

WARPP - A Toolkit for Simulating High-Performance Parallel Scientific Codes

S.D. Hammond, G.R. Mudalige, J.A. Smith, S.A. Jarvis
Department of Computer Science
University of Warwick
Coventry, United Kingdom
{sdh,esviz,jas,saj}@dcs.warwick.ac.uk

J.A. Herdman and A. Vadgama
Supercomputing Solution Centre
Atomic Weapons Establishment
Aldermaston
United Kingdom

ABSTRACT

There are a number of challenges facing the High Performance Computing (HPC) community, including increasing levels of concurrency (threads, cores, nodes), deeper and more complex memory hierarchies (register, cache, disk, network), mixed hardware sets (CPUs and GPUs) and increasing scale (tens or hundreds of thousands of processing elements). Assessing the performance of complex scientific applications on specialised high-performance computing architectures is difficult. In many cases, traditional computer benchmarking is insufficient as it typically requires access to physical machines of equivalent (or similar) specification and rarely relates to the *potential capability* of an application. A technique known as *application performance modelling* addresses many of these additional requirements. Modelling allows future architectures and/or applications to be explored in a mathematical or simulated setting, thus enabling hypothetical questions relating to the configuration of a potential future architecture to be assessed in terms of its impact on key scientific codes.

This paper describes the Warwick Performance Prediction (WARPP) simulator, which is used to construct application performance models for complex industry-strength parallel scientific codes executing on thousands of processing cores. The capability and accuracy of the simulator is demonstrated through its application to a scientific benchmark developed by the United Kingdom Atomic Weapons Establishment (AWE). The results of the simulations are validated for two different HPC architectures, each case demonstrating a greater than 90% accuracy for run-time prediction. Simulation results, collected from runs on a standard PC, are provided for up to 65,000 processor cores. It is also shown how the addition of operating system jitter to the simulator can improve the quality of the application performance model results.

Categories and Subject Descriptors

C.4 [Performance of Systems]: Measurement techniques, Modelling techniques, Performance attributes; I.6.8 [Simulation and Modelling]: Type of Simulation - Discrete event

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIMUTools 2009 Rome, Italy

Copyright 2009 ICST 978-963-9799-45-5.

General Terms

Performance Modelling, Simulation

Keywords

Application Performance Modelling, Simulation, High Performance Computing

1. INTRODUCTION

Assessing the potential performance of parallel scientific codes on existing and future HPC architectures is crucial for national laboratories such as the Los Alamos National Laboratory (LANL) and the United Kingdom Atomic Weapons Establishment (AWE). Procurement decisions are largely directed by the need to ensure, and understand, future scientific capability. Therefore, being able to assess the potential scientific delivery from one HPC architecture, against a rival system, is paramount. Organisations such as LANL and AWE are also keen to ensure that their scientific codes, often developed over many decades, make best use of new architectural innovations and that any code development is done in such a way that it improves potential performance rather than hinders it.

Application performance modelling – that is, assessing application/architecture combinations through modelling – is an established academic field, and there are several examples of where the application of such approaches prove to be advantageous: input/code optimisation [22], efficient scheduling [26], post-installation performance verification [19], and the procurement of systems for the United States Department of Energy [19]. The process of modelling itself can be generalised to three basic approaches; modelling based on analytic (mathematical) methods, (e.g. LogP [6], LogGP [2], LoPC [9]), modelling based on tool support and simulation (e.g. PACE [11, 4] and DIMEMAS [10, 20]), and a hybrid approach which uses elements of both (e.g. POEMS [1]). Modelling based on tool support has a number of advantages over its more mathematical counterpart: firstly, it is often based on (source) code analysis, which absolves the user from translating lengthy programmatic features into abstract analytical program models; secondly, tool support allows larger-scale problems to be tackled, opening up the possibility of full-scale application analysis, as opposed to analysis based on small, core application kernels; thirdly, mathematical models often hide the mechanics of execution, subsuming complex, synchronised activities into collective mathematical expressions - in parallel codes in particular, understanding this complex synchronisation amongst processes is often the key to understanding application performance.

Despite these benefits, tool-supported application modelling techniques *for scale* are difficult to develop. Typical applications codes

can be tens or hundreds of thousands of lines in length, and blend a variety of programming languages and libraries. To add to this, modern HPC architectures consist of tens of thousands of processing elements, have increasing levels of concurrency (threads, cores, nodes), deeper and more complex memory hierarchies (register, cache, disk, network), mixed hardware sets (CPUs and GPUs) and layered interconnects (node, processor, blade, chassis). It is not feasible therefore to simulate the behaviour of each application instruction on each element of the target hardware (if one is to scale beyond a few thousand processor cores).

In this paper we introduce the Warwick Performance Prediction (WARPP) tool kit which comprises a suite of tools designed specifically to support the rapid and efficient generation of accurate, flexible performance models for high-performance parallel scientific codes. The centrepiece of this tool kit is an efficient and portable discrete-event simulator, which can scale to tens of thousands of processor cores yet at the same time provide high levels of modelling accuracy. In order to demonstrate the capabilities of WARPP, we document its application to an industry-strength procurement benchmark developed and maintained by the United Kingdom Atomic Weapons Establishment (AWE), one of the UK's largest users of supercomputing resources. The focus of this paper is therefore, to (1) provide a detailed description of the tool kit and simulator, (2) demonstrate its application to a real-world high-performance parallel scientific code and (3) illustrate the capabilities of this simulation approach *to scale*, that is, to model real application behaviour on HPC systems beyond 50,000 processing elements, with complex layered interconnects and in the context of background operating systems noise.

The specific contributions of this paper are:

- The presentation of a new discrete event simulation-based toolkit, which supports the modelling of industry-strength parallel scientific codes on modern HPC architectures that scale to tens of thousands of processor cores. The predictive accuracy of the resulting models exceeds 90%;
- A detailed description of the simulator's support for multiple networks within a single simulation allowing for the evaluation of application performance on complex network and machine topologies with high degrees of accuracy;
- A description of the use of coarse-grained computational modelling, permitting rapid and accurate simulations of computational behaviour - a key technique in improving the scalability of the simulations;
- The development of a simulation-based performance model for an AWE HPC benchmark code on two high performance computing architectures, with extended projections for up to 65,000 processor cores. The simulation is also used to elucidate several properties of the runtime behaviour of the code, including a breakdown in terms of computation/communication, parallel efficiency with increasing problem size input and performance in the presence of operating system noise. An evaluation of the simulator's performance in providing these insights is also examined.

The remainder of this paper is organised as follows: Section 2 discusses related work; Section 3 introduces the WARPP tool kit and details the discrete event simulator; in Section 4 we describe the application of the modelling toolkit to AWE Chimaera and study the performance of the benchmark for two high performance computing systems; the paper concludes in Section 5.

2. RELATED WORK

Simulation-based performance studies employ specialised simulator hardware or software to remove the requirement for the user to have expertise in model construction or computing hardware. The simulator will attempt to replicate the behaviour of the code with respect to a set of input parameters such as machine processor count, network latency *etc.* Examples include the Wisconsin Wind Tunnel [23], PROTEUS [3] and the PACE toolkit [4, 11] also developed at the University of Warwick. The notable problem with this previous research has been that in order for simulations to achieve appreciable levels of accuracy, each individual program instruction must be simulated directly. As scientific codes and modern machine sizes have grown in size and complexity, this approach has led to intractable simulation times. Whilst many of the techniques which underpin these toolkits are still relevant, and indeed are used as a basis for the work presented in this paper, the lengthy simulation times which have resulted, as well as limited support for complex networking models, make these toolkits less plausible solutions for users who require accurate models of large, complex parallel codes.

The more recent DIMEMAS project [10, 20] alleviates the instruction-based simulation approach through replay of traces obtained during a run of the application. Evaluation in the context of different machine sizes is supported through the regeneration of a trace, subject to the user's specification. This approach has been successfully demonstrated on machine sizes of up to 1000 processor cores. The reliance on traces however, acts as an inhibitor to the manual tuning or changing of a performance model, since the code behaviour is implicitly contained within the trace rather than explicitly described in a user-editable model. Editing such complex structures is non-trivial and the initial creation requires that the code actually be written in the first place - modelling-led prototyping of algorithms is therefore not possible. The traces employed are also large in size requiring considerable disk space and system memory, placing severe limits on the maximum model size that can be processed on an individual workstation in feasible timeframes.

In [14], Grove and Coddington develop the Performance Evaluating Virtual Parallel Machine (PEVPM) which provides lightweight and rapid predictions of performance and execution through the use of directives, effectively removing the need to simulate complete application execution. The development of the MPIBench benchmarking tool aids the PEVPM by providing high fidelity network models with probability distributions for the variance in transmission times. The work presented in this paper draws on similar techniques to the PEVPM, but offers three additional contributions: (i) complex network models; (ii) simulation to large-scale and, (iii) the impact on the simulation of computational noise.

Denzel *et al* present MARS - a toolkit for the simulation of high-performance computing systems in [8]. This system is primarily constructed to elucidate the performance of computing hardware including complex network topologies. The mechanism employed is similar to DIMEMAS in that application traces are replayed but, the simulator itself is considerably more flexible in the design of simulated hardware. The work presented in this paper differs from MARS in that our focus is directed to the development of application performance models which provide insight into the behaviour of algorithms and applications - the MARS system is built to assess the performance of computing hardware in the context of an application.

3. THE WARPP TOOLKIT

The WARwick Performance Prediction (WARPP) toolkit presented

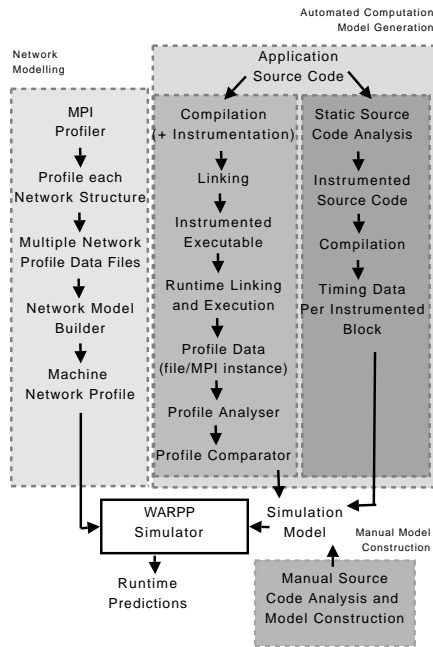


Figure 1: The WARPP Modelling Process

in this paper is a prototype semi-automatic performance prediction environment which supports the exploration and analysis of a code's performance on machines consisting of thousands of processors. More specifically, our interest is on the accurate simulation of complex scientific codes on modern Massively Parallel Processor (MPP) machines, which may be constructed from multi-core, multi-cabinet components, each of which may have complex interconnects and communication protocols requiring modelling. Note that the abstraction of a machine into a set of virtual processors and a set of networks permits modelling of distributed computational resources, including SMP-machines.

Figure 1 presents the workflow which is associated with the development of a simulation model using the WARPP toolkit. The modelling process requires four stages: (1) model construction, (2) machine benchmarking using a reliable MPI benchmarking utility [12, 17], a filesystem I/O benchmark [25] and an instrumented version of the application, (3) the post-execution analysis of machine benchmarking results to produce simulator inputs and finally (4) simulation.

The reader will note that three methods of model construction are proposed in Figure 1 - (1) hand-coded simulation script programming, (2) automated script generation from static source code analysis and (3) automated script generation from post-execution trace analysis [15]. In this paper we focus exclusively on method 1 - manual model development - since it provides the most accurate performance models and enables us to focus on how simulations are carried out without the added complexity of discussing the tools required for automated model generation. Tools to support methods 2 and 3 are in development.

3.1 Model Construction

The WARPP Simulator accepts simulations which are written in a C-like language designed specifically to reduce the knowledge requirements in developing a simulation model. As the simulator is based on discrete event methods the script can essentially be thought of as a program which generates events that are of inter-

Table 1: Events Supported by the WARPP Simulator

Event	Parameters
Compute	Time
Network Send	Destination, Message Size, Type, Tag, Blocking
Network Recv	Origin, Message Size, Type, Tag, Blocking
Wait/Idle	Time
I/O Read	Read Length
I/O Write	Write Length

est to the simulator. Six types of events are currently supported (see Table 1) in the simulation scripting language, which represent the most common activities associated with the execution of a parallel code. During the execution of a simulation the model script is evaluated, with the simulator halting script execution to process each event as it is generated. Each event is generated by a directive placed into the simulation script.

Computation modelling in the WARPP simulator is based on the use of coarse grained 'blocks' of code. This differs from previous toolkits such as PACE [4, 11], which employed per-instruction simulation and more recent simulators such as DIMEMAS which use trace-profiles. The WARPP toolkit can be considered somewhere in-between these two approaches with the focus on a 'block' of computation approximately equal to those used by compilers in the generation of code. A block might therefore be thought of as a group of instructions but is likely to be smaller and finer than whole sections of computation recorded during a trace-based profile.

The timings for each 'block' are obtained by the direct instrumentation of source code with timing routines, with multiple blocks being instrumented within a single application. The instrumentation is currently performed by hand, however we are in the process of developing Fortran and C-language tools to perform instrumentation immediately prior to compilation. Each block corresponds to a single compute event during simulation. The placement of timing routines should therefore represent a trade-off between simulator performance and flexible modelling - too large blocks reduce the experimentation possible in the simulations but improve simulation times, whilst too small blocks increase the flexibility of a model at the cost of increasing the number of events the simulator has to process. From our experience of modelling using the simulator, the instrumentation of 'blocks' of code should roughly follow the notion of a 'basic-block' used by compilers, which is typically a section of code such as a loop body or a group of statements between function calls. This provides for adequate flexibility, which often also correlates with the user's conceptual structuring of the code, but does not overwhelm the simulator with a large number of events to process.

By means of an example consider an entire loop. The loop body contains a single block of code which corresponds to one compute event per iteration. The time for the block, and thus the simulated time for each iteration, is obtained by placing a start timer call immediately prior to the loop starting and an end-timing call immediately after the loop has completed. The time the loop takes to execute is divided by the number of iterations and recorded in a manner as to map the time to the equivalent event during simulation. For non-loop blocks the time for the event corresponds to the time between the start and end timer calls. Since each block is liable to be executed multiple times in a scientific code, the time for each block is averaged over the course of an execution. When any MPI function is encountered in the application source code it is wrapped by a call to stop the code timing and then a clock restart immediately after completion. This maintains a clear division be-

tween the timing of computation and communication within the executing application.

The introduction of instrumentation to a code does have an impact on the performance of the application at runtime and in some cases can disrupt the optimisation process during compilation. These overheads are nevertheless less than those generated when profiling. This is because the timing calls are compiled directly into the application, resulting in efficient, in-context timing, which does not require control to pass from the application to a large external profiling library. Note that the use of timers in the code can be substituted for toolkits such as PAPI, which provides low overhead, processor/instruction information, or other advanced machine measurement utilities. In the experiments conducted for this paper, the overheads resulting from the use of timing statements amounts to less than 1% of total execution time, as the timed blocks of code are extremely large when compared to the timing routine used. For codes where the block size is considerably smaller we are developing methods of obtaining timing information using light-weight methods such as reading processor clock registers. Instrumenting on a per-block basis allows us to gain a valuable insight into the times associated with each individual section of code. We therefore gain finer-grained information on the computational behaviour of the code, itself directed by the user. Such flexibility is not possible through profiling alone where the profiling information is often recorded as a coarser (typically function) level.

We also note that since the timings used for each compute event are simply numerical values relating to the wall time being used for processing, they can be generated through methods such as low-level processor simulation, statistical analysis based on existing results or analytical modelling. Thus users are able to develop flexible performance studies on a per-event basis, swapping instrumented compute times for abstract models where desired or developing entire simulation studies using a combination of these approaches. Since the simulation environment also exposes a rich scripting language, a limited degree of non-deterministic behaviour is able to be modelled through user refinement. We are also investigating methods for the limited pre-processing or in-simulation processing of input decks to guide execution behaviour, potentially enabling data dependent runtimes to be modelled.

3.2 Developing Simulator Inputs and Modelling Machine Networks

Following code instrumentation, the timed code as well as reliable MPI [12, 17, 24, 13] and file I/O benchmarks [25] are executed on the target machine. Only a limited number of processors are required for this purpose, since the timings which are obtained can then be used to produce estimates for individual events in the context of increased input sizes or processor counts. MPI and I/O timings are used to produce the notion of a “time per byte,” so that the simulator can produce predictions of communication or read/write times without requiring large lookup tables to be created in memory. In this subsection we describe how the timings recorded during benchmarking are processed prior to simulation, so that accurate models of computation, I/O and networking can be developed.

The computation times per event are recorded from an instrumented run on the application. The output of this run is collated and entered into a ‘globals’ file ready for simulation. The separation between the timing of each event and the simulation script itself helps to promote reusability of the model between different simulations and provides for greater degrees of flexibility.

Modern machines frequently contain more than one network - by means of example consider a large multi-core machine composed of SMP nodes connected via an InfiniBand interconnect. In this

machine there are at least three networks which will be utilised at runtime - the low latency, high bandwidth core-to-core bus, a fast processor-to-processor bus within each SMP node and the slower, lower bandwidth InfiniBand network. Each of these networks has complex performance properties which must be modelled if the simulation of communications is to be accurate. In [5] the author’s show that in a number of high performance codes the use of local communications (*i.e.* those within a single node) can be up to 50% of the total messages sent during execution, demonstrating a substantial motivation for complex network simulation mechanisms to be developed.

This process is achieved in the WARPP toolkit by the construction of multiple network ‘profiles’. A profile represents one network within the machine. Within each profile the message space is divided into distinct regions to enable the modelling of networks which utilise multiple protocols - for instance, many machines will utilise a special small message protocol and then use an alternative for larger messages and so forth - or packet based chunking of network transmissions for which there may be various complex behaviours. The performance of the interconnect in each region is described by two parameters - latency and bandwidth. The latency and bandwidth values used for each region are calculated by performing a least-squares regression over the data obtained during network benchmarking. Multiple regions are then grouped to form a profile.

The topology of the machine is relayed to the simulator by a set of triples - each containing the identifiers of two virtual processors and the network which connects them. In a dual-core, dual-processor SMP machine, virtual processors 0 and 1 will be mapped to the core-to-core profile, with processors 0 and 2 mapped to the processor-to-processor profile and so forth.

As with all inputs to the WARPP simulator, network topologies and profiles are communicated in plain text allowing for easy modification by end-users. We are also in the process of building automatic topology description tools which relay information obtained during the scheduling of a job or from post-execution analysis of MPI benchmarks. The separation of machine topology from the simulation model and input parameters allows for further reusability between simulations and the rapid creation of alternative topological investigations which may be generated by the machine scheduler, workload analysis or experimentation.

A similar process to the construction of a network model is applied to output from the filesystem I/O benchmark to form the machine’s I/O model.

3.3 Simulation using the WARPP Simulator

Following the creation of all required simulator inputs, the simulation of the code is now possible. Simulation is conducted on the WARPP simulator, which is written entirely in Java to aid portability between machines and the reproducibility of results between runs and installations. Four inputs are required for accurate simulation - a simulation script (which contains the control flow and event structure of the application), a set of ‘global’ values which contains the respective timing for each computational event, the machine’s network model and finally an I/O model.

A simulation is carried out by the creation of a set of ‘virtual processors’ within the simulator - with each being responsible for maintaining control-flow and expression stacks as well as a localised timeline. A virtual processor is an abstract processing element - this represents a physical processor core or in the case of uni-core processors a complete processor. The execution of the simulation script proceeds by the swapping of control flow between one of the virtual processors in the system, and handlers within the

simulator which process the events being generated. A virtual processor simply executes the control flow of the simulation model, halting when an event is reached and passing control flow back to the simulator for processing. In order to improve performance the simulation script is compiled to Java bytecode by the creation of multiple small functions which comply with the requirement for execution to halt when an event is reached.

Computation and wait/idle events are the most easily processed since they require the virtual processor's timeline to be incremented by a specific time - for compute events this is resolved by locating the time associated with the event in the simulator 'globals' input, for wait events the simulator generates the next event in the virtual processor and then resolves how long the processor would have been idle.

Networking and I/O events are passed to special handlers within the simulator that check that both the sender and receiver are ready and have posted the correct event details (*i.e.* the message size, tag, destination are all aligned). The time associated with the sending transmission is obtained by finding the correct network associated with the sender and receiving virtual processor and then calculating the time required for the transmission of the message using the latency and bandwidth values from the respective region. Where transmissions between the sender and receiver do not occur at the same point in virtual time, the simulator will stall the respective processor until needed, recording this time as a wait event. The time for each event is recorded against the virtual processor's localised timeline.

A simulation completes once all virtual processors have fully executed the simulation script, after which timeline summaries are displayed to the user. The simulator includes several options to change its behaviour to suit user preferences, one of which records the entire timeline for each virtual processor into a series of traces allowing post-simulation inspection of behaviour.

4. CASE STUDY: MODELLING THE AWE CHIMAERA BENCHMARK

The Chimaera benchmark is a three-dimensional particle transport code developed and maintained by the United Kingdom Atomic Weapons Establishment (AWE). It employs the wavefront design pattern which is described briefly in Section 4.1. The purpose of the benchmark is the replication of operational behaviour of larger internal codes which occupy a considerable proportion of parallel runtime on the supercomputing facilities of AWE. The code shares many similarities with the ubiquitous Sweep3D application [16, 18] developed by the Los Alamos National Laboratory (LANL) in the United States, but is considerably larger and more complex in its operation. Unlike Sweep3D, Chimaera employs alternative sweep orderings within the data array, a convergence criteria to halt simulation (Sweep3D always executes precisely 12 iterations) and extended mathematics. The AWE Chimaera code is also relatively unknown in academic literature with only one existing analytic model [22] - there are no existing simulations of the benchmark. It is worth noting that the modelling of both Chimaera and Sweep3D continues to be of interest to laboratories such as LANL and AWE because of the considerable proportions of execution time consumed, thus accurate models of existing and future machines can help to direct procurement and tuning so that each machine is run at its maximum performance.

4.1 Generic Wavefront Algorithm

The wavefront algorithm, originally the "hyperplane" method, is based on work by Lamport from the 1970s which investigated meth-

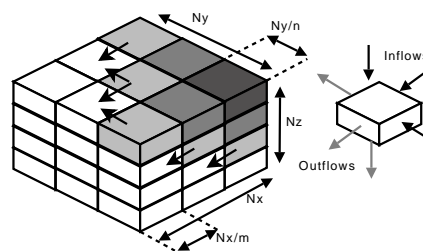


Figure 2: Wavefront Executing through a 3D Data Array

Table 2: Benchmark Machine Specification

	Francesca	Skua
Architecture	Distributed Cluster	Shared-Memory
Nodes	240	1
Processor	Intel Xeon 5660	Intel Itanium-II
Instruction Set	x86-64	IA-64
Processor Clock	3.0Ghz	1.6Ghz
Processors/Node	2	56
Cores/Processor	2	1
Total Cores	960	56
Memory per Node	8GB	112GB
Total Memory	1.92TB	112GB
Network Interconnect	4x SDR-InfiniBand	SGI NUMalink
File System	12TB (IBM-GPFS)	3.7TB
Operating System	GNU/Linux	GNU/Linux
Compiler Toolkit	Intel 10.0	Intel 9.0

ods for the parallelisation of Fortran DO-loops [21]. The basic three-dimensional wavefront problem executes over a data array of size $N_x \times N_y \times N_z$. The array is distributed over a two-dimensional processor array of size $m \times n$ giving each processor a 'column' of data of size $N_x/m \times N_y/n \times N_z$. The decomposition of this data is presented graphically in Figure 4. For discussion purposes it helps to consider the column of data as a stack of N_z 'tiles' each of size $N_x/m \times N_y/n \times 1$.

The wavefront algorithm proceeds by executing a series of *sweeps* through the data array. In the usual course of execution Chimaera executes 8 sweeps, one for each vertex of the three-dimensional data array. Note that this is not a strict requirement, the LU [27] code developed by NASA, which also employs the wavefront algorithm, requires only two sweeps to complete.

A sweep begins on a processor at a vertex of the processor array. Computation required for the first tile is completed and the boundary information is sent to the two downstream neighbouring processors. The originating processor then computes the second tile in its data stack, while the two neighbours compute their first tile. Following computation the processors communicate boundary information with their downstream neighbours. A sweep is complete once all processors in the data array have computed all tiles in their data stack. Figure 4 presents a sweep executing through the data array. Darkened grey cells have been solved in previous steps.

A full iteration of the wavefront algorithm is complete when all 8-sweeps in Chimaera have finished executing. For the standard input decks supplied during benchmarking, Chimaera executes 419 full iterations.

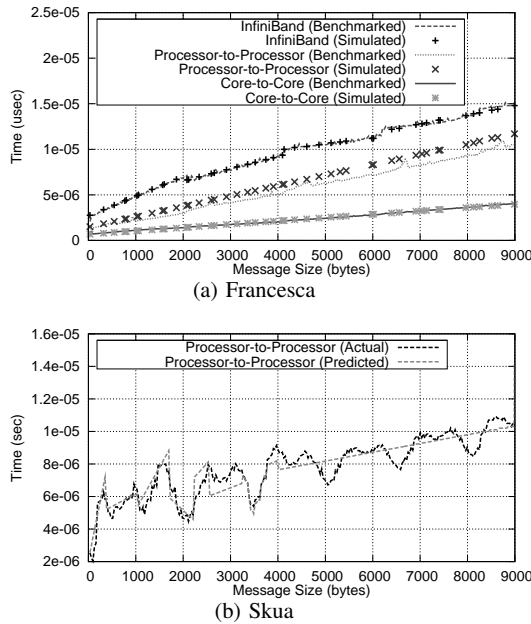


Figure 3: Benchmarked MPI Performance for Francesca and Skua (Intel MPI Benchmark 3.0)

4.2 Benchmark Machines

A simulation model for the Chimaera benchmark has been developed manually and evaluated on two supercomputers. The two machines used to obtain sample computation times as well as validations of the simulator’s accuracy are: (1) a recently installed 11.5 TFLOP/s IBM Intel Xeon InfiBand machine (*Francesca*) and (2) an older SGI Altix 3700BX2 (*Skua*). Both are production machines operated by the Centre for Scientific Computing at the University of Warwick. The specification for each machine is shown in Table 2. Note that as both machines are used for production runs at the University, job runtimes exhibit as much as 15% variance due to the inconsistent allocation of resources within the machine, background load and contention for resources arising from node sharing.

The networks used by both the Francesca and Skua machine have been benchmarked using the Intel MPI Benchmarking utility [17] version 3.0 in order to obtain a set of network profiles suitable for simulation. We note that several more advanced MPI benchmark utilities are available [13, 24] for complex network benchmarking, however, for our purposes the Intel benchmark is sufficient to obtain accurate point-to-point communication times which support the application modelling process. The results from this benchmarking are presented in Figures 3(a) and 3(b) respectively along side simulated results for the associated network models. As described in our earlier example of network modelling, the Francesca machine uses three inter-core communication networks - each of these are modelled as a separate profile for simulation. The InfiBand model, which is the key contributor to the accuracy of simulations for the Francesca machine, has a root mean squared error of 1.8×10^{-7} seconds. We note that the considerable variability of the NUMALink network used in Skua poses a difficult set of values for the network modelling approach used in this simulator, however, an RMSE of 6.97×10^{-7} seconds is also obtained demonstrating a good level of accuracy for such a volatile performance profile. We also note that in the region of interest for the Chimaera code (which

Table 3: Simulation Validations - Francesca Machine (Intel Xeon, InfiBand, OpenMPI 1.2.5, Intel Fortran 10 Compiler)

Core Count	Problem Size	Actual (secs)	Predicted (secs)	Error (secs)	Error (%)
4	60 ³	95.05	95.32	0.27	0.29
8	60 ³	50.18	50.58	0.40	0.80
16	60 ³	26.65	26.67	0.03	0.10
64	60 ³	9.00	9.63	0.64	7.08
128	60 ³	5.69	6.22	0.52	9.20
256	60 ³	3.86	3.96	0.10	2.56
32	120 ³	196.86	196.90	0.04	0.02
64	120 ³	56.72	55.22	1.50	2.64
128	120 ³	32.56	33.15	0.60	1.82
256	120 ³	18.64	19.47	0.83	4.44
128	240 ³	225.65	211.63	14.02	6.21
256	240 ³	129.65	118.59	11.06	8.53

Table 4: Simulation Validations - Skua Machine (SGI-Altix, NUMALink, Intel Fortan 9 Compiler)

Core Count	Problem Size	Actual (secs)	Predicted (secs)	Error (secs)	Error (%)
4	60 ³	345.78	324.38	21.40	6.19
8	60 ³	182.73	167.60	15.13	8.28
12	60 ³	126.19	113.73	12.45	9.87
16	60 ³	97.61	86.07	11.54	11.82
18	60 ³	86.68	78.36	8.32	9.59
20	60 ³	78.67	70.02	8.65	10.99
20	120 ³	605.85	531.21	74.63	12.32
24	120 ³	494.19	446.17	48.02	9.72
28	120 ³	433.64	402.39	29.25	6.75
32	120 ³	377.75	339.78	37.97	10.05

uses small messages) the model demonstrates exceptionally high levels of predictive accuracy as can be seen by close relationship between predicted and benchmarked values in Figure 3(b).

4.3 Model Validation

Tables 3 and 4 present validations of the simulation-based model of the AWE Chimaera benchmark for the Francesca and Skua machines respectively. The ‘actual’ runtime figure presented is averaged over 5 runs to ensure a representative runtime is provided for comparison to the prediction. Variations of up to 15% are seen in these runs which we attribute to machine noise, node sharing and background network load. The average error for both machines is less than 10% despite there being significant differences in machine architecture, reflecting the ability of the simulator to process simulations on a variety of hardware platforms. Note that the validations presented are the largest which are possible within the resource constraint policies used on the Francesca machine. Although it is difficult to predict whether similar error bounds will hold at considerably larger scale, the projection of larger runs based on accurate, validated smaller runs is common place in high performance computing modelling literature and serves to provide insight into potential performance at machine sizes which may not even be commercially available.

The computation times for both simulations are taken from a single run of the Chimera executable on 4 processing elements on each

Table 5: Simulation Validations - Cray XT3 and XT4 installations at AWE and ORNL (Chimaera 240³ Problem)

Core Count	Machine	Actual (secs)	Predicted (secs)	Error (%)
256	AWE XT3	333.54	309.92	7.08
1024	AWE XT3	100.01	92.16	7.91
1024	ORNL XT4	110.43	101.48	8.11
4096	ORNL XT4	54.13	48.18	10.99

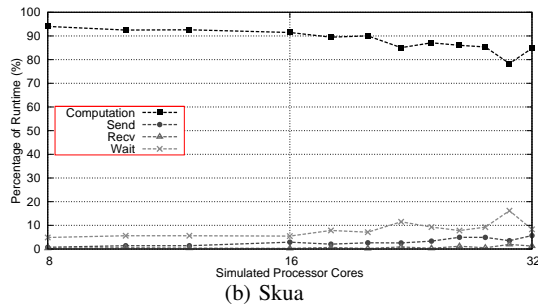
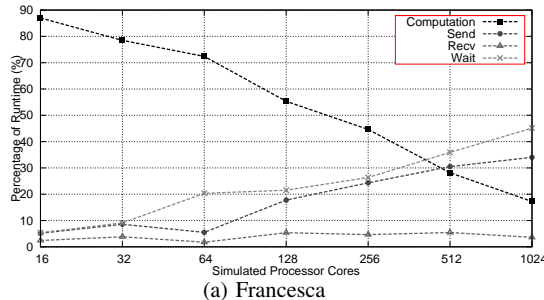


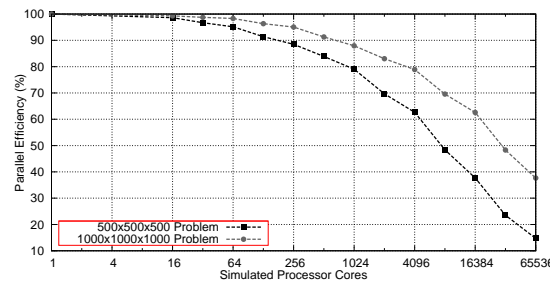
Figure 4: Runtime Breakdown for the Chimaera 60³ Problem

machine. The compute event times have then been extrapolated for larger processor configurations by developing a per-cell computation cost and then multiplying this by the number of cells a compute event represents. Similarly, the network model was constructed using only a single process on two nodes for Francesca and two processors for Skua. From these simple set of measurements we are able to construct each of the simulations shown. The reliance on so few measurements is of considerable advantage during procurement since sample machines, which are often only single blades or at most a group of blades, can be benchmarked and their performance at scale predicted, even if such a large machine does not currently exist.

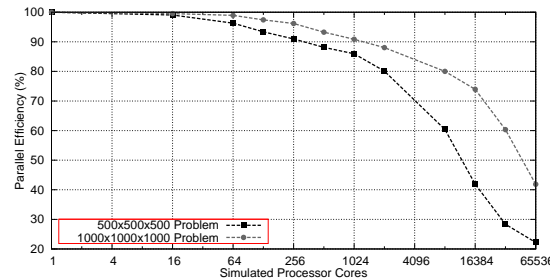
We are also able to show accuracy of the simulations at scale (Table 5). WARPP simulations of the Chimaera benchmark have been run at AWE and the Oak Ridge National Laboratory on Cray XT3 and XT4 systems respectively. For executions exceeding 4000 cores, the simulations demonstrate a greater than 90% accuracy. These results also correlate with those from a recent analytical study in [22].

4.4 Benchmark Behaviour Breakdown

Obtaining a breakdown of the performance of a parallel code is often a difficult challenge. The use of parallel profiling tools can provide limited insight, but such tools often perturb the runtime potentially limiting the applicability of the results. Analytic models also provide only limited insight since much of the complexity of



(a) Francesca



(b) Skua

Figure 5: Parallel Efficiency for the 500³ and 1000³ Chimaera Problems on Enlarged Francesca and Skua Machines

the code's operation is hidden in the highly abstracted mathematics of the performance model. For this reason, simulation-based models often provide more accurate insight into the behaviour of parallel codes, which in turn can support more accurate determination of code or machine bottlenecks and potential sources of optimisation.

Figures 4(a) and 4(b) present the proportion of runtime attributed to computation, network sends/receives and processor wait time on a processor at the centre of the processor array for Francesca and Skua respectively. Processors on the edge of the array will communicate less, resulting in a lower proportion of time in send or receive, because they have at least one neighbour missing.

In comparison both machines demonstrate similar behaviour up to 32 processors with computation accounting for between 80-90% of runtime. At scale, the results for the Francesca machine show that the decline in the proportion of time accounted for by computation is relatively smooth with the networking exhibiting slightly more volatile behaviour. We attribute this to the length of messages being communicated breaking over one of the 2048-byte boundaries where different performance characteristics exist.

4.5 Large Problem Size Inputs

The procurement of future systems is often oriented not only to improving the performance of existing codes and problem sizes but also to the execution of more complex or larger inputs. Our experience has been that it is common for procurement exercises to look at enlarging the computational capability of a system by two or three times in each subsequent purchase. Similarly, the increase in processor core counts which usually accompanies any new machine purchase is expected to be used efficiently - simply increasing the core counts assigned to applications can result in limited speed improvements yet consume much more of the machine's resources. Systems managers will therefore expect to see high levels of utilisation and efficient machine use. To this end we have used the existing simulation model and extended the input parameters extrapolating the existing computational costs associated with the

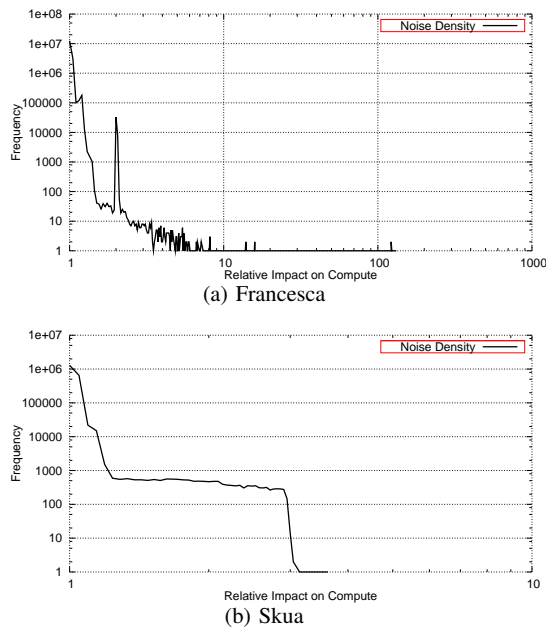


Figure 6: Relativized P-SNAP System Noise Profiles for Francesca and Skua when executing fixed-quanta equal to the computation of the Chimaera 60^3 Problem

240³ Chimaera problem for two larger inputs - 500³ and 1000³. To our knowledge Chimaera has never been run at such scale. The target of our analysis is the identification of the point at which the code breaches 50% parallel efficiency. This figure is targeted by computing sites such as AWE and LANL because it represents efficient use of the machine, trading further reductions in runtime for processor availability.

The result of our simulations is shown in Figure 5. As can be seen from the figure, the 500³ problem becomes less than 50% parallel efficient shortly before 8192 cores and the 1000³ problem shortly before 32768. Skua has marginally higher levels of parallel efficiency because the processors used are slower and therefore it takes longer for the network to dominate the execution. Note that Skua still has the slowest runtimes. By providing quantitative estimates of the point at which the code's efficiency falls below 50% we are able to inform procurement activities and provide user's with an upper limit on the number of processors which should be employed to comply with the 50% metric.

4.6 Impact of System Noise

System noise, also referred to as operating system jitter, arises from the use of background daemons and other user's processes [7]. It is common that commercial, more general purpose, operating systems such as Linux experience higher levels of noise than hand tuned or specialised light-weight kernel processes such as Cray Catamount. The effect of noise on a single individual processor is generally considered to be low, however, when all the processors executing a parallel application experience noise, the effect can be compounded by processors arriving late for communication or taking longer to compute than their neighbours.

In this final set of experiments we attempt to provide quantitative estimates of the impact of noise on Chimaera by injecting randomised noise into the simulation during execution. The random noise distribution is taken from an execution of the P-SNAP bench-

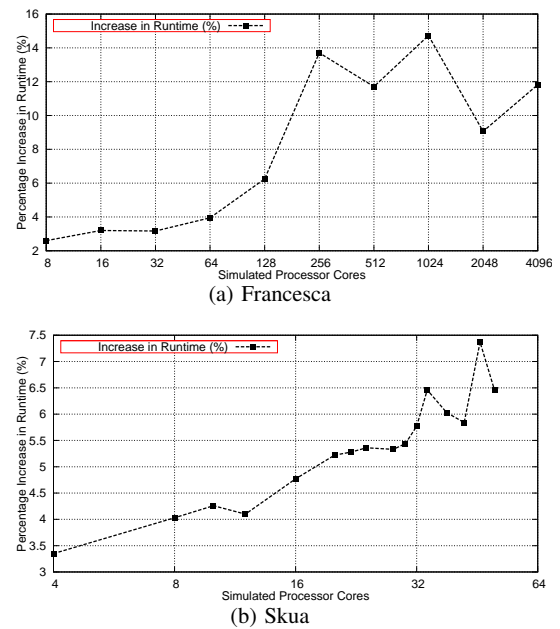


Figure 7: Percentage Increase in Predicted Runtime of Chimaera with Noise Profile Applied During Simulation

mark developed by the Los Alamos National Laboratory. P-SNAP executes a fixed quantum of work for which a specific time period of execution is expected, when the actual recorded execution time varies this is recorded. By analysing the distribution of multiple executions of the fixed-quantum a distribution of noise can be generated. Figures 6(a) and 6(b) present the results of this benchmark when executing fixed-quanta approximately equal to the computational work in Chimaera. The noise for the Skua machine is considerably smoother and less frequent than that observed on Francesca, which may be attributable to operating system tuning by SGI or fewer background daemons being executed on the system.

The impact of the application of noise during simulation is shown in Figures 7(a) and 7(b). For both machines the introduction of a noise profile to execution results in reasonable increases in runtime which are up to 15% for Francesca and 8% for Skua. We believe that these values give some indication of the variation a user might expect in the runtimes of their job - the 15% figure correlates with our own experiences from executions of Chimaera on the Francesca machine on configurations up to 256 processors. The increase in impact on runtime shown at higher processor counts also follows expectation - the more processors in a system the higher the probability that random noise will interfere with any single processor in the run.

We are currently developing specific network noise features within our simulator which allow for the simulation of networks which experience blocking or congestion. An open question remains - how best to efficiently identify the parameters to such a noise profile from benchmarking in a shared system.

4.7 Simulator Performance

In the introduction to this paper we commented on the observation that many existing simulation toolkits designed to replicate the behaviour of parallel scientific codes are simply unable to provide tractable execution times because of the reliance on individual instruction simulation. The problem is particularly acute when simu-

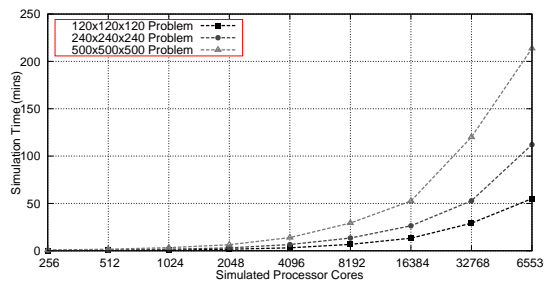


Figure 8: Time Required to Simulate Chimaera 240³, 500³ and 1000³ Problems

lating machines with tens of thousands of processor cores.

In Figure 8 we present the time required for our own simulator to simulate large problem inputs to Chimaera. These times were taken from execution of the simulation on a single Intel Xeon dual-core 2.33Ghz workstation with 8GB of memory. The Sun JDK 1.6 (64-bit) edition was used as the Java virtual machine. Approximately the same simulation time is required for both Francesca and Skua. All simulations for 16384 cores or less are completed in under an hour. Simulations for 1024 processors or less - the usual size of existing jobs at AWE and the University of Warwick - run in under a minute. All simulations require less than 2GB of system memory.

Whilst we believe that these times represent a clear advance for simulations of parallel codes we are still actively investigating methods to optimise and improve the performance of the simulator still further. Potential avenues for optimisation include parallelisation, since the existing simulator is entirely single threaded, and just-in-time/ahead-of-time optimisation of the simulation script prior to compilation in Java.

5. CONCLUSIONS

In this paper we introduce the WARwick Performance Prediction (WARPP) toolkit - a discrete event simulation-based suite of tools which have been designed to support the rapid and accurate modelling of parallel scientific codes. At the heart of the toolkit lies a newly designed discrete event simulator which executes C-style scripts to generate events representing the behaviour of a parallel application. The specific contributions of the simulator which are explored in this paper are:

- The use of coarse grained ‘basic-block’ computational modelling as opposed to individual instructions seen in existing work. The use of larger computational blocks, for which the times are recorded through instrumentation, allows for considerable increases in performance and scalability whilst remaining more flexible than the coarse-grained sections of code which result from trace-profiling;
- Support for complex models of machine networks. Network models can now be composed of multiple ‘profiles’ each defined by a set of ‘regions’. Each region itself represents a specific range in the message space. The topological map of the system is then relayed to the simulator through virtual processor pairs being linked to a profile. By permitting complex topologies to be created the simulator is able to provide accurate models of machines which employ multiple complex internal networks;
- Direct compilation of model scripts into Java bytecode allowing for a considerable improvement in simulation perfor-

mance;

- The ability to model machines containing thousands or tens of thousands of virtual processors. This represents a clear advantage of existing research, which is typically demonstrated on configurations of up to 1000 processor cores. We note that the simulator is able to deliver simulations for exceptionally large machine sizes within very acceptable execution time;
- Full recording of events and code behaviour information during runtime which is summarised to the user upon completion. The ability to analyse the results and output of the simulation in more detail is of significant advantage to users who wish to use the system as means to discover code or machine bottlenecks.

In the latter half of this paper we applied our toolkit to modelling the industrial strength Chimaera benchmark developed by the United Kingdom Atomic Weapons Establishment on two high performance computing architectures employed at the University of Warwick. This paper represents the first reported simulation of the Chimaera code. Model validations for both platforms were conducted on a variety of input sizes and processor configurations demonstrating accuracies which exceeded 90%. Similarly, network models developed for InfiniBand and NUMalink revealed RMSE errors of less than 1.8×10^{-7} and 6.97×10^{-7} respectively, despite complex multi-profile multi-region models having to be employed. Each of these simulations and network models was developed with only a small number of benchmarks being required - in the case of Francesca only two nodes were used and for Skua only two processors - which is of clear benefit during procurement when only limited resources are available for testing or benchmarking.

Following the development of an accurate simulation we were able to further explore the performance of Chimaera for existing and enlarged machine configurations and problem sizes. We highlighted the parallel efficiency of the code at a problem size which has never before to our knowledge been executed. These results are of utility to system designers, code developers and procurement managers. As well as providing key insights into the breakdown of the code’s runtime for two platforms, we were able estimate the runtime variance of the code in the context of machine noise - the maximum impact of which is estimated to be 15% when executing on the Intel Xeon-based Francesca machine.

The toolkit presented in this paper is actively being developed to provide rapid, accurate and where possible automated generation of runtime predictions. Many of these tools are still in the process of being prototyped and extended in our collaborative work with academic and industrial partners. The specific results presented in this paper, demonstrate that the simulator, which lies at the very centre of the toolkit and WARPP modelling process, offers accurate and reliable predictions of code behaviour in compact timeframes. The ability to develop accurate models rapidly, as well as to simulate these quickly, represents a significant improvement over existing simulation-based performance modelling toolkits which cannot offer the scalability or necessary features to support the demands of modern application performance modelling.

Acknowledgements

Access to the Francesca and Skua machines was provided by the Centre for Scientific Computing at the University of Warwick with support from the Science Research Investment Fund.

Access to the Chimaera benchmark and the development of the

WARPP toolkit are part-funded by the United Kingdom Atomic Weapons Establishment (AWE) under grants CDK0660 (*The Production of Predictive Models for Future Computing Requirements*) and CDK0724 (*AWE Technical Outreach Programme*).

Tool Availability

The WARPP Simulator and a set of exemplar simulation models are available as a free download from the High Performance Systems Group at the University of Warwick. The accompanying code analysis and instrumentation tools are currently in the process of being extended and re-developed prior to being made available.

6. REFERENCES

- [1] V. Adve, R. Bagrodia, J. Browne, E. Deelman, A. Dubeh, E. Houstis, J. Rice, R. Sakellariou, D. Sundaram-Stukel, P. Teller, and M. Vernon. POEMS: End-to-End Performance Design of Large Parallel Adaptive Computational Systems. *Software Engineering*, 26(11):1027–1048, 2000.
- [2] A. Alexandrov, M. Ionescu, K. Schauser, and C. Scheiman. LogGP: Incorporating Long Messages into the LogP Model for Parallel Computation. *Journal of Parallel and Distributed Computing*, 44(1):71–79, 1997.
- [3] E. Brewer, C. Dellarocas, A. Colbrook, and W. Weihl. PROTEUS: A High-Performance Parallel-Architecture Simulator. In *Measurement and Modeling of Computer Systems*, pages 247–248, 1992.
- [4] J. Cao, D. Kerbyson, E. Papaefstathiou, and G. Nudd. Performance Modelling of Parallel and Distributed Computing Using PACE. *IEEE International Performance Computing and Communications Conference, IPCCC-2000*, pages 485–492, Feb. 2000.
- [5] L. Chai, Q. Gao, and D. Panda. Understanding the Impact of Multi-Core Architecture in Cluster Computing: A Case Study with Intel Dual-Core System. *CCGRID 2007*, pages 471–478, May 2007.
- [6] D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a Realistic Model of Parallel Computation. In *Principles Practice of Parallel Programming*, pages 1–12, 1993.
- [7] P. De, R. Kothari, and V. Mann. Identifying Sources of Operating System Jitter Through Fine-Grained Kernel Instrumentation. In *IEEE Cluster*, 2007.
- [8] W. Denzel, J. Li, P. Walker, and Y. Jin. A framework for end-to-end simulation of high-performance computing systems, booktitle = Simutools '08: Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops. 2008.
- [9] M. Frank, A. Agarwal, and M. Vernon. LoPC: Modeling Contention in Parallel Algorithms. In *Principles Practice of Parallel Programming*, pages 276–287, 1997.
- [10] S. Girona and J. Labarta. Sensitivity of Performance Prediction of Message Passing Programs. In *Proc. International Conference on Parallel and Distributed Processing Techniques and Applications*, 1999.
- [11] G.R. Nudd, D.J. Kerbyson, E. Papaefstathiou, J.S. Harper, S.C. Perry and D.V. Wilcox. PACE: A Toolset for the Performance Prediction of Parallel and Distributed Systems. *The International Journal of High Performance Computing*, 4:228–251, 2000.
- [12] W. Gropp and E. Lusk. Reproducible Measurements of MPI Performance Characteristics. In *PVM/MPI*, pages 11–18, 1999.
- [13] D. Grove and P. Coddington. Precise MPI performance measurement using MPIBench. In *In Proceedings of HPC Asia*, 2001.
- [14] D. Grove and P. Coddington. Modeling Message-Passing Programs with a Performance Evaluating Virtual Parallel Machine. *Performance Evaluation*, 60(1-4):165 – 187, 2005. Performance Modeling and Evaluation of High-Performance Parallel and Distributed Systems.
- [15] S. Hammond, J. Smith, G. Mudalige, and S. Jarvis. Predictive Simulation of HPC Applications. *IEEE 23rd International Conference on Advanced Information Networking and Applications (AINA-09)*, May 2009.
- [16] A. Hoisie, G. Johnson, D. Kerbyson, M. Lang, and S. Pakin. A Performance Comparison through Benchmarking and Modeling of Three Leading Supercomputers: Blue Gene/L, Red Storm, and Purple. In *Proc. IEEE/ACM SuperComputing*, Tampa, FL, October 2006.
- [17] Intel Corp. MPI Benchmark Utility. 2008.
- [18] D. Kerbyson, A. Hoisie, and H. Wasserman. A Comparison between the Earth Simulator and AlphaServer Systems using Predictive Application Performance Models. *Computer Architecture News (ACM)*, December 2002.
- [19] D. Kerbyson, A. Hoisie, and H. Wasserman. Use of Predictive Performance Modeling During Large-Scale System Installation. In *Proceedings of PACT-SPDSEC02*, Charlottesville, VA., August 2002.
- [20] J. Labarta, S. Girona, and T. Cortes. Analysing Scheduling Policies using DIMEMAS. *Parallel Computing*, 23(1), April 1997.
- [21] L. Lamport. The Parallel Execution of DO Loops. *Commun. ACM*, 17(2):83–93, 1974.
- [22] G. Mudalige, M. Vernon, and S. Jarvis. A Plug and Play Model for Wavefront Computation. In *International Parallel and Distributed Processing Symposium 2008 (IPDPS'08)*, Miami, Florida, USA, 2008.
- [23] S. Reinhardt, M. Hill, and J. Larus. The Wisconsin Wind Tunnel: Virtual Prototyping of Parallel Computers. In *Measurement and Modeling of Computer Systems*, pages 48–60, 1993.
- [24] R. Reussner, P. Sanders, and M. Muller. SKaMPI: A detailed, accurate MPI benchmark. In *Recent advances in Parallel Virtual Machine and Message Passing Interface*, pages 52–59. Springer, 1998.
- [25] H. Shan and J. Shalf. Using IOR to Analyze the I/O Performance for HPC Platforms. Technical Report LBNL-62647, Lawrence Berkeley National Laboratory, June 2007.
- [26] D. Spooner, S. Jarvis, J. Cao, S. Saini, and G. R. Nudd. Local Grid Scheduling Techniques using Performance Prediction. *IEE Proc. Comp. Digit. Tech., Nice, France*, 150(2):87–96, April 2003.
- [27] M. Yarrow and R. V. der Wijngaart. Communication Improvement for the LU NAS Parallel Benchmark: A Model for Efficient Parallel Relaxation Schemes. Technical Report NAS- 97-032, NASA Ames Research Center, November 1997.