Performance Prediction for Parallel Numerical Software on the White Rose Grid

Giuseppe Romanazzi and Peter K. Jimack
School of Computing, University of Leeds, UK
{roman,pkj}@comp.leeds.ac.uk

Abstract

The development of Grid environments over recent years now allows scientists access to a range of shared resources which may be located across different sites. These include computational resources such as large parallel computers which may be used to produce accurate numerical simulations in a relatively short time. Since different computer architectures have different performance characteristics, different levels of availability and different access charges, to be able to make optimal decisions concerning their selection and utilization for a given scientific application, we need to be able to predict the application's performance on each available parallel environment. In this research we seek to model the performance of typical state-of-the-art numerical software, based upon multigrid techniques, on different distributed memory parallel machines for varying numbers of available CPUs. The goal of the work is to use these models to support decisions as to which resources should be requested or reserved: including the possibility of solving a single problem across multiple resources when they are available.

1 Introduction

Computational Grids should enable the effective utilization of geographically distributed resources by distributed teams of researchers in a transparent and seamless manner. Their potential has already been successfully demonstrated across a very wide range of scientific applications involving distributed data analysis, remote visualization and large-scale computation. It is the latter aspect of Grid computing and e-Science that is addressed by this work. In particular, we investigate the problem of how to access the wide variety of heterogeneous compute resources potentially available on a Grid in a manner that maximizes their efficiency of utilization.

Previous work on this issue has, justifiably, viewed this problem as one of resource management, [9], and projects such as Globus [1], Nimrod/G [5] and Gridway [7] have all addressed related scheduling issues with noteworthy success. Our viewpoint is rather different however and is based upon an analysis of the sort of computational problems that Grid users may wish to solve. Our goal is to provide automatic predictive capabilities that will allow optimal decisions to be made concerning the selection of Grid resources for a given computational problem. In order to be able to support such decisions there is a fundamental requirement to be able to predict the performance characteristics of a given piece of application software on an arbitrary resource. This paper describes an attempt to achieve this in the context of a general class of computational algorithm, known as multigrid.

Multigrid is one of the most power numerical techniques to have been developed over the last 30 years [3, 4, 12]. As such, state-of-the-art parallel numerical software is now increasingly incorporating multigrid implementation in a variety of application domains [2, 8, 10]. In the next section we provide a very brief introduction to parallel multigrid algorithms for the solution of elliptic or parabolic partial differential equations (PDEs) in two space dimensions. This is followed by an analysis of the performance of two such codes on an abstract distributed memory architecture. The analysis is then used to build a predictive model for this class of code, that is designed to allow estimates of run times to be obtained for large numbers of processors, based upon observed performance on very small numbers of processors. The paper concludes with a description of some numerical tests to assess the accuracy and robustness of these predictions and a discussion of the outcomes obtained. Further extensions of the work are also suggested.

2 Multigrid and Parallel Implementation

The general principal upon which multigrid is based is that when using many iterative solvers for the systems of algebraic equations that result from the discretization of PDEs, the component of the error that is damped most quickly is the high frequency part [4, 12]. This observation leads to the development of an algorithm which takes a very small number of iterations on the finite difference or finite element grid upon which the solution is sought, and then restricts the residual and equations to a coarse grid, to solve for an estimate of the error on this grid. This error is then interpolated back onto the original grid before a small number of further iterations are taken and the process repeated. When the error equation is itself solved in the same manner, using a still coarser grid, and these corrections are repeated recursively down to a very coarse base grid, the resulting process is known as multigrid.

Further details of this procedure may be found in [3, 4, 12], for example, however it is sufficient to note here that multigrid, and related multilevel techniques, represent the state-of-the-art in computational methods for PDEs and, as such, provide one of the most important potential applications for high performance computing. Any parallel implementation of such an algorithm requires a number of components to be implemented to run concurrently:

- application of the iterative solver at each grid level,
- restriction of the residual to a coarse level,
- exact solution at the coarsest level,
- interpolation of the error to a fine level,
- a convergence test.

There is also a variant of the algorithm (primarily designed with nonlinear problems in mind), known as FAS (full approximation scheme) [12], which requires the solution as well as the residual to be restricted to the coarser grid at each level.

In this work we consider the parallel implementation of two multigrid codes: one is standard and the other uses the FAS approach. In both cases they partition a two-dimensional finite difference grid across a set of parallel processors by assigning blocks of rows to different processors. Note that if the coarsest mesh is partitioned in this manner, then if all finer meshes are uniform refinements of this they are automatically partitioned too: see Figure 1 for an illustration.

It is clear from inspection of the meshes in Fig. 1 that each stage of the parallel multigrid process requires communication between neighbouring processes (iteration, restriction, coarse grid solution, interpolation, convergence test). The precise way in which these are implemented will vary from code to code, however the basic structure of the algorithm will remain the same. For example, it is typical for each processor to be responsible for one block of rows but also to store an additional dummy row (sometimes referred to as a ghost row) of unknowns immediately above and immediately below its own subdomain. These rows are used for storing copies of the top and bottom rows owned by the processors below and above respectively. In this work we consider two different implementations, both based upon MPI [11], referred to as m1 and m2.

2.1 The algorithm m1

This algorithm solves the steady-state equation

$$
\begin{aligned}
&-\nabla^2 u = f \ \text{ in } \Omega,\\
&\Omega = [0,1]\times[0,1],\\
&u_{|\partial\Omega} = 0.
\end{aligned}
$$

The discretization is based upon the standard five point finite difference stencil on each grid. The iterative solver employed is the well-known Red-Black Gauss-Seidel (RBGS) method [8], which is ideally suited to parallel implementation but requires two neighbour-to-neighbour communications for each update sweep through the grid points.

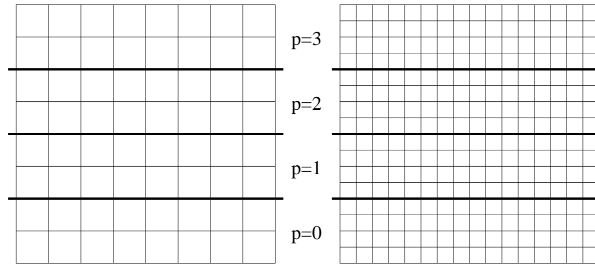

Figure 1: Partitioning of a coarse and a fine mesh across four processors by assigning a block of rows to each processor (bold lines represent the partition boundaries)

2.2 The algorithm m2

This algorithm, described in more detail in [8], uses an unconditionally-stable implicit time-stepping scheme to solve the transient problem

$$\frac{\partial u}{\partial t} = \nabla^2 u + f \text{ in } (0,\mathrm{T}] \times \Omega,$$
$$\Omega = [0,1] \times [0,1],$$
$$u_{|\partial\Omega} = 0,$$
$$u_{|t=0} = u_0.$$

As for m1, the discretization of the Laplacian is based upon the standard five point finite difference stencil. Hence, at each time step it is necessary to solve an algebraic system of equations for which multigrid is used. Again RBGS is selected as the iterative scheme and so communications are required between neighbour processors after each red and black sweep. The algorithm m2 is implemented using the FAS approach [12].

3 Scalability and a Predictive Model

The goal of most parallel numerical implementations is to be able to solve larger problems than would be otherwise possible. For the numerical solution of PDEs this means solving problems on finer grids, so as to be able to achieve higher accuracy. Ideally, when increasing the size of a problem by a factor of np and solving it using np processors (instead of a single processor), the solution time should be unchanged. This would represent a perfect efficiency and is rarely achieved due to the parallel overheads such as inter-processor communications and any computations that are repeated on more than one processor. In this research our aim is to be able to predict these overheads in the situation where the size of the problem (the number of discrete unknowns, N^2, on the finest mesh) scales in proportion to the number of processors. Consequently, the basic assumption that we make is that the parallel solution time (on np processors) may be represented as

$$T_{parallel} = T_{comp} + T_{comm}. \quad (1)$$

In (1), T_{comp} represents the computational time for a problem of size $N^2/np \ \ (= N(1)^2, \text{ say})$ on a single processor, 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 execution of a problem of size $N(1)^2$ on a single processor. One of the major attractions of the multigrid approach is that this time should be proportional to the size of the problem on the single processor. Figure 2 demonstrates that this is indeed the case for both of the implementations, m1 and m2, considered in this paper. Note that in Fig. 2 results are presented for a single processor of two different computers, denoted as WRG2 and WRG3. Further details of these architectures are provided in the next section. It is clear however that for each code, on each processor, T_{comp} is proportional to $N(1)^2$, although the constant of proportionality is of course different in each case. Since our interest is in scaling the problem size with the number of processors, np, then T_{comp} remains independent of np.

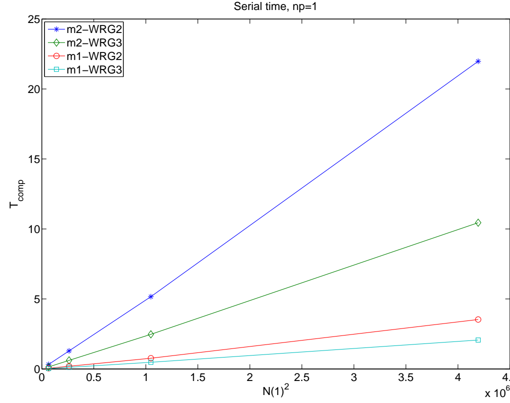

Figure 2: T_{comp}, the serial time for $N(1) = 256, \ldots, 2048$

The more challenging task that we have, therefore, is to model T_{comm} in a manner that will allow predictions to be made for large values of np. Note that any additional work that is undertaken when computing in parallel is associated with the dummy rows that are stored on each processor, as is the communication overhead. When solving on a fine mesh of size N^2 the value of T_{comm} may be estimated, to leading order, as

$$T_{comm} \propto T_{start} + kN. \qquad (2)$$

Here T_{start} relates to the start-up cost for the communications whilst kN represents the number of tasks of size N at each multigrid cycle (which will depend upon the number of communications and the number of additional iterative solves on duplicated rows). Furthermore, it is important to note that k itself may not necessarily be independent of N. In fact, slightly more than np times the work is undertaken by the parallel multigrid algorithm on an N^2 fine mesh than by the sequential multigrid solver on an N^2/np fine mesh. This is because in the former case either there are more mesh levels, or else the coarsest grid must be finer. In all of the examples discussed in this paper we consider each problem using the same coarse level N_0^2 (with N_0 a power of 2) and the same work per processor at the finest level (N^2 on np processors, where $N = \sqrt{np}N(1)$). Therefore, the number of grid levels used, $nlevels$, depends on the number of processors np, that is

$$nlevels = log_2(N/N_0) + 1. \qquad (3)$$

We consider two different models for T_{comm}, based upon (2). In the first of these we do assume that k is approximately constant to obtain

$$T_{comm} = a + bN. \qquad (4)$$

In the second model we add a quadratic term $\gamma N(1)^2$ that allows a nonlinear growth of the overhead time

$$T_{comm} = \alpha + \beta N + \gamma N(1)^2. \qquad (5)$$

The quadratic term introduced in (5) is designed to allow a degree of nonlinearity to the overhead model, reflecting the fact that k in (2) may not necessarily be a constant. As will be demonstrated below, the importance (or otherwise) of this nonlinear effect depends upon the specific characteristics of the processor and communication hardware that are present on each particular parallel architecture.

In order to obtain values for (a, b) and (α, β, γ) in (4) and (5) respectively, the parallel performance of a given code must be assessed on each target architecture. Note from Fig. 1 that the communication pattern is identical regardless of np: requiring only neighbour to neighbour communications. Hence our next assumption is that (a, b) or (α, β, γ) may be determined using just a small number of processors. In this work we choose $np = 4$ in order to approximate (a, b) or (α, β, γ) by fitting (4) or (5) respectively to a plot of $(T_{parallel} - T_{comp})$ against N. Note that, from (1), $T_{comm} = T_{parallel} - T_{comp}$ and T_{comp} is known from the data collected on a single processor.

A summary of the overall predictive methodology is provided by the following steps.

In the following notation np is the target number of processors and N^2 is the largest problem that can be solved on these processors without swapping effects causing performance to be diminished. Similarly, $N(1)^2 = N^2/np$ is the largest such problem that can fit onto a single processor.

1. For $\ell = 1$ to m

   Run the code on a single processor with a fine grid of dimension $(2^{1-\ell}N(1))^2$.

   In each case collect T_{comp} based upon average timings over at least 5 runs.

2. For $\ell = 1$ to m

   Run the code on 4 [or 8] processors, with a fine grid of dimension $(2^{2-\ell}N(1))^2$ [or $(2^{2-\ell}N(1) \times 2^{3-\ell}N(1))$].

   In each case collect $T_{parallel}$ based upon average timings over at least 5 runs.

3. Fit a straight line of the form (4) through the data collected in steps 1 and 2 to estimate a and b, or fit a quadratic curve through the data collected to estimate α, β and γ in (5).

4. Use either model (4) or model (5) to estimate the value of T_{comm} for larger choices of np and combine this with T_{comp} (determined in step 1) to estimate $T_{parallel}$ as in (1).

4 Numerical Results

The approach derived in the previous section is now used to predict the performance of the two multigrid codes, m1 and m2, on the two parallel architectures WRG2 and WRG3. These computers form part of the University of Leeds' contribution to the White Rose Grid [6]

- 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 and Sun Grid Engine Enterprise Edition provides resource management.

- 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 switching and Sun Grid Engine are used for communication and resource management respectively.

Table 1 shows values of (a, b) and (α, β, γ) obtained by following steps 1 to 4 described above, using $N(1) = 2048$ and $m = 4$. Each of the codes is run on each of the selected parallel architectures. Clearly the precise values obtained for (a, b) and (α, β, γ) depend upon a number of factors.

- Because users of WRG2 and WRG3 do not get exclusive access to the machines, or the Myrinet switches, there is always some variation in the solution times obtained in steps 1 and 2.

- On WRG2 there are both 2.4GHz and 2.2GHz processor nodes, hence the parameters will depend on which processors are used to collect execution times in steps 1 and 2.

- On WRG3 the situation is made more complex by the fact that each node consists of two processors and 4 cores. Users may therefore have access to a core on a node where other users are running jobs or else they may have an entire node to themselves. This creates significant variations in the timings for both sequential and small parallel runs.

As outlined in steps 1 and 2 above, a simple way to reduce the effects of these variations is simply to take average timings over five runs (say), however such a crude approach may not always be sufficient. Hence, for WRG2 it makes sense to ensure that the slower processors are used in all of the model timings since when running a large parallel job it is inevitable that some of the processors will be the slower 2.2GHz ones. Furthermore, in order to better control the effects of multiple cores on WRG3, we have chosen to undertake all of the sequential runs using four copies of the same code: all running on the

		m2-WRG2	m2-WRG3	m1-WRG2	m1-WRG3
1 node	$a =$	-0.2864	$5.139e-04$	$-6.917e-02$	$2.943e-04$
	$b =$	$4.795e-04$	-0.1658	$1.069e-04$	-0.2250
2 nodes	$a =$		$4.083e-04$		$3.191e-04$
	$b =$		$-8.518e-03$		-0.2316
1 node	$\alpha =$	$3.230e-02$	-0.2079	$-5.760e-02$	$6.012e-02$
	$\beta =$	$1.385e-04$	$5.704e-04$	$9.135e-05$	$-8.833e-05$
	$\gamma =$	$2.877e-07$	$-4.765e-08$	$1.309e-08$	$3.228e-07$
2 nodes	$\alpha =$		-0.4211		$-1.698e-02$
	$\beta =$		$9.6205e-04$		$3.106e-05$
	$\gamma =$		$-4.6713e-07$		$2.430e-07$

Table 1: Parameters (a, b) and (α, β, γ) of the T_{comm} models (4) and (5) respectively, obtained for $np = 4$ and for $N(4) = 512, \ldots, 4096$ with coarse grid of size 32

same node. This simulates the situation that will exist for a large parallel run. In addition to this, for WRG3, two predictive timings are obtained. The first of these is obtained by running the 4 process parallel job on a single node, whilst the second is obtained by running an 8 process parallel job across two nodes. The latter is designed to allow both intra-communication (communication between processes in the same node) and inter-communication (communication between processors in different nodes) overheads to be captured by our model (the former will only capture intra-communication costs).

Given the values shown in Table 1, it is now possible to make predictions for the performance of the multigrid codes on greater numbers of processors. In this paper, we consider executing the codes on $np = 64$ processors, with the grids scaled in size in proportion to np. Figure 3 shows results using the linear model, (4), whilst Figure 4 shows similar results based upon the quadratic model (5). In each case all four combinations of algorithms (m1,m2) and computer systems (WRG2,WRG3) are presented. Note that for the runs on WRG3 two predictions (Tpredict1 and Tpredict2) are presented: these are based upon the parameters in Table 1 when 1 node (4 cores) or 2 nodes (8 cores) are used in step 2 respectively.

Comparing Figs. 3 and 4 we see that for both codes the linear model does just as good a job as the quadratic model on the single core architecture of WRG2. In all cases the error between the predictions and the actual timings is around about 20-30% for the largest problem considered ($N(64) = 16384$). This relatively large under-prediction is almost certainly due to the effects of the slower processors not being taken into account correctly when capturing the parameters that are presented in Table 1.

From Fig. 4 it is apparent that using quadratic model can provide significantly better predictions for both multigrid implementations on WRG3, provided that Tpredicted2 is used. Recall that this prediction is based upon the use of 8 cores in step 2 of the methodology described in the previous section (as opposed to 4 cores for Tpredicted1). This clearly demonstrates that, since the large parallel job is likely to run across all available cores on each node, and to use multiple nodes, both intra- and inter-communications must be captured by the predictive model.

5 Conclusion and Future Work

In this paper we have proposed a simple methodology for predicting the performance of parallel multigrid codes based upon their characteristics when executed on small numbers of processors. The initial results presented are quite encouraging and demonstrate that reasonably accurate and reliable predictions are possible provided that sufficient care is taken with the construction of the model and the evaluation of its parameters. In particular, it has been possible to demonstrate that the effects of a multicore architecture can be captured and that the models proposed can be applied to two quite different multigrid codes.

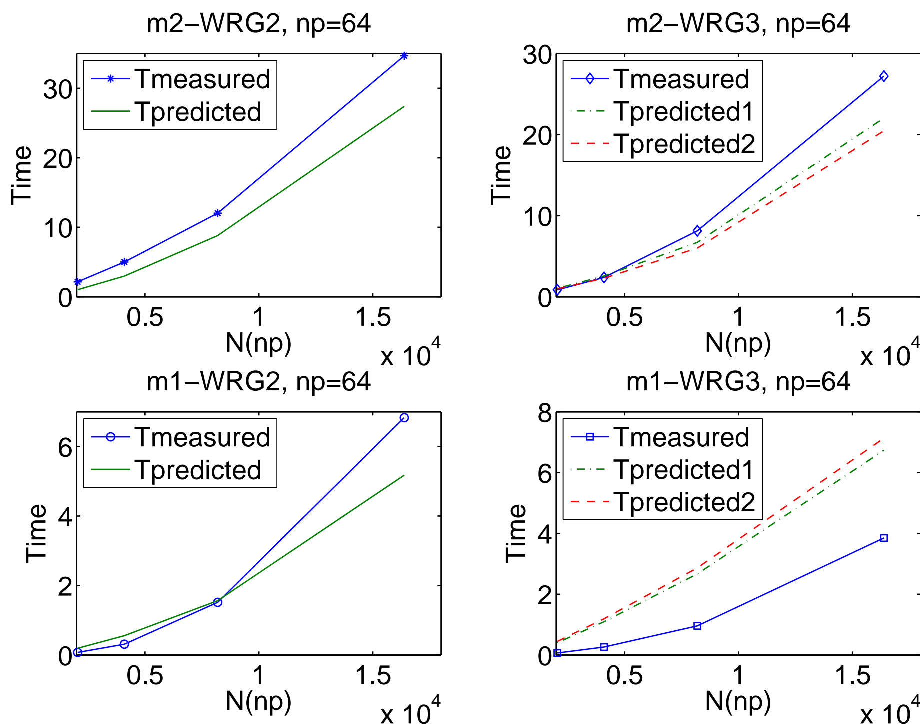

Figure 3: Linear model $T = T_{comp} + a + bN$ for Time predicted, and Time measured on $np = 64$ processors, $N(64) = 2048, \ldots, 16384$

Nevertheless, it is clear that further research still needs to be undertaken before we can reach our goal of guaranteeing optimal resource utilization across computational Grids. For example, where an architecture is not completely homogeneous (e.g. WRG2) more care needs to be taken in the evaluation of the model parameters. Furthermore, additional investigations are required in order to better understand the sensitivity of the predicted times with respect to these parameter values.

Other issues that will also need to be considered in future work include the incorporation of a global convergence test into the model and of an exact solver at the coarsest level. These have not been important in the results presented here since only a fixed number of multigrid V-cycles have been used in the numerical tests and the coarsest grid solves have just used a large fixed number of RBGS sweeps. In practice however the number of cycles and the number of coarse grid sweeps will not be fixed and will depend upon whether or not convergence has been achieved.

Finally, we observe that, in addition to deciding which single computational resource to use in order to complete a given computational task, there will be occasions when multiple Grid resources are available and might be used together. Consequently, in future work we intend to extend our models to provide predictions that will allow decisions to be made on how best to split the work across more than one resource and to determine the likely efficiency (and cost-effectiveness) of so doing.

References

[1] Globus Project, http://www.globus.org (2007).

[2] Bank, R.E. and Holst, M.J., A New Paradigm for Parallel Adaptive Meshing Algorithms. *SIAM Review* **45**, 292–323 (2003).

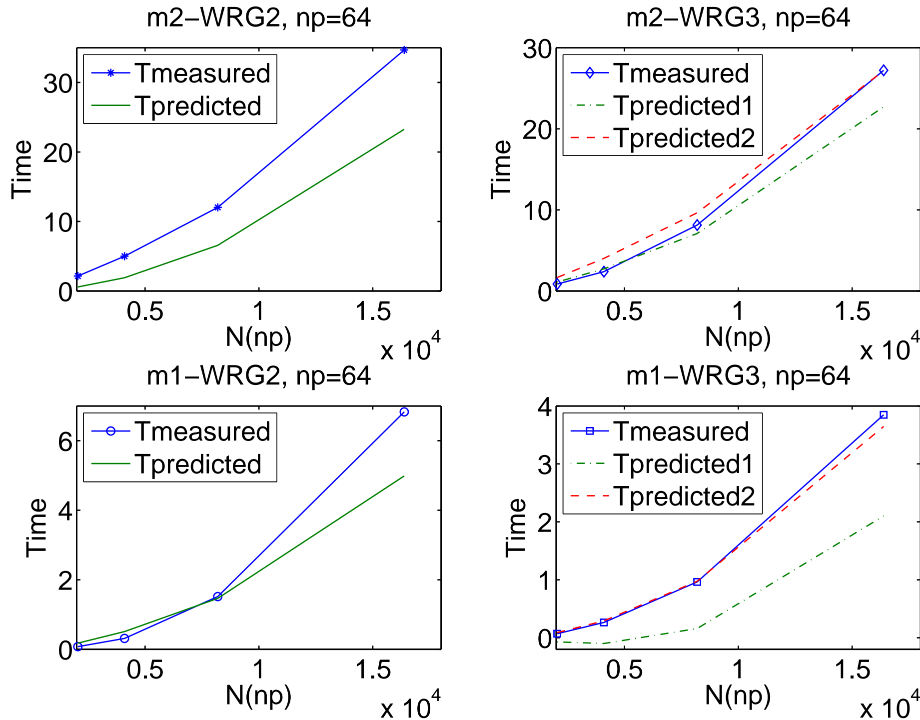

Figure 4: Quadratic model $T = T_{comp} + \alpha + \beta N + \gamma N(1)^2$ for Time predicted, and Time measured on $np = 64$ processors, $N(64) = 2048, \ldots, 16384$

[3] Brandt, A., Multi-level adaptive solutions to boundary value problems. *Mathematics of Computation* **31**, 333–390 (1977).

[4] Briggs, W.L., Henson, V.E. and McCormick, S.F. *A Multigrid Tutorial.* SIAM (2000).

[5] Buyya, R., Abramson, D. and Giddy, J.G., Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid. In *Proceedings of the 4th Int. Conf. on High Performance Computing in Asia-Pacific Region.* IEEE Computer Society (2000).

[6] Dew, P.M., Schmidt, J.G., Thompson, M. and Morris, P., The White Rose Grid: practice and experience. In: Cox, S J (editor) *Proceedings of the 2nd UK All Hands e-Science Meeting.* EPSRC (2003).

[7] Huedo, E., Montero, R.S. and Llorente, I.M., A modular meta-scheduling architecture for interfacing pre-WS and WS Grid resource management services. *Future Generation Computing Systems* **23**, 252–261 (2007).

[8] Koh, Y.Y., *Efficient Numerical Solution of Droplet Spreading Flows.* Ph.D. Thesis, University of Leeds (2007).

[9] Krauter, K., Buyya, R. and Maheswaran, M., A taxonomy and survey of Grid resource management systems. *Int. J. of Software: Practice and Experience* **32**, 135–164 (2002).

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

[11] Snir, M., Otto, S.W., Huss-Lederman S., Walker D.W. and Dongarra J., *MPI – The Complete Reference.* MIT Press (1996)

[12] Trottenberg, U., Oosterlee, C.W. and Schüller, A., *Multigrid.* Academic Press (2003).