Parallelization of MATLAB for Euro50 integrated modelling

M. Browne*[a], T. Andersen[b], A. Enmark[b], D. Moraru[b], A. Shearer[a]
[a]Dept. of Information Technology NUI Galway, Galway, Rep. of Ireland.
[b]Lund Observatory, Box 43, SE-221 00 Lund, Sweden.

ABSTRACT

MATLAB and its companion product Simulink are commonly used tools in systems modelling and other scientific disciplines. A cross-disciplinary integrated MATLAB model is used to study the overall performance of the proposed 50m optical and infrared telescope, Euro50. However the computational requirements of this kind of end-to-end simulation of the telescope's behaviour, exceeds the capability of an individual contemporary Personal Computer. By parallelizing the model, primarily on a functional basis, it can be implemented across a Beowulf cluster of generic PCs. This requires MATLAB to distribute in some way data and calculations to the cluster nodes and combine completed results. There have been a number of attempts to produce toolkits to allow MATLAB to be used in a parallel fashion. They have used a variety of techniques. Here we present findings from using some of these toolkits and proposed advances.

Keywords: MATLAB, Euro50, Simulation, MPI, Parallel

1. INTRODUCTION

Euro50[1] as rendered in figure 1 is a proposed European Extremely Large Telescope (ELT). It has a segmented primary mirror with an elaborate control system ("live optics"), dual conjugate adaptive optics, automatic alignment of the secondary mirror and numerous rotators, in addition to the normal primary motions of the telescope. An integrated model[2] has been built in MATLAB to study the performance of this system. The model includes the dynamic performance of the telescope structure, the primary mirror segment alignment system, the adaptive optics, the atmosphere and the optics of the telescope. Disturbances caused by wind and detector noise are also included.

Fig. 1. Rendering of Euro50 on-site.

MATLAB by Mathworks[3] Ltd., its Toolboxes and companion product Simulink are commonly used 'desktop tools' in systems modelling and other scientific disciplines. However, the requirements of the cross-disciplinary end-to-end simulation of the telescope's behaviour exceeds the capability of an individual contemporary Personal Computer (PC). By parallelizing the model, primarily on a functional basis, it can be implemented across a Beowulf cluster of generic

Modeling and Systems Engineering for Astronomy, edited by Simon C. Craig, Martin J. Cullum, Proc. of SPIE Vol. 5497 (SPIE, Bellingham, WA, 2004)
0277-786X/04/$15 doi: 10.1117/12.551317

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 09/23/2013 Terms of Use: http://spiedl.org/terms

PCs. This requires the MATLAB environment to be extended to in some way distribute data and calculations to the cluster nodes and return completed results. Cleve Moler[7] of Mathworks has clearly stated that a parallel MATLAB is not viable, however that was in 1995 and High Performance Computing (HPC) practice has changed considerably since then, as the economic advantage and performance of clusters has risen.

2. MODEL STRUCTURE

The model has three execution phases; initialization, execution and post processing. A functional parallelization of the system is used where different nodes represent and simulate different parts of the telescope. The master node, on which the primary differential equation solver is executed, controls interaction between the individual nodes.

The solver is capable of handling subsystems with different dynamical performance, see figure 2, hence spending less computer time on subsystems with slow dynamics than on those with fast. The system is organised in a modular way so that it can be easily changed to model other designs. By implementing the model on a cluster, two disadvantages of a traditional single PC approach can be challenged; speed and memory footprint.

Many systems modelling problems use very large matrices. While today's PC processors can often manipulate the matrices more quickly than specialist hardware, the size of the matrices exceeds the capacity of commodity hardware memory subsystems and of conventional operating systems. As a result it is sometimes necessary to use traditional HPC hardware for example SGI machines (e.g. calvin.nuigalway.ie Origin 3800) purely for memory and operating system reasons. This has several problems; such hardware is rare and gaining access can be difficult and expensive. If a problem is loosely coupled, does not require massive I/O or is not highly parallel, computational performance can even decrease as processors in such systems are generally slower than commodity PC processors, though much better catered for in terms of I/O.

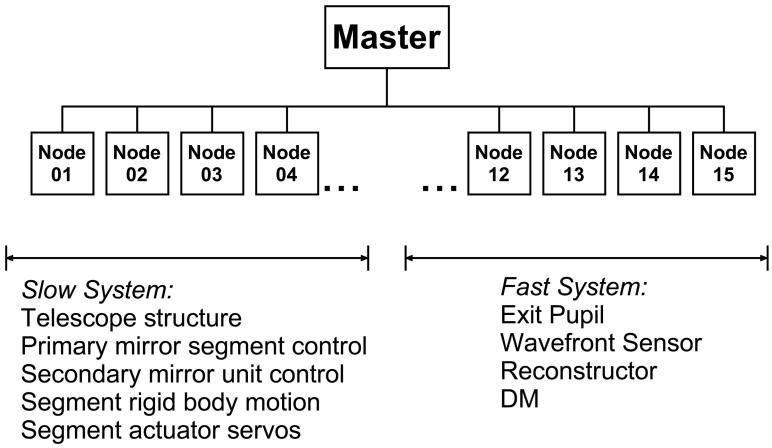

Fig. 2. Functional parallelization strategy.

By spreading the memory requirement across a cluster in such a way that the communications requirement is minimised, as the functional decomposition does, a simulation which was previously impossible can be accomplished with the possibility of using the additional processors to then increase performance.

3. MATLAB TOOLKITS

MATLAB does not include any parallel functionality however there are basic interfaces to the TCP/IP network stack, C and Fortran. By building upon these interfaces there have been of the order of 30 attempts[4] to produce 'toolkits' to allow

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 09/23/2013 Terms of Use: http://spiedl.org/terms

MATLAB to be used in a parallel fashion. They have used a variety of techniques and approaches. Communication between the nodes is generally handled in either of two ways, via a shared file system e.g. NFS or a TCP/IP connection. Some of the toolkits require a MATLAB daemon and hence license on each node others do not. The commands called from within MATLAB to distribute the computation vary. Several of the toolkits take a low-level approach using commands that are syntactically and functionally very similar to the popular Message Passing Interface (MPI) libraries. Others use a more high level approach choosing to use simpler commands that resemble more closely those of MATLAB.

MPI is a widely used standard for writing message passing programs. MPI is standardised by the Message Passing Interface Forum (MPIF), with participation from over 40 other organizations. MPI is a message passing library specification; it is not a language or compiler specification. There are several popular implementations such as MPICH and LAM MPI as well as vendor specific implementations. MPI is almost always used with C or Fortran. Relative to development in MATLAB it is considered difficult to use.

The suitability of MATLAB for model development, visualisation and general scientific and engineering problem solving is beyond question. However, prior to experimenting with various toolkits the suitability of MATLAB for large-scale parallel computation relative to traditional C and Fortran was evaluated. This was done by taking representative problems and recoding the computational intensive parts in C or Fortran. MATLAB was then made to execute the compiled code in place of its own internal functions to give a performance comparison.

To increase performance the current release of MATLAB (version 6.5) uses just in time (JIT) compilation similar in concept to that used in many Java implementations. Instead of simply interpreting m code as in previous releases, code is actively profiled and performance critical code is optimised and compiled without any user intervention. Experiment showed that MATLAB could indeed compete with the traditional HPC languages in raw performance. Typically attaining 90% of the performance of Fortran 90 on calculations large enough to minimise the overhead of moving variables to and from the conventional MATLAB environment with a few caveats. The process of using C or Fortran code within MATLAB is relatively complex, error prone and often unfamiliar to MATLAB users. This further emphasises the need to use multiple machines from within MATLAB, using MATLAB calls, to increase model performance.

A selection of toolkits, that were considered to be relevant and in popular use were compared. An examination of their capabilities, performance and ease of use was carried out. An example of one of the best of such toolkits is "Matlab Parallelization Toolkit" by Einar Heiberg[5]. The toolbox is released as Open Source. It is most suitable to problems where the amount of communication is low. The toolkit adopts a Master/Slave paradigm. As it is based on the network stack rather than shared memory, it is suitable for use on clusters. The master process is responsible for starting the required number of slaves either on the same machine or on other machines within a cluster. The master and slaves each run a full MATLAB session. Communication between the master and slaves is carried out via TCP/IP pipes. A useful selection of low-level calls modelled on the most common MPI functions are available along with several high level commands such as parallelized for-loops, profiler and debugging. One of the easiest commands to understand is the high-level parallelized for-loop, assuming sufficient slaves have been initialised and the variables are in vector form each iteration of the loop is executed separately and simultaneously. For finer control in more complex problems calls to send variables and MATLAB commands to specific slaves and retrieve results are available.

MatlabWS is a parallel MATLAB toolkit developed by co-author D. Moraru specifically for the Euro50 modelling effort. It is general purpose in design and has many possible applications in HPC. It extends the notion of workspaces within MATLAB to include those of other MATLAB engines which can be running on separate machines. Remote MATLAB engines can be started and controlled from a master engine. Variables can be readily moved to and from remote engines. The computational resources of the remote engine can thus be used to work on transferred variables and return the results. The initial version of MatlabWS was conceptually simple with a straightforward syntax. However it suffered latency problems similar to those of other toolkits as it too used MATLAB's own communication routines.

Communications between 2GHz Pentium IV based machines over via dedicated 100Mbps ethernet can easily saturate the available bandwidth, slowing transmission of large variables. In practise, a greater problem is latency. Tests using the Heiberg toolkit have shown that on 100Mbps ethernet there is minimum period of 35ms of latency involved in any

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 09/23/2013 Terms of Use: http://spiedl.org/terms

communications between MATLAB instances on separate cluster nodes regardless of message size, see figure 3. While moving to gigabit ethernet would increase bandwidth, it would have little impact on latency.

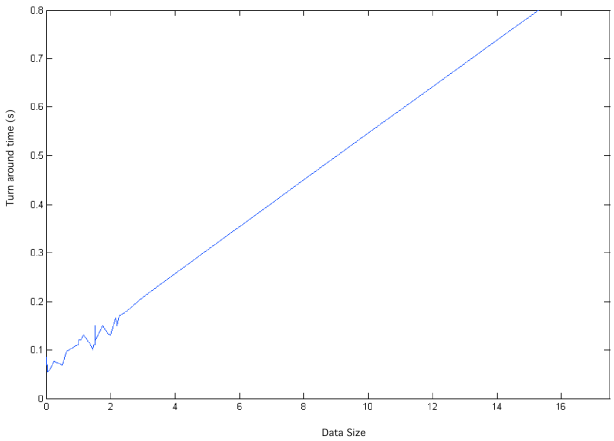

Fig. 3. Minimum MATLAB turn around time.

Figure 4 shows the minimal difference in latency between Gigabit ethernet and Fast ethernet (100 Mbps) at low throughput rates. For higher throughput levels using larger messages the difference is of less importance because the total transfer time will greatly exceed the latency. The uneven and lower than expected Gigabit ethernet performance at higher throughput levels are attributed to non-optimal drivers, in this case for Linux. Considerable differences in performance between models of network interface cards can also be seen. Gigabit card performance can suffer if installed in regular 32 bit PCI slots. Switching equipment must also be considered.

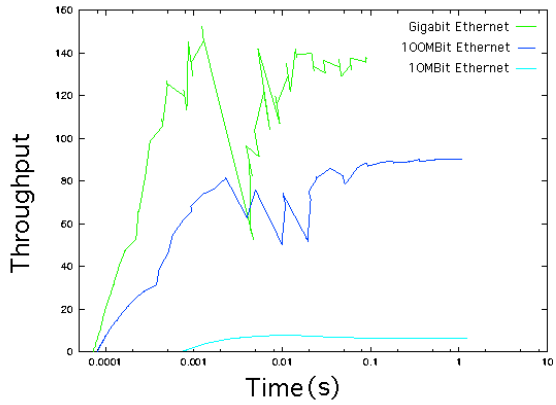

Fig. 4. Ethernet latency comparison[8]

Recent work has changed the manner in which MatlabWS handles communications, with very positive results. Additional communication and connection management routines have been written in C++ using the Trolltech Qt libraries. This code is compiled into MEX files which are directly callable from MATLAB.

The master MATLAB engine runs a multi-threaded application. The MatlabWS event-loop runs as a separate thread within the context of the MATLAB interpreter process. This results in very fast access to the local workspace and interpreter, as no interprocess communication is required. Standard techniques in multi-threaded programming such as semaphores and mutexes are used to control access to shared variables. The most common method of serialising MATLAB variables for transport or storage is to write out a MAT file containing them. To avoid this overhead

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 09/23/2013 Terms of Use: http://spiedl.org/terms

MatlabWS can efficiently serialise built-in data types in memory. In the future support for user-defined objects and cell structures may be added along with data compression.

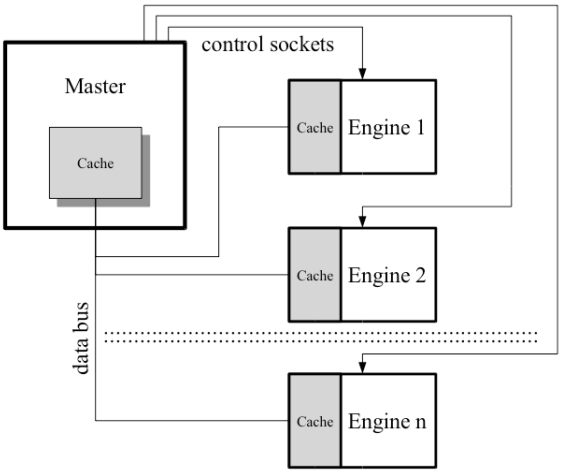

Fig. 5. MatlabWS structure.

Transmission of variable data and engine control commands use separate network sockets, see figure 5. All MATLAB engine control is carried out via the master node but data transmission can also be on a slave-to-slave basis. Remote MATLAB engines are started using either SSH or RSH. The slave STDIN and STDOUT streams are redirected to the master node for monitoring and debugging. The slave engines run a single threaded application. This executes entirely within a MEX file. It takes input from the master on its control socket and maintains data socket connections to other engines. The MEX file terminates if the connection to the master is lost. A simple cache of variables destined for a particular engine is also maintained; the engine then fetches the variables from cache as they are needed. This is transparent to the user. The following is an example MatlabWS's intuitive syntax:

```
% start a MATLAB engine with its own workspace called '1' on node01
engOpen(1,'node01');

% Create a matrix of random values and assign them to variable
% A in workspace '1'
engPutVariable(1,'A',rand(1000));

% On engine '1' calculate the inverse of A and return the results
% to a variable B in the master workspace
B = evalin(1, 'inv(A)');

% Close the engine on node01
engClose(1);
```

Figures 6 shows the data throughput achieved by MatlabWS for a range of packet sizes. For larger packets it is very close to the theoretical maximum. For smaller packets, less efficient use is made of the available bandwidth and so the data transfer rate is lower.

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 09/23/2013 Terms of Use: http://spiedl.org/terms

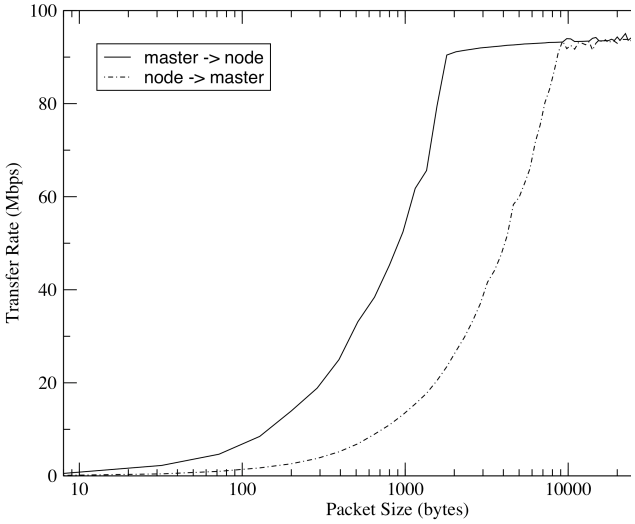

Fig. 6. MatlabWS data throughput.

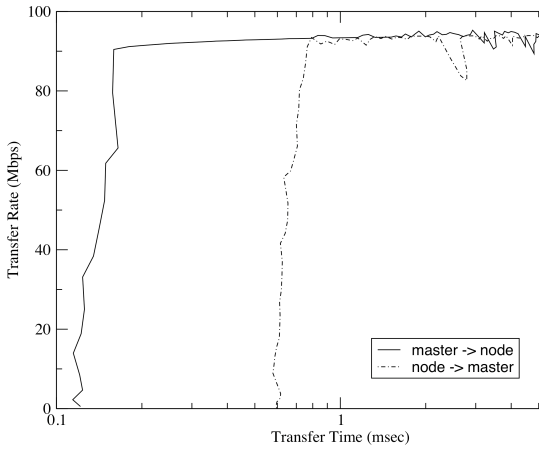

Fig. 7. MatlabWS latency performance.

Figure 7 shows the very much lower latencies achieved by MatlabWS by not using MATLAB's built-in communications routines. Typical latencies are below 0.5 ms for small data transfers and commands. For larger transfers latencies are higher but less important as in such cases the overall transfer time is dominant.

4. CONCLUSIONS

To make development easier we intend to write wrappers for existing commands, this will allow us to consistently present the best commands from a selection of toolkits with a MATLAB like syntax. In addition, a simple yet valuable feature to be added is "clusterless operation", such that if a cluster is not available, commands will just execute locally without any code changes. This will be particularly useful during code development.

We investigated a low latency MATLAB message passing scheme using operating system interprocess communication (IPC) calls. This could sit alongside network transport code and provide lower latency messaging when hardware permits. Initial results from calvin give average latencies of 270μs. On a single processor 2GHz Pentium IV the figure is

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 09/23/2013 Terms of Use: http://spiedl.org/terms

300μs. However these times are of the same order as MatlabWS with the requirement of symmetric multiprocessing, (SMP). This indicates that there is room for improvement in these figures. In addition another toolkit MPITB[6] offers comparable latencies over ethernet, using native MPI. MPITB uses the LAM MPI implementation. However this toolkit is more complex to configure and use. MatlabWS's combination of the ease of use of the Heiberg toolkit with the performance of MPITB make it a very compelling product.

The improvements in latency have resulted in a reduction in typical model run time from 70 hours to 24 hours. It is hoped that architectural changes that better exploit a lower latency environment can reduce this still further. Tests will shortly be conducted using a gigabit ethernet based cluster to examine the available performance gains with minimal hardware expenditure and no software changes.

ACKNOWLEDGEMENTS

The NUI, Galway aspect of this work has been carried out as part of the Science foundation Ireland (SFI), WebCom-G programme & the Enterprise Ireland Cosmogrid programme.

REFERENCES

1. Lund Observatory, *http://www.astro.lu.se/~torben/euro50/*, 2004.
2. Torben Andersen, Michael Browne, Anita Enmark, Dan Moraru, Mette Owner-Petersen and Holger Riewaldt. *"Integrated Modelling of the Euro50"*, 2nd Bäckaskog Workshop on Extremely Large Telescopes 2003.
3. *http://www.mathworks.com*
4. R. Choy. *MATLAB Survey. http://supertech.lcs.mit.edu/%7Ecly/survey.html*, 2001.
5. Einar Heiberg. *"Matlab parallelization toolkit". http://hem.passagen.se/einar heiberg/index.html*, 2001.
6. http://atc.ugr.es/javier-bin/mpitb_eng, 2004.
7. C. Moler. *"Why there isn't a parallel matlab"*. Cleve's corner, Mathworks Newsletter, 1995.
8. Ebert et al. "*Gigabit Ethernet and Low-Cost Supercomputing".* Ames Laboratory, Iowa State University, 1996.

*brownemi @ it.nuigalway.ie; phone +353 (0)91 524411 ext. 3180

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 09/23/2013 Terms of Use: http://spiedl.org/terms