Predictive Simulation of HPC Applications

S.D. Hammond, J.A. Smith, G.R. Mudalige, S.A. Jarvis
High Performance Systems Group
University of Warwick
Coventry, CV4 7AL, UK
{sdh, jas, esviz, saj}@dcs.warwick.ac.uk

Abstract

The architectures which support modern supercomputing machinery are as diverse today, as at any point during the last twenty years. The variety of processor core arrangements, threading strategies and the arrival of heterogeneous computation nodes are driving modern-day solutions to petaflop speeds. The increasing complexity of such systems, as well as codes written to take advantage of the new computational abilities, pose significant frustrations for existing techniques which aim to model and analyse the performance of such hardware and software.

In this paper we demonstrate the use of post-execution analysis on trace-based profiles to support the construction of simulation-based models. This involves combining the runtime capture of call-graph information with computational timings, which in turn allows representative models of code behaviour to be extracted. The main advantage of this technique is that it largely automates performance model development, a burden associated with existing techniques. We demonstrate the capabilities of our approach using both the NAS Parallel Benchmark suite and a real-world supercomputing benchmark developed by the United Kingdom Atomic Weapons Establishment. The resulting models, developed in less than two hours per code, have a good degree of predictive accuracy. We also show how one of these models can be used to explore the performance of the code on over 16,000 cores, demonstrating the scalability of our solution.

1 Introduction

Modern supercomputing features as diverse a set of architectures as at any point during the last twenty years. Contemporary machines range from single large shared-memory structures to multiple cabinets containing small nodes connected via optimised interconnects. At the node level, the multitude of processor architectures, core configurations and threading abilities helps to increase the complexity still further. On the horizon, the arrival of the IBM RoadRunner machine, as well as advances in specialised computational support from vendors such as ClearSpeed, NVIDIA [5] and AMD [1], look set to herald the arrival of heterogeneous architectures to the supercomputing mix.

In the face of this myriad of complex architectures, machine designers, scientific programmers and system procurement managers must design, optimise and administer intricate hardware and software installations with a view to reducing costs and improving performance. Developing an understanding of code and machine behaviour is therefore a vital prerequisite to achieving these goals. In practical terms, the modelling of code behaviour, and the insights obtained by evaluating the resulting models, have been shown in a number of studies to provide support for efficient scheduling [25], procurement, code optimisation [21] and post-installation machine verification [17].

Approaches to performance modelling span two broad categories - those based on analytical modelling and those based on simulation. Typically, analytical models are developed through the manual inspection of source code and subsequent development of mathematical formulae to represent the execution time of the application's critical path. For the vast majority of such studies, a modelling framework (e.g. LogP [6], LogGP [2], LoPC [9]) is employed to reduce the burden on the performance modeller. The reliance on manual code inspections, however, has been a substantial inhibitor to the scalability of the approach for large, modern scientific applications which may contain hundreds of thousands of lines of code.

Simulation-based performance modelling techniques, on the other hand, employ specialised simulation software to replicate code behaviour on a virtual representation of hardware. The benefit of this approach is the reduced burden on the performance modeller, since many existing simulators do not require the development of a hand-crafted model ahead of time. Instead, the application source code or even the executable itself is used by either the simulator software or pre-simulation analysers to construct a simulation model. Existing approaches include the Wisconsin Wind Tunnel [22], PROTEUS [4] and the PACE toolkit [13, 14, 25], which replicate code behaviour through the simulation of each individual program instruction. Whilst this approach has been largely successful in reducing the overheads associated with analytical model development, the execution time required to simulate each program instruction on the large core counts found in modern supercomputing machinery is a significant barrier to the simulation of large codes or even moderately large machine configurations.

In this paper we introduce the WARwick Performance Prediction (WARPP) toolkit, which has been designed to support the modelling of modern parallel scientific codes on large processor configurations. In order to alleviate the overheads associated with existing simulation techniques, the simulator makes use of coarse-grained computational timings, effectively aggregating multiple instructions into computational blocks; thereby reducing the execution time associated with simulation and increasing the scalability of the approach. To overcome the time consuming process of model development, we also introduce a new method for the construction of simulation models using the automated analysis of trace-based profiles.

We demonstrate the effectiveness of this toolkit on both an industry benchmark developed by the United Kingdom Atomic Weapons Establishment (AWE) and the widely referenced NAS Parallel Benchmark suite [7, 3, 24, 21, 26]. Each model presented in this paper has been constructed using the analysis tools contained within our toolkit, no manual user intervention has been necessary. Furthermore, each model has been built, and its simulations run, in under two hours. We believe that this represents a substantial improvement on existing model generation strategies and offers a technique that considerably lowers the knowledge barrier present in contemporary approaches. Specific contributions of this paper include:

- A new method for the creation of simulation-based performance models, which employs a lightweight trace-based profiler to obtain code call-graph and computation timings during runtime. The tools require minimal user input and no understanding of the code structure, problem domain or implementation languages is needed;

- Validation of these tools on both an industry benchmark developed by AWE and the NAS Parallel Benchmark suite, demonstrating the versatility of this approach to the modelling of a variety of scientific codes;

- An exploration of the performance of the NAS-LU benchmarking application on up to 16,384 processor cores, demonstrating the scalability of the technique.

The remainder of this paper is organised as follows: Section 2 describes related work; Section 3 documents our new simulation-based toolkit and Section 4 demonstrates its application to the NAS Parallel Benchmark suite, including extended analysis for NAS-LU (Section 4.2). An industry-strength case study is presented in Section 5, and the paper concludes in Section 6.

2 Related Work

Performance Modelling is principally charged with providing techniques to model and analyse code behaviour in relation to hardware performance.

A large collection of work has been based on analytical methods, which represent the critical path of a code using mathematical formulae, themselves often expressed within a modelling framework such as LogP [6], LogGP [2] or LoPC [9]. While effective, analytical approaches require considerable manual analysis of source code which, apart from being time consuming, often requires in-depth knowledge of the code domain as well as the implementation languages and communication paradigm being employed. Analytical studies are often limited therefore to core computational kernels as opposed to full-blown applications.

An alternative approach is to use tool support, including simulation. Examples of such systems include the Wisconsin Wind Tunnel [22], PROTEUS [4] and the PACE toolkit [13, 14, 25], a precursor to this work. In each of these systems the code being modelled is replayed instruction-by-instruction and the use of machine resources is recorded by the simulator. The more recent DIMEMAS project [10, 18] also uses traces of an application taken at runtime, effectively removing the dependence on per-instruction based simulation. The work presented in this paper differs from DIMEMAS in that a single compact and parameterisable model of application performance is produced from a single program trace; in so doing we permit later modification and fine-tuning, something which is generally not possible when using large trace files. Our work is also differentiated from DIMEMAS by its ability to scale to machines with tens to hundreds of thousands of processor cores - far beyond existing simulation toolkits which scale to tens or hundreds of processors at best.

Our work therefore represents a significant advance for simulation-based methods of parallel application analysis. We offer benefits in three main areas: (1) improvement in both simulator performance and scalability by making use of coarse-grained computational events (as opposed to instruction-based simulation), (2) automated model generation from trace profiles, thus providing user-editable and tunable simulation models and, (3) highly parameterisable models of network performance, which may include multi-

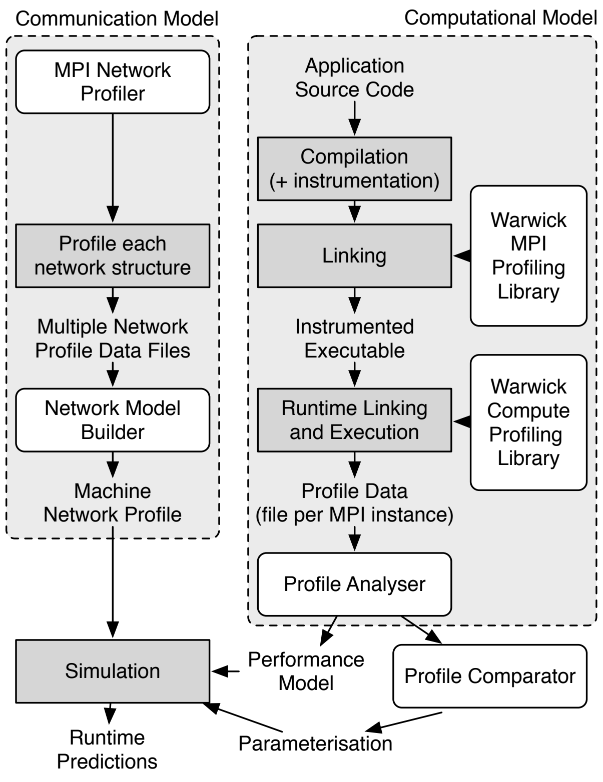


Figure 1. The WARPP Modelling Process

layered profiles relating to each interconnect employed by the machine.

3 The WARPP Toolkit

The WARwick Performance Prediction (WARPP) toolkit has been designed to support the rapid generation and simulation of performance models for large scientific codes executing on massively parallel processor machines (MPPs). These machines may contain multi-core, multi-processor nodes each exhibiting complex performance properties. Simulation models, which are encoded in a human readable C-like scripting language, can be developed by hand, by automated source code analysis, or from trace-based profiles. This paper focusses on the last of these techniques.

Developing a performance model using trace-based profiles comprises fives stages: (1) obtaining an instrumented version of the code either by injection of profiler calls into the source code, or profiler hooks added to object code during compilation; (2) communication profiling using the Warwick MPI Profiling Library - a PMPI-based profiler for MPI operations; (3) execution of the instrumented code together with a reliable MPI and/or filesystem benchmarking utilities on the target architecture (see [11, 12]); (4) post-execution profile analysis and model construction, and finally, (5) model simulation.

The simulator component, which is written entirely in Java to aid portability, uses a discrete event system to replicate the behaviour of parallel codes. The toolkit includes support for network transmissions (sends, receives), computation, file I/O and processor idle time. The smallest unit of computation granularity in WARPP differs significantly from existing toolkits. Instead of using single application instructions the toolkit uses coarser grained computational blocks, which significantly increases the scalability of the result. Run times for these computational blocks are obtained either through timer instrumentation of the code or through traditional profiling. An additional advantage of this coarser-grained approach is that hybrid models (combining analytical and simulation-based approaches) can be built, and evaluated within the same simulator.

The simulator also includes a significant new contribution in its support for the topological description and modelling of complex networks which may have ‘multiple layers of complexity.’ This allows the modelling of the high bandwidth, low latency, core-to-core bus found within a single processor, the inter-processor communication networks found within multi-processor nodes, and the lower bandwidth, higher latency interconnects such as InfiniBand or the Cray SeaStar. Although multiple network layers may be present in existing network simulators, they are not part of existing application performance simulators.

Each sub-network is described in the simulator as a set of latency, bandwidth pairs (a ‘profile’) mapped to a message-size space, with the topology of the system encoded as an assignment of MPI rank pairs to each profile. For example, a profile may contain the latency and bandwidth associated with message sizes of between 64 and 2048 bytes of an InfiniBand interconnect; an additional profile will contain details of larger messages *etc*. When the simulator processes a transmission from one virtual process to another, the respective network is identified within the simulator and its properties used to derive the time required. This approach is also able to support machines such as the IBM BlueGene which features multiple networks.

Note that the separation of computational timings, network topology description and performance models enables alternative machine configurations or runtime scenarios to be explored with only minor changes to a specific simulation input. In effect, simulation components are reusable between models, further reducing the development time associated with future studies.

3.1 Profiling Computation and Communication

The Warwick Computational Profiling Library is a dynamic library that is resolved immediately prior to execution. The

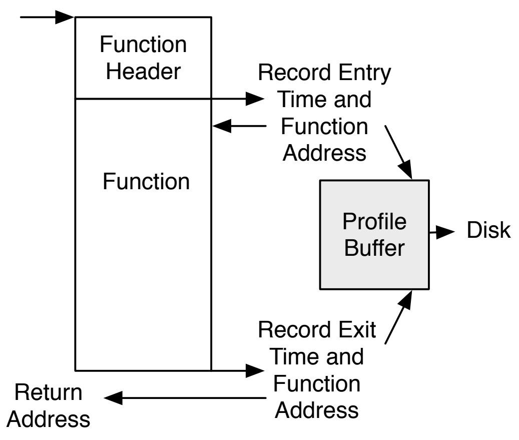

Function Header
Function
Record Entry Time and Function Address
Profile Buffer
Disk
Record Exit Time and Function Address
Return Address

Figure 2. The WARPP Computational Profiler

code exports a series of functions which enable it to be linked either to manually inserted profile statements or compiler instrumented hooks. When a profiling hook is encountered during execution the control passes from the application to the profiler, which then records the function address currently being executed, the current process time and whether the function is being entered or exited. Two timers are currently available - a wall clock timer, which will incorporate time spent in I/O or idle states, and a computational timer, which records only the actual processor time consumed by the application.

The profiler can be used with codes that have been compiled with either the GNU or Intel compiler toolsets with the `-finstrument-functions` compiler option enabled. During execution this has the effect of calling the profiler every time a function is entered or exited, thus, the data recorded implicitly encodes the call-graph of the application and the computation time spent in each function. To reduce the overheads associated with profiling a large code, a configurable buffer is employed to aggregate profiling data, preventing frequent, inefficient small disk writes which can cause significant pressure on network and I/O resources.

Recording the behaviour of MPI codes is essential for the development of parameterised communication models for simulation. The supporting Warwick MPI Profiling Library provides a masking layer for MPI functions, which is added to the application during linking. Each MPI function has a skeleton wrapper in the library. When called, the skeleton records the operation parameters (e.g. the size of transmission, datatype, communicator group) of the MPI operation along with the current process time, and then passes control to its PMPI [23] equivalent where the actual MPI transmission is carried out. This has the effect of intercepting calls to the system MPI libraries so that the recording of operations can be carried out. The MPI profiler also buffers recorded profile information to reduce the stress on filesystem I/O.

The output from a profiled execution implicitly contains the application call graph, since the profile data records when functions are entered or exited and the time the data is recorded. During profile analysis the data is read sequentially, and a record is made of which calls each function makes, and how much time is spent in each and between each child call. When this information is fully aggregated, a call graph is constructed in memory and annotated with compute times per call. At this point the analysis engine is able to compare the profile output from multiple nodes to check for inconsistencies; where these exist, call graph specialisations are made on a per MPI rank basis. The in-memory model of the call graph is then analysed for MPI operations. Where these are found the MPI parameters are located in the records taken by the MPI profiling library. Each MPI operation in the call-graph is then annotated with MPI specific details to enable parameterisation later in the modelling process. Finally the call-graph is serialised to disk in the form of a simulation model.

Wherever possible, the analysis engine will attempt to parameterise the number of calls made to child functions, the computation time spent in each function and any additional information such as MPI operation parameters. This enables the user to explore alternative application execution scenarios at a later date.

4 Modelling the NAS Parallel Benchmarks

The NAS Parallel Benchmark suite [3, 7] is a set of benchmarks devised by the NASA Ames Research Centre as a mechanism to examine and contrast the performance of supercomputing resources. The suite contains 8 benchmarks comprising of 5 kernels (EP, MG, CG, FT and IS) and three applications (BT, LU and SP), each of which can be executed in one of four problem input classes - A, B, C or D. In this paper we use version 2.4 of the benchmark and restrict our analysis to class B and class C data.

The target architecture is a 11.5 TFLOP/s IBM machine, comprising of 240 dual-Intel Xeon 5160 dual-core nodes each sharing 8GB of memory (giving 1.92TB in total). Each core supports the SSE and SSE2 instruction sets for floating point operations. Nodes are connected via a QLogic InfiniPath 4X, SDR (raw 10Gb/s, 8Gb/s data) QLE7140 host channel adapter (HCA) interconnected by a single 288-port Voltaire ISR 9288 switch. The processor core to HCA ratio is 4:1. Each compute node runs the 64-bit SUSE Linux Enterprise Server 10 operating system and has access to a 12TB IBM GPFS parallel file system. This machine is typical of commodity-based solutions currently adopted by medium-sized organisations such as universities, scientific research centres and engineering firms.

For our study the GNU gcc/gfortran version 4.1.2 toolset was employed in conjunction with OpenMPI 1.2.5 [8] and the PBS Pro scheduler. By default, jobs launched under

NAS Benchmark	PE Count (Cores)	Average Error (%) Class B and C
CG	16	20.11
CG	32	25.01
CG	64	13.91
EP	16	17.17
EP	32	22.27
EP	64	36.72
LU	16	18.1
LU	32	18.98
LU	64	16.3
MG	16	2.78
MG	32	20.02
MG	64	17.39
SP	16	3.49
SP	64	19.4

Table 1. Percentage Error in Simulation Prediction vs. Actual Runtime.

PBS are allocated ‘freely’ in the system - *i.e.* to any free core which meets the wall time or memory resources requested by the job, which can cause runtime variation of up to 15%. Nodes and processors are shared between jobs unless specifically requested during submission. For the experiments presented here we have queued jobs in the system to share resources, as this best reflects real world use. Runtimes presented are averaged over 3 successive runs to reduce the effect of a single ‘bad’ allocation of resources.

Three principle network interconnects are employed - a low latency core-to-core bus, an intra-node, processor-to-processor bus and finally the InfiniBand blade-to-blade network. The Intel MPI Benchmarking utility [12] is used to characterise each of these networks, and these are encoded as a ‘network profile’ for simulation. The network profiles themselves generate predictions with an accuracy exceeding 95%.

Runtime predictions obtained by simulating each NAS benchmark model are shown in Table 1. The predictive error averages 17.6% across all simulations. Note that the generation of each model was fully automated, no hand tuning of the models has been conducted and the entire process from initial profiled execution to completion of simulation required no longer than 2 hours per code.

4.1 Profiling Overhead

The very nature of code instrumentation and profiling means that the execution behaviour of code is changed from its unmodified/uninstrumented form. The addition of timer calls is not only an overhead at execution time, but it can also prevent some compiler optimisations such as function inlining, which in turn leads to further runtime perturbation.

Figure 3 shows the increase in executable size which

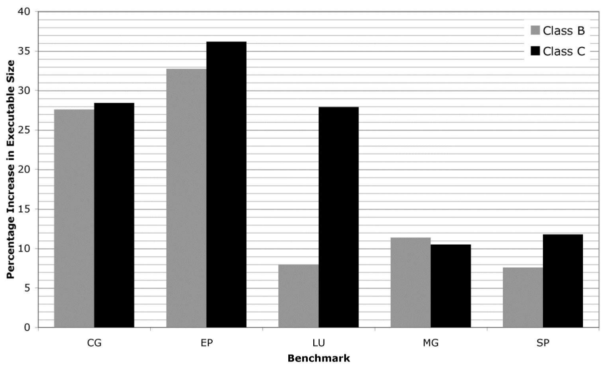


Figure 3. Percentage Increase in Executable Size with Instrumentation and MPI Profiling.

Benchmark	Class B		Class C
	16 PEs	32 PEs	64 PEs
CG	3.07%	0.67%	6.89%
EP	1.65%	5.00%	5.81%
LU	4.72%	14.22%	18.25%
MG	4.98%	1.63%	9.50%
SP	1.48%	N/A	4.88%

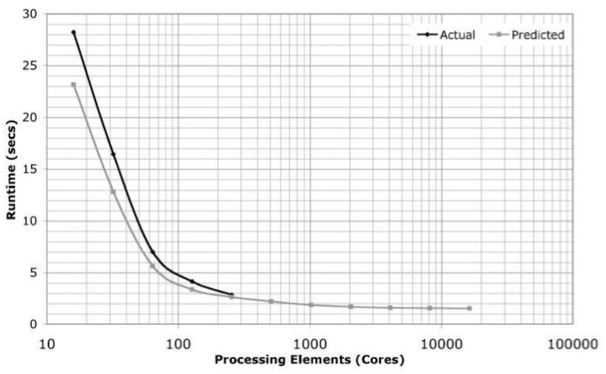
Table 2. Percentage Increase in Runtime when using Instrumented Code, MPI Profiling Library and Computational Profiling Module.

arises from using code instrumentation via the GNU compiler toolkit. The effect of this increase on runtime is shown in Table 2. Although there is a correlation between the increased size of the executable and the runtime perturbation, the change in executable size is not a clear determinant of by how much the runtime will increase. We believe that when the impact on runtime is larger, this derives from the repeated calling of small functions - in this scenario the profiler steps in frequently, upsetting the arrangement of caches and causing more context switches to obtain timer values. Smaller functions are more heavily affected because if the execution time is short, a higher proportion of time is spent using the less efficient cache reads whilst the cache is repopulated.

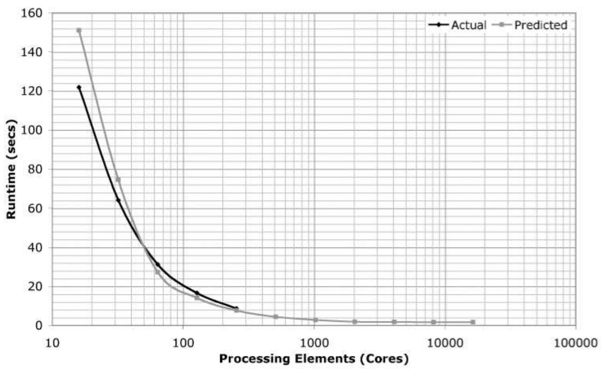
4.2 Extended Analysis of NAS-LU

In an extension to the initial model of NAS-LU we have use profiles obtained during 16, 32 and 64 PE runs to parameterise the model for larger processor configurations. The effect is to produce a factor by which the number of function calls and the time spent in each function changes as the PE count is increased. Figure 4 presents results obtained by simulating this parameterised model for machine configurations up to 16,384 cores.

The predictions obtained through our simulations



(a) NAS-LU Class B



(b) NAS-LU Class C

Figure 4. Predicted Runtimes for NAS-LU running on an enlarged Francesca-style machine.

demonstrate that for both data classes there is little improvement in scalability to be gained by increasing the processor core count beyond 1,024 cores, as at this core count the communication is the dominant component of runtime. Further increases to the PE count beyond this point require additional communication, which in turn reduces the parallel efficiency to less than the 50% value targeted by supercomputing centres such as AWE.

5 Modelling the 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 [16, 15, 20]. Unlike Sweep3D, however, the code employs a different internal sweep ordering, makes more frequent use of filesystem I/O and utilises a complex convergence criteria to decide when execution is complete. For an overview of the Chimaera application, we refer the reader to [21]. A more comprehensive discussion of the algorithmic basis for wavefront codes, including

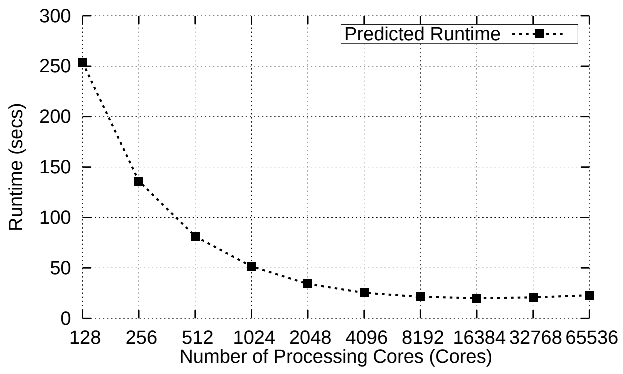


Figure 5. Projections of the Chimaera 240^3 Problem

Chimaera, Sweep3D and LU, can be found in Lamport's original description of the "hyperplane" method [19].

5.1 Predicting the Performance of the Chimaera Benchmark

Table 3 presents predictions for the time to solution of the Chimaera benchmark when run in multiple processor configurations of the Francesca machine. Note that the runtimes presented here are for the complete time to solution. The average predictive error overall is 14.34% with errors of 14.7% and 13.62% for the 60^3 and 120^3 problems respectively. A major advantage of the technique we report in this paper is the ability to refine the automatically generated models at a later date. The simulation errors obtained when simple hand tuning of simulation input parameters and control flow is applied is also shown in Table 3. Note that average predictions errors are now reduced to 7.63%. The results of further simulations for an enlarged machine and increased input size using the hand-tuned performance model are shown in Figure 5 demonstrating the ability of the technique to handle complex machine and code analysis at scale.

Table 4 presents the predicted breakdown of each of the Chimaera runs into proportion of runtime spent in computation and communication. For each, there is a large proportion of time spent in computation which comes from the significant time spent reading the benchmark problem data at startup - filesystem I/O and processor idle times for these runs are almost negligible. Whilst the actual computational code itself scales very well [21], also demonstrated by the smooth curve in Figure 5, the initial startup codes become increasingly more expensive as a proportion of runtime when the PE count is increased, hence the proportion of runtime spent in communication remains very small.

Problem Size	Processor-Core Count	Actual (sec)	Prediction (sec)	Prediction Error (sec)	Prediction Error (%)	Hand-Tuned Error (%)
60^3	16	30.32	24.53	5.79	19.10	0.10
60^3	32	19.29	21.47	2.18	11.28	3.58
60^3	64	15.67	18.10	2.53	16.28	14.03
60^3	128	13.57	15.22	1.65	12.16	9.20
120^3	128	49.76	61.86	12.10	24.32	11.41
120^3	256	41.23	42.43	1.20	2.91	7.50
240^3	128	225.65	253.79	28.14	11.09	10.48
240^3	256	129.65	136.16	6.51	4.78	4.78

Table 3. Predictions for Time to Solution of the AWE Chimaera Benchmark.

Problem Size	PEs	Comp (%)	Comm. (%)
60^3	16	97.39	2.61
60^3	32	96.13	3.87
60^3	64	96.10	3.90
60^3	128	95.40	4.70
120^3	128	96.96	3.04
120^3	256	95.56	4.44

Table 4. Predictions for Proportion of Time Spent in Communication and Computation for the AWE Chimaera Benchmark.

6 Conclusions

We present a new toolkit designed to support the automated development of simulation-based performance models. This toolkit includes new MPI and Computation Profiling libraries, a post-execution profile analysis, an automated model constructor and an efficient, discrete event simulator. Several innovative methods are employed, including:

- Coarse-grained computation event modelling, which alleviates the significant execution time associated with simulating individual instructions on large processor/core configurations;
- Support for multiple network profiles within individual simulations, enabling the modelling of machines which employ multi-core processors, multi-processor nodes, and complex network interconnects;
- Automated model construction from post-execution profile analysis, reducing the time and expertise required to construct a simulation-based performance model.

The effectiveness of this toolkit is validated through its application to the NAS Parallel Benchmark suite, and to the industry-strength benchmark code AWE Chimaera. Performance models for each code were constructed and evaluated within two hours of code installation, and in most cases the models exhibited a greater than 80% accuracy.

Two additional features of this modelling toolkit were demonstrated. First, the use of the models to speculate about the scaling behaviour of the code. Using the models it was possible to show at what point the NAS-LU and Chimaera benchmarks would fail to realize a parallel efficiency of 50%, despite having only run the codes on much smaller processor configurations. Second, it was shown how hand-tuning of the models could generate runtime predictions with an accuracy exceeding 90%. Despite the models being based on profile-derived traces, the abstract model which is extracted from these traces is highly configurable.

We have already deployed this toolkit in an industry setting, and our experience to date suggests that the toolkit provides a platform for faster model generation, requires less expertise in terms of the application and its supporting programming language(s), requires no additional knowledge of the supporting abstract programming models (e.g. LogGP), and provides simulations for a wide range of application and processor configurations very rapidly. Hand-tuning the models clearly brings increased levels of predictive accuracy, but the basic models produced by this toolkit provide an excellent starting point for further model development and refinement.

Acknowledgements

We acknowledge machine support from the Centre for Scientific Computing at the University of Warwick and the associated Science Research Investment Fund. This work is funded in part by the Atomic Weapons Establishment (AWE) under grants CDK0660 (*The Production of Predictive Models for Future Computing Requirements*) and CDK0724 (*AWE Technical Outreach Programme*).

References

[1] Advanced Micro Devices Corporation. AMD Stream Computing: Software Stack. 2007.

[2] A. Alexandrov, M.F. Ionescu, K.E. 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] D.H. Bailey, L. Dagum, E. Barszcz, and H.D. Simon. NAS Parallel Benchmark Results. In *Supercomputing*, pages 386–393, 1992.

[4] E.A. Brewer, C. Dellarocas, A. Colbrook, and W.E. Weihl. PROTEUS: A High-Performance Parallel-Architecture Simulator. In *Measurement and Modeling of Computer Systems*, pages 247–248, 1992.

[5] NVIDIA Corporation. *NVIDIA CUDA: Compute Unified Device Architecture, Programming Guide 1.1*. November 2007.

[6] D.E. Culler, R.M. Karp, D.A. Patterson, A.Sahay, K.E. Schauser, E. Santos, R. Subramonian, and Thorsten von Eicken. LogP: Towards a Realistic Model of Parallel Computation. In *Principles Practice of Parallel Programming*, pages 1–12, 1993.

[7] D. H. Bailey, E. Barszcz, J. T. Barton *et al.* The NAS Parallel Benchmarks. *The International Journal of Supercomputer Applications*, 5(3), 1991.

[8] E. Gabriel, G.E. Fagg, G. Bosilca *et al.* Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, 2004.

[9] M. Frank, A. Agarwal, and M.K. 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 *in Proc. International Conference on Parallel and Distributed Processing Techniques and Applications*, 1999.

[11] W. Gropp and E.L. Lusk. Reproducible Measurements of MPI Performance Characteristics. In *PVM/MPI*, pages 11–18, 1999.

[12] Intel Corp. MPI Benchmark Utility. 2008.

[13] S.A. Jarvis, D.P. Spooner, H.N. Lim Choi Keung, J. Cao, S. Saini, and G.R. Nudd. Performance prediction and its use in parallel and distributed computing. *Future Generation Computer Systems*, 22(7):746–754, August 2006.

[14] D. J. Kerbyson, J. S. Harper, A. Craig, and G. R. Nudd. PACE: A Toolset to Investigate and Predict Performance in Parallel Systems. In *European Parallel Tools Meeting, ONERA, Paris*, October 1996.

[15] D.J. Kerbyson, A. Hoisie, and S.D. Pautz. Performance Modeling of Deterministic Transport Computations. In *Performance Analysis and Grid Computing*, October 2003.

[16] D.J. Kerbyson, A. Hoisie, and H.J. Wasserman. A Comparison between the Earth Simulator and AlphaServer Systems using Predictive Application Performance Models. *Computer Architecture News (ACM)*, December 2002.

[17] D.J. Kerbyson, A. Hoisie, and H.J. Wasserman. Use of Predictive Performance Modeling During Large-Scale System Installation. In *Proceedings of PACT-SPDSEC02*, Charlottesville, VA., August 2002.

[18] J. Labarta, S. Girona, and T. Cortes. Analysing Scheduling Policies using DIMEMAS. *Parallel Computing*, 23(1), April 1997.

[19] L. Lamport. The Parallel Execution of DO Loops. *Commun. ACM*, 17(2):83–93, 1974.

[20] G.R. Mudalige, S.A. Jarvis, D.P. Spooner, and G.R. Nudd. Predictive Performance Analysis of a Parallel Pipelined Synchronous Wavefront Application for Commodity Processor Cluster Systems. *Cluster 2006*.

[21] G.R. Mudalige, M.K. Vernon, and S.A. Jarvis. A Plug and Play Model for Wavefront Computation. In *International Parallel and Distributed Processing Symposium 2008 (IPDPS'08)*, Miami, Florida, USA, 2008.

[22] S. Mukherjee. Wisconsin Wind Tunnel II: A Fast and Portable Parallel Architecture Simulator. In *Workshop on Performance Analysis and Its Impact on Design*, June 1997.

[23] P.S. Pacheco. *Parallel Programming with MPI*, page 271. Morgan Kaufmann, 1997.

[24] D.M. Pressel. Results from Measuring the Performance of the NAS Benchmarks on the Current Generation of Parallel Computers and Observations Drawn from these Measurements. *Users Group Conference (DOD UGC04)*, June 2004.

[25] D. P. Spooner, S. A. 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.

[26] M. Yarrow and R Van der Wijngaart. Communication improvement for the LU NAS Parallel Benchmark. Technical Report 97-032, NASA Ames Research Center, November 1997.