Parallel Performance Prediction for Numerical Codes in a Multi-Cluster Environment

by

Giuseppe Romanazzi & Peter K Jimack

April 2008
Abstract. We propose a model for describing and predicting the performance of parallel numerical software on distributed memory architectures within a multi-cluster environment. The goal of the model is to allow reliable predictions to be made as to the execution time of a given code on a large number of processors of a given parallel system, or combination of systems, by only benchmarking the code on small numbers of processors. This has potential applications for the scheduling of jobs in a Grid computing environment where informed decisions about which resources to use in order to maximize the performance and/or minimize the cost of a job will be valuable. The model is tested using a particular class of numerical code, based upon the multigrid solution of discretized partial differential equations, and despite its simplicity it is demonstrated to be extremely accurate and robust with respect to both the processor and communications architectures considered. The potential significance of this work is illustrated via two scenarios which consider a Grid user who wishes to use the available resources either (i) to obtain a particular result as quickly as possible, or (ii) to obtain results to different levels of accuracy.

Keywords: Parallel Distributed Algorithms; Grid Computing; Cluster Computing; Performance Evaluation and Prediction.

1 Introduction

As Grid computing becomes available as a practical commodity for computational science practitioners the need for reliable performance prediction becomes essential. In particular, when a variety of computational resources are available to a scientific research team they need to be able to make informed decisions about which resources to use, based upon issues such as the size of the problem they wish to solve, the turn-around time for obtaining their solution and the financial charge that this will incur. In order to make such decisions in a reliable way, it is necessary that they are able to predict the performance of their software across different combinations of these resources.

In this work we present a novel and robust methodology for predicting the performance of parallel numerical software across different clusters (in terms of both processor and communications architectures) and across combinations of these clusters. The methodology is first described and its predictive capability is then assessed for five different cluster configurations, using a typical parallel multigrid code. Following [7] multigrid has been selected due to its high relevance to modern computational simulation (e.g. [4]) however the methodology proposed is not limited to such algorithms.

2 Related Work

This work builds upon a very substantial body of research into performance modeling that varies from analytical models designed for a single application
through to general frameworks that can be applied to many applications on a large range of high performance computing (HPC) systems. For example, in [5] detailed models of a particular application are built for a range of target HPC systems, whereas in [1] an application trace is combined with some benchmarks of the HPC system used in order to produce performance predictions. Both approaches have been demonstrated to be able to provide accurate and robust predictions, although each has its potential drawbacks: significant code specific knowledge being required for deriving the analytic models, whereas the trace approach may require significant computational effort.

Our approach lies between these two extremes. We use relatively simple analytic models (compared to [2] for example), that are applicable to a general class of algorithms and then make use of a small number of simulations of the application on a limited number of CPUs of the target architecture in order to obtain values for the parameters of these models. Predictions as to performance of the application on larger numbers of processors may then be made. This idea has already been shown to work well when the communication scales well with the number of processors in [7] and in this work we improve the robustness of our model to also capture architectures that do not scale so well.

3 Parallel Numerical Software

Most numerical methods for the solution of partial differential equations (PDEs) are based upon the use of a spatial mesh for performing the discretization (as in finite difference, element, etc.). The main goal for the parallel numerical solution of PDEs is to be able to solve problems on finer grids than would be otherwise possible, so as to achieve greater accuracy. With multigrid algorithms, when the problem size is increased by a factor of \( np \) then the solution time also grows by this factor, and so when solving on \( np \) processors (instead of a single processor), the solution time should be unchanged. This would represent a perfect efficiency but is rarely achieved due to parallel overheads such as inter-processor communications and computations that are repeated on more than one processor.

In this research our aim is to be able to predict the execution time, including these overheads, of parallel numerical software running on \( np \) processors. We restrict our attention to mesh-based PDE solvers, in this case considering a finite difference code with a series of sends and receives in MPI, that solves a model PDE problem over a square two-dimensional domain (of size \( N \times N \), say), see \( m1 \) in [7]. This domain is uniformly partitioned across the processors by assigning contiguous rows of the mesh to each processor in turn. In the case of a multigrid solver, partitioning the coarsest mesh in this manner ensures all finer meshes are uniformly partitioned too (see [6] for further details). The right-hand diagram in Figure 1 illustrates a typical partition when \( np = 4 \). Each stage of the parallel numerical solver requires communications between neighbouring processors in order to update their neighbouring rows.
4 The Predictive Model

The underlying observation upon which our model is based is that when we scale the size of our computational problem with the number of processors, the parallel overheads observed using just a small number of processors can describe the communication pattern of the problem for a much larger number of processors. The next basic assumption that we make is that the parallel solution time (on \( np \) processors) may be represented as

\[
T_{\text{parallel}} = T_{\text{comp}} + T_{\text{comm}}.
\]  

In (1), \( T_{\text{comp}} \) represents the computational time for a problem of size \( N \times N(1) \) on a single processor (where \( N(1) = N/np \)), and \( T_{\text{comm}} \) represents all of the parallel overheads (primarily due to inter-processor communications).

The calculation of \( T_{\text{comp}} \) is straightforward since this simply requires the execution of a problem of size \( N \times N(1) \) on a single processor. Note that it is important that the precise dimensions of the problem solved on each processor in the parallel implementation are maintained for the sequential solve in order to obtain an accurate value for \( T_{\text{comp}} \).

The more challenging task is to model \( T_{\text{comm}} \) in a manner that will allow predictions to be made for large values of \( np \). Recall that our goal is to develop a simple model that will capture the main features of this class of numerical algorithm with just a small number of parameters that may be computed based upon runs using only a few processors. We present this model in (2) and then justify its simplicity in the remainder of the section.

\[
T_{\text{comm}} = \alpha(np) + \gamma(np) \cdot \text{work}.
\]  

In (2) the term \( \text{work} \) is used to represent the problem size on each processor, and is expressed in MBytes of the memory required. Also note that the length of the messages \( (N) \) does not appear in this formula since it is assumed that for a given size of target problem (e.g. a mesh of dimension 65536 \( \times \) 65536) the size
of the messages is known \textit{a priori} (in this case, since the partition is by rows, the messages will be of length 65536). Hence there is no need to include \( N \) in the model as it is fixed in advance. This is the primary reason that the expression (2) can be so simple.

Furthermore, we will assume that the following relations also hold:

\[
\alpha(np) \approx c + d \log_2(np) \tag{3}
\]

\[
\gamma(np) \approx \text{constant}. \tag{4}
\]

The justification for this model and the above assumptions is based upon a substantial body of empirical evidence on different parallel architectures. Two such illustrations are provided in Figures 2 and 3. These show plots of overhead against work for two different systems: one based upon a Fast Ethernet switch and the other based upon Myrinet. In each case we observe an almost linear growth in overhead with work, where the slope is approximately constant and there is an almost constant difference between graphs as \( np \) is doubled. Note that the length of the messages is the same in all of these runs (see Figure 1 for constant work with two different choices of \( np \) and Figure 4 for the same \( np \) but half the work per processor).

![Fig. 2. Overhead \((T_{\text{comm}})\) on an Ethernet system with a fixed size of messages \((N)\).](image)

In order to be able to use the model (2) it is necessary to evaluate the parameters \( c, d \) and \( \gamma \). These are determined using measurements taken for \( np = 4 \) and \( np = 8 \): \( \gamma = \gamma(8) \) and \( c \) and \( d \) are obtained using a simple linear fit through the two data points.
Fig. 3. Overhead ($T_{comm}$) on a Myrinet system with a fixed size of messages ($N$).

A summary of the overall predictive methodology is provided by the following steps. We define as $N \times N$ and $np$ the target problem size and number of processors respectively (i.e. we wish to predict a code’s performance for these values). Also, let $N(1) = N/np$ and define $N \times N(1)$ to be the size of problem on each processor in the target configuration.

1. For $\ell = 1$ to 2
   (a) Run the code on a single processor with a fine grid of dimension $N \times (4^{1-\ell}N(1))$, define $work \propto N \times (4^{1-\ell}N(1))$.
   In each case collect $T_{comp}$ based upon average timings over at least 5 runs.

2. For $\ell = 1$ to 2
   (a) Run the code on $np0 = 4, 8$ processors, with a fine grid of dimension $N \times np0 \times (4^{1-\ell}N(1))$.

Fig. 4. Scaling the work per processor whilst maintaining the communication volume.
In each case collect $T_{\text{parallel}}$ based upon average timings over at least 5 runs, and then compute $T_{\text{comm}} = T_{\text{parallel}} - T_{\text{comp}}$.

3. Fit a straight line as in Eq. (2) (for both choices of $np = np0$) through the data collected in steps 1 and 2 to estimate $\alpha(np0)$ and $\gamma(np0)$.

4. Fit a straight line as in Eq. (3) through the points $(2, \alpha(4))$ and $(3, \alpha(8))$ to estimate $c$ and $d$: based upon Eq. (3) now compute $\alpha(np)$ for the required choice of $np$.

5. Use model in Eq. (2) to estimate the value of $T_{\text{comm}}$ for the required choice of $np$ (using the values $\gamma(np) = \gamma(8)$ and $\alpha(np)$ determined in steps 3 and 4 respectively).

6. Combine $T_{\text{comm}}$ from step 5 with $T_{\text{comp}}$ (determined in step 1, for $\ell = 1$) to estimate $T_{\text{parallel}}$ as in Eq. (1).

5 Numerical Results

The approach described in the previous section is now used to predict the performance of a typical numerical code running on two different clusters, either individually or together.

5.1 The White Rose Grid

The White Rose Grid is a collaborative project involving the Universities of Leeds, Sheffield and York [3]. In these tests we make use of two clusters on this Grid.

– WRG2 (White Rose Grid Node 2) is a cluster of 128 dual processor nodes, each based around 2.2 or 2.4GHz Intel Xeon processors with 2GBytes of memory and 512 KB of L2 cache. Either Myrinet or Fast Ethernet switching may be used to connect the nodes.

– WRG3 (White Rose Grid Node 3) is a cluster of 87 Sun microsystem dual processor AMD nodes, each formed by two dual core 2.0GHz processors. Each of the $87 \times 4 = 348$ batched processors has L2 cache memory of size 512KB and 2GBytes of physical memory. Again, both Myrinet and Fast Ethernet switching are available.

In addition to running jobs on either cluster, using either switching technology, it is also possible to run a single parallel application across both clusters together (using Fast Ethernet only).

Because users of WRG2 and WRG3 do not get exclusive access to their resources some variations in the execution time of the same parallel job can be observed across different runs. A simple way to reduce such effects in the predictive methodology is to take average timings on a limited number of runs (five runs, say). However, this approach alone is not sufficient since specific hardware features must also be accounted for.

For cluster WRG2, for example, there are (75) 2.4GHz and (53) 2.2GHz dual processors, hence it is necessary to ensure that all runs used in the parameter
estimation phase make use of at least one slower processor. This is because if only
the faster processors are used to estimate $T_{comm}$ and $T_{comp}$, then the resulting
model will under-predict solution times on large numbers of processors (where
some of the processors will be 2.2GHz rather than 2.4GHz). Similarly, on the
multicore cluster WRG3, care needs to be taken to account for this architectural
feature. For example, all of the sequential runs are undertaken using four copies
of the same code: each running on the same node. Again, this decision is made
bearing in mind the situation that will exist for a large parallel run in which
all the available cores in a node are likely to be used. Moreover on this cluster
the 8 core runs, distributed as two full nodes, are able to catch both intra- and
inter-node communications, see [7] for further details.

5.2 Methodology for Inter-Cluster applications

As mentioned above, it is also possible to run a single job across both clusters
using Switched Ethernet. Figure 5 illustrates a typical partition, for which the
work per processor may be different on each cluster. In this example a target
configuration with $np_2$ processes on WRG2 (each working with a sub-mesh of
size $N \times N_2(1)$) and $np_3$ processes on WRG3 (each working with a sub-mesh of
size $N \times N_3(1)$) is assumed. In order to predict the overall solution time for such a
multi-cluster run we make the assumption that the inter-cluster communication
costs, whilst greater than those within each cluster, will generally be negligible
compared to the inevitable imbalance of execution times between the clusters.
Hence our methodology is to use the approach of the previous section to predict
$T_{parallel}(2)$ for the problem of size $N \times (np_2 \times N_2(1))$ assigned to the $np_2$ processes
of WRG2 and $T_{parallel}(3)$ for the problem of size $N \times (np_3 \times N_3(1))$ on the $np_3$
processes of WRG3. We then take the simple estimate

$$T_{parallel} = \max (T_{parallel}(2), T_{parallel}(3)).$$

Fig. 5. Example partitions by rows of a fine square mesh across two clusters: WRG2
and WRG3.
5.3 Results

We have tested our models for a wide range of problems with five different cluster architectures and present a selection of typical results in Tables 1 and 2 below. These tables are focused around potential applications of the predictions within a Grid environment (we refer to these as scenarios), however the key observation that we make here is the consistent accuracy of the predictions when compared to the actual run times that have subsequently been computed.

Scenario 1

In this scenario, it is assumed that a problem of a particular size must be solved and that two clusters are scheduled to be partially available, with \( np_2 \) and \( np_3 \) processors free on WRG2 and WRG3, respectively. Specifically, we consider the case \( np_2 = 64, np_3 = 32 \) for a target problem size of \( N \times N \) with \( N = 65536 \). The memory requirement across some different combinations of processors is shown in the fourth row of Table 1. The columns entitled “\( np_2=64 \)” show two sets of predicted and actual results using 64 processors on WRG2: based upon Ethernet and Myrinet switching respectively. The columns entitled “\( np_3=32 \)” are empty, reflecting the fact that insufficient memory is available to execute a job of this size on 32 cores of WRG3 alone. The final column shows predicted and actual results when the job is split equally between the two clusters (using 64 and 32 processes on WRG2 and WRG3 respectively). In all cases, the model is demonstrated to provide excellent predictions to the actual execution time.

<table>
<thead>
<tr>
<th>proc</th>
<th>np2=64</th>
<th>np2=64</th>
<th>np3=32</th>
<th>np3=32</th>
<th>(np2,np3)=(64,32)</th>
</tr>
</thead>
<tbody>
<tr>
<td>switch</td>
<td>Ethernet</td>
<td>Myrinet</td>
<td>Ethernet</td>
<td>Myrinet</td>
<td>Ethernet</td>
</tr>
<tr>
<td>size</td>
<td>65536^2</td>
<td>65536^2</td>
<td>65536^2</td>
<td>65536^2</td>
<td>65536^2</td>
</tr>
<tr>
<td>mem/proc</td>
<td>2GB</td>
<td>2GB</td>
<td>4GB</td>
<td>4GB</td>
<td>(1GB, 2GB)</td>
</tr>
<tr>
<td>prediction</td>
<td>1715.7</td>
<td>983.9</td>
<td>-</td>
<td>-</td>
<td>max(997.1, 1044.4)</td>
</tr>
<tr>
<td>measurement</td>
<td>1703.9</td>
<td>1014.9</td>
<td>-</td>
<td>-</td>
<td>1104.6</td>
</tr>
<tr>
<td></td>
<td>error</td>
<td></td>
<td></td>
<td></td>
<td>5.45%</td>
</tr>
</tbody>
</table>

The purpose of this scenario is to illustrate a situation in which the user wishes to decide which of a number of combinations of available resources will deliver the required answer in the shortest time. Here the user is able to determine whether it will be better to use 64 processors of WRG2 alone or a combination of these processors along with the 32 available cores of WRG3. In this particular case, if only Ethernet is available on WRG2 then the latter approach is faster whereas the former would be better if Myrinet is available on WRG2. Assuming that pricing information is available to the user (based upon
a different rate per cpu hour on each cluster) it is also possible to predict the financial cost of each option in advance.

Other combinations of processors and job partition may be assessed in the same manner according to what resources are scheduled to be available at any given time. For example if there are an equal number of processors available on WRG2 and WRG3 then it is likely to be desirable to give the faster cluster more than half of the computational domain to work with.

**Scenario 2**

In the second scenario that we present, a user wishes to consider solving a problem with different levels of mesh resolution. That is, given two Grid resources that are simultaneously available, they can either choose to solve on the larger of these two resources or else they can make use of both resources together in order to solve a problem with even more unknowns (using the memory of both resources together). In the latter case it will clearly be possible to get more resolution but the user may wish to know how much extra this will cost, and will therefore need a reliable estimate of the solution time for each alternative.

Table 2 shows five different predictions, along with the corresponding measured runs times, for different cluster configurations. It is assumed that up to 32 processes may be used on either of WRG2 or WRG3, or on each of them together. In the single-cluster cases the largest problem that may be solved for which $N$ is a power of 2 is $32768 \times 32768$, which corresponds to approximately 1Gb of memory per processor. By combining the two clusters however it is possible to obtain a solution with $N = 65536$. As for the previous table of results, it is again clear that the predictions obtained using the methodology described in this paper prove to be remarkably robust given their simplicity.

<table>
<thead>
<tr>
<th>switch</th>
<th>mem/proc</th>
<th>prediction</th>
<th>measurement</th>
<th>error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethernet</td>
<td>1GB</td>
<td>739.2</td>
<td>776.7</td>
<td>4.83%</td>
</tr>
<tr>
<td>Myrinet</td>
<td>32768$^2$</td>
<td>628.6</td>
<td>628.3</td>
<td>0.05%</td>
</tr>
<tr>
<td>Ethernet</td>
<td>1GB</td>
<td>451.9</td>
<td>444.4</td>
<td>1.69%</td>
</tr>
<tr>
<td>Myrinet</td>
<td>32768$^2$</td>
<td>259.5</td>
<td>281.0</td>
<td>7.65%</td>
</tr>
<tr>
<td>Ethernet</td>
<td>1GB</td>
<td>max(1686.0, 1044.4)</td>
<td>1645.5</td>
<td>2.46%</td>
</tr>
</tbody>
</table>

The significance of this scenario is that, in a Grid computing environment, our predictions provide users with the tools required to make an informed decision as to what resources they wish to request. By combining resources from two different clusters it is possible to solve a problem with greater mesh resolution however the cost of doing so may be substantial. In this specific case if, for
example, WRG2 is charged at 1 unit per cpu hour and WRG3 is charged at 2 units per cpu hour, then the financial cost (both predicted and actual) of obtaining the greater resolution by using both clusters would be approximately 10 times the cost of the $32768 \times 32768$ resolution run using WG3 with Myrinet.

6 Discussion

In this paper we have proposed a simple methodology for predicting the performance of parallel numerical codes within a multi-cluster environment based upon their characteristics when executed on small numbers of processors. The methodology has been demonstrated to be robust and accurate in five combinations of parallel architectures for a multigrid code and its successful application in two different Grid scenarios has also been shown. Furthermore, the approach is applicable to a general class of parallel numerical software and to very general cluster configurations, including multicore and inhomogeneous architectures. Although not exploited here, we note that using additional information about the numerical codes, such as knowledge of their performance as a function of the problem size (work), we can also predict the computational time $T_{\text{comp}}$ based on its behaviour on smaller (computationally) problems. In this way, we are able to reduce the required access to the machines during the prediction process.

The next stage of this work will be to attempt to predict the computational performance of more complex numerical software, such as [4]. An additional factor that will be included in this modeling process is the ability to consider different domain decomposition strategies (e.g. partitioning the data into blocks rather than strips) and predict their relative performance for a given problem on a given architecture.

Acknowledgements

We are very grateful to Drs. Chris Goodyer and Jason Wood for their valuable input to this research which is supported by EPSRC grant EP/C010027/1.

References