SHARED MEMORY FOR MATLAB AND OCTAVE: ANOTHER PACKAGE FOR DISTRIBUTED PROGRAMMING

Mariusz Kamola *

* *Warsaw University of Technology,*
ul. Nowowiejska 15/19, 00-665 Warsaw, Poland
M.Kamola@ia.pw.edu.pl
Research and Academic Computer Network,
ul. Wąwozowa 18, 02-796 Warsaw, Poland

Abstract: The need of distributed computation in math environments as Matlab or Octave is nowadays unquestionable. A parallelisation package is presented that gives the user possibility of distributed programming using common memory model and synchronisation mechanisms, also while programming cross-platform Octave and Matlab. Functionality and technical details of the package are described. Also, a benchmarking test is done for comparison of this package and Matlab Distributed Computing Toolbox efficiency. The test problem is Google PageRank algorithm. The results of tests are promising, but the package still needs further maintenance and development.

Keywords: parallel algorithms, mathematical programming.

1. INTRODUCTION

Parallel processing is gaining a respected place among functionalities offered by math environments. However, native support for parallelism was denied any significant role — at least in the case of Matlab. Only recently has the Distributed Processing Toolbox (DCT) provided rich and efficient mechanisms for parallel processing. However, nature abhors a vacuum and in the meantime there sprang a number of third-party parallel computation packages, responding to an apparent demand for such functionality.

In this paper we are going to present yet another package enabling parallel computations for math environments. We also want to provide a rationale behind making the package, especially when DCT and dozen of other products exist. Originally, reasons against native parallelism of Matlab, as explained in (Moler, 1995), were threefold: memory model, granularity and business situation. The memory model has changed since then in the sense that there appeared computational problems so big that their data do not fit into memory of a single computer, and distributed processing becomes a necessity, regardless of the assumed inferior performance of parallelised tasks. Next, it is the granularity that drives for parallel computation. Bigger and bigger problems can be decomposed into long-running tasks that exchange data between themselves rarely, if at all. The above two factors improve at the same time the business situation, finally making DCT a viable product.

The economic aspect is one of the main reasons that third-party alternatives to DCT exist and are in use. They simply provide a cheaper alternative to do a numerical job, especially that some of them allow cooperation of Matlab and Oc-

tave instances, too. Functionality offered by those products is generally a subset of what DCT has. Therefore, there is always some field or niche one can target while writing a parallelisation package like the one presented here.

Our package provides functionality of a virtual shared memory, and synchronisation methods. Its detailed description comes in Section 3, following an overview of the existing solutions, given in Section 2. Next, efficiencies of DCT and the presented package are compared in Section 4, and closing remarks are given in Section 5.

2. EXISTING PARALLELISATION PACKAGES FOR MATLAB AND OCTAVE

There are three main ways (Choy and Edelman, 2005) to enrich non-parallel math environments (as Octave or older Matlab releases) with support for parallel processing. The first one is to transcribe the original program source (script) to some language with support for parallelisation (e.g. C++ plus PVM). In such case the original math environment is used only for basic prototyping. The second way is to connect the environment to some backend software that supports parallel processing. In this case the original math environment is used as a terminal to the backend solver (or solvers), and its functionality is limited to user friendly handling, manipulation and presentation of data.

The third way is to make a number of math environment instances interact in order to solve the problem jointly. This technique is the most conservative in the sense that the biggest part of the environment functionalities remains in use during parallel job solving. It also means that necessary changes and additions that make parallel processing possible can be really few and easy to implement. That may be the reason the third-party parallelisation packages of this kind proliferate. Efficiency of such technique can be high due to the fact that optimised native numerical procedures are still in use. On the other hand, synchronisation and data transmission are the things math environment are not optimised for. DCT and the presented new parallelisation package fall in this category of techniques, so we will focus further on them.

There are basically two ways a number of math environment instances can be put to work together. For embarrassingly parallel (a.k.a. coarse-grained) problems single instruction multiple data (SIMD) approach is applied. This means in practice that one math environment instance becomes the coordinator and distributes the work across the remaining available instances that play the workers. There is no interaction between algorithms processing parts of the problem, and synchronisation procedures are needed only while scattering initial data and gathering results.

The other type of interaction is accomplished through message passing. This gives more flexibility for the programs to interact. In particular, the programs may differ, and may address each other directly by sending any data structure — the message. Let us look at capabilities of packages of both types.

2.1 Parallel Matlab

Packages designed for embarrassingly parallel problems are often used (and sometimes written) by researchers or engineers with little knowledge about distributed programming. Therefore, their most valued features are ease of configuration, use, and portability. A short comparison is done on a representative group of them in Table 1. All they make it possible to run the same script on many Matlab instances, and provide means for data scattering/gathering. The problem can be decomposed by hand or automatically, with load balancing. To make the decomposition more intuitive, constructs like `parfor` are provided; alternatively the way the input data is arranged (e.g. in a matrix with an extra dimension) determines how the job will be distributed. The easiest and most robust technology employed in the packages is a common file system with standard features like file locking or exclusive file creation. Other implementations use TCP based communication, either directly or through another packages. In the course of their development, some packages (Parmatlab, DistributePP) started to offer point-to-point messaging.

Packages designed to solve problems using messaging approach are summarised in Table 2. As it turns out, many of them operate on common filesystem, too. It means that the basic functionality of common filesystem is universal enough for both message passing and scatter/gather packages. In fact, both kinds must have messaging and synchronisation of a kind; their classification depends mostly on what type of communication (high-level i.e. scatter/gather vs. low-level i.e. send/recv) is given to the user. Naturally, using a message passing package requires far better knowledge of parallel programming techniques — and is preferred more by academia than industry. If not the common filesystem, MPI or PVM are the technologies chosen for interprocess communication. A remarkable fact is that there is only one package (MATmarks) presenting shared memory programming model, and it is quite an old one.

Table 1. Packages for solving embarrassingly parallel problems in Matlab

Package Name	Release Year	Communication via	Remarks
MULTI	2000	nfs[a]	Implemented purely using `.m` files. Product apparently discontinued.
Paralize	2006⋆[b]	nfs	Purely using `.m` files. Default distribution along 3rd matrix dimension. A separate Matlab instance needed to run a kind of server. MEX used.[c]
PMI	1999⋆	engine[d]	
PLab	2002	TCP[e]	MEX
Parmatlab	2001⋆	TCP	Parallelisation up to 5 arbitrary dimensions. MEX. Many Matlab versions and operating systems supported.
MPI	2005⋆	engine	Successor of PMI.
DistributePP	2004⋆	nfs	MEX
MULTICORE	2007⋆	nfs	Purely using `.m` files.

[a] Network File System or any equivalent technique allowing computers to work on common file system. The communication and synchronisation mechanisms are made using files.
[b] A star (⋆) means that the package is present in Matlab Central, `www.mathworks.com/matlabcentral`.
[c] MEX is Matlab Executable, a technology to make programs written in C language and Matlab scripts communicate.
[d] Matlab Engine is Matlab functionality for reading and writing data into workspace of remote Matlab instances.
[e] Communication accomplished by ordinary use of IP sockets.

Table 2. Packages for solving parallel problems in Matlab by message passing

Package Name	Release Year	Communication via	Remarks
MultiMatlab	2005	MPI[f]	Available high-level parallelisation commands. MEX+MPICH
CMTM	2006	MPI	MEX+MPI/Pro
DP-Toolbox	2005	PVM[g]	Available high-level parallelisation commands. MEX
MPITB/PVMTB	2000	PVM/MPI	Gives access to PVM or MPI commands from Matlab. MEX+MPI/LAM or PVM
MATmarks	1998	TCP	Shared memory approach; synchronisation of local instances of Matlab objects done occasionally. Based on TreadMarks package.
MatlabMPI	2004	nfs	MPI functionality. Purely using `.m` files.
pMatlab	2005	nfs	Uses MatlabMPI. Provides high-level parallelisation commands.

[f] Specific third-party Message Passing Interface (MPI) implementation is used for communication and synchronisation.
[g] Parallel Virtual Mode (PVM) mechanism used for message passing.

2.2 Parallel Octave

There are not so many packages devoted explicitly for parallelisation of Octave scripts. Three initiatives have to be mentioned, however:

- There exist a bunch of distributed programming primitives maintained by Octave Forge group. Those include commands like send or receive; everything is based on IP socket communications. This group of functions can constitute a base for development of a complete distributed programming package.
- Parallel-Octave — a package that allows interfacing Octave scripts trough MPI/LAM.
- MPITB — a package of the same name and by the same authors as for parallel Matlab (cf. Table 2). Similarly, it allows calling MPI functions from within Octave scripts.

Let us recall that a number of parallelisation packages for Matlab have been written in an orthodox way, using only `.m` files and a common file system. Considered high compatibility of Octave and Matlab, it should be possible to use those packages directly in Octave. However, this solution is not advertised on Octave or Octave Forge websites.

As it turns out, there is a number of alternatives for Matlab DCT, and there is a number of parallel programming solutions for Octave. Many of them are currently maintained, which means that computations on machines with individual Matlab licenses and a free parallelisation package can be more appealing than the acquisition of DCT. As regards Octave, it is planned to provide Octave with DCT-equivalent capabilities (Eaton and Rawlings, 2003).

As regards parallel computations performed by a mixed Matlab/Octave cluster, the need for an appropriate parallelisation package is not pronounced widely. However, in our opinion the rationale for it is at least twofold:

(1) A single Matlab instance may act as a frontend station for a cluster of Octaves. One can take the advantage of superior Matlab data presentation interface and ergonomics of its workspace.
(2) Stations capable of running Matlab can take care of tasks which cannot be done by Octave (e.g. running Simulink or sophisticated toolboxes), while at the same time the rest of stations in a cluster is running Octave in order to perform auxiliary parallel operations in the task (e.g. calculating products, performing directional searches in optimisation).

These are two of many other reasons our parallelisation package has been created.

3. NEW PACKAGE ARCHITECTURE AND FUNCTIONALITY

The idea lying behind creation of yet another parallelisation package was that it is natural to program a distributed application with notion of common memory, available to all processors. It would be convenient for people with programming background to have some mechanisms analogous to interprocess communications (IPCS) while programming on Matlab or Octave cluster. However, providing a natural way of addressing common memory objects is quite a tedious work, and we appreciate the effort done by authors of MATLAB*p, and the DCT itself. The drive for simplicity led us to such solution: common objects are maintained by a server, and up- or downloaded on demand by Matlab and Octave instances. Therefore, at the cost of performance (degraded by unnecessary duplication of data) highly logical model has been created. [1]

3.1 Package operations

In our package all cluster nodes are treated equally since the storage and synchronisation tasks are pushed to the server. The stations can perform the following main operations:

- `matrix_conf(hostname,port)` — specifies the computer name and port number on which the server is listening. This is the only configuration command, issued before any other commands.
- `matrix_put(name,x)` — store the local matrix `x` in the common memory, under name `name`. If an object of such name exist, its data are overwritten with `x`.
- `matrix_get(name)` — get from the common memory the object named `name`; return the object.
- `mutex_lock(name)` — lock the mutex `name`. If the mutex does not exist, it is created and locked.
- `mutex_unlock(name)` — unlock the mutex specified by `name`.
- `mutex_trylock(name)` — try to lock mutex `name`. If it is already locked, error code is returned; otherwise the mutex is locked.
- `barrier(name,strength)` — wait on barrier named `name` until the number of waiting processes reaches `strength`.

[1] One can see an analogy in how the computer works: the data stored in RAM has to be fetched to a CPU register before performing an operation; similarly the result has to be put back to RAM.

Therefore, a simple program calculating product `c` of matrices `a` and `b` might look like that:

```
matrix_conf('localhost',7575);
a=matrix_get('a');
b=matrix_get('b');
ci=a(i,:)*b;
mutex_lock('lock');
c=matrix_get('c');
c(i,:)=ci;
matrix_put('c',c);
mutex_unlock('lock')
```

In the example, `i` denotes the row index the cluster node is responsible for. Matrices `a`, `b` and `c` pre-exist in the common memory. One row of the product `a*b` is computed by each node; after that the solution `c` in the common memory is updated with partial results calculated by a node.

3.2 Package architecture

For each operation, the operation code and any arguments are serialised and sent over freshly opened socket to the server, where they are processed. The result is sent back the same way, deserialised and returned to the caller. The operation call is blocking, so it depends on the server whether the caller will hang, which is important for accomplishing synchronisation. The communication procedures are written in C++ language and interfaced with Matlab (via MEX) or Octave (via MEX equivalent). The client code is therefore a dynamically loaded library. No multithreading or forking is done on the client side.

The server has been implemented in Java, as it has good support for synchronisation and networking. For each incoming request, a new thread is created. Next, the thread carries out the submitted operation request. Implementation of synchronisation between threads is particularly clear and stable this way.

3.3 Discussion

If the package being presented was to be classified according to criteria in Section 2, it should be considered (by negative selection) as a message passing one, although there is no explicit massaging. Instead, we have a commonly available repository of data and a number of synchronisation routines capable together of providing functionality equivalent to that offered by packages from Table 2. Conceptually, our package is close to MATmarks and the DCT itself, but uses far simpler handling of data: just copying. As regards communication technologies, it is close to PLab and MATmarks again (by the use of plain TCP) and also to Paralize (by running a central server).

Major advantages of our package are the common memory programming model, i.e. the one any programming novice learns, and autonomous repository and synchronisation server. The fact that the server is implemented in Java guarantees portability and opens up possibilities of linking the cluster controlled this way with any other service present in the network. Also the fact that the data is stored outside Matlab or Octave mechanisms guarantees interoperability of Matlab and Octave interacting through this package.

The disadvantages are also a few, especially that the package is not mature nor maintained and developed regularly. Another one is that whole objects must be transmitted, and not their parts (as in Paralize or Parmatlab). Also, basic data types are now supported: the real and the complex. This diminishes appeal of the package, particularly that object oriented programming is in fashion — and must be fixed quickly. Finally, there is some potential in code optimisation; especially in the way the data connections are maintained. Probably keeping one socket connection to the server open all the time would increase the speed (connection establishment and adequate sizing of the transmission window usually takes much time), but at the cost of lesser robustness and more complicated code.

4. BENCHMARKING

Performance of our package has been compared in a series of tests to that of DCT. The testing environment was a cluster of PCs with Intel Pentium4 processors operating at 3GHz, with 2GB RAM each, and 1Gbps network cards. The operating system was Linux.

4.1 Test problem and test plans

The algorithm selected for the benchmark was PageRank routine used by Google (Page *et al.*, 1998) for assigning ranks to web pages. In fact, the process of following links pointing to other web pages can be modelled by a Markov process with states corresponding to web pages. State transition probabilities between n web pages are stored in a n-by-n matrix $\mathbf{A}$. Page ranks are just the elements of an eigenvector $\mathbf{x}$, i.e. $\mathbf{x} = \mathbf{A}\mathbf{x}$.

The structure of $\mathbf{A}$ guarantees that the solution can be found by a simple iteration procedure with step i: $\mathbf{x}^{(i)} = \mathbf{A}\mathbf{x}^{(i-1)}$. It must be noted, however, that $\mathbf{A}$ is not sparse, although the matrix of incidencies for graph of web pages (nodes) and links between them (edges) is sparse. This is because the action of going to any location address typed in the location bar in the browser must be modelled by nonzero probabilities where normally zeros should be. Therefore we are facing dense matrix multiplication problem of size in the order of billions.

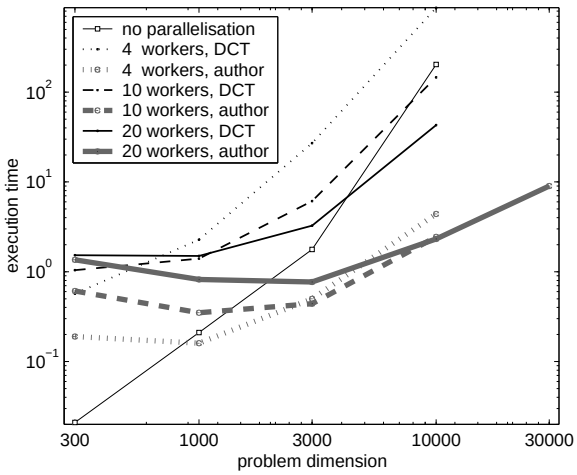

Fig. 1. Average execution times of the test problem for various mechanisms, problem dimension and number of CPUs used.

The problem can be decomposed across k nodes: each station m computes its own part of the solution, $(x_{1+m*n/k}, \ldots, x_{(m+1)*n/k})$ using some (probably outdated) approximation of the solution stored in a shared memory. The stations maintain that commonly available approximation of the solution; they upload their part of the solution approximation periodically (alternatively, they broadcast their solutions to the others every so often.)

The test plan covers comparison of computation times achieved by original (single-station) and distributed (DCT) algorithms. Also, the frequency of synchronisation between stations will be examined. Next, DCT will be compared with our package in terms of efficiency.

4.2 Test results

A series of tests have been performed in various configurations, taking as sole performance metrics the total execution time, i.e. the time until all local solutions converge. The termination criterion used was $\max_{j=1,\ldots,n} |x_j^{(i)} - x_j^{(i-1)}| < \varepsilon$. All the major quantitative results have been presented synthetically in Fig. 1.

Quite expectedly, this problem shows in practice its numerical complexity of order $O(n^2)$, represented by 'no parallelisation' graph, until RAM resources are exhausted ($n = 10000$) and intense swapping takes place. Decomposition of the problem and parallel solving using DCT gave results indicated by thin black lines of patterns getting

dense as the number of processor grows (dotted, dashed and solid lines). As it turns out, parallel processing introduces substantial overhead so that parallelisation pays off only in case when the problem was not processing power but RAM storage, which is consistent with what was stated by (Moler, 1995) a dozen of years ago. Results obtained with four workers are particularly discouraging because DCT always performed an order of magnitude worse than single-station computations.

There were attempts to improve things by updating the common solution approximation every 3, 10, 30 or 100 steps, instead of every single step $\mathbf{x}^{(i)} = \mathbf{A}\mathbf{x}^{(i-1)}$. This was done in hope that reduced network I/O operations will give more time for calculations and finally compensate the fact that synchronisation is not done so often. However, things have not changed much in any of the tested cases.

The results of running the author's parallelisation package (presented in Fig. 1 with thick grey lines) proved to be quite interesting. For the problem dimension n growing, the package initially imposes an overhead similar to DCT, only to surpass single-station performance for as small n as 1000. A peculiar phenomenon is that the computation times have their minima not for the smallest n but for problems that are bigger, i.e. of size correlated with the number of workers used. Such behaviour is difficult to explain; it may be caused by scheduler or TCP operation for extremely small portions of data. In any case, the package seems to be useful for problems of size reaching 30000 — where DCT hangs or reports communication errors, and single-station processing is impossible due to memory limits. Another positive observation is that computation time for $k = 20$ and $n = 3000 \div 30000$ grows an almost constant rate (on log-log graph).

The test problem was also solved using single-station Octave. The execution times were about three times longer than in Matlab, which means in particular that coupling Octaves with the parallelisation package still gives results quicker than computations by a single Matlab instance.

5. CLOSING REMARKS

The parallelisation package presented here can complement the already existing similar solutions for Matlab and Octave. Providing an imitation of common memory can be appealing for those with programming experience, especially at universities. For the considered test problem the package performs definitely better than DCT when used in *pmode*. Unfortunately, only one problem has been tested so far, and the results given in Section 4 have to be considered tentative.

Also, there is much to be done: the current version does not have support for sparse matrices; neither it offers accessing a slice of matrix stored in common memory. These and other ways of development are open for the public and supported by the author since the package is publicly available.

From the economic perspective, the package can be attractive alternative for institutions already possessing a number of Matlab licenses. Otherwise, one has to raise some EUR 2,300 for a decent cluster configuration (one DCT license plus eight distributed computing engine licenses). The package can address especially those not interested in full DCT functionality. Once again, academic institutions where Matlab licenses stay idle when there are no labs can perform computationally demanding jobs off the hours.

ACKNOWLEDGMENTS

The package has been done by the expansion of student projects by Ms. Katarzyna Pietron, Mr. Dariusz Dudek and Mr. Marek Krajewski.

The cluster for tests was located in Photonics and Web Engineering Research Group Laboratory/Electronics for High Energy Physics Experiments Laboratory, Warsaw Univ. of Technology. The author wants to thank Mr. Michał Matuszewski and all the laboratory staff for their help.

The author thanks Mr. Andrzej Karbowski, PhD for his valuable remarks.

REFERENCES

Choy, R. and A. Edelman (2005). Parallel MATLAB: Doing it right. In: *Proceedings of the IEEE*. Vol. 93. pp. 331–341.

Eaton, John W. and James B. Rawlings (2003). Ten years of Octave — recent developments and plans for the future.

Moler, C. (1995). Why there isn't a parallel MATLAB.In Cleve's Corner at `www.mathworks.com`.

Page, Lawrence, Sergey Brin, Rajeev Motwani and Terry Winograd (1998). The pagerank citation ranking: Bringing order to the web. Technical report. Stanford Digital Library Technologies Project.