComm, master, firstIter, lastIter, iMeth, dispLevel)` is the synchronization point at the start of loop execution. A dynamic load balancing environment is created on `iComm`. The information about this environment is maintained in `DLBinfo`. The scheduler will have the rank specified by `master`. The loop range is `[firstIter, lastIter]`. The loop scheduling method that will be used is indicated by `iMeth` (see `DLB_StartChunk`). The amount of debugging information to be displayed is `dispLevel` (0 = no tracing; 1 = trace the sends and receives; and 2 = trace the sends, receives and probes).

`function done = DLB_Terminated(DLBinfo)` returns true if all loop iterations have been executed.

`function [iStart iSize DLBinfo] = DLB_StartChunk (DLBinfo)` returns a range for a chunk of iterations. This range starts with iteration `iStart` and contains `iSize` iterations. Let `N` and `P` denote the number of iterations and the number of processors, respectively, and let `remaining` denote the number of unscheduled iterations at any time during loop execution. The following loop scheduling methods are used by the scheduler for determining the chunk sizes:

- `iMeth=0`: `iSize=N/P` (static scheduling);
- `iMeth=1`: `iSize=1` (self scheduling);
- `iMeth=2`: `iSize=(N/P)/log2(N/P)` (fixed size chunking, with the same number of chunks as in factoring; see `iMeth=4`);
- `iMeth=3`: `iSize=(remaining/P)` (guided self scheduling [8]); and
- `iMeth=4`: `P` chunks, each of `iSize=remaining/(2*P)` (factoring [9]).

Other loop scheduling methods such as variants of adaptive weighted factoring [10, 11, 12, 13] and adaptive factoring [14, 15] will be added in the future. Internally, the scheduler probes for requests for chunk data from workers and responds by sending `[iStart iSize]` if `remaining>0`; otherwise, the scheduler sends `iSize=0`, which will be interpreted by workers as the termination signal. If there are no requests, the scheduler sets `iSize=1` and proceeds to execute one iteration.

`function DLBinfo = DLB_EndChunk(DLBinfo)` signals the end of execution of a chunk of iterations. Internally, a worker processor requests from the scheduler the start and size of the next chunk.

`function [iIters iTime] = DLB_EndLoop(DLBinfo)` (optional) marks the end of loop execution. `iIters` is the number of iterations done by the calling processor, and `iTime` is the cost measured using the MATLAB function `etime`. `iIters` and `iTime` are useful for assessing the performance gains

achieved by dynamic load balancing. For example, the sum of the `iTimes` from all participating processors gives an estimate of the cost of executing the loop on a single processor.

3.3 Submission to a Linux Cluster

MatlabMPI provides a `MPLRun.m` script to initiate the parallel execution of a MATLAB program on a collection of machines. On a Linux cluster managed by PBS, the PBS script for the parallel MATLAB job must first create the list of machines to be supplied as an argument to `MPLRun`, and then start a MATLAB session to execute `MPLRun`. In order to avoid initiating this extra MATLAB session on a Linux cluster, we moved the essential functionality of `MPLRun` to the PBS script. Thus, the PBS scripts for our MatlabMPI jobs first create the parallel MATLAB scripts, and then start the remote MATLAB sessions to run these scripts. In contrast to those generated by `MPLRun`, our parallel MATLAB scripts include the statements for the remote MATLAB sessions to create `MPI_COMM_WORLD`, instead of loading it from disk.

4 Evaluation

We have integrated the toolkit into nontrivial MATLAB programs. This section presents results of testing the toolkit on two such programs. The tests were conducted on the heterogeneous general-purpose EMPIRE cluster of the Mississippi State University [16]. The cluster has a total of 1038 processors organized into 16 racks. A rack contains 32 nodes of dual 1.0GHz or 1.266GHz Pentium III processors and 1.25GB RAM. Each node is connected to a 100Mb/s Ethernet rack switch. The rack switches are connected by a gigabit Ethernet cluster switch. Installed software includes RedHat Linux and PBS. The general submission queue allows 64-processor, 48-hour jobs; a special queue allows 128-processor, 96-hour jobs from a restricted set of users. According to the Top 500 Supercomputer Sites list published in June 2002, EMPIRE then was the 126th fastest computer in the world.

The first example is a MATLAB program used to verify the correctness of a filtered back projection algorithm for tomographic imaging [17]. An outline of the computationally-intensive section of the program and its parallel version with the toolkit is illustrated by Figure 2. The other sections of the program are executed redundantly by all participating processors.

Figure 3 compares the performance of the imaging application on 1, 4, 8 and 12 processors. In addition to the parallelization, the application variable `rays` was increased in value from 16 to 64 to generate a parallel loop with 256 iterations. For this problem with a short runtime, the speedups on the loop execution (loop time plus gather time) are 1.83, 3.43 and 4.94 for 4, 8 and 12 processors, respectively.

The second example is a program to benchmark several methods for solving nonnegative quadratic programming problems (<http://www.cis.upenn.edu/feisha/codes/nqpBench.m>). In particular, the program compares the convergence

```

%serial version      | % parallel version
for x=x_min:x_max2, | comm = MPI_COMM_WORLD;
    for y=y_min:y_max2, | info = DLB_StartLoop(comm,0,x_min,x_max2,4,2);
        sum=0.0;        | while ( ~DLB_Terminated(info) )
        for i=1:angles, |     [iStart iSize info] = DLB_StartChunk(info);
            ...          |     for x=iStart:(iStart+iSize-1)
            sum = sum + ... |         (The "for y=y_min:y_max2..." is unchanged)
        end;            |     end;
        f(x,y)=sum;      |     info = DLB_EndChunk(info);
    end;                | end;
end;                    |         % gather results
                        | if (comm.rank==0) % 0 receives remote chunks
                        |     for i=1:info.numChunks
                        |         iStart = info.chunkMap{1,i}(1,1);
                        |         iSize = info.chunkMap{1,i}(1,2);
                        |         src = info.chunkMap{1,i}(1,3);
                        |         f(iStart:iStart+iSize-1,:) = ...
                        |             MPI_Recv(src,iStart,comm);
                        |     end
                        | else % workers send executed chunks to 0
                        |     for i=1:info.numChunks
                        |         iStart = info.chunkMap{1,i}(1,1);
                        |         iSize = info.chunkMap{1,i}(1,2);
                        |         MPI_Send(0,iStart,comm,...
                        |             f(iStart:iStart+iSize-1,:));
                        |     end
                        | end;

```

Fig. 2. Outline of the computationally-intensive section in a MATLAB program for tomographic imaging (left) and its parallel version (right) with code to gather results into one of the processors

Fig. 3. Total time of tomographic imaging application

Table 1. Performance of benchmark program

Processors	1	3	4	5
Wall time (sec)	59096.5	23495.1	17655.5	11854.5
Speedup	-	2.52	3.35	4.99

rate of a recently developed multiplicative updates technique [18] to the rates of MATLAB’s `quadprog()` with preconditioned conjugate gradients and `quadprog()` with direct solver. The program generates `numTrial=1000` random problems and computes the average solution time for each technique. Table 1 summarizes the performance of the program on 1, 3, 4 and 5 processors when `numTrial=10000`. The 5-processor limit is due to the number of concurrent users allowed for a MATLAB toolbox required by the benchmark program.

5 Conclusions and Future Work

MatlabMPI and similar packages enable the execution of MATLAB programs on parallel processors. However, load balancing is necessary to fully exploit the advantages of parallelization and to make more productive use of expensive parallel computing resources.

We developed a toolkit based on MatlabMPI to ease the parallelization and integration of load balancing into MATLAB programs that contain computationally-intensive parallel loops. The toolkit implements a dynamic scheduler-worker load balancing strategy where loads are determined through loop scheduling techniques. The toolkit drastically reduces the burden of program developers in writing efficient parallel MATLAB code.

We successfully tested the toolkit by integrating it into a number of nontrivial MATLAB programs and executing these on a general-purpose Linux cluster. Results of sample tests indicate significant speedups are achievable with the toolkit. We are currently integrating the toolkit into other applications such as a design optimization application. Performance improvements will be reported in the future. We are also implementing adaptive loop scheduling techniques into the toolkit as additional options for determining processor loads.

References

1. Kepner, J.: Parallel Programming with MatlabMPI. (2005) <http://www.ll.mit.edu/MatlabMPI>.
2. The MathWorks: MATLAB - The Language of Technical Computing. (2005) <http://www.mathworks.com/products/matlab>.
3. MPI Forum: The Message Passing Interface Standard. (1995) <http://www-unix.mcs.anl.gov/mpi>.
4. Moler, C.: Why ther isn’t a parallel MATLAB (1995) http://www.mathworks.com/company/newsletters/news_notes/pdf/spr95cleve.pdf .

5. Choy, R., Edelman, A.: Parallel MATLAB: Doing it right. *Proceedings of the IEEE* **93** (2005) 331–341
6. Cariño, R.L., Banicescu, I.: A load balancing tool for distributed parallel loops. *Cluster Computing: The Journal of Networks, Software Tools and Applications* **8**(4) (2005) To appear
7. Altair Grid Technologies: Portable Batch Systems. (2003) <http://www.openpbs.org>.
8. Polychronopoulos, C., Kuck, D.: Guided self-scheduling: A practical scheduling scheme for parallel supercomputers. *IEEE Transactions on Computers* **C-36**(12) (1987) 1425–1439
9. Hummel, S.F., Schonberg, E., Flynn, L.E.: Factoring: A method for scheduling parallel loops. *Communications of the ACM* **35**(8) (1992) 90–101
10. Banicescu, I., Velusamy, V.: Performance of scheduling scientific applications with adaptive weighted factoring. In: *Proceedings of the 15th IEEE International Parallel and Distributed Processing Symposium - 10th Heterogenous Computing Workshop (IPDPS-HCW 2001)* CDROM, IEEE Computer Society (2001)
11. Banicescu, I., Velusamy, V., Devaprasad, J.: On the scalability of dynamic scheduling scientific applications with adaptive weighted factoring. *Cluster Computing: The Journal of Networks, Software Tools and Applications* **6** (2003) 215–226
12. Cariño, R.L., Banicescu, I.: Dynamic scheduling parallel loops with variable iterate execution times. In: *Proceedings of the 16th IEEE International Parallel and Distributed Processing Symposium - 3rd Workshop on Parallel and Distributed Scientific and Engineering Computing With Applications (IPDPS-PDSECA 2002)* CDROM, IEEE Computer Society (2002)
13. Cariño, R.L., Banicescu, I.: Load balancing parallel loops on message-passing systems. In Akl, S., Gonzales, T., eds.: *Proceedings of the 14th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2004)*, ACTA Press (2002) 362–367
14. Banicescu, I., Liu, Z.: Adaptive factoring: A dynamic scheduling method tuned to the rate of weight changes. In: *Proceedings of the High Performance Computing Symposium (HPC) 2000*. (2000) 122–129
15. Banicescu, I., Velusamy, V.: Load balancing highly irregular computations with the adaptive factoring. In: *Proceedings of the 16th IEEE International Parallel and Distributed Processing Symposium - 11th Heterogeneous Computing Workshop (IPDPS-HCW 2002)* CDROM, IEEE Computer Society (2002)
16. Mississippi State University ERC: EMPIRE: The ERC's Massively Parallel Initiative for Research and Engineering. [http://www.erc.msstate.edu/computing/empire/\(2001\)](http://www.erc.msstate.edu/computing/empire/(2001))
17. Rao, R.P., Kriz, R.D., Abbott, A.L., Ribbens, C.J.: Parallel implementation of the filtered back projection algorithm for tomographic imaging. <http://www.sv.vt.edu/xray-ct/parallel/Parallel.CT.html> (1995)
18. Sha, F., Saul, L.K., Lee, D.D.: Multiplicative updates for nonnegative quadratic programming in support vector machines. In S. Becker, S.T., Obermayer, K., eds.: *Advances in Neural Information Processing Systems 15*. MIT Press, Cambridge, MA (2003) 1041–1048