Published in IET Software
Received on 16th January 2009
Revised on 8th June 2009
doi: 10.1049/iet-sen.2009.0007

In Special Issue on Performance Engineering

ISSN 1751-8806

Performance prediction and procurement in practice: assessing the suitability of commodity cluster components for wavefront codes

S.D. Hammond[1] *G.R. Mudalige*[1] *J.A. Smith*[1] *J.A. Davis*[1] *A.B. Mills*[1] *S.A. Jarvis*[1] *J. Holt*[2] *I. Miller*[3] *J.A. Herdman*[3] *A. Vadgama*[3]

[1]*High Performance Systems Group, University of Warwick, Coventry, UK*
[2]*Tessella PLC, Reading, Berkshire, UK*
[3]*Supercomputing Solution Centre, Atomic Weapons Establishment, Aldermaston, UK*
E-mail: sdh@dcs.warwick.ac.uk

Abstract: The cost of state-of-the-art supercomputing resources makes each individual purchase a length and expensive process. Often each candidate architecture will need to be benchmarked using a variety of tools to assess likely performance. However, benchmarking alone only provides a limited insight into the suitability of each architecture for key codes and will give potentially misleading results when assessing their scalability. In this study the authors present a case study of the application of recently developed performance models of the Chimaera benchmarking code written by the United Kingdom Atomic Weapons Establishment (AWE), with a view to analysing how the code will perform and scale on a medium sized, commodity-based InfiniBand cluster. The models are validated and demonstrate a greater than 90% accuracy for an existing InfiniBand machine; the models are then used as the basis for predicting code performance on a variety of alternative hardware configurations which include changes in the underlying network, the use of faster processors and the use of a higher core density per processor. The results demonstrate the compute-bound nature of Chimaera and its sensitivity to network latency at increased processor counts. By using these insights the authors are able to discuss potential strategies which may be employed during the procurement of future mid-range clusters for wavefront-rich workloads.

1 Introduction

Modern supercomputing resources are constantly evolving. Where once a 'supercomputer' may have been a shared memory machine comprising tens of processors housed in a single structure, today supercomputing resources commonly utilise multiple sub-structures such as cabinets, multiple-processor nodes and more recently multiple-core processors. When combined with the complex network interconnects found in modern systems, identifying and analysing the performance properties of the machine as a whole becomes a significant challenge. With the growing core counts of modern machines and the ever increasing complexity of each system, the task of procuring the 'right' computing machinery for purpose is rapidly becoming a lengthy and intricate process. Pure benchmarking of applications on candidate architectures serves only limited purpose – the results only highlight the performance of specific codes and often only for specific inputs. For organisations who want the very best machine performance, a deeper knowledge of

code behaviour with respect to each prospective platform is needed.

Performance modelling has been used as a basis for machine comparison [1, 2] and post-installation performance verification [3], and has been shown in a number of examples to address many of the questions that may arise during procurement. While serving as a showcase for many performance modelling techniques, the focus of these studies has been on very large emerging architectures and not on the small- to medium-sized commodity or near-commodity clusters common to many research organisations. In these procurement activities similar issues must be addressed but with hardware that may have a lower specification, be arranged differently or have alternative behaviour to the expensive components that are common place in high-end supercomputing systems.

In this paper we utilise two recently developed performance models to explore the performance of the Chimaera neutron transport benchmark developed and maintained by the United Kingdom Atomic Weapons Establishment (AWE), targeting a processing element (PE) count of up to 4096 cores. The direct use and cross-comparison of predictions from two different performance modelling techniques aids not only in elucidating specific code and machine behaviour, but also in increasing the accuracies of our observations. This work does not report on the respective costs of each procurement strategy, but instead provides some degree of quantitative exploration of various hardware and application configurations, which can in turn support the queries that may arise during the early stages of a procurement. The specific contributions of this work are:

- The presentation of a performance study for the AWE Chimaera benchmark on commodity or near-commodity hardware. This is the first such study for the Chimaera benchmark and is designed to support future procurement activities for mid-range supercomputing resources at AWE. We use two approaches in verifying our predictions: (i) based on analytic modelling methods utilising the recently developed 'plug and play' reusable wavefront model from [4] and (ii) using a new discrete event simulation toolkit [5]. Both approaches show predictive accuracies of over 90% and provide higher confidence in the conclusions obtained from our performance engineering study.

- A quantitative exploration of the key parameters that affect the performance of wavefront codes on modern commodity HPC systems, supporting the exploration of prospective machine configurations for procurement.

- An exploration of the contention costs arising on a CMP-processor-based cluster when executing Chimaera and the implications for code runtime and machine procurement.

- A comparison of three compiler toolkits for Linux with a projection for the performance of each at large processor counts, demonstrating the ability to examine the implications of software stack choice on application runtime.

- A method for assessing the performance of individual processors within the machine through the recording and graphical representation of data obtained during simulation. The graphical representation of networking and idle times provide a quick method for the determination of machine bottlenecks, which may help to expose machine design flaws or potential areas within the communication structures which may be a candidate for optimisation.

The remainder of this paper is organised as follows: Section 2 provides a brief overview of the two main approaches to application performance modelling – those based on analytical studies and those based on simulation; the Chimaera benchmark is introduced in Section 3, continuing our discussion in Section 4 by describing the development of two performance models using analytical techniques and a new simulation-based toolkit; Sections 5 and 6 contain a case study in which we benchmark an existing 11.5 TFLOP/s InfiniBand system and project runtimes for a variety of alternative application and machine configurations; Section 7 discusses the performance behaviour associated with message passing interface (MPI) rank allocation on the machine once the hardware topology is known; a comparison of generally available compiler toolkits for Linux is presented in Section 8, with model-based runtime projections for each at large scale; finally the paper concludes in Section 9 with a summary of the results and a review of the implications for procuring a small- to medium-sized cluster for sustained wavefront-rich workloads.

2 Performance modelling

Application performance modelling is principally charged with the derivation of models by which code behaviour can be analysed and predicted. In the main, the interest in such models is in analysing how the computation and communication structures in a code change with respect to an increased processor count or problem size. By developing a deeper insight into the runtime fluctuations resulting from such changes, an understanding of code bottlenecks, software optimisations and optimal runtime configurations can be developed.

Current techniques for developing application performance models fall into two distinct categories – those based on analytical studies and those based on simulation. Although some conceptual work on a binding of the two is discussed in the POEMS framework [6], there has been little practical demonstration reported in academic literature. Analytical studies [7–9], which seek to represent code behaviour by a series of mathematical formulae, are often developed within some modelling framework or abstraction methodology (e.g. LogP [10], LogGP [11] and LoPC [12]). The use of rigid frameworks for modelling helps alleviate some of the complexity involved in modelling and provides a generic basis

doi: 10.1049/iet-sen.2009.0007

upon which code behaviour can be judged. The challenges of using an analytical approach include identifying the key application parameters that affect runtime behaviour and understanding how best to represent each mathematically. The analysis of code for modelling is often based on manual code inspection which, although time consuming, allows the performance modeller to develop a deeper understanding of specific code behaviour from which further behavioural insights may be garnered.

An overview of the recently developed 'plug and play' reusable wavefront model [4], which is used as the basis for our analytical exploration of Chimaera, is presented in Section 4.1. Note that the development of a reusable model serves to reduce the time required to model future wavefront codes, since a flexible framework can now be applied to any wavefront application; this approach also permits cross-application comparisons to be made within a highly abstract and algorithm-specific framework.

Simulation-based performance tools (e.g. the Wisconsin Wind Tunnel [13], PROTEUS [14] and the PACE toolkit [15, 16]) were originally envisaged as mechanisms to decrease the burden of performance modelling by eliminating the need to manually inspect application source code. The automated replay of applications, either in source or binary form, allowed developers and performance modellers alike to experiment by making changes to the application and simulating execution without requiring direct access to the specific machine in question. In practice, the simulation environments developed to date have attempted to directly simulate individual application instructions, making the simulation of large industrial codes infeasible in realistic time frames. When application complexity is compounded with the increasing sophistication of emerging clusters, the use of simulation quickly becomes intractable as a source of fast and effective performance evaluation. In Section 4.2 we describe a simulation toolkit, which seeks to overcome some of the problems discussed; it includes the use of coarser-grained computational timings (as opposed to individual instruction timings) and a 'layered' approach to network modelling, which results in significantly reduced simulation times, while providing prediction accuracies commensurate with leading analytical models. The focus of this paper is on the application of the reusable analytic model and the simulation toolkit to one particular benchmark, further examples of their application (to the NAS Parallel Benchmark Suite) can be found in [17].

3 Chimaera benchmark

The Chimaera benchmark is a three-dimensional neutron transport code developed and maintained by the United Kingdom Atomic Weapons Establishment (AWE). On first inspection the code shares a similar structure with the now ubiquitous Sweep3D application described in numerous analytical performance studies [2, 8, 18]. Unlike Sweep3D, however, the code employs a different internal sweep ordering and utilises a complex convergence criteria to decide when execution is complete. To support the description of the performance models, we present a concise description of the wavefront algorithm employed by both Sweep3D and Chimaera. Our discussion is deliberately brief as existing papers describe the behaviour of the wavefront algorithm (e.g. [19]) in more detail.

3.1 Generic wavefront algorithm

The generic three-dimensional wavefront algorithm operates over a data array of size $N_x \times N_y \times N_z$. The data array is decomposed over a two-dimensional processor array sized $m \times n$. Each processor receives a 'column' of data sized $N_x/m \times N_y/n \times N_z$. For the purposes of our discussion it helps to consider this column as a stack of N_z tiles, each being $N_x/m \times N_y/n \times 1$ in size. The algorithm proceeds by executing sweeps through the data which pass from one vertex to its opposite corner. For Chimaera and Sweep3D eight sweeps are used – one for each vertex in the three-dimensional space.

A sweep originates at a vertex of the processor array (the origins of each sweep for Chimaera are shown in Fig. 2). The computation required to solve the first tile in the originating processor's stack is completed and boundary information is exchanged with the two neighbouring processors. Once exchanges are complete the two neighbouring processors solve the first tile in their stack, whereas the originating processor solves its second tile and so on. On completion, boundary information is again passed downstream to neighbouring processors. A sweep completes once all tiles in the last processor have been solved. Fig. 1 shows a partially complete sweep with dark grey tiles having been solved in previous stages; light grey tiles are currently executing and white tiles are awaiting boundary information from upstream processors (arrows are used to show visible communications to downstream processors). A full 'iteration' of the wavefront algorithm in Chimaera requires all eight sweeps to have completed.

4 Modelling Chimaera

The modelling of Chimaera has been conducted using two approaches – analytical modelling based on the 'plug and

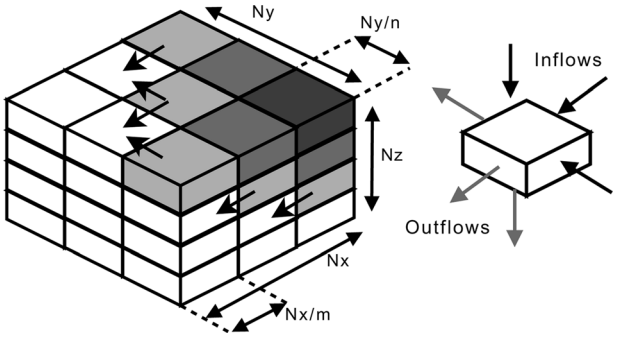

Figure 1 *Sweep execution through the data array in Chimaera*

Table 1 Reusable wavefront model application parameters

Model parameter	Chimaera value
N_x, N_y, N_z	problem input size
W_g	measured
$W_{g,\text{pre}}$	0
H_{tile}(cells)	1
n_{sweeps}	8
n_{full}	4
n_{diag}	2
message size$_{\text{EW}}$ (bytes)	$8H_{\text{tile}} \times \#\text{angles} \times N_y/m$
message size$_{\text{NS}}$ (bytes)	$8H_{\text{tile}} \times \#\text{angles} \times N_x/n$

play' reusable model [4] and using the WARPP simulation toolkit developed by the University of Warwick [5].

4.1 Plug and play analytical model

The 'Plug-and-play' reusable wavefront model developed in [4] represents the culmination of three individual application performance studies for the Sweep3D, Chimaera and NAS-LU benchmarks. By using the insights obtained in modelling these three wavefront codes, the authors have extracted and abstracted the common parameters (shown in Table 1) which affect application runtime into a generic model. The computation time, W_g, and the computation time per cell prior to the algorithm kernel, $W_{g,\text{pre}}$, are the only machine-specific values for which benchmarking of the application is required. For our study these values were obtained using a manually instrumented version of the benchmark that times the core computational kernel of the wavefront algorithm. $W_{g,\text{pre}}$ is set to zero in Chimaera as there are no computational sections in the sweep algorithm prior to the main kernel.

The sweep ordering parameters, n_{sweeps}, n_{full} and n_{diag} represent the total number of sweeps per iteration, the number of full sweeps, and the number of half sweeps, respectively. The concept of 'full' and 'half' sweeps relates to the ability of sweeps within the application to overlap. Recall the sweep ordering presented in Fig. 2. Sweep 2 originates on the processor located in the top right corner of the processor array. Once this sweep has successfully passed through the bottom right corner (the starting location for sweep 3) the next sweep will begin. If this starts prior to sweep 2 finishing on the bottom left processor, overlapping occurs, which serves to increase the efficiency of the code. Overlapping can only occur if sweep i finishes at the starting location for sweep $i+1$ while other downstream processors are still processing sweep i. This occurs twice in Chimaera (sweep pairs 2, 3 and 6, 7) giving an n_{diag} value of 2. The full reusable model is presented in Table 2 with the complete equation for runtime given in (r5). Explanations of each sub-equation are given in [4]. Note that in [4] the authors develop a complex LogGP communications model for the Cray XT4 architecture. In this work we develop a simpler, but equally effective, regionalised least squares regression model to obtain times for MPI send and receive operations (see in Section 5.1).

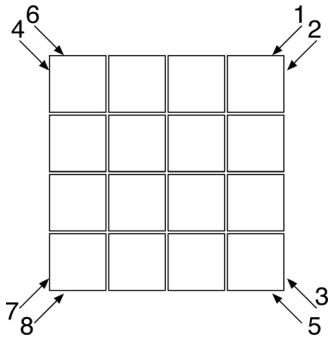

Figure 2 *Start locations for each sweep within the two-dimensional processor array employed by Chimaera*

4.2 Simulation using the WARPP toolkit

The WARwick performance prediction (WARPP) toolkit [5] has been designed to support performance prediction and code analysis on machines containing thousands of

Table 2 Plug-and-play LogGP model: single core per node

$W_{\text{pre}} = W_{g,\text{pre}} \times H_{\text{tile}} \times N_x/n \times N_y/m$	(r1a)
$W = W_g \times H_{\text{tile}} \times N_x/n \times N_y/m$	(r1b)
Start $P_{1,1} = W_{\text{pre}}$	(r2a)
Start $P_{i,j} = \max(\text{Start } P_{i-1,j} + W_{i-1,j} + \text{Total\_comm}_E + \text{Receive}_N, \text{Start } P_{i,j-1} + W_{i,j-1} + \text{Send}_E + \text{Total\_Comm}_S)$	(r2b)
$T_{\text{diagfill}} = \text{Start } P_{1,m}$	(r3a)
$T_{\text{fullfill}} = \text{Start } P_{n,m}$	(r3b)
$T_{\text{stack}} = (\text{Receive}_W + \text{Receive}_N + W + \text{Send}_E + \text{Send}_S + W_{\text{pre}})N_z/H_{\text{tile}} - W_{\text{pre}}$	(r4)
time per iteration $= n_{\text{diag}}T_{\text{diagfill}} + n_{\text{full}}T_{\text{fullfill}} + n_{\text{sweeps}}T_{\text{stack}} + T_{\text{nonwavefront}}$	(r5)

doi: 10.1049/iet-sen.2009.0007

processors. More specifically, we intend for this toolkit to provide accurate simulations for modern massive parallel processor (MPP) machines which might consist of multi-core, multi-processor cabinet structures each having their own complex interconnect or protocol. As the sizes of future machine architectures continue to grow, we expect that additional sub-structures will be required to support increasing core density and on-board characteristics such as memory and bus topology. With this in mind, the structure of a machine is relayed to the simulator by a series of 'profiles.' Each profile has unique performance properties such as network latency, outbound bandwidth etc. When developing a simulation the user is required to specify the respective values for each property and also a mapping of MPI processes to profiles for the specific machine configuration being analysed. By providing a generic basis for the description of a machine, arbitrarily complex (and heterogeneous) hardware models can be developed, enabling the exploration of not only machine structures but also future multi-structured computing resources.

Simulations developed using WARPP build on the observation that parallel codes are ordered executions of basic blocks separated by control flow, calls to network transmissions or I/O operations. Like previous simulators we recreate application behaviour by replaying the code's control flow, pausing during execution to directly simulate computation, communication and I/O events. Communication between processes are simulated fully ensuring that transmissions between nodes block when the transmissive partner is otherwise engaged. Computation is, however, modelled quite differently to existing work in that it does not simulate each application instruction directly. Instead, the toolkit replaces basic blocks within the control flow with the estimated (or actual) time that the block requires for execution on the target platform. The switch from instruction-level simulation to coarser grained computational timings significantly reduces the time required for individual simulations, it also significantly improves the scalability of the simulator to processor counts considerably higher than in the previous toolkits. An issue that arises in moving to coarser-grained computational timings is precisely how the time for the block is extracted from the application. To alleviate the manual instrumentation of code to obtain such timings, the toolkit includes an automated code analyser which injects timing routines into the application source code directly, thus creating an instrumented benchmark of the code. The analyser also generates a control flow representation of the code, detailing where each block can be found and identifying its associated execution time from the instrumented application output.

4.2.1 Developing a simulation in WARPP: Developing a WARPP simulation involves three stages. First the application source code is analysed using automated code analysis tools – these are responsible for diagnosing the 'basic blocks' of the application and extracting a control flow graph for each process in the parallel application. Basic blocks are defined as being separated by either a change in the address counter (as would be caused by a branching statement or loop) or a communication (such as an `MPI_Send` or `MPI_Recv`). Once the basic blocks have been found, each is instrumented with timing routines to record the wall time that is required for execution. Two outputs are produced at this stage of simulation – an instrumented version of the application's source code, and a basic performance model that describes the control flow of the application as well as the arrangement of basic blocks within this control flow and the points at which communication and I/O occurs.

The second stage of simulation requires the user to benchmark the target machine using the instrumented version of the code and a reliable MPI benchmarking utility (such as MPPTest [20] or the Intel MPI Benchmark [21]). The output of these benchmarks, which takes the form of a 'work time' for each sequential block and a set of network latencies and bandwidths, is then fed into the third stage of simulation; here the control flow is replayed, using the wall-clock times of each block to calculate the compute time to which the communication behaviour in the application directly simulated and added to obtain a complete model.

During a simulation, data relating to the application's performance and machine utilisation is recorded, which enables the performance modeller to replay the simulated execution at a later date and analyse where execution time was spent (for example, time spent in communication, computation, idle etc). Recent work has also demonstrated the utility of such analysis in directing potential code optimisations ahead of implementation [22].

5 Modelling code performance on a commodity high-performance cluster

We present the results of a benchmarking and modelling exercise conducted on the 11.5 TFLOP/s Warwick Centre for Scientific Computing (CSC) IBM supercomputer. This machine (Francesca) is typical of a large, sub-million pound commodity cluster available today, comprising of 240 dual-Intel Xeon 5160 dual-core nodes each sharing 8 GB of memory (giving 1.92 TB in total). Nodes are connected via a QLogic InfiniPath 4X, SDR (raw 10 Gb/s, 8 Gb/s data) QLE7140 host channel adapters (HCAs) connected to a single 288-port Voltaire ISR 9288 switch. Processor core to HCA ratio is 4:1. Each compute node runs the SUSE Linux Enterprise Server 10 operating system and has access to the IBM GPFS parallel file system [23]. For our study the Intel C/Fortran 10 compiler suite was used in conjunction with OpenMPI 1.2.5 [24] and the PBS Pro scheduler. By default, jobs launched under the CSC PBS installation are allocated 'freely' in the system – that is to any free core that meets the wall time or memory resources requested by the job. Nodes and processors are shared between jobs unless specifically

requested during submission. Runtimes can therefore vary by as much as 10–15% between successive jobs because of the varying placement of processes within the machine and the potential sharing of nodes.

The benchmarked values from this machine serve two purposes – firstly to allow us to verify our performance models against a set of known runtimes ensuring accuracy, and secondly to form the basis of projections for alternative machine configurations that may be considered during a procurement exercise.

5.1 Network benchmarks and models

The results of machine benchmarking demonstrating raw MPI latency and bandwidths are shown in Table 3. Note that the network benchmarking is partitioned into two regions by message size. The point at which the split in network performance occurs is at 2048 bytes, indicating that the InfiniBand management system may be configured for a maximum transmission unit (MTU) size of 2 Kbytes (a maximum of 4K is supported by the HCA and switch).

For both performance studies (analytical and simulation based) we model the communication time for a message of length x bytes as $t_{\text{send}}(x) = (1/B)x + n_l$, with the bandwidth ($B$) and latency ($n_l$) associated with the appropriate region for x. The time for a receiver is modelled by $t_{\text{recv}}(x) = (1/B)x$, since the receiver does not experience the latency required to establish the connection but must spend at least the actual transmission time in a locked state accepting data from the network interconnect.

5.2 Performance model validation

Table 4 presents validations of the analytical and simulation-based performance models for the CSC-Francesca machine. The average prediction error is 10.46% for the analytical model and 9.03% for the simulation, demonstrating the high degree of accuracy in the models and the strong correlation between both studies.

Table 3 Benchmarked network performance for the CSC-Francesca machine (measurements taken using the Intel MPI benchmarking utility version 3.0 [21]; using Intel C compiler version 10 with default system MPI libraries)

Network profile	Message size (bytes)	Latency (n_l) (microsec)	Bandwidth (B) (Gbytes/s)
on-chip (core to core)	≥ 0	0.655	2.70
off-processor (processor to processor)	<2048	0.69	2.80
	≥ 2048	0.91	3.83
off-node (node to node)	<2048	2.64	0.46
	≥ 2048	3.63	0.73

Note that the vast majority of the predicted runtimes are below the actual execution time – this is to be expected as both performance models assume as 'perfect' allocation of processor cores within the machine, assuming that neighbouring MPI ranks will be allocated as closely as physically possible. In practice, the free placement of processes causes some degree of increased execution time because of the higher network costs experienced. Similarly, the natural load and noise that occur from shared resources helps to create variation in execution. Additionally, predictions are taken from averaged estimates of machine parameters for which rounding and measurements errors may also occur.

5.3 Processor performance breakdown

Since every event processed by the WARPP simulator is categorised into either compute, network send, network

Table 4 Model validations on the CSC-IBM Francesca machine using Intel Fortran 10.0 with `-O2` optimisation; OpenMPI 1.2.5; runtimes are wall time for sweeping components in seconds; negative values indicate under-predictions

Core count	Problem size	Actual runtime (s)	Analytical pred. (s)	Sim. pred. (s)	Analytical error (%)	Sim. error (%)
32	120^3	107.18	88.76	89.58	−17.19	−16.42
64	120^3	56.72	47.59	48.75	−16.09	−14.04
128	120^3	32.56	28.20	28.98	−13.40	−11.01
81	240^3	342.33	326.45	330.46	−4.64	−3.47
96	240^3	297.03	268.71	277.56	−9.54	−6.55
100	240^3	278.37	243.36	248.32	−12.58	−10.79
128	240^3	225.65	205.50	207.18	−8.93	−8.18
169	240^3	174.35	174.35	177.09	−0.88	1.57
256	240^3	129.65	115.58	117.98	−10.85	−9.01

doi: 10.1049/iet-sen.2009.0007

receive or idle states, accurate breakdowns of processor behaviour can be obtained by keeping running totals of the time spent in each state. Fig. 3 shows four sets of data obtained from the simulation of a 256-core execution of the 240^3 Chimaera problem on two types of machine – one with an entirely homogeneous InfiniBand network (Figs. 3*a* and *b*) and the other from a machine that is structured as a dual-core, dual-processor system similar to Francesca (Figs. 3*c* and *d*). These graphics show the processors of the machine represented as a two-dimensional array, with MPI rank 0 located at the bottom left and rank 255 at the top right. Ranks are allocated columnwise according to a node-fill allocation strategy in which MPI ranks are allocated contiguously to nodes without any over-subscription. For the purposes of discussion, processor idle or wait times represent any event where the PE being simulated was required to halt execution pending the completion of another system operation. In parallel applications these pauses in execution typically relate to the time spent waiting for a blocking network operation to complete. In order to provide higher fidelity analysis, the simulator distinguishes the time spent waiting for an MPI operation to complete from the time spent actually conducting the transmission, thus is it possible to analyse the proportion of time the processor spends conducting useful activities for each individual PE in the machine.

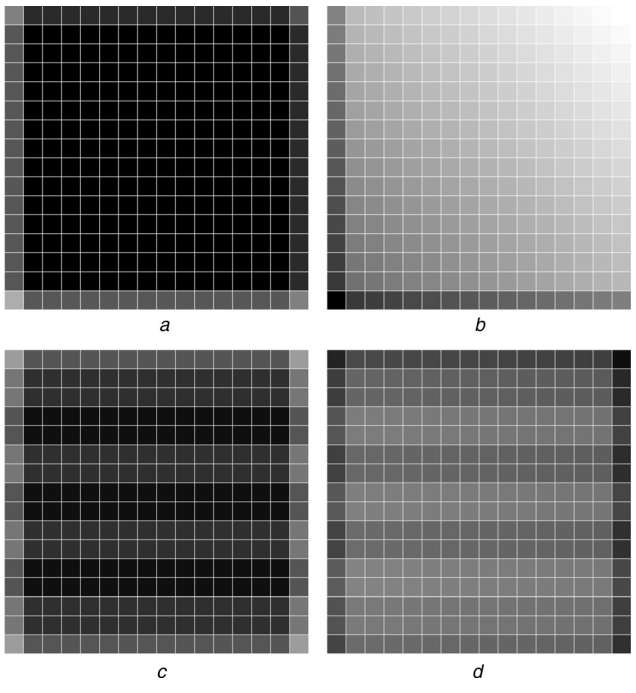

Figure 3 *Relative time spent in network and wait states for a 256 (16 × 16) processor array computing the Chimaera 240^3 problem (darker indicates more time in operation, processors are arranged by MPI rank, rank 0 at the bottom left corner, rank 255 at the top right corner)*

a Communications (single sweep, homogeneous network)
b Wait/idle (single sweep, homogeneous network)
c Communications (8 sweeps, Francesca)
d Wait/idle (8 sweeps, Francesca)

This represents an advantage over analytical techniques which focus on the critical path execution time.

Figs. 3*a* and *b* show a single sweep through the processor array which is reflected in the gradient-like shading that occurs for processor idle times. The idle states are darker towards the bottom left of Fig. 3*b* since the sweep originates at the top right corner and hence downstream processors must wait for their upstream counterparts to complete before starting their first computation. The communications pattern shown in Fig. 3*a* utilises an entirely homogeneous network (i.e. there are no intra-core or processor-to-processor communications present) and is very even, only the edges of the processor array experience lower communication time as they have no neighbours on at least one side with which to spend time communicating.

In Figs. 3*c* and *d* we present the relative times spent in communication and idle states when all eight sweeps are executed on Francesca. In this simulation, MPI ranks are allocated using a node fill mechanism – that is all free cores on a node are allocated contiguously. The introduction of multiple cores and processors to each node creates a less consistent performance across the processor array with dark bandings appearing where communications require the use of the InfiniBand interconnect. By using similar combinations of events, for example relative time spent in computation, we can use the data recorded during simulation to diagnose potential bottlenecks in either the machine or application's performance. In this particular case the InfiniBand interconnect represents a bottleneck as slower network behaviour is experienced at the edge of the nodes communications.

6 Procurement: assessing the suitability of machine components

Following the benchmarking of the CSC-Francesca machine and validation of the performance models, we present several sub-studies exploring alternative machine and application configurations. In the following studies we analyse the effect on code runtime of (i) an increase in problem size, (ii) moving to a Gigabit Ethernet or Myrinet 2000 network, (iii) the installation of InfiniBand resources with identical bandwidth but increased latency, (iv) a change in the performance of individual processor-cores and (v) a doubling of processor-core density.

6.1 Large problem sizes

New computing machinery is often purchased with the aim of increasing the complexity or size of problem which can be solved. The decision of which machine to purchase may be governed by expectations of how future users intend using the system. Fig. 4 shows the expected parallel efficiency of an increased input size with increasing processor count. Note that there is a significant decline in efficiency for each input size as the processor-core count

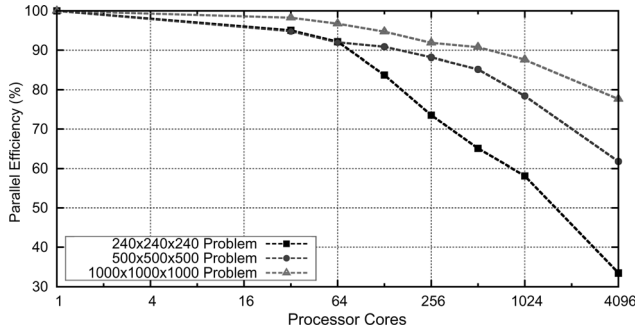

Figure 4 *Parallel efficiency of large problem sizes using the infiniband interconnect*

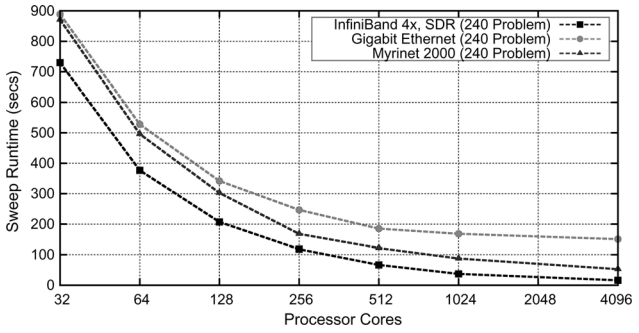

Figure 5 *Chimaera runtime using InfiniBand (4x, SDR), Myrinet 2000 and gigabit Ethernet interconnects*

rises. This effect is attributable to the increasing proportion of runtime accounted for by communication, resulting from a decrease in computation time per processor and an increase in the number of network transmissions in the system as a whole.

The measure of parallel efficiency is of particular interest to AWE as parallel jobs are mandated to run at greater than 50% parallel efficiency wherever possible; users will specifically choose processor-core counts to target this value. For the 240^3 problem this turning point occurs between 1024 and 2048 cores indicating the approximate core count which may be required per job if targeted specifically for a 50% parallel efficiency. Depending on how many simultaneous jobs the organisation hopes to execute at this level of an approximate core count for procurement can be deduced. For larger problem sizes a similar analysis is also possible – one would expect significantly more cores to be required before the 50% parallel efficiency turning point is reached.

6.2 Choice of networking interconnect

For any machine intended to execute high-performance parallel codes the choice of interconnect is particularly acute. The precise mix of latency, bandwidth capacity and cost must be balanced to support the compute resources in delivering a smooth and consistent performance. At the time of procurement it is common to assess not only which interconnect will provide the best raw performance, but also what the effect of changing the interconnect or choosing a slightly lower specification will have on overall runtime. We have modelled two such choices: (i) whether to select a gigabit Ethernet, a Myrinet 2000 network or an InfiniBand interconnect and (ii) the effect of purchasing an InfiniBand network with identical 4x, SDR bandwidths but with 25, 50 and 75% higher latencies.

Fig. 5 presents the predicted runtimes for a hypothetical machine in which we exchange an InfiniBand interconnect with gigabit Ethernet and Myrinet 2000 network. The gigabit runtime is consistently over 100 s slower than the InfiniBand-based system, reflecting the impact of increased latency and a significant decrease in bandwidth. The Myrinet 2000 interconnect offers similar bandwidth to gigabit Ethernet yet reduced latency. The effect of this arrangement is runtimes more consistent with the InfiniBand system at large processor configurations when the runtime becomes dominated by network activity.

In analysing the results the reader should consider the economics of purchasing either fewer processors and a more expensive InfiniBand network, or a greater number of processors and a less expensive gigabit interconnect; a typical decision that will be faced in any procurement activity. For the Chimaera benchmark the results demonstrate that between two and four times as many processors will be required to offset the degradation of using a slower interconnect – a significant increase which will in turn make the machine more expensive to run and potentially more difficult to administer. While this example in network performance might seem extreme, similar comparisons can be made between any two networks that have similar performance characteristics – for example two different vendor InfiniBand offerings.

In Fig. 6 we demonstrate predictions for the percentage increase in runtime resulting from the use of an 4x SDR InfiniBand interconnect with 25, 50 and 75% higher latencies. For small processor counts (less than 1000) the increase in runtime is less than 6% in all cases. After this, where communication begins to become a higher proportion

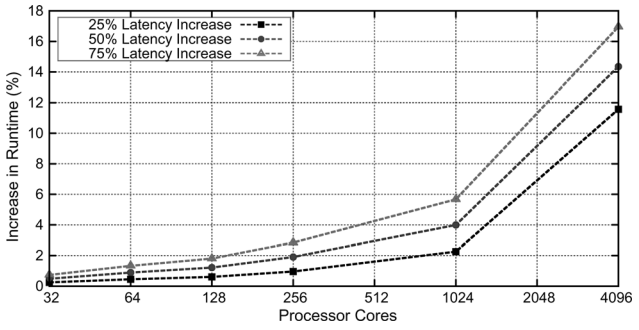

Figure 6 *Increase in runtime from a 4x SDR InfiniBand network with varying increases in latency*

doi: 10.1049/iet-sen.2009.0007

of runtime, the runtime begins to increase rapidly (by at least 10%). In this scenario the purchase of a lower specification system may be acceptable if the intention is to limit the maximum processor count of each job to 1024 cores or less.

Machine configurations for node counts greater than 288 will also cause increases in wire latencies as fat-tree-based switch topologies will need to be employed in order to cope with the extra port count. These costs are not included in this work as benchmarked values to support a predictive model are not currently available; Johnson *et al.* [25] suggest that contention within InfiniBand switches may be reduced in future systems through the use of advanced routing algorithms. Fig. 6 provides some indication of how sensitive the structured communication pattern used in Chimaera is to even minor increases in network latency.

6.3 Machine compute performance

The compute resources of the machine are usually the feature that draws the most attention. While only part of the picture for parallel systems, the computational aspects of a code are often better understood by domain experts and developers. With increased variation in processors – increased core counts and arrangement, clock speed differences and varying cache implementations – choosing the 'right' processor for an application can be difficult. We present several studies which attempt to quantify the performance benefit of choosing either 10 or 20% faster processors, 10 or 20% slower processors, or in substituting existing dual-core Intel Xeon 5160 processors for quad-core chips with the same per-core performance but high core-density per processor.

6.3.1 Increased individual core performance: Fig. 7 presents the predicted change in runtime from using dual-core processors with individual core performances of +10, +20, −10 and −20%. The diminishing returns demonstrate the respective points at which communication begins to dominate runtime. In each case the change in runtime performance is approximately equal to the change in per core performance for small processor counts. As the processor count rises the impact on runtime is reduced because of the increased proportion of runtime accounted for by communication, reducing the contribution of faster compute resources to the runtime. Note that at increased processor counts the impact on runtime of using a slower processor is also reduced. The choice of core performance should therefore be considered in the context of job size – at small job sizes the runtime is significantly improved by using the fastest processors possible; as the total core count in use rises, there are diminishing returns from employing faster computational resources.

6.3.2 Increased core density – dual against quad core: With an increasing variety of multi-core processors becoming available, including dual-, quad- and oct-core configurations, a common issue arising in procurement is which core density to select in designing the machine's compute architecture. On initial consideration the economic advantages of higher core densities are consolidation and reduced power or cooling demands per core, however, the increasing density often impacts on runtime performance.

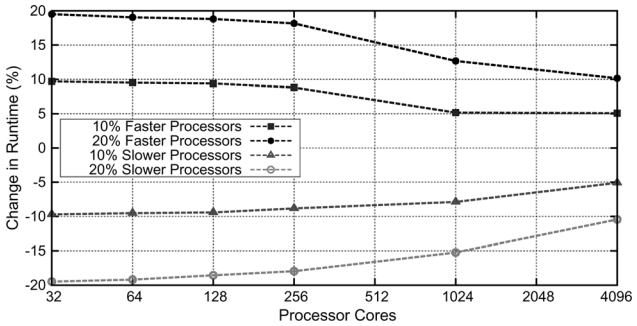

Figure 7 *Change in runtime from varying individual processor-core processing performance*

In Figs. 8*a* and *b* we show a set of results obtained from running the Intel MPI benchmark in three configurations – one, two and four MPI processes per node, respectively. The increasing number of processes per node (which is the effect of higher core densities) reduces the per-core network performance. The increased time to perform an MPI send, and the decreased per-core bandwidth, results from high levels of contention for the single InfiniBand HCA per node. Each process must wait longer before having exclusive access to the machine network. Note also the increased volatility of the network's performance that arises from the contention, the effect of this, which appears as spikes in

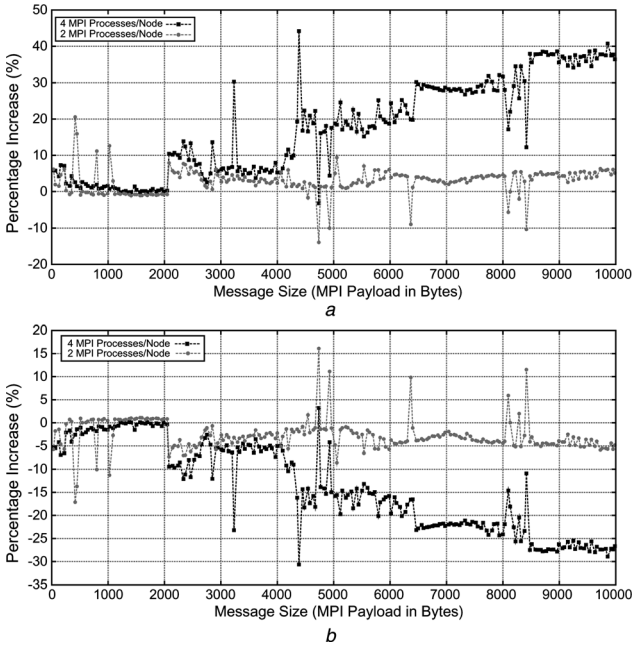

Figure 8 *Percentage change in network performance when increasing the MPI processes-per-node from one to two or four*

a Time to send
b Per-core bandwidth

Table 5 Predicted quad against dual-core performance (the quad-core configuration is modelled with an increased time to send and reduced bandwidth to account for contention)

Total core count	Dual-core runtime (s)	Quad-core runtime (s)	Runtime percentage change (%)
32	729.97	726.19	−0.52
64	376.46	373.62	−0.75
128	207.18	207.92	0.36
256	117.98	118.50	0.44
1024	66.64	66.33	7.88
4096	37.29	40.45	8.46

the network's performance, is that the communication performance of the machine is less consistent and therefore, the runtimes present a greater degree of variance. If core densities continue to increase then there will be an even greater impact on performance unless the issue of contention is addressed by increasing the number of networking channels per node – the economic effect of this may be a significant addition to procurement cost.

We have modelled the effect on runtimes of replacing each existing dual-core processor with a quad-core equivalent in which the per-core compute performance remains identical. The network latency for the InfiniBand network has been left the same for message sizes of less than 2048 bytes, increased by 10% for message sizes ranging from 2048 to 4096 bytes and increased by 20% for larger messages. Network bandwidth is decreased in the same fashion. The changes in latency and bandwidth are drawn from the observed values shown previously. Table 5 presents the predicted runtimes for the quad-core machine compared with the existing dual-core machine. Initially, performance is improved since there are more cores utilising the fast core-to-core transmission. Once the core counts reach 1024 processors the increased latency and reduced bandwidth create up to an 8% increase in runtime.

7 MPI rank allocation strategies to improve performance

Once a specific set of hardware has been purchased it is left to system managers and users to configure the system in a manner that provides the best performance in the context of the jobs being executed. Identifying how the basic parameters, which govern the placement of MPI ranks or allocation of processors, will affect performance prior to machine purchase is not only advantageous when selecting a faster machine but is also useful for priming system administrators to the likely behavioural characteristics of a machine before it is installed. In this study we evaluate

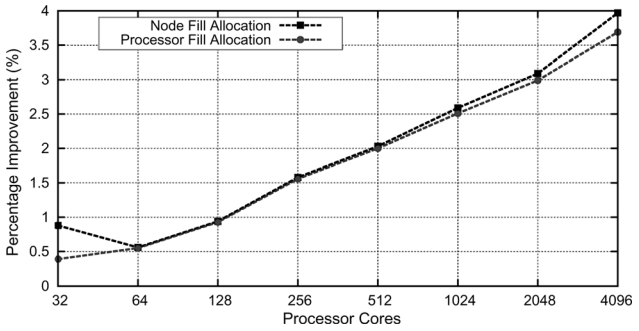

Figure 9 *Simulated performance improvement relative to round-robin allocation for Chimaera 240^3 problem using node and processor fill strategies*

three potential rank allocation strategies that may be used in high-performance parallel environments:

- Round robin scheduling, where ranks are allocated by looping over all of the available hardware nodes assigning the next MPI rank in sequence until all have been allocated.

- Node fill allocation, in which each node is filled completely in turn ensuring that contiguous blocks of MPI ranks are allocated to each node. In a dual-core, dual-processor system ranks 0–3 are allocated to a single node and so forth.

- Processor fill allocation, where the cores of a single processor are assigned in each pass around the hardware nodes. This allocation helps to ensure that smaller contiguous blocks of MPI ranks are created in comparison to the Node Fill Allocation scheme but that rank topology is more suited to two-dimensional processor grids such as those used by Chimaera. In a dual-core, dual-processor machine, ranks 0 and 1 are allocated to a node, 2 and 3 to the next node and so forth until all ranks have been allocated.

Fig. 9 presents the relative performance improvement of each strategy over a simple round-robin allocation. Note that the round-robin allocation gives the slowest runtimes since the layout of the ranks does not make use of the near-neighbour communications that are present in the two-dimensional processor decomposition employed by Chimaera. Both the node and processor fill allocations result in improvements in runtime when compared with round-robin allocations of between 0.5 and 4.0% for processor configurations of up to 4096 cores; the improvement is similar for both schemes. These performance improvements also represent the reduction in runtimes which is achieved through diligent use of intra-node communications, providing some indication of how highly this should be prioritised when determining scheduling allocation policies.

8 Compiler selection and performance at scale

The choice of compiler toolkit for a machine is usually driven by availability as well as vendor preference. In Fig. 10 we present

doi: 10.1049/iet-sen.2009.0007

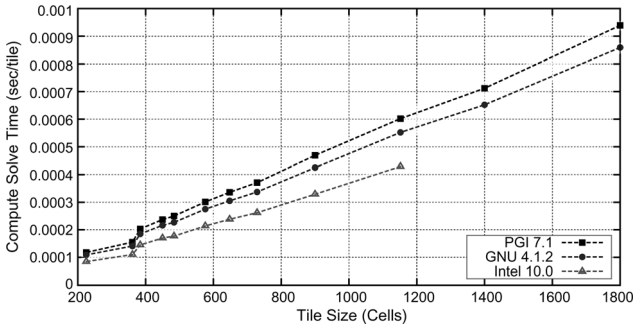

Figure 10 *Per-tile solve time for Intel, PGI and GNU compiler toolkits (−O3 optimisation, OpenMPI 1.2.5)*

the per-tile solve time required when using three compiler toolkits for Linux: (i) GNU 4.1, (ii) Intel 10.0 and (iii) PGI 7.1.2. In each of these experiments the 240^3 problem was decomposed over a variety of processor grids to achieve the tile sizes of interest. Similar code executions compiled using −O2 optimisation achieves solve times within 0.5% of those shown above. While further platform-specific optimisations may be possible with each compiler, the majority of general users apply few optimisation flags to the build processes in order to maintain predictable runtime behaviour. Therefore the use of single optimisation flags is more representative of final code performance than execution using fully tuned compilation settings. Note the absence of times for the Intel Fortran compiler (Fig. 10) at larger tile sizes – this results from the Chimaera executable producing segmentation faults for the low processor counts required with this large tile size. For the tile sizes that do complete execution, the Intel compiler is up to 40% faster in tile-solve time than the GNU and PGI toolkits. When translated into a series of simulations at large scale, the effects of the slower compute times for the GNU and PGI tools are shown in Fig. 11. As the proportion of time accounted for by communications grows with the increasing processor count, the effect of the faster Intel compiler is reduced, offering less than 10% performance advantage at 16 384 cores.

The ability to differentiate the performance of potential software stacks is of particular interest to procurements that utilise generic offerings such as Linux, where a wide variety

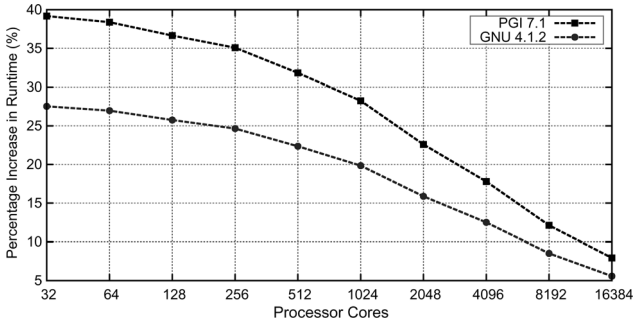

Figure 11 *Compiler performance at scale (relative to Intel 10.0, −O3 optimisation, OpenMPI 1.2.5)*

of toolchains may be available. Performance modelling in this respect allows an analysis of the performance cost ratio of more expensive licences against generally available open source systems such as GNU. Similar analysis can be conducted on the impact on runtime of different software configurations, where minor tuning may improve execution times at scale.

9 Conclusions

We present a series of case studies detailing the use of two application performance models – one based on analytical techniques and the other based on simulation. The case studies are focused on support for mid-range commodity clusters for a wavefront-rich workload. The paper explores the performance and scalability of the Chimaera benchmark code written and maintained by the United Kingdom AWE.

The performance models for the Chimaera benchmark are accurate to greater than 90% for a variety of processor configurations and input sizes. The cross-correlation of predictions from two contrasting performance modelling techniques serves to increase our confidence in the predictions and the insights obtained during our subsequent analysis.

More specifically, this paper shows:

- Quantitative estimates for the parallel efficiency of existing and future problem sizes that are of interest to AWE.

- That a system with a low-performance network will require a greater number of processors to offset the effect of higher latencies and lower bandwidth. We demonstrate this by projecting the performance of a gigabit ethernet network in comparison to a faster InfiniBand-based system, showing that between two and four times as many processors are required by the ethernet-based system to achieve comparable levels of performance at core counts of less than 1024.

- Reducing the latency by a factor of two, results in up to a 10% improvement in overall runtime.

- For small processor counts the overall runtime varies by the factor of improvement in per-core performance, but as the core count increases, the contribution of faster per-core performance provides diminishing returns.

- That increasing the core density per processor will reduce the performance due to contention for memory and network resources. We estimate the quantitative degradation of overall runtime when doubling core-density from dual- to quad-core processors to be approximately 8% for 4096 cores on the commodity InfiniBand system studied.

- The use of MPI node or processor fill allocation results in a performance improvement of up to 4% for 4096 cores when compared with a round-robin MPI rank allocation. This is because the network topology is more effectively exploited

by the near-neighbour communications patterns employed by Chimaera.

Our results show that the selection of machine configuration and processor count should be directed by the average size of jobs the machine is intended to execute. For multiple small jobs, individually faster processors should be prioritised over a faster interconnect, since the code is predominantly compute bound at this scale. For larger jobs, the interconnect plays a more significant role in performance indicating that a more expensive, low latency network should be targeted during procurement.

The predictive performance models used in this study provide an efficient, low-cost and rapid approach to gathering quantitative and qualitative insights into questions which arise during procurement for both currently available and future systems. In contrast, traditional approaches such as direct benchmarking require significant execution time and vendor support to arrive at a subset of conclusions that are limited by the currently available machine configurations.

10 Acknowledgments

Access to the Chimaera benchmark was provided under grants CDK0660 (The Production of Predictive Models for Future Computing Requirements) and CDK0724 (AWE Technical Outreach Programme) from the United Kingdom Atomic Weapons Establishment. Access to the CSC-Francesca machine was provided by the Centre for Scientific Computing at the University of Warwick with support from the Science Research Investment Fund.

11 References

[1] HOISIE A., JOHNSON G., KERBYSON D.J., LANG M., PAKIN S.: 'A performance comparison through benchmarking and modeling of three leading supercomputers: blue gene/l, red storm, and purple'. Proc. IEEE/ACM SuperComputing, Tampa, FL, October 2006

[2] KERBYSON D.J., HOISIE A., WASSERMAN H.J.: 'A comparison between the Earth Simulator and AlphaServer systems using predictive application performance models'. International Parallel and Distributed Processing Symp. (IPDPS), 2003, Comput. Arch. News (ACM), 2002

[3] KERBYSON D.J., HOISIE A., WASSERMAN H.J.: 'Use of predictive performance modeling during large-scale system installation'. Proc. PACT-SPDSEC02, Charlottesville, VA, August 2002

[4] MUDALIGE G.R., VERNON M.K., JARVIS S.A.: 'A plug and play model for wavefront computation'. IEEE Int. Parallel and Distributed Processing Symp. 2008 (IPDPS'08), Miami, Florida, USA, April 2008, pp. 1–14

[5] HAMMOND S.D., MUDALIGE G.R., SMITH J.A., JARVIS S.A.: 'WARPP – a toolkit for simulating high-performance parallel scientific codes'. Second Int. Conf. Simulation Tools and Techniques (SIMUTools09), Rome, Italy, March 2009

[6] ADVE V.S., BAGRODIA R., BROWNE J.C., *ET AL.*: 'POEMS: end-to-end performance design of large parallel adaptive computational systems', *Softw. Eng.*, 2000, **26**, (11), pp. 1027–1048

[7] KARP R.M., SAHAY A., SANTOS E.E., SCHAUSER K.E.: 'Optimal broadcast and summation in the LogP model'. ACM Symp. Parallel Algorithms and Architectures, 1993, pp. 142–153

[8] KERBYSON D.J., HOISIE A., PAUTZ S.D.: 'Performance modeling of deterministic transport computations'. Performance Analysis and Grid Computing, October 2003

[9] SUNDARAM-STUKEL D., VERNON M.K.: 'Predictive analysis of a wavefront application using LogGP'. PPoPP'99: Proc. Seventh ACM SIGPLAN Symp. Principles and Practice of Parallel Programming, 1999, pp. 141–150

[10] CULLER D.E., KARP R.M., PATTERSON D.A., *ET AL.*: 'LogP: towards a realistic model of parallel computation', *Princ. Pract. Parallel Program.*, 1993, pp. 1–12

[11] ALEXANDROV A., IONESCU M.F., SCHAUSER K.E., SCHEIMAN C.: 'LogGP: incorporating long messages into the LogP model for parallel computation', *J. Parallel Distrib. Comput.*, 1997, **44**, (1), pp. 71–79

[12] FRANK M., AGARWAL A., VERNON M.K.: 'LoPC: modeling contention in parallel algorithms', *Princ. Pract. Parallel Program.*, 1997, pp. 276–287

[13] REINHARDT S.K., HILL M.D., LARUS J.R., LEBECK A.R., LEWIS J.C., WOOD D.A.: 'The Wisconsin wind tunnel: virtual prototyping of parallel computers', *Meas. Model. Comput. Syst.*, 1993, pp. 48–60

[14] BREWER E.A., DELLAROCAS C., COLBROOK A., WEIHL W.E.: 'PROTEUS: a high-performance parallel-architecture simulator', *Meas. Model. Comput. Syst.*, 1992, pp. 247–248

[15] NUDD G.R., KERBYSON D.J., PAPAEFSTATHIOU E., HARPER J.S., PERRY S.C., WILCOX D.V.: 'PACE: a toolset for the performance prediction of parallel and distributed systems', *Int. J. High Perform. Comput.*, 1999, **4**, pp. 228–251

[16] KERBYSON D.J., HARPER J.S., CRAIG A., NUDD G.R.: 'PACE: a toolset to investigate and predict performance in parallel systems'. European Parallel Tools Meeting, ONERA, Paris, October 1996

[17] HAMMOND S.D., SMITH J.A., MUDALIGE G.R., JARVIS S.A.: 'Predictive simulation of HPC applications'. IEEE 23rd Int. Conf. Advanced Information Networking and Applications (AINA-09), May 2009

doi: 10.1049/iet-sen.2009.0007

[18] MUDALIGE G.R., JARVIS S.A., SPOONER D.P., NUDD G.R.: 'Predictive performance analysis of a parallel pipelined synchronous wavefront application for commodity processor cluster systems'. IEEE Int. Conf. Cluster Computing 2006, 2006

[19] LAMPORT L.: 'The parallel execution of DO loops', *Commun. ACM*, 1974, **17**, (2), pp. 83–93

[20] GROPP W., LUSK E.L.: 'Reproducible measurements of MPI performance characteristics', PVM/MPI, 1999, pp. 11–18

[21] Intel Corp: 'MPI Benchmark Utility', 2008

[22] MUDALIGE G.R., HAMMOND S.D., SMITH J.A., JARVIS S.A.: 'Predictive analysis and optimisation of pipelined wavefront computations'. Proc. 11th Workshop on Advances in Parallel and Distributed Computational Models (APDCM2009), 23rd IEEE Int. Parallel and Distributed Processing Symp. (IPDPS 2009)

[23] SCHMUCK F., HASKIN R.: 'GPFS: a shared-disk file system for large computing clusters'. Proc. First Conf. File and Storage Technologies (FAST), January 2002, pp. 231–244

[24] GABRIEL E., FAGG GR.E., BOSILCA G., *ET AL.*: 'Open MPI: goals, concept, and design of a next generation MPI implementation'. Proc. 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary, September 2004, pp. 97–104

[25] JOHNSON G., KERBYSON D.J., LANG M.: 'Optimization of InfiniBand for scientific applications'. Int. Parallel and Distributed Processing Symp. 2008 (IPDPS'08), Miami, FL, USA, April 2008

[26] HAMMOND S.D., MUDALIGE G.R., SMITH J.A., JARVIS S.A.: 'Performance prediction and procurement in practice: assessing the suitability of commodity cluster components for wavefront codes'. UK Performance Engineering Workshop, Imperial College, London, UK, July 2008