

Reliable Performance Prediction for Parallel Scientific Software in a Multi-Cluster Grid Environment

Giuseppe Romanazzi¹, Peter K. Jimack¹ and Chris E. Goodyer¹

¹School of Computing, University of Leeds, Leeds LS2 9JT, UK

Keywords: Parallel Distributed Algorithms, Cluster Computing, Performance Evaluation and Prediction, Multilevel software.

Abstract

We propose a model for describing and predicting the performance of practical parallel engineering numerical software on a multi-cluster environment with different distributed memory architectures. 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, by only benchmarking the code on small numbers of processors. The model is tested using a practical engineering multilevel code. Despite its simplicity the model demonstrates to be accurate and robust with respect the cluster architectures considered.

Keywords: Parallel Distributed Algorithms, Cluster Computing, Performance Evaluation and Prediction, Multilevel Software.

1 Introduction

As Grid computing becomes available as a practical and viable commodity for computational engineering practitioners the need for reliable performance prediction becomes essential. In particular, when a variety of computational resources are available to an engineering research team they need the ability 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 be able to make such decisions in a reliable way, it is necessary that they are able to predict the performance of their software on different individual resources and across combinations of these resources.

In this paper we present results from our recent research into the modelling of practical engineering application software [5] that exploits state-of-the-art multilevel solvers [6]. The parallelization of this software is based upon standard geometric

domain decomposition techniques and is implemented using the portable communications library MPI. This ensures that the software may be executed in a multi-cluster environment involving both heterogeneous and multicore processor systems, each with different levels of performance, availability and access charges.

The long-term goal of our research is to allow reliable predictions to be made as to the execution time of any given parallel code on a large number of heterogeneous and multicore processors, possibly distributed on different clusters, by benchmarking the code on small numbers of processors. When extra memory and processors are available, parallel multilevel implementations are able to solve problems numerically on finer meshes, so as to achieve greater accuracy than would be otherwise possible. We wish to be able to estimate the performance prior to actually running on these very fine meshes with large numbers of processors.

We show the performance of our predictive model across a range of computational resources: single and multicore, homogeneous and heterogeneous. Prior work has shown this approach to be successful for a number of “model problems” based upon state-of-the-art parallel multigrid solvers, see [11]. The work described in this paper demonstrates the feasibility of our proposed approach when extended to practical multilevel engineering software, rather than the more straightforward model problems previously considered.

2 Related work

2.1 Performance modelling

This work builds upon a very substantial body of research into performance modelling 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 [8] detailed models of a particular application are built for a range of target HPC systems, whereas in [4] and in [9] 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 [3] 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 of the performance of the application on larger numbers of processors may then be made. This idea has already been shown to work well for simple multigrid codes running on different parallel architectures and in a multi-cluster environment, see [11, 12].

2.2 Background to the physical problem

Elastohydrodynamic lubrication (EHL) plays an important role in many mechanical devices such as journal bearings or gears where, under very heavy loads, the extreme pressure in the lubricant causes elastic deformation of the contacting elements. This is typically modelled via a thin-film approximation for the lubricant flow, coupled with a film-thickness equation which captures the elastic deformation. With a suitable non-dimensionalization (see, [14] for further details) the following equations are obtained on a two-dimensional domain $(X_{\min}, X_{\max}) \times (Y_{\min}, Y_{\max})$:

$$\frac{\partial}{\partial X} \left(\frac{\rho H^3}{\eta \lambda} \frac{\partial P}{\partial X} \right) + \frac{\partial}{\partial Y} \left(\frac{\rho H^3}{\eta \lambda} \frac{\partial P}{\partial Y} \right) - u_s \frac{\partial(\rho H)}{\partial X} = 0 \quad (1)$$

and

$$H(X, Y) = H_{00} + \frac{X^2}{2} + \frac{Y^2}{2} + \frac{2}{\pi} \int_{Y_{\min}}^{Y_{\max}} \int_{X_{\min}}^{X_{\max}} \frac{P(X', Y')}{\sqrt{(X - X')^2 + (Y - Y')^2}} dX' dY' . \quad (2)$$

Here P and H are the unknown pressure and film-thickness respectively, λ and u_s are constants, H_{00} is an unknown offset value which can be determined indirectly through a force balance constraint,

$$\int_{Y_{\min}}^{Y_{\max}} \int_{X_{\min}}^{X_{\max}} P(X, Y) dX dY = \frac{2\pi}{3} , \quad (3)$$

and the density ρ and viscosity η are given by the following empirical relations:

$$\begin{aligned} \rho(P) &= \frac{0.59 \times 10^9 + 1.34 p_h P}{0.59 \times 10^9 + p_h P} \\ \eta(P) &= \exp \left\{ \frac{\alpha p_0}{z_i} \left[-1 + \left(1 + \frac{p_h P}{p_0} \right)^{z_i} \right] \right\} . \end{aligned}$$

The coefficients p_h , p_0 , α and z_i are assumed to be known constants.

The numerical method, that has been implemented in parallel, for solving the above model is described in detail in [5]. The code, which we refer to as *mEHL*, is based upon a finite difference approximation to (1) and a simple quadrature scheme for (2). The efficient solution of the resulting discrete system depends critically upon the use of multilevel methods:

- Parallel nonlinear multigrid is used for the solution of the discrete form of (1).
- Parallel multilevel multi-integration (MLMI) is used to evaluate the discrete form of (2).

The latter scheme is especially important since it allows the cost of evaluating the film thickness over the entire domain, approximated on a finest mesh of size $N^f \times N^f$, to be reduced from $O((N^f)^4)$ to $O((N^f)^2(\log N^f)^2)$. Note however that the

parallel implementation of the MLMI requires each process to work with the entire computational domain at the coarsest mesh level. This is fundamentally different from the case with the parallel multigrid solver, which allows the meshes at each level to be partitioned across the processes. This difference has an important impact on the way in which the parallel performance of the software should be predicted, compared to a pure multigrid solver such as in [11, 12].

Precise models for the computational complexity of the code, as well as the parallel computational and memory complexities, are provided in [5]. These are *not* required for the purposes of this work however, which seeks to base the performance prediction on much simpler empirical models. Indeed, the following section summarizes in equations (5)-(7) all of the *a priori* knowledge that is required about the complexity of the *mEHL* software. Consequently, no further details are provided here.

3 The Predictive Model

The ultimate goal of our model is to allow reliable predictions to be made as to the execution time of any given code running on parallel clusters within a Grid environment. Initially however we focus on the important, and growing class of software that uses parallel multigrid methods on large numbers of heterogeneous and/or multicore processors. Our philosophy is to benchmark the code on small numbers of processors in order to aim to predict its performance on larger numbers. Hence, when sufficient numbers of processor are available, it is possible to solve problems numerically on finer grids, so as to achieve greater accuracy than would be otherwise possible. We would like to be able to estimate the performance prior to actually running on these very fine meshes, so as to predict the resource implications of so doing.

In our model we represent the parallel execution time as consisting of two components:

$$T_{parallel} = T_{comp} + T_{comm}, \quad (4)$$

where T_{comp} is the computational time and T_{comm} is the parallel overhead, primarily due to inter-processor communications. Our method is applied to parallel codes that equally distribute their (two-dimensional) domain across np processors using a partition by rows, as in [11]. Let $N_x^f \times N_y^f$ be the size of the finest grid used in the problem then the size of the problem associated to each processors is $N_x^f \times N_y^f(1)$ where $N_y^f(1) = N_y^f/np$, see Figure 1.

3.1 Predicting the computational time T_{comp}

In full multigrid codes, since the computational time scales linearly with the size of the problem, T_{comp} can be easily determined through runs of the code on a single processor, see [10]. It should be noted however that, in order to obtain accurate predictions of T_{comp} , care needs to be taken to ensure that the geometric shape of the subdomains for the parallel runs are respected. This permits the model to describe accurately the

caching, and other memory access patterns, see [11]. In this work we represent T_{comp} as a sum of two components

$$T_{comp} = T_{mgrid} + T_{nmgrid},$$

where T_{mgrid} is the computational time associated with the multigrid operations and T_{nmgrid} is the rest of the computational cost of the code. Consequently, assuming the parallel multigrid is run on np processors with a partition by rows, T_{mgrid} can be measured through a run on a single processor of the code with size of the problem equal to $N_x^f \times N_y^f(1)$. The strategy for predicting T_{nmgrid} depends however on the particular code considered. Information known about the computational code behaviour can help to obtain a predictive strategy for T_{nmgrid} . This constitutes the fundamental contribution of this work as a generalization of the multigrid methodology described in [?, 12], which only considers the case $T_{nmgrid} \approx 0$.

3.1.1 A computational analysis of $mEHL$

The multilevel multi-integration (MLMI) software for EHL described in [5], has two fundamental computational parts: a set of multigrid V-cycles that controls the general convergence of the code and a series of MLMI cycles associated with the stages of (pre- and post-)smoothing of the multigrid V-cycle. The computational time spent in a V-cycle is therefore the sum of two terms: the multigrid cost T_{mgrid} , that is proportional to the (finest) size $N_x^f \times N_y^f(1)$ of the problem assigned to each processor and the remaining computational cost T_{nmgrid} associated to the multilevel multi-integration. We remark that we define as $N_x^c \times N_y^c$, $N_x^f \times N_y^f$, and np the coarsest mesh, the finest mesh of the target problem size, and the target number of processors, respectively (i.e. we wish to predict the code's performance for these values).

As described in [5], the cost T_{nmgrid} (associated to each processor) has a multi-summation term that is quadratic with respect to the size of the overall coarsest mesh, $N_x^c \times N_y^c$, to leading order in powers of the dimensions of the problem N_x and N_y . Therefore we have

$$T_{nmgrid} \propto \frac{(N_x^c N_y^c)^2}{np}, \quad (5)$$

$$T_{mgrid} \propto \sum_{k=c+1}^f \frac{N_x^k N_y^k}{np}, \quad (6)$$

$$T_{comp} \propto \gamma_1 \frac{(N_x^c N_y^c)^2}{np} + \gamma_2 \sum_{k=c+1}^f \frac{N_x^k N_y^k}{np}, \quad (7)$$

where c and f are the coarse and fine grid levels respectively in the multilevel hierarchy. See Figure 1 for an illustration of this notation.

A parallel run of $mEHL$ with nVC V-cycles performs the multi-summation over the coarsest mesh on each processor

$$nsum = nVC * [ncoarse + (ngrid - 1) * (npre + npost + 2)] \quad (8)$$

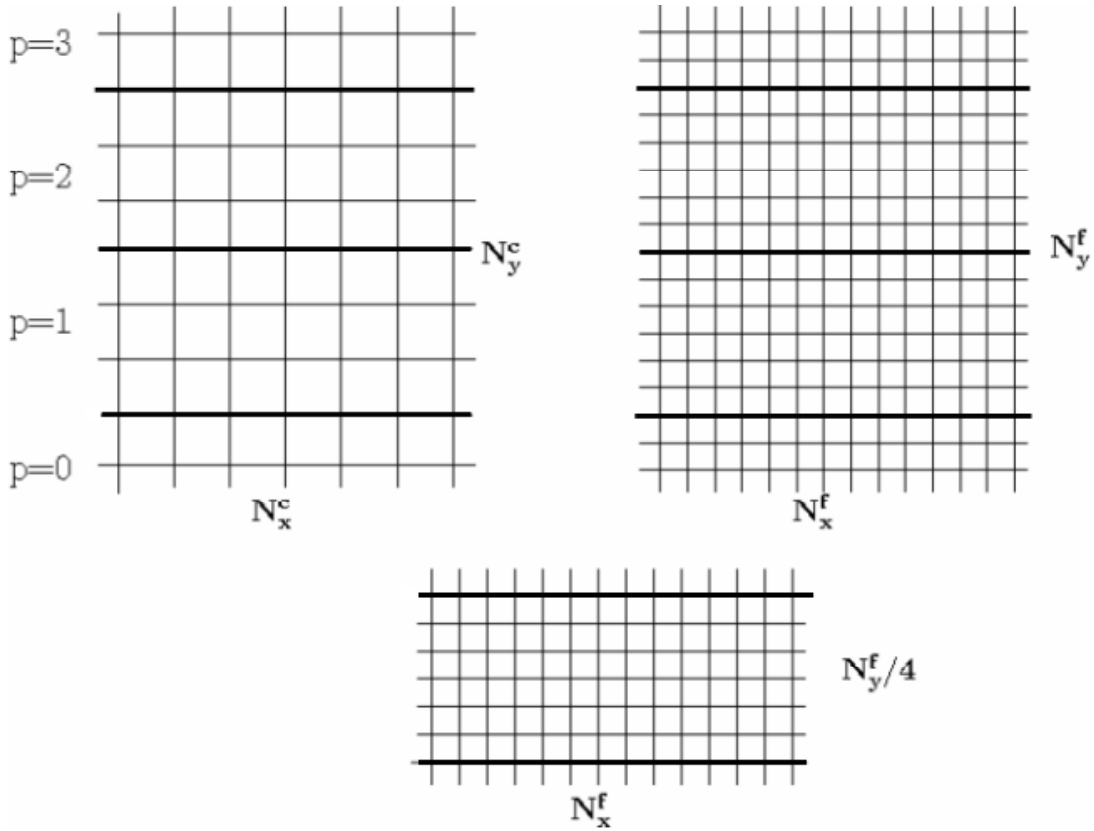


Figure 1: Coarse and fine mesh of a partitioning across four processors. The picture shown at the bottom represents the size problem used in the serial runs for determining T_{mgrid} .

times. In (8) $ngrids(= \log_2(N_x^f/N_x^c) + 1)$ is the number of grids used; $ncoarse$ is the number of smoothing sweeps at the coarsest level, $npre$ is the number of pre-smoothing sweeps and $npost$ is the number of post-smoothing sweeps in a single V-cycle. In order to predict the multi-summation effect in T_{comp} (this is the quadratic term of T_{nmgrid} in (5)), we need to exploit its dependence with respect to both $nsum$ in (8) and T_{comp} in (7). First, we observe that the same multi-summation work can be obtained across a sequence of serial runs of the code with a coarsest mesh $N_x^c \times N_y^c$ and different finest levels $2^l N_x^c \times 2^l N_y^c$ (for $l = 1, 2, \dots$), so long as we keep $nsum$ constant through all these runs. The associated execution times obtained are denoted as T_l in the following part of this section. The parameter $nsum$ is kept constant by appropriate variation of the parameter $ncoarse$, according to the equation (8). In fact the possibility of changing this parameter in (8) permits us to obtain the same $nsum$ (equal to the value used in the target problem that we wish to predict) through all the sequential runs (associated to $l = 1, 2, \dots$) where a different number of grids ($ngrids$) is used. According to equation (7), we can then obtain a straight line through all the points (l, T_l) . Therefore, we can obtain T_{nmgrid} (as expressed in (5)) using the value

extrapolated in $l = 0$ of the line plotted through the points (l, T_l) and dividing it by np .

The methodology for obtaining T_{nmgrid} can be therefore described through the following steps:

1. run the code on a single processor with the coarsest mesh $N_x^c \times N_y^c$ and finest mesh $2^l N_x^c \times 2^l N_y^c$ for $l = 1, 2, 3$ In each case collect the execution time T_l obtained;
2. determine the least square fitting line through the points (l, T_l) for $l = 1, 2, 3$
3. extrapolate the least square fitting line obtained in step 2 to $l = 0$;
4. get as prediction for T_{nmgrid} the quotient obtained by dividing the extrapolated value obtained in step 3 by the number of processors np (see Equation (5)).

In order to predict T_{mgrid} we need, as explained before, to run the code on a single processor with a finest mesh of size $N_x^f \times N_y^f(1)$. This is analogous to our earlier work in [12], see also Figure 1. Since now we are only interested to catch the computational cost associated with the multigrid, we do not consider terms due to the multi-summation at the coarsest mesh, These will be therefore removed from the computational cost.

The methodology for determining T_{mgrid} is then described through the following steps:

1. run the code on a single processor with coarsest mesh $N_x^c \times N_y^c(1)$, with $N_y^c(1) = \frac{N_y^c}{np}$ and finest mesh $N_x^f \times N_y^f(1)$, saving the execution time as $T_{nlevels-1}$;
2. run the code on a single processor with the same coarsest mesh as in step 1 and with the finest mesh equal to $2^l N_x^c \times 2^l N_y^c$ for $l = 1, 2, 3$;
3. determine the least square fitting line through the points (l, T_l) for $l = 1, 2, 3$;
4. extrapolate the least square fitting line obtained in step 2 to $l = 0$, obtaining the value T_0 ;
5. get as prediction for T_{mgrid} the difference between $T_{nlevels-1}$ and T_0 , (the former represents T_{mgrid} plus some MLMI work and T_0 is an estimate of this MLMI work, which must therefore removed).

Finally, the computational time predicted, T_{comp} , is the sum of T_{nmgrid} and T_{mgrid} obtained from the methodology described.

Table 1: Best fit slopes for the overhead patterns observed in Figure 2.

procs	np=2	np=4	np=8	np=16	np=32	np=64	np=128
slope	57.9089	68.2272	82.1748	67.6626	69.9855	81.6459	62.4410

3.2 Predicting the overhead time T_{comm}

The goal of this section is to develop a *simple* model that captures the main features of the parallel overheads with just a small number of parameters that may be computed based upon runs using only a few processors. The model presented here is slightly less simple than that described in [12], because the code *mEHL* includes both point-to-point and global communications, as opposed to only point-to-point communications in the multigrid code described in [12]. Similar to [12] however, we use the model

$$T_{comm} = \alpha(np) + \gamma(np) \cdot work. \quad (9)$$

In (9) the term *work* is used to represent the problem size on each processor at the finest level, it can be 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 16385×16385) the size of the messages is known *a priori* (in this case, since the partition is by rows, the messages on the finest mesh will be of length 16385). This is the primary reason that the expression (9) can be so simple.

Furthermore, we will assume that the following relations are valid:

$$\alpha(np) \approx c + d \log_2(np) + e \log_2(np)^2, \quad (10)$$

$$\gamma(np) \approx constant. \quad (11)$$

The prediction of the parallel overheads requires the inclusion of a quadratic term in the model of $\alpha(np)$. This is the main difference with respect to the multigrid overhead model described in [12], where a linear model is sufficient to capture the overheads due to local communication patterns. The justifications for this model and the above assumptions are based upon a substantial body of empirical evidence. An illustration of this is provided in Figure 2. This figure shows plots of overhead against work at the finest level for different numbers of processors. In almost each case we observe a linear growth in overhead with work, with an almost constant slope (i.e. the slope is approximately independent of np). We also note that in the case when the growth of the overhead is not perfect linear (as for $np = 64, 128$) the least square fitting line for the overhead pattern plotted has an almost equal linear slope of that observed when fewer processors are used, see Table 1.

We also observe that the overhead increases non-linearly as the number of processors used (np) is doubled, (hence the need for the quadratic term in (10)). Note that the length of the messages is the same in all of these runs.

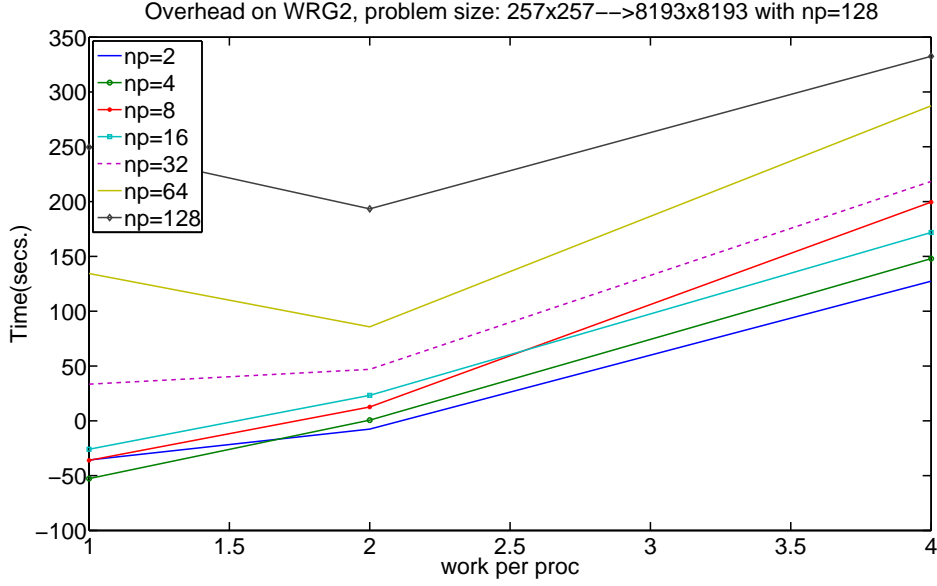


Figure 2: Overhead time (T_{comm}) on the heterogeneous parallel system WRG2 with a fixed size of messages ($N_x = 257, \dots, 16385$). The overhead patterns plotted are obtained using $T_{para} - T_{comp}$, where T_{para} is the effective measure of parallel time and T_{comp} is obtained using the methodology described in Section 3.1.1

The predictive methodology for T_{comm} that follows (and that is similar to that described in [12]) is based on a series of runs on a limited number of processors $np = 1, 2, 4, 8$.

1. For $\ell = 1$ to 3
Using the methodology described in the previous paragraph, determine T_{comp} for a parallel run across np processors with a fine grid dimension $N_x^f \times (2^{1-\ell} N_y^f)$, and define $work \propto N_x^f (2^{1-\ell} N_y^f (1))$.
2. For $\ell = 1$ to 3
Run the code on $np0 = 2, 4, 8$ processors, with a fine grid of dimension $N_x^f \times np0(2^{1-\ell} N_y^f (1))$.
3. Fit a straight line as in Eq. (9) through the data collected in steps 1 and 2 to estimate $\alpha(np0)$ and $\gamma(np0)$, for the three cases $np0 = 2, 4$ and 8.
4. Fit a parabola as in Eq. (10) through the points $(1, \alpha(2))$ $(2, \alpha(4))$ and $(3, \alpha(8))$ to estimate c , d and e : based upon Eq.(10) with these coefficient values now compute $\alpha(np)$ for the required choice of np .
5. Use the model in Eq. (9) 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, for $\ell = 1$) to estimate $T_{parallel}$ as in Eq. (4).

4 Numerical Results

We have tested our model for two different cluster architectures (referred to here as WRG2 and WRG3), which are both available on the White Rose Grid [15].

4.1 Implementation of the methodology across the White Rose Grid

We make use of two clusters on this grid, with the following characteristics:

- 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. Myrinet switching is 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 access to 8GBytes of physical memory. Again, Myrinet switching is used.

Users of WRG2 and WRG3 do not get exclusive access to their resources and hence some variations in the execution time of the same parallel job can be observed across different runs. These variations can be limited when the methodology take into account the specific hardware features of these clusters, see discussion in [11, 12]. We have also to consider the situation that will exist for a large parallel run. In WRG2, for example, since a large parallel run will inevitably involve some of the slower processors (with 2.2GHz instead of 2.4GHz) we need to run all the runs for determining both T_{comp} and T_{comm} using at least one slower processor. Similarly, for the multicore cluster WRG3, a large parallel run will use all four cores for each node. As a consequence, in the methodology for predicting T_{comm} in Section 3.2 we need to use $np0 = 4, 8, 16$ in WRG3 with all cores taken in single nodes (1, 2 or 4 respectively) as opposed to $np0 = 2, 4, 8$ used in WRG2.

4.2 Discussion and results

In this section we present a selection of results of the prediction model applied to the code *mEHL* using the two clusters WRG2 and WRG3. We have tested the model for the two following problems

1. $np = 64, N_x^c \times N_y^c = 257 \times 128,$
 $N_x^f \times N_y^c = 16385 \times 8193,$
 $nVc = 9, npre = 3, npost = 1,$ Maximum memory used per proc *756MB*;

2. $np = 128$, $N_x^c \times N_y^c = 257 \times 257$, $N_x^f \times N_y^c = 16385 \times 16385$, $nVc = 9$
 $npre = 3$, $npost = 1$, Maximum memory used per proc $763MB$.

The problem size chosen in these tests represents a typical scenario in which a user wishes to use as many processors as possible in order to numerically solve a problem with the highest possible level of mesh resolution. In fact the problem size proposed for these tests is that associated with the largest problem that can be solved on the given number of processors (64 or 128).

In Tables 2 and 3 are shown the measurements, the predictions and the errors obtained using the methodology for the clusters WRG2 and WRG3 respectively. We observe that in both cases the methodology shows high accuracy for the prediction of the code’s performance. The error measured is less than 5% in all the combinations examined. However, it may be observed that low accuracy is achieved when the parallel overheads become predominant with respect to the computational time. This can be seen with both clusters WRG2 and WRG3, as shown in Figures 3 and 4. In these figures, the plots of the execution time running on 128 processors are shown with respect to the memory usage at the finest level per processor (work on finest grid). These plots provide evidence that the methodology is able to describe the irregular patterns of the performance of the code as the work per processor increases, but very accurate predictions are obtained only when the work per processor is sufficiently high. This feature is almost certainly due to the model for T_{comp} being more reliable than our simple model for T_{comm} so that when the latter dominates the accuracy deteriorates. Fortunately, for efficient parallel applications T_{comp} is dominant and so this is unlikely to be a problem in practice.

Table 2: Predictions and measurements (both quoted in seconds) for WRG2.

procs size	np=64 16385×8193	np=128 16385×16385
prediction	1051.31	1242.86
measurement	1074.86	1260.24
error	2.91%	1.38%

Total time predicted for $np = 64, 128$ over WRG3 are represented in the Table 3

5 Conclusions

In this paper we have proposed a simple methodology, extending that presented in [12], for predicting the performance of a complex parallel numerical multilevel code that combines both multigrid and multilevel multi-integration computations. This methodology has been demonstrated to be robust and accurate across different parallel architectures, including multicore and inhomogeneous architectures.

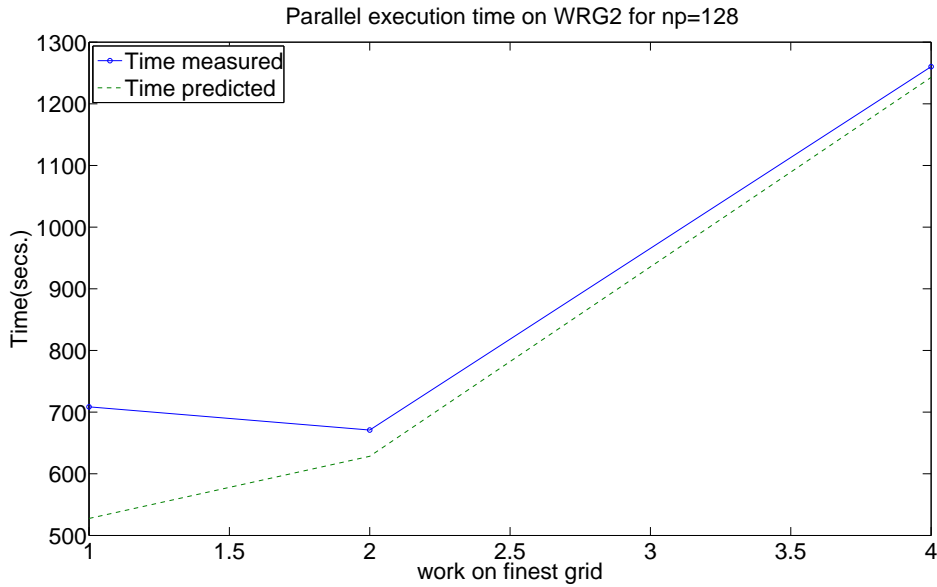


Figure 3: Prediction on WRG2 with a fixed size of messages with $N_x^f = 16385$. The three points plotted with work=1,2 and 4 are associated to the work on the finest mesh $N_y^f = 4097, 8913$ and 16385 respectively.

Table 3: Predictions and measurements (both quoted in seconds) for WRG2.

procs size	np=64 16385×8193	np=128 16385×16385
prediction	904.39	1107.79
measurement	908.44	1124.19
error	0.44%	1.45%

The next stage of this work is to model the overhead patterns when different domain decomposition strategies are used (e.g. partitioning the data into blocks rather than strips), and to consider running the software across multiple Grid resources as part of a single computation.

Acknowledgements

This work is supported by EPSRC grant EP/C010027/1.

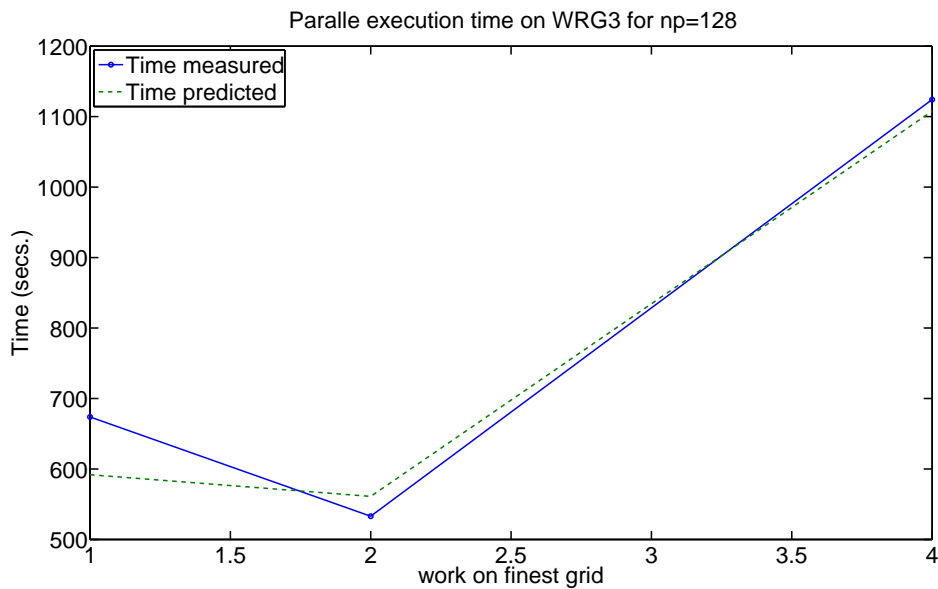


Figure 4: Prediction on WRG3 with a fixed size of messages with $N_x^f = 16385$. The three points plotted with work=1,2 and 4 are associated to the finest mesh (N_y^f) equal to 4097, 8913 and 16385 respectively.

References

- [1] A. Brandt, “Multi-level Adaptive Solutions to Boundary Value Problems”, *Mathematics of Computation*, 31, 333–390, 1977.
- [2] W.L. Briggs, V.E. Henson, and S.F. McCormick, “A Multigrid Tutorial”, SIAM (2000).
- [3] D.E. Culler, R.M. Karp, D.A. Patterson, A. Sahay, K.E. Schauer, E. Santos, R. Subramonian, and T. von Eicken, “LogP: towards a realistic Model of Parallel Computation”, *SIGPLAN Not.*, 28, 7, 1–12, 1993.
- [4] L. Carrington, M. Laurenzano, A. Snively, R. Campbell, and L.P. Davis, “How well can Simple Metrics represent the Performance of HPC Applications?”, in “Proceedings of SC2005”, 2005.
- [5] C.E. Goodyer and M. Berzins, Parallelization and Scalability issues of a Multi-level Elastohydrodynamic Lubrication Solver, “Concurrency and Computation: Practice and Experience”, 19, 369–396, 2007.
- [6] C.E. Goodyer, M. Berzins, P.K. Jimack, L.E. Scales, A Grid-Enabled Problem Solving Environment for Parallel Computational Engineering Design. *Advances in Engineering Software*, vol.37, 439–449, 2006.

- [7] 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. Numer. Meth. Fluids*, 56, 979–1002, 2008.
- [8] 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 SC2001”, 2001.
- [9] G. Rodriguez, R.M. Badia, and J. Labarta, “Generation of Simple Analytical Models for Message Passing”, in “Proceedings of Euro-Par 2004”, (Editor), M. Danelutto et al. (LNCS 3149, Springer), 183–188, 2004.
- [10] G. Romanazzi and P.K. Jimack, “Performance Prediction for Parallel Numerical Software on the White Rose Grid”, in “Proceedings of UK e-Science All Hands Meeting”, ed. S.J. Cox (ISBN 978-0-9553988-3-4), 517–524, 2007.
- [11] G. Romanazzi and P.K. Jimack, “Parallel performance prediction for multigrid codes on distributed memory architectures” in “High Performance Computing and Communications (HPCC-07)”, (Editor), R. Perrott et al. *Lecture Notes in Computer Science* 4782, Springer, 647–658, 2007.
- [12] G. Romanazzi and P.K. Jimack, “Parallel performance prediction for Numerical codes in a Multi-Cluster Environment”, Preprint of School of Computing, University of Leeds, 2008.
- [13] U. Trottenberg, C.W. Oosterlee, and A. Schüller, “Multigrid”, Academic Press (2003).
- [14] C.H. Venner and A.A. Lubrecht, “Multilevel Methods in Lubrication”, Elsevier, Amsterdam, 2000.
- [15] 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”, (Editor), S.J. Cox, EPSRC, 2003.