Proceedings of the International Multiconference on
Computer Science and Information Technology pp. 467–474

ISBN 978-83-60810-14-9
ISSN 1896-7094

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

Giuseppe Romanazzi, Peter K. Jimack
School of Computing
University of Leeds
LS2 9JT Leeds, United Kingdom
Email: {roman,pkj}@comp.leeds.ac.uk

***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, and on a 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 methodology is built and tested for a particular class of numerical code, based upon the multilevel 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. Furthermore, results are also presented which demonstrate that excellent predictions may also be obtained for numerical algorithms that are more general than the pure multigrid solver used to motivate the methodology. These are based upon the use of a practical parallel engineering code that is briefly described. 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.**

***Index Terms*—Parallel Distributed Algorithms; Grid Computing; Cluster Computing; Performance Evaluation and Prediction; Meta-Scheduling.**

I. 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 robust methodology for predicting the performance of parallel numerical multilevel software across different clusters (in terms of both processor and communications architectures) and across combinations of these clusters. The long term goal of this research is to model numerical software that requires a large computational cost, in a simple and cheap way using only few parallel runs across few processors.

Multilevel software (such as multigrid) has been selected for this work due to its growing importance in practical high performance computing software: as the maturity of multilevel algorithms continues to develop, it is able to provide excellent efficiency for very wide classes of problem [1], [2], [3], [4].

The methodology is first described and its predictive capability is then assessed for five different cluster configurations, using a typical parallel multigrid code. It is of course desirable that the predictive methodology proposed should be appropriate to the widest possible classes of numerical algorithms and the paper concludes with a discussion of these issues along with an illustrative example.

II. Related Work

In previous work [5] we have begun to consider the use of simple (and cheap to implement) predictive models for the solution of certain classes of parallel multigrid codes when executed on distributed memory hardware. Whilst the results obtained in [5] are very encouraging, in this work we develop the ideas further in a number of significant ways.

1) A more general model for inter-processor communication is used which enables less-scalable communications patterns to be captured than previously. This is important when there are all-to-all communications at any point in the code and/or when the hardware does not scale well (e.g. Ethernet switching). The additional generality of this work also ensures that both blocking and non-blocking communication patterns can be reliably captured and modelled.
2) We extend our previous work to consider inter-, as well as intra-, cluster communications. Specifically, we now permit a single parallel job to be split across two entirely different clusters and the performance to be reliably predicted in advance.
3) In addition to reporting on the performance of our model as applied to benchmark multigrid codes, we also provide preliminary results which demonstrate that this performance is also achieved when applied to a practical multilevel engineering code [2].

978-83-60810-14-9/08/$25.00 © 2008 IEEE

There is a very substantial body of research into performance modelling [6] 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 [7] detailed models of a particular application are built for a range of target HPC systems, whereas in [8] or [9] an application trace is combined with some benchmarks of the HPC system that is being 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. Moreover, in the former approach, when a different HPC system is used it would generally be necessary to change the model, adding new parameters for example. Instead, in the latter, we need to add or to find new benchmarks when a new code is used. Considering these limits, the choice between the two approaches can depend also on other factors. For example, when it is more important to predict the run-time of a large-scale application on a given set of systems, as opposed to comparing the performance of the systems in general, researchers (like those in the LANL group [7]) prefer to study deeply their application in order to obtain its own analytic model for the available set of HPC systems. On the other hand, when it is more interesting to compare performances of different machines on some real-applications, the latter approach is preferable; in that case different benchmark metrics can be used and convoluted with the application trace file.

Our approach lies between these two extremes. We use relatively simple analytic models (compared to the LogP model [10] for example), that are applicable to a general class of multigrid 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.

As already indicated, our emphasis in this paper is to provide computational science practitioners with the tools to be able to make informed decisions concerning the Grid resources that they request. Indeed, the scenarios that we consider specifically relate to situations in which the Grid users are aware of which resources are immediately available (and can be reserved) or they are able to reserve resources at some future point in time. More generally however exactly the same information regarding the predicted execution time of a code on different resources, and different combinations of resources, is required by a Grid meta-scheduler for it to be able to work effectively. The job of such a scheduler is to evaluate different candidate resource sets and to select the “most suitable” resources for the execution of the application, e.g. [11]. It is with this in mind that our relatively light-weight approach to performance prediction becomes particularly attractive, since it is both simple and cheap to execute automatically.

There is of course a significant body of literature relating to performance models for large Grid environments. An excellent recent example is the research described in [12] which breaks the execution time of a parallel application into two parts, representing computation and communication costs, that are subsequently estimated for the target platform. Unlike our approach [12] is restricted to tasks that run on a single Grid resource, however the situation in which the load on the resource varies dynamically is included. Other researchers have also considered this situation, including the possible use of stochastic information to predict an application’s behaviour when there is contention for resources [13], [14]. In our work we assume that once a set of resources have been allocated they will be held exclusively by the application for the duration of the run or the reserved time slot, whichever is the shorter. Hence we do not consider this issue of contention here.

A variety of other papers on the subject of performance modelling in both dedicated and non-dedicated environments are described in [6] or [12], for example, so we do not repeat such reviews here. However, we finish this introduction by noting that the precise scheduling mechanism that is used for executing jobs on a Grid may have a significant influence on the performance of the prediction models themselves. Throughout this work we are focused on the situation where we are interested solely in the computational resources that are either available and ready to be used immediately, or the resources that may be reserved for use at some specified time in the future. All of the tests that were undertaken for this work were executed without the intervention of a scheduler. Instead, available resources were reserved and then the required jobs were launched.

III. 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.), see for reference [15], [16].

Using parallel resources we are able to solve problems on finer grids than would be otherwise possible, so as to achieve greater accuracy. When the work per processor is kept constant, a parallel numerical software is considered efficient if there is only a slow increase in the execution time as the number of processors used grows. 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. In some of the runs that follow we use more than one core per physical processor and for other runs we use a parallel architecture with a single core per physical processor. In each case we use the generic term processor to refer to each core or processor respectively. As suggested above, we restrict our attention to mesh-based PDE solvers, in this case considering a finite difference code with a series of non-blocking sends and receives in MPI, that solves a model PDE problem over a square two-dimensional domain (of size $N \times N$, say), see the multigrid code $m1$ in [5]. 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, the partitioning of the coarsest mesh ensures that all finer meshes are uniformly partitioned too (see [3], [17] for further details).

The top diagram in Fig. 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. This is typical in parallel numerical software of this type, e.g. [2], [3], [17].

IV. The Predictive Model

The underlying observation upon which our model is based is that when we scale the size of our computational problem with respect to the number of processors used, the parallel overheads observed using just a small number of processors can describe the communication pattern for runs using a much larger number of processors. This occurs when the problem size per processor is kept fixed. In our methodology we therefore use parallel runs across few processors for predicting the performance of the parallel run across a large number of processors (np), with the same work assigned to each processor across all these runs. For convenience, here we define as “work per processor” the memory required by each processor: this is because the work load per processor in a multigrid code is proportional to the problem size assigned and therefore to the associated memory required by each processor.

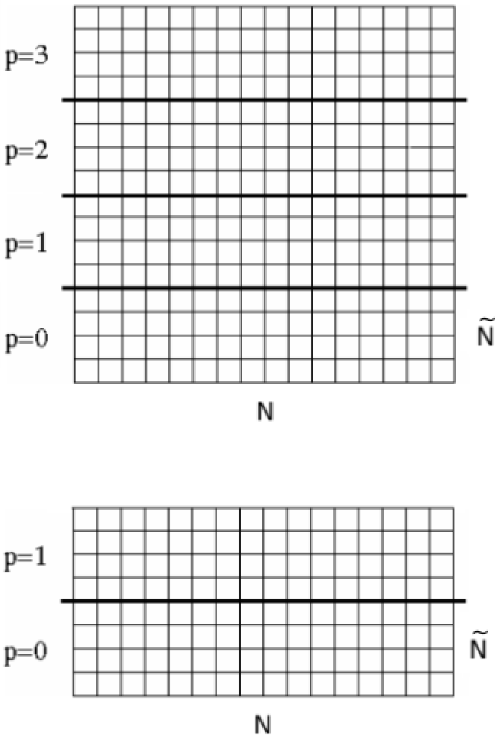

Fig. 1. Partitioning of a square mesh across four processors (top) and the equivalent problem considered on two processors (bottom).

The next basic assumption that we make is that the parallel solution time (on np processors) may be represented as

$$T = T_{comp} + T_{comm}. \tag{1}$$

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

The calculation of T_{comp} is straightforward since this simply requires the execution of a problem of size $N \times \widetilde{N}$ 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_{comp}. This is because the memory access and contention patterns observed in the parallel runs (such as cache and multicore effects at the node-level) vary with respect to the geometrical dimensions of the memory allocated to each processor, and they can consequently influence the computational time measured.

The more challenging task is to model T_{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.

$$T_{comm} = \alpha(np) + \gamma(np) \cdot work. \tag{2}$$

In (2) the term $work$ is used to represent the work on each processor, and is expressed in MBytes of the memory required, which is proportional to the computational cost. 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 65536×65536) the size of the messages is known *a priori* (in this case, since the partition is by rows, the largest 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:

$$\begin{aligned} \alpha(np) &\approx c + d\log_2(np) && (3)\\ \gamma(np) &\approx \text{constant.} && (4) \end{aligned}$$

The justification for this model and the above assumptions are based upon our own empirical evidence gained using different parallel architectures. Two such illustrations are provided in Fig. 2 and Fig. 3. These show plots of overhead against work

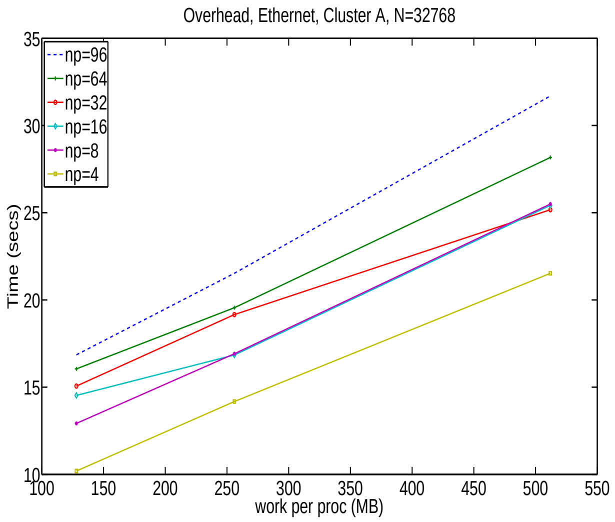

Fig. 2. Overhead (T_{comm}) associated to a fixed size of messages (N) using Fast Ethernet switching

for two different systems: one based upon a Fast Ethernet switching 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 Fig. 1 for constant work with two different choices of np and Fig. 4 for the same np but half the work per processor).

In order to be able to use the model (2) it is necessary to evaluate the parameters c, d and γ. These are determined using measurements taken for $np = 4$ and $np = 8$: $\gamma = \gamma(8)$ whilst c and d are obtained using a simple linear fit through the two data points.

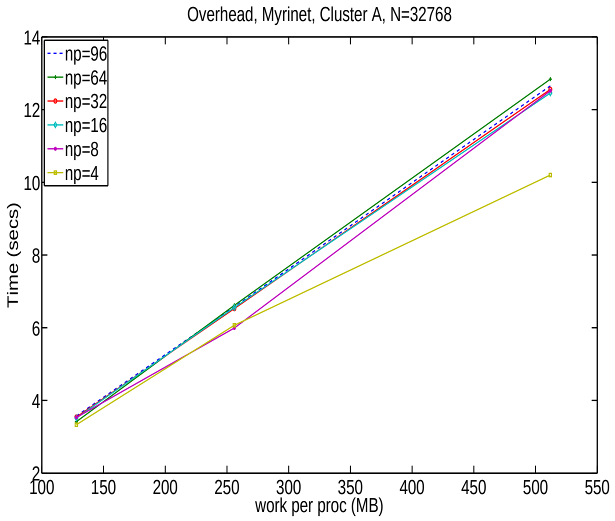

Fig. 3. Overhead (T_{comm}) associated to a fixed size of messages (N) using Myrinet switching.

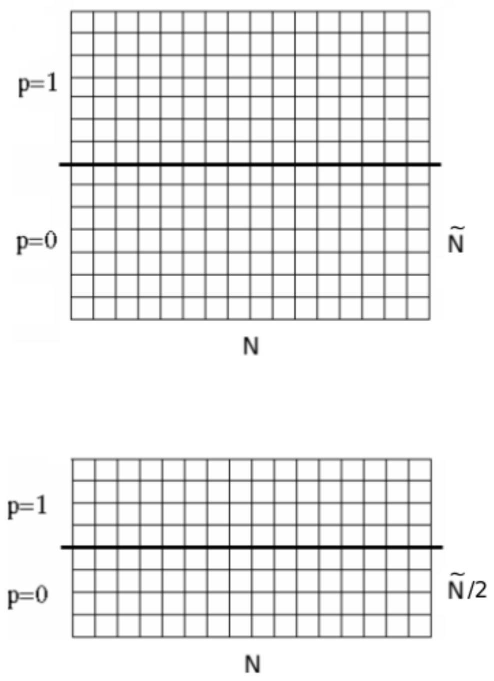

Fig. 4. Scaling the work per processor whilst maintaining the communication volume.

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 $\widetilde{N} = N/np$ and define $N \times \widetilde{N}$ to be the size of problem on each processor in the target configuration.

1) Run the code on a single processor with a fine grid of dimension $N \times \widetilde{N}$ and then with dimension $N \times \frac{\widetilde{N}}{4}$. In each case collect the computational time T_{comp} and define as $work$ the memory allocated in the processor.
2) Run the code on $np0 = 4, 8$ processors, with a fine grid of dimension $N \times (np0 * \widetilde{N})$ and $N \times \left(np0 * \frac{\widetilde{N}}{4}\right)$. In each case collect the parallel time T and then compute $T_{comm} = T - T_{comp}$.
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 the model in Eq. (2) to estimate the value of T_{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_{comm} from step 5 with T_{comp} (determined in step 1, with finest size $N \times \widetilde{N}$) to estimate T as in Eq. (1).

In the parallel runs described in step 2, we use messages at all levels with lengths equal to those used in the parallel run that we are interested to predict. As we show in the next section, this permits us to describe accurately the communication patterns at all mesh levels of the multigrid code.

V. 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.

A. The White Rose Grid

The White Rose Grid is a collaborative project involving the Universities of Leeds, Sheffield and York [18]. In these tests we make use of two clusters on this Grid.

- Cluster A (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.
- Cluster B (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 access to 8GBytes 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 clusters A and B 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. However, this approach alone is not sufficient since specific hardware features must also be accounted for.

For cluster A, 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 B, 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 (four-core) 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 [5] for further details. This strategy permits to reproduce the effects [19] of the memory contention at the node-level in a multi-core architecture.

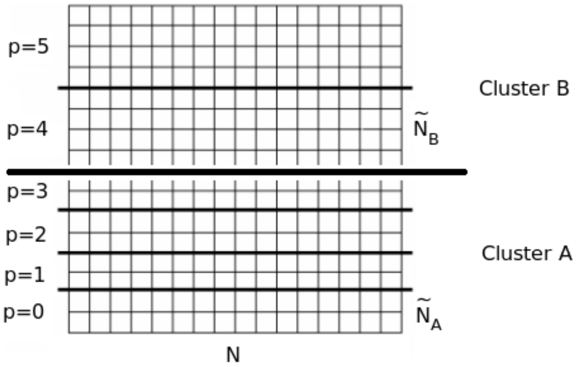

Fig. 5. Example partitions by rows of a fine square mesh across two clusters: A and B.

B. Methodology for Inter-Cluster applications

As mentioned above, it is also possible to run a single job across both clusters using Fast Ethernet switching. Fig. 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_A processors on cluster A (each working with a sub-mesh of size $N \times \widetilde{N}_A$) and np_B processors on cluster B (each working with a sub-mesh of size $N \times \widetilde{N}_B$) 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_A for the problem of size $N \times (np_A * \widetilde{N}_A)$ assigned to the np_A processors of cluster A and T_B for the problem of size $N \times (np_B * \widetilde{N}_B)$ on the np_B processors of cluster B. We then take the simple estimate

$$T = \max(T_A, T_B). \tag{5}$$

C. Results

We have tested our models for a range of problems with five different cluster architectures and present a selection of typical results in Tables I and II below. These tables are focused around two potential applications of the predictions within a Grid environment, which we refer to here as scenarios. However the key observation that wish to 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_A and np_B processors free on clusters A and B, respectively. Specifically, we consider the

TABLE I
MEASUREMENTS AND PREDICTIONS (BOTH QUOTED IN SECONDS) FOR SCENARIO 1.

procs switching size mem. per core/proc	$np_A = 64$ Ethernet 65536^2 2GB	$np_A = 64$ Myrinet 65536^2 2GB	$np_B = 32$ Ethernet 65536^2 4GB	$np_B = 32$ Myrinet 65536^2 4GB	$(np_A, np_B) = (64, 32)$ Ethernet 65536^2 (1GB, 2GB)
measurement	1703.9	1014.9	-	-	1104.6
prediction	1715.7	983.9	-	-	1044.4
$\|\text{error}\|$	0.69%	3.05%	-	-	5.45%

case $np_A = 64$, $np_B = 32$ for a target problem size of $N \times N$ with $N = 65536$, see Table I. The memory requirement across some different combinations of processors is shown in the fourth row. The columns entitled "$np_A = 64$" show two sets of predicted and actual results using 64 processors on cluster A: based upon Ethernet and Myrinet switching respectively. The columns entitled "$np_B = 32$" are empty, reflecting the fact that insufficient memory is available to execute a job of this size on 32 cores of cluster B alone. The final column shows predicted and actual results when the job is split equally between the two clusters (using 64 and 32 cores on clusters A and B respectively). In all cases, the model is demonstrated to provide excellent predictions to the actual measured run times.

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 cluster A alone or a combination of these processors along with the 32 available cores of cluster B. In this particular case, if only Ethernet is available then the latter approach is faster whereas the former would be better if Myrinet is available on cluster A. 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 cluster A and B 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 II shows five different predictions, along with the corresponding measured runs times, for different cluster configurations. It is assumed that up to 32 processes are available on either of cluster A or B, 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 32768×32768, which corresponds to approximately 1GByte 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.

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, cluster A is charged at 1 unit per cpu hour and cluster B 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 32768×32768 resolution run using cluster B with Myrinet.

Note that, as with scenario 1, it is possible to consider other combinations of available processors using this same approach. Unlike scenario 1 however, in this case our focus is on maximizing the amount of memory available rather than minimizing the run time required.

VI. DISCUSSION

In this paper we have proposed a simple methodology for predicting the performance of parallel numerical codes within a multi-cluster environment. The philosophy upon which this methodology is based is to produce a general empirical model that involves a minimum number of parameters, and then to determine appropriate values for these parameters for any given combination of code and hardware resources. These parameter values are determined based upon the characteristics of the code when it is executed on much smaller numbers of processors than are ultimately required. This allows resources that are not currently available to be reserved for future execution based upon the predicted need. Results presented in the previous section demonstrate that the methodology is both robust and accurate across five different combinations of parallel architecture for a given multigrid code. Furthermore,

TABLE II
MEASUREMENTS AND PREDICTIONS (BOTH QUOTED IN SECONDS) FOR SCENARIO 2.

procs switching size mem per core/proc	$np_A = 32$ Ethernet 32768^2 1GB	$np_A = 32$ Myrinet 32768^2 1GB	$np_B = 32$ Ethernet 32768^2 1GB	$np_B = 32$ Myrinet 32768^2 1GB	$(np_A, np_B) = (32, 32)$ Ethernet 65536^2 (2GB, 2GB)
measurement	776.7	628.3	444.4	281.0	1645.5
prediction	739.2	628.6	451.9	259.5	1686.0
$\lvert\text{error}\rvert$	4.83%	0.05%	1.69%	7.65%	2.46%

two different Grid scenarios have been considered, for which the performance prediction is of clear practical value.

Although the results presented in this work have been computed without the aid of any automatic scheduling software, it is clear that the performance prediction capability that has been demonstrated is of great potential value to Grid middleware and meta-schedule developers. When applications are submitted to a Grid, the scheduler needs accurate information regarding the potential performance of those applications on different resource combinations in order to be able to make optimal choices regarding the allocation of jobs to resources. We hope to explore this feature of our work further in future research. In order to be of maximum value however it will be necessary to demonstrate the generality of our approach to other numerical software.

In addition to the standard linear multigrid code that has been used for testing here, the methodology can be shown to extend to other parallel multilevel software too. Examples from our current work include the simulation of the spreading of fluid droplets [3] and the simulation of nonlinear lubrication problems involving fluid-structure interaction [2]. Details of the practical application to these engineering problems on a single cluster form the subject of another publication [20], however sample results are included here as evidence of the generality of our approach. Table III illustrates timings and predictions for the code described in [2], where we use the same methodology as described in this paper, based upon the separate prediction of T_{comp} and T. In this case the code has additional components to the pure multigrid codes used for the rest of this paper and the work no longer scales linearly with memory. Nevertheless, as Table III clearly shows, provided this is taken into account the basic methodology that we propose again provides excellent predictions.

In addition to applying and testing our methodology to practical scientific codes in 2-d, one of the next steps that we wish to undertake is the application in 3-d. When the same linear partition of the problem is used then it is expected that the approach will be equally successful however further developments are required in order to deal with more general partitioning strategies. It is also our intention to assess the quality of the methodology when applied to other numerical schemes than the multilevel finite difference and finite element codes so far investigated. Candidates for a successful application includes other structured approaches such as Lattice-Boltzmann simulations [21].

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. We also thank anonymous referee for their comments concerning Grid meta-scheduling.

REFERENCES

[1] R. E. Bank, and M. J. Holst, “A New Paradigm for Parallel Adaptive Meshing Algorithms,” *SIAM Review* vol. 45, 2003, pp. 292–323.

[2] C. E. Goodyer, and M. Berzins, “Parallelization and scalability issues of a multilevel elastohydrodynamic lubrication solver,” *Concurrency and Computation*, vol. 19, 2007, pp. 369–396.

[3] P. H. Gaskell, P. K. Jimack, Y. Y. Koh, and H. M. Thompson, “Development and application of a parallel multigrid solver for the simulation of spreading droplets,” *Int. J. Num. Meth. Fluids*, vol. 56, 2008, pp. 979–1002.

[4] P. Ladeveze, A. Nouy, and O. Loiseau, “A multiscale computational approach for contact problems”, *Comput. Meth. Appl. Mech. Engrg.*, vol. 191, 2002, pp. 4869-4891.

[5] G. Romanazzi, and P. K. Jimack, “Parallel performance prediction for multigrid codes on distributed memory architectures”, in *High Performance Computing and Communications (HPCC-07)*, ed. R. Perrott et al. (LNCS 4782, Springer), 2007, pp. 647–658.

[6] S. Pllana, I. Brandic and S. Benkner, “A survey of the state of the art in performance modeling and prediction of parallel and distributed computing systems”, *Int. J. Comput. Intel. Res. (IJCIR)*, vol. 4, 2008, pp. 17–26.

[7] D. J. Kerbyson, H. J. Alme, A. Hoisie, F. Petrini, H. J. Wasserman and M. Gittings, “Predictive performance and scalability modeling of a large-scale application”, in *Proceedings of SuperComputing 2001*, 2001.

[8] G. Rodriguez, R. M. Badia, and J. Labarta, “Generation of simple analytical models for message passing”, in *Euro-Par 2004 Parallel Processing*, ed. M. Danelutto et al. (LNCS 3149, Springer), 2004, pp. 183–188.

[9] L. Carrington, M. Laurenzano, A. Snavely, R. Campbell and L. Davis, “How well can simple metrics represent the performance of HPC applications?”, in *Proceedings of SuperComputing 2005*, 2005.

[10] D. E. Culler, R. M. Karp, D. A. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian and T. von Eicken, “LogP: towards a Realistic Model of Parallel Computation”, *SIGPLAN Not.*, vol. 28, 1993, pp. 1–12.

[11] A. Petitet, S. Blackford, J. Dongarra, B. Ellis, G. Fagg, K. Roche and S. Vadhiyar, “Numerical libraries and the grid: The grads experiments with scalapack”, *J. of High Performance Applic. and Supercomputing*, vol. 15, 2001, pp. 359–374.

[12] H. A. Sanjay and S. Vadhiyer, “Performance modeling of parallel applications for grid scheduling”, *J. Parallel Dist. Comput.*, vol. 68, 2008, pp. 1135–1145.

[13] J. Schopf and F. Berman, “Performance prediction in production environments”, in *Proceedings of 12th International Parallel Processing Symposium*, Orlando, USA, 1998.

[14] J. Schopf and F. Berman, “Using stochastic information to predict application behaviour on contended resources”, *Int. J. Found. Comput. Sci.*, 12, 2001, pp. 341364.

[15] J. Blazek, *Computational Fluid Dynamics: Principles and Applications*, Elsevier, 2002.

[16] S. S. Rao, *The Finite Element Method in Engineering*, Butterworth-Heinemann, 2005.

TABLE III
MEASUREMENTS AND PREDICTIONS (BOTH QUOTED IN SECONDS) FOR THE COMPUTATIONAL ENGINEERING APPLICATION SOFTWARE DESCRIBED IN [2], BASED UPON OUR PREDICTIVE METHODOLOGY ASSESSED FOR CLUSTERS A AND B USING MYRINET SWITCHING.

procs cluster A size	$np_A = 64$ 16385×8193	$np_A = 128$ 16385×16385
measurement	1074.86	1260.24
prediction	1051.31	1242.86
$\vert\mathrm{error}\vert$	2.91%	1.38%
procs cluster B size	$np_B = 64$ 16385×8193	$np_B = 128$ 16385×16385
measurement	908.44	1124.19
prediction	904.39	1107.79
$\vert\mathrm{error}\vert$	0.44%	1.45%

[17] S. Lang, and G. Wittum, “Large-scale density-driven flow simulations using parallel unstructured grid adaptation and local multigrid methods”, *Concurrency and Computation: Practice and Experience,* vol. 17, 2005, pp. 1415–1440.

[18] P. M. Dew, J. G. Schmidt, M. Thompson, and P. Morris, “The White Rose Grid: practice and experience,” *in Proceedings of the 2nd UK All Hands e-Science Meeting,* ed. S.J. Cox, EPSRC, 2003.

[19] M. D. McCool, “Scalable programming models for massively multicore processors”, *Proceedings of the IEEE*, vol. 96, 2008, pp. 816-831.

[20] G. Romanazzi, P. K. Jimack, and C. E. Goodyer, “Reliable performance prediction for parallel scientific software in a multi-cluster grid environment”, in *Proceedings of the Sixth International Conference on Engineering Computational Technology,* Civil-Comp Press, 2008, to appear.

[21] A. R. Davies, J. L. Summers, and M. C. T. Wilson, “Simulation of a 3-D lid-driven cavity flow by a parallelised lattice Boltzmann method”, in *Parallel Computational Fluid Dynamics: new Frontiers and Multi-Disciplinary Applications*, 2003, pp. 265-271.