Survey of HPC Performance Modelling and Prediction Tools

Rob Allan
Computational Science and Engineering Department,
STFC Daresbury Laboratory, Daresbury, Warrington WA4 4AD

Alistair Mills
High Performance Systems Group,
Department of Computer Science,
University of Warwick, Coventry CV4 7AL

Contact e-Mail: r.j.allan@dl.ac.uk, alistair.mills@warwick.ac.uk

June 1, 2009

Abstract

Parallel and distributed computing systems are designed with performance in mind. Significant past research effort has been invested in the developing approaches for performance modelling and prediction of applications running on HPC systems. This report compares some tools for realistically modelling and predicting computer performance for a range of architectures and applications. We consider current activity worldwide which embraces a range of approaches from high level mathematical modelling to instruction level simulation. The different approaches are contrasted and three tools chosen for further evaluation.

Contents

1 Introduction

This report compares some tools for the realistic modelling and computer performance prediction of a range of HPC architectures and applications. Understanding the performance and cost implications of selecting a particular computing platform for a chosen application is difficult. There are, however, methods that allow alternative vendor systems to be compared before purchase, and the achieved performance of a system to be verified against reliable forecasts once it has been installed. These methods, based on application performance modelling, have tended to remain the preserve of a select group of performance experts.

There are many possible uses for performance modelling, and these have been discussed elsewhere, e.g. [31]. These applications can be arranged in five important categories as follows.

1. Procurement studies
    - Compare machines
    - Code mapping to machines
    - Upgrades to machines
    - Vendor result validation
2. Validation of installation
    - On delivery
    - Routine assurance
3. Design of future systems
    - Impact of future systems on workload
    - Influence the future offerings of vendors
4. Code optimisation and r-edesign
    - Re-engineering
    - Parameter selection
5. Workflow
    - Co-scheduling
    - Maximising scientific delivery
    - User work planning

There are a number of challenges facing the 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). In many cases, traditional computer benchmarking is insufficient to understand the performance of a large scientific application on such a system, as it typically requires access

to physical machines of equivalent (or similar) specification and rarely relates to the "potential capability" of the 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. Modelling could therefore be used as a key tool in the software design phase of computational science.

Some background information can also be found in other reports [6, 38]. We note that there are a larger number of tools used for network performance prediction. Many of these tools also use discrete event simulation [19] and may be worth reviewing.

1.1 Methodology

Application performance modelling, that is, assessing application and 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 and code optimisation, efficient scheduling, post-installation performance verification, and the procurement of systems.

The process of modelling itself can be generalised to three basic approaches:

1. Modelling based on analytic (mathematical) methods, e.g. LogP [16], LogGP [3], LogPC [29];
2. Modelling based on tool support and simulation, e.g. PACE [35, 12], POSE [41, 42], DiMeMaS [20, 33] and WARPP [21];
3. A hybrid approach which uses elements of both, e.g. POEMS [1] and Performance Prophet [37].

Modelling based on tool support, as proposed here, has a number of advantages over its more mathematical counterpart: (i) it is often based on (source) code anaysis, which absolves the user from translating lengthy programmatic features into abstract analytical program models; (ii) 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; and (iii) 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 features.

There are three different approaches to model construction:

1. Hand coded simulation script from "expert" code analysis;
2. Automated script generation from static source code analysis;
3. Hand coded or automated script generation from post-execution trace analysis.

Hammond et al. [21] noted that method (1) provides the most accurate performance models, especially when the abstract model is built from execution profiles of fairly small blocks of code, similar to a

"basic block" which would be considered by a compiler (e.g. a loop body or a sequence of statements between function calls). There is a trade-off between accuracy (smaller block sizes) and complexity of model leading to long simulation times. Instruction level simulators as used in chip design were found to be infeasible for large scale scientific applications for this reason. In our previous work we used method (3), but some of the models were adjusted with "expert" input from the application developers.

2 CCLRC Methodology c.2002

The following discussion is taken from [6]. The methodology adopted by Rob Allan, Martyn Guest and Paul Sherwood at Daresbury Laboratory used a method of hand crafted model building using information from run time traces, mathematical models and other fitting procedures to capture scalability with problem size and processing element count.

2.1 Code Analysis using VAMPIR

Our code analysis tool of choice in 2002 was VAMPIR [48]. Alternatives identified in a related report [5] included Paragraph, Apprentice and VT. To collect the required information it would also be possible to use simpler non-visual event tracing. All these methods were time consuming and required repeated runs of the codes on large problem sizes with large numbers of nodes. As noted below, the traces obtained are notoriously hard to analyse. Such tools are nevertheless useful as part of the code optimisation process because they can help to identify bottlnecks and inconsistent run time behaviour.

VAMPIR (Visualisation and Analysis of MPI Resources) is a commercial post-mortem trace visualisation tool [48]. It uses profiling extensions to MPI (see [5]) and permits analysis of message events during parallel execution. An event trace file is produced using the VAMPIRTrace library. Event ordering, message lengths and times can then be analysed viaually via the VAMPIR X11 interface.

Typical steps in code analysis are:

1. Instrumentation – trace calls are added around time critical program sections to start and stop the trace and at user defied event flags;
2. Amdahl's Law fitting may be carried out for cpu phases if a detailed complexity analysis is not available. Other more sophisticated analytic fits to measured data might be used [22];
3. Message lengths can be identified as a function of problem size and a fitting procedure used;
4. Use approximate models for MPI timings (latency/ bandwidth, logP etc. [22, 16];
    - However there is no treatment of contention, SMP etc. could not be modelled at this time;
    - Could replace fits with data from simulation, lookup tables etc.

2.2 Detailed Cost Models

Following VAMPIR characterisation and fits to the observed data, and as an alternative to simulation, detailed models were built of the time determining steps of each code. These had been identified partly from the traces and partly from a knowledge of the code itself. Typically the steps taken in modelling and predicting parallel code performance were:

1. Identify scalability rules for code kernels. We did this mainly using VAMPIR as described above;
2. Identify scalability rules for computational parts of code depending on parameters which define the scientific problem;
3. These scalability rules should be based on reasonable architecture parameters;
4. Build an application module for a simulator encapsulating the above rules;
5. Build an module for the simulator encapsulating architecture parameters;
6. Combine the modules in a two tier structure which is used to predict the application performance for given problem and architecture parameters.

In our case the simulator was written as straightforward Fortran 90 code which accepted as input data the number of nodes, size of problem and distribution of data for the job etc. In the two tier construction, architecture specific information is isolated from application or algorithm specific information. The simplest case is if the computational and communication parts of the applications can be separated. If this is not the case, with asynchronous communications or i/o, it must be known at what point in the code the communications or i/o start and finish. The time taken is then the higher of the communications or i/o and computation time for that section of code. The use of BSP like "supersteps" was found useful in doing this [9]. Within the modelling methodology we attempted to identify "algorithm classes" which span a number of applications and can be given their own sub-models. These were reflected in the model API which had methods for each class, point to point communication and sequential computation being the simplest ones. The class methods each inherit the machine parameters and input data and yield a wall clock time.

Parameters for a "flat MPI" paradigm were used initially, but some predictions of tuned performance have also been attempted using "hypercube" algorithms incorporating SMP parameters for fine grained reduction operations. Values of parameters used in the architecture models thus reflected and distinguished the algorithm was used, e.g. memory intensive, point to point or collective. This basic data was obtained by benchmarking algorithm classes with these features and from discussion with vendors.

CPU parameters implicitly included information about cache, memory, page and TLB structure which were not included separately. We did not attempt to do this accurately as in some other work and in particular we did not consider machine state. However the use of algorithm classes enabled a reasonable account of the machine behaviour in typical conditions for the applications of interest. A similar approach was used by Burkhart et al. [11].

For either computational or message passing parts of the code, calls to methods in the model simulator yielded estimated elapsed times. Summation of these times yielded results that compared well to

measured benchmark performance. However we did not strive for ultimate accuracy, but rather focused on scalability and differentiating between different algorithmic approaches.

The application codes considered using the above methodology were ANGUS, CETEP, and DL_POLY [6].

3 Barcelona Supercomputing Centre

Jesús Labarta is head of the Computer Sciences Department at BSC, and Judit Giménez is leader of the Performance Tools Group. Her group is working on the design of tools to instrument, analyse and predict the behaviour of applications on large parallel systems. Flexibility, simplicity and the appropriate combination of qualitative and quantitative information are some of the issues considered in the design of these tools. Scalability and management of the high volumes of performance data are also two issues that need to be considered to handle long running applications that use hundreds or thousands of processors. The team is also working in the definition of methodologies and procedures that would simplify and facilitate the process of analysis and optimization. See `http://www.bsc.es/plantillaF.php?cat_id=52`.

The group currently supports the following.

- Paraver: A very powerful performance visualization and analysis tool based on traces that can be used to analyse any information that is expressed on its input trace format;
- DiMeMas: Simulation tool for the parametric analysis of the behaviour of message passing applications on a configurable parallel platform;
- Instrumentation packages: Set of programs and libraries to generate or translate Paraver and Dimemas traces. We have packages for instrumenting different programming models (MPI, OpenMP, mixed) under different platforms (IBM AIX, Linux, SGI IRIX, HP Alpha) and translators from IBM AIXtrace and LTT formats;
- Utilities: Set of small programs to process Paraver traces to cut, summarise, translate or accumulate the performance data. They can be used independently but they are also integrated within Paraver.

There seems to be little current information available about these tools, so the following notes are taken from our previous work [6].

3.1 DiMeMas

DiMeMas [45] is an execution simulation tool that was for some time commercially available from PALLAS GmbH. There appears to have been no active development of this tool since c.2005. It is however still in use at Barcelona Supercomputer Centre [13].

DiMeMas [20, 33] alleviates the instruction based simulation approach through replay of traces obtained during a run of the application. Evaluation of the context of different machine sizes is supported

through the re-generation of a trace, subject to the user's specification. This approach has been successfully demonstrated on machine sizes of up to 1,000 processing elements. 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 captured in a user editable abstract model. Editing such a complex structure is non-trivial and the initial creation requires that the code actually be written and run. 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 a feasible time.

3.2 CCLRC work using DiMeMas c.2002

The DiMeMas tool was designed to model MPI programs running on SMP and distributed memory architectures, using a VAMPIR or Paraver trace file produced on p processes to simulate performance on n SMP nodes having p/n processes each. DiMeMas simulation can be usedto add knowledge about what is contained in a trace file because paramenters can be changed to estimate:

1. Effect of overlapping computation and communication;
2. Effect of contention:
   - Number of links between nodes and network;
   - Number of messages on network;
3. Performance on different architectures:
   - Assume uniform cpu scaling;
   - Use known communication parameters;
   - Must have same total number of processors as measured data;
4. Simulation of MPI model on SMP architecture:
   - Intra vs. inter node communication parameters can be included;
   - Mapping of tasks to SMP nodes;
5. Modified algorithms for collective operations etc. can be tested

We perceived the advantages of the DiMeMas approach, as compared to qualitative latency/ bandwidth models, to be that input parameters can account for the overlapping of computation and communication and contention can be incorporated as arising from a limited number of links between nodes and the network. It can also include a limit to the number of messages active on the network at any time.

Differentiation between intra- and inter-node communication capabilities allows the simulation of MPI models on SMP architectures. It is possible to consider the effects of different mappings of tasks to SMP nodes, and of modified algorithms for collective communications, domain decomposition etc.

In our work, DiMeMas used the trace file produced by VAMPIRtrace which contains information produced using the MPI profiling interface. Since the trace file does not contain information about

the point to point message passing underlying collective operatins (a feature of the MPI profiling procedure) we had to build a separate set of kernels for these operations. The kernels comprised of point to point routines based on a binary spanning tree which could replace the usual MPI library collective operations. Performance of this was compared to the real MPI routines in the applications considered [6]. We noted that an accurate cpu timer or elapsed timer was needed, which was not then available on every platform.

Inherent disadvantages of DiMeMas (and other similar tools) are:

1. There is no extrapolation to different numbers of processors (nodes);
2. The simulation is for the same code parameters as the measured tracefile, because there is no built in information about the way the problem scales;
3. No scaling information for MPI collective operations;
4. The length of time required to collect the necessary trace files and amount of data to be handled;
5. Simulates MPI only and cannot predict effects of algorithm changes.

4 LANL: Los Alamos National Laboratory

The Performance and Architecture Laboratory (PAL) at Los Alamos National Laboratory uses a parametric approach developed by Darren Kerbyson, formerly at University of Warwick, which expresses the execution time of a application on a machine as a mathematical logPC model [29, 30, 32].

$T(S, M) = T_{comp}(S, M) + T_{comm}(S, M) + T_{mem}(S, M)$

where T_{comp} is the computation time, T_{comm} is the communication time, and T_{mem} is the time spent for memory contention within a multi-processor node. By expressing the performance of the whole program with a such a simple expression structural information, for instance the control flow of the program, is not preserved. This approach may therefore not be suitable for many cases of model based performance evaluation. Paremeters are obtained via measurements on a small system.

In the PAL approach, the development of a workload model is based on a detailed static analysis of the source code of the program, such as counting computation and communication operations. This yields the parameterised mathematical model but requires significant human effort. The machine is not modeled separately. Systems are characterised by a set of parameters such as number of processors per node, number of nodes, number of communication links per node, communication latency and bandwidth and sequential processing speed. These parameters serve as input for the model.

Kerbyson et al. [32] report performance prediction results for various systems with average error between 4% and 12%. In the PAL approach, the performance effects of the overlapping of computation and communication phases are not considered. Therefore, models that are built based on PAL may provide accurate results only for the class of programs where this does not significantly affect the overall program performance.

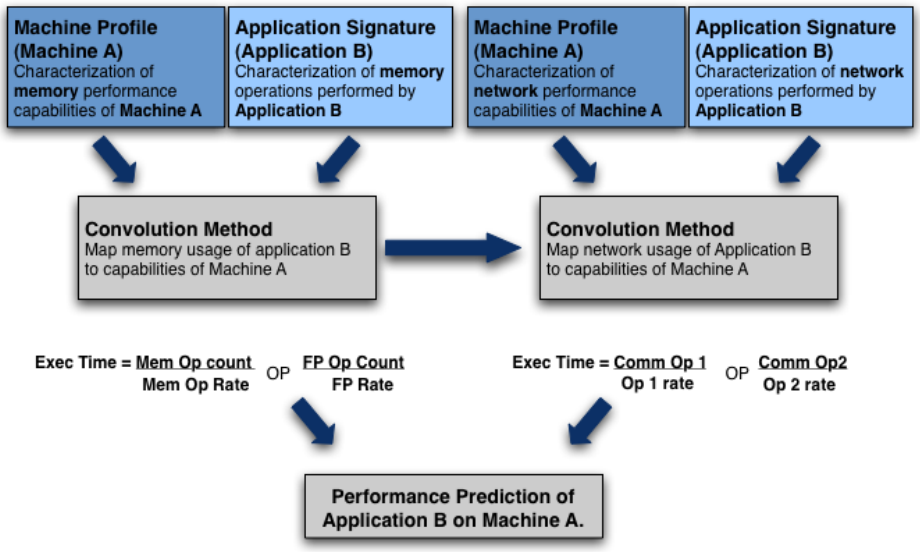

Figure 1: PMaC Framework Overview

The methodology has been used to study the SAGE application on ASCI (Accelerated Strategic Computing Initiative) Q system. More recently they have published papers on the performance of applications on Intel Nehalem processors, Sweep3d on IBM/ Sony Cell/BE, buffering in quad core Opteron and Cell/BE and the architecture and performance of the IBM Roadrunner system.

For more information on PAL see `http://www.ccs3.lanl.gov/pal`.

5 PMaC Precision Framework

The Performance Modelling and Characterization Group at San Diego Supercomputer Centre is led by Allan Snavely. For more information on PMaC see `http://www.sdsc.edu/pmac`.

The PMaC Prediction Framework is an implementation of an automated prediction model. In this context, a prediction model is a calculable expression that takes as parameters and attributes of application software and target machine hardware plus other factors to compute expected performance, in this case the expected runtime of the application.

The PMaC Framework operates by using the following three elements.

- Machine Profile – Characterization of the rates at which a machine can (or is projected to) carry out fundamental operations abstract from any particular application.
- Application Signature – Detailed summaries of the fundamental operations carried out by the application independent of any particular machine.
- Convolution Methods – Algebraic mappings of the Application Signatures on to the Machine profiles to arrive at a performance prediction.

In the PMaC approach, the machine profile is obtained from basic benchmarking of a system's fundamental parameters. In addition to CPU performance, performance of the memory sub-system is measured using PMaC's MultiMAPS benchmark on a single node, see `http://www.sdsc.edu/pmap/projects/mmaps.html`. MultiMAPS measures the bandwidth achieved by the machine while retrieving a variety of data sizes using different strides and line sizes. Network performance is measured by a simple MPI ping-pong benchmark run on 2 nodes of the system in various ways.

The application signature is principally based on memory and network operations. Information is obtained by instrumetation and tracing. PMaCinst is a binary re-writer that instruments memory and floating point operations. The memory address stream of those operations is processed on the fly through a cache simulator which is capable of simulating more than 20 different cache structures. The PSiNS tracer is a link time instrumentation tool which captures information about MPI calls. PSiNS combines the minimal features of MPIDTrace and IPM tools. MPI calls are traced using MPIDTrace which is part of the Dimemas project, see Section 3.1. MPIDTrace creates an event trace file, which is used by DiMeMas for simulation in one of the convolution steps.

Following this characterisation, two convolution steps are applied. The PMaC Convolver uses the trace results from PMaCinst and the memory profile data to produce an estimated time for the work done on the processor in between communication events on the target machine for that application. PSiNS simulator is similar to Dimemas and uses the estimated time from the PMaC convolver along with MPI call trace produced by different tools (PSiNS tracer, MPIDTrace) to simulate the execution of the application on the target system and emits the execution time as well as significant information on the decompostion of execution time to program components similar to the information given by IPM. DiMeMas can also be use the estimated time from the PMaC convolver along with the event trace produced by MPIDtrace to simulate the full execution time.

The PSiNS tracer and simulator can be obtained here `http://www.sdsc.edu/pmac/projects/psins.html`.

5.1 IPM: Integrated Performance Monitoring

IPM is a portable profiling infrastructure for parallel codes which binds together a number of lightweight instruments, see `http://www.sdsc.edu/pmac/projects/IPM.html`. In this way it can provide a low overhead performance summary of the computation and communication in an application. The amount of detail reported can be selected at run time via environment variables or through a MPI_Pcontrol interface. IPM has extremely low overhead, is scalable and easy to use requiring no source code modification.

The current version is in use at NSF, DOE, and DOD HPC centres in the USA. IPM has unique features that make it effective for ongoing monitoring of application performance by system administrators as well as application scientists.

IPM brings together several types of information important to developers and users of parallel HPC codes. The information is gathered in a way that tries to minimize the impact on the running code, maintaining a small fixed memory footprint and using minimal amounts of CPU. When the profile is generated the data from individual tasks is aggregated in a scalable way.

The monitors that IPM currently integrates are as follows.

- MPI – communication topology and statistics for each MPI call and buffer size;
- HPM – PAPI (many) or PMAPI (AIX) performance events;
- Memory – wallclock, user and system timings;
- Switch – Communication volume and packet loss.

NSF awarded a grant from 2007-2010 for further development of the IPM tools to provide: (1) a tool for capturing program's performance data with special emphasis on low overhead and scalability for up to millions of processors; (2) easy to understand application profiles which capture communication volumes and patterns, processor and memory system counter information, and topology-aware counters from network adapters and switches; (3) a database backend for workload characterisation and comparison of architectures; and (4) support for community driven enhancements through portable, extensible, open source software.

IPM can be obtained from `http://ipm-hpc.sourceforge.net/`.

6 POEMS: Performance Oriented End to end Modelling System

The aim of POEMS project [1] was to develop an environment for performance modelling of parallel computing systems. The methodology was to use multiple evaluation tools developed by a consortium of researchers from several universities in the USA. The project ran from 1997-2000 and, whilst included for completeness, will not be considered it further.

The system model was composed of component models. POEMS authors stated that each component of the system model could be evaluated by the corresponding evaluation tool – the output of a tool serving as input for the subsequent tool [1]. In general, the component models may be of different kinds and at different levels of abstraction. It is therefore difficult to see how the output of one evaluation tool may to be interpretable by the subsequent evaluation tool unless very careful design considerations are made.

POEMS devised a graphical "workflow" representation for parallel applications which is based on task graphs. Each node of the task graph may represent a set of parallel tasks. Edges of the task graph may represent data flow or task precedence. However, POEMS did not provide the corresponding tool support for graphical model composition. There was however an automatic task graph generator for High Performance Fortran (HPF) programs [2]. The tool support was provided by an extended version of the dHPF compiler [44].

The automatic development of a machine model was not adequately addressed. The authors claimed that the machine model may be automatically composed from existing components but not all the required components were developed. The processor and memory sub-systems were simulated with the SimpleScalar tool [10], the network is simulated based on the Parsec simulation language and the I/O subsystem was simulated with PIOSIM [7].

For more information on POEMS see `http://pages.cs.wisc.edu/~vernon/poems.publications.html`.

7 University of Illinois at Urbana Champaign

POSE: Parallel Object Oriented Simulation Environment [41] is a parallel discrete event simulator. POSE is used for simulation of the performance behaviour of applications that are executed on large scale machines such as IBM's BlueGene [42].

A detailed simulation of such large machines may require processing and memory resources that are not available on the desktop, therefore, such simulations are themselves executed on multi-processor systems. POSE is implemented on an existing object oriented parallel programming environment based on Charm++ [28], which is a parallel C++ library. The simulation entities of POSE are represented as Charm++ objects. Each object has a data member for tracking the simulation time and a set of methods for event handling. Typically, sequential discrete event simulators are about ten times faster than parallel discrete event simulators if a single processor machine is used for model evaluation. This is due to the synchronisation and communication overhead of parallel code. If a machine with more than ten processors is available for model evaluation, then the parallel simulator will perform best [41].

Based on POSE and Charm++, a specific simulator called BigSim was developed for the BlueGene-L [43, 42]. The machine is modelled as a set of inter-connected nodes, each node may having a set of processing elements. The effort for the development of a detailed machine model that supports execution driven simulation is high. Evaluating different applications may also be time consuming as the user has to restructure the source of the model.

It has not been possible to assess the efficiency of this approach since we were unable to find a comparison of prediction results of BlueGene-L simulator with measurement results on the real BlueGene-L machine.

It is possible to download Charm++ and BigSim from the Web site [43] along with examples of modelling the NAMD application.

8 Rice University

RSIM: Rice Simulator for ILP Multiprocessors, is a simulator of cache coherent non-uniform memory access (CC-NUMA) shared memory machines [47, 26]. A distinguishing feature of RSIM is the ability to simulate processors that use instruction level parallelism (ILP).

RSIM currently supports SPARC processors. Input for the RSIM simulator can be applications that are compiled and linked (i.e. executables) on SPARC/ Solaris systems. The application therefore has to exist and be compiled which makes this approach un-suitale for algorithm design. During the simulation, RSIM interprets the executable of the program and provides as output the number of executed cycles, and statistics on the utilization of components of the machine.

RSIM comprises a detailed cycle level machine model that allows the analysis of the performance

effects of architectural parameters. The main components of the machine model include processors, the memory hierarchy (L1 cache, L2 cache, local and remote memory), and the interconnection network. Because of this detailed level of simulation RSIM execution is several thousands times slower than the program execution on the real machine, so it is not suitable for evaluation of various designs of real world applications.

It may be possible to adapt the Rice simulator to use with CC-NUMA system on offer from vendors such as SGI, although its utility for large scale applications is questionable for the above reason.

9 SDSA; Science Driven System Architecture

This is a project of the SDSA Group at NERSC aimed at ensuring the appropriate procurement of future computer systems and efficient utilisation of existing ones. They use a combination of performance modelling based on a convolution of measured system and application charateristics and full workload analysis. The NERSC application workload is encapsulated in the Sustained System Performance (SSP) benchmark, see `http://www.nersc.gov/projects/SDSA/software/?benchmark=ssp`.

System information is gathered using IPM, see Section 5.

For more information on SDSA see `http://www.nersc.gov/projects/SDSA`.

10 University of Vienna

Performance Prophet (PP) [46, 37] is a tool for performance modelling and prediction of parallel and distributed computing systems. It provides a GUI, which simplifies the tasks of specification and modification of the performance model. The user specifies the model graphically using the Unified Modelling Language (UML) [36]. PP then automatically transforms the model from UML to C++ and evaluates it by simulation.

Because it is easy to change the model, PP is suitable for exploring a large set of possible application versions or numerical algorithms. This rapid evaluation capability of PP is due to a methodology that involves model simplification and the combination of mathematical modeling with discrete event simulation. The aim is to combine the model evaluation efficiency of mathematical performance models with the structure awareness of simulation models. The behaviour of the whole computing system is divided into action states and waiting states. Mathematical modelling is used for the performance behaviour of action states, whereas the performance behaviour of waiting states is simulated.

The machine model, in this case for clusters of SMPs, is composed automatically based on user specified parameters such as the number of nodes and the number of processors per node.

PP thus uses a hybrid approach to simplify a detailed model that combines simulation and analytical techniques to reduce the time needed to evaluate the model. Its authors [37] have assessed the accuracy by modelling and simulating a real world materials science application that comprises about 15,000 lines of code. The average prediction error was about 7%.

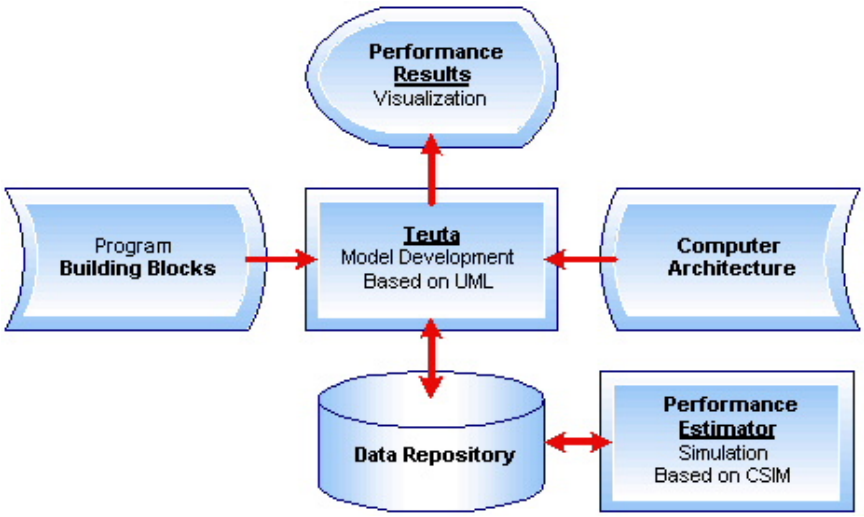

Figure 2: Performance Prophet Architecture

The Teuta graphical editor is used for the UML based modelling of parallel applications and Grid workflow applications in Performance Prophet.

At present Teuta supports the following types of UML diagrams: Activity, Collaboration, Deployment, and Class. Its features include:

- Model checking;
- Model traversing;
- Easy to extend with new types of UML diagrams and modeling elements;
- Generation of multiple model representations;
- XML based configuration (menus, toolbars, ..);
- Platform independent, requires Java programming language, J2SE 1.5.

A version of Teuta is distributed free of charge for educational and research purposes, see `http://www.par.univie.ac.at/project/prophet/node4.html`. This version is suitable for modelling parallel, but not Grid applications.

11 University of Warwick

11.1 PACE: Performance Analysis and Characterisation Environment

PACE [35, 12], developed by the High Performance Systems Group at the University of Warwick, is a performance prediction system that provides quantitative data concerning the performance of

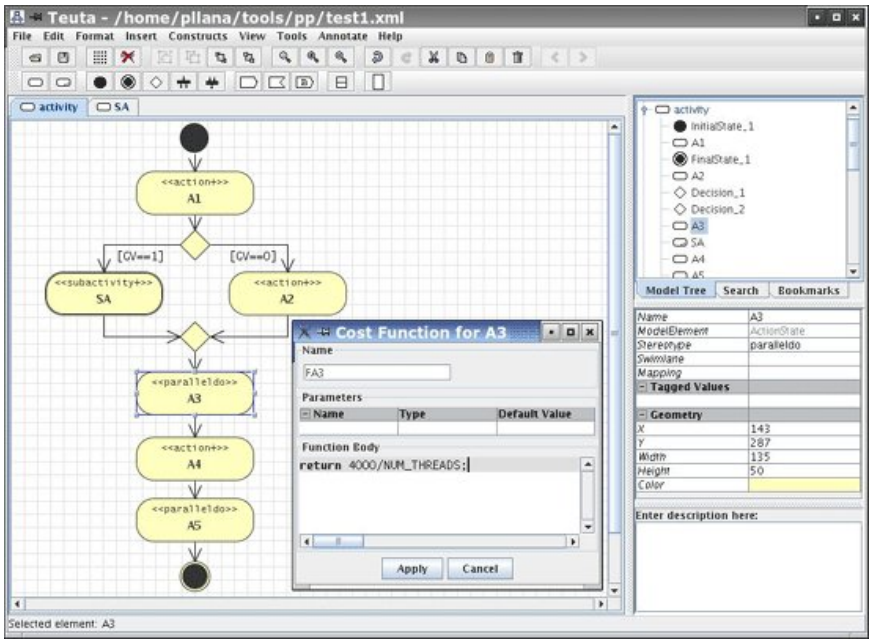

Figure 3: Teuta Graphical User Interface

applications running on high performance parallel and distributed computing systems. The system works by characterising the application and the underlying hardware on which the application is to be run, and combining the resulting models to derive predictive execution data. PACE claims to provide the capability for the rapid calculation of performance estimates without sacrificing performance accuracy. PACE offers a mechanism for evaluating performance scenarios, for example the scaling effect of increasing the number of processors, and the impact of modifying the mapping strategies (of process to processor) and underlying computational algorithms [**?**].

Details of the PACE toolkit can be seen in Fig. 1. An important feature of the design is that the application and resource modelling are separated and there are independent tools for each.

The PACE application tools provide a means to capture the performance aspects of an application and its parallelisation strategy. Static source code analysis forms the basis of this process, drawing on the control flow of the application, the frequency at which operations are performed, and the communication structure. The resulting performance specification language (PSL) scripts can be compiled to an application model. Although a large part of this process is automated, users can modify the performance scripts to account for data dependent parameters and also utilise previously generated scripts stored in an object library.

The capabilities of the available computing resources are modelled by the PACE resource tools. These tools use a hardware modelling and configuration language (HMCL) to define the performance of the underlying hardware. The resource tools also contain a number of benchmarking programs that allow the performance of the CPU, network and memory components of a variety of hardware platforms to be measured and modelled. The HMCL scripts thus provide a resource model for each hardware component in the system. These models are static and, once a model has been created for a particular hardware, it can be archived and reused.

Once the application and hardware models have been generated, they can be evaluated using the PACE evaluation engine. Using PACE the following studies can be performed: evaluation of time predictions for different systems, mapping strategies and algorithms; exploration of the scalability of the application and resources; predction of system resource usage, network, computation, idle time, etc.; and generation of predictive traces through the use of standard visualisation tools.

The capabilities of PACE have been validated using ASCI high performance demonstrator applications [**?**, **?**]. The toolkit provides a good level of predictive accuracy, with an approximate 5% average error, and the evaluation process typically completes in a matter of seconds on a desktop PC.

[Fig.1. An outline of the PACE system including the application and resource modelling componentsa nd the parametric evaluation engine which combines the two.]

PACE has been used in a number of other high performance settings; these include the performance optimisation of financial applications, real time performance analysis and application steering and the predictive performance and scalability modelling of the Sweep3D application. This work is different from previous research of the group, in that the data predicted by PACE is integrated in and applied to a dynamic workload steering environment. To enable such an application, new techniques have been devised that allow PACE performance data to be generated, published and queried in real time.

11.2 WARPP: Warwick Performance Prediction Toolkit

WARPP, the WARwick Performance Prediction Toolkit Simulator, is a prototype semi-automatic performance prediction environment which supports the exploration and analysis of an application's performance on machines consisting of thousands of processing elements. It was designed to support the automated generation of performance models [21].

WARPP builds upon the PACE toolkit described above and is now a Java application available for free download from `http://www2.warwick.ac.uk/fac/sci/dcs/people/research/csrcbc/research/wppt`. The tools have been used again to model the Sweep3D benchmark application, the NAS Parallel Benchmark Suite and several larger scientific MPI applications. WARPP employs discrete event simulation to model code runtimes and behaviour on large supercomputing resources containing tens of thousands of processors. The authors claim accuracies of greater than 90%.

The results obtained from the WARPP toolkit are currently being written up for a series of publications [21]. In addition, the Warwick group is developing new profiling tools as well as automated code analysis environments which will be integrated into the toolkit in the future.

11.2.1 WARPP Modelling Process

The WARPP toolkit combines work from multiple research projects and includes an automated code analyser, code instrumentation facilities, process/ wall time profilers and an aggressive out of order, discrete event simulator.

The WARPP modelling process features three distinct stages (shown in grey in the Figure 4). The first, source code instrumentation, is completed via automated code analysis tools. Each basic block within

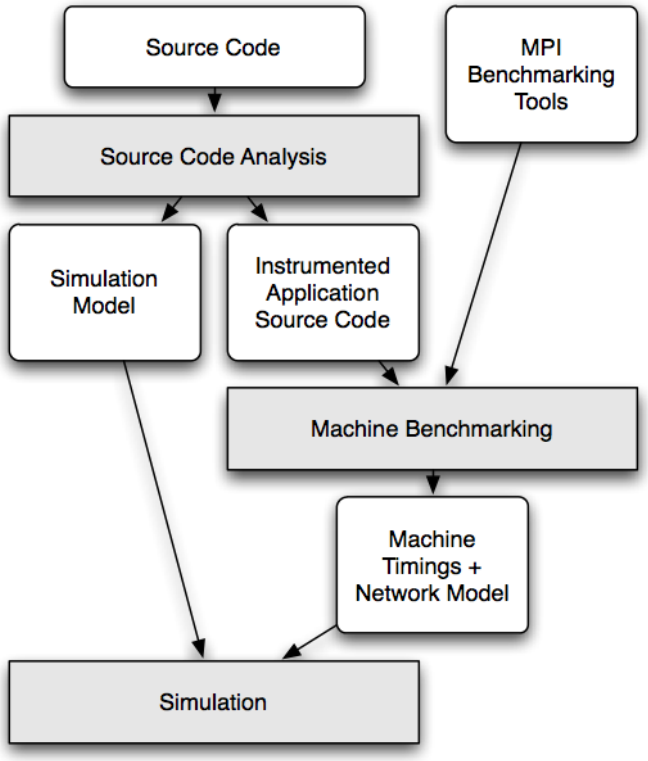

Figure 4: WARPP Modelling Process

the application source code is identified and instrumented with a start and stop timer for each entry and exit point respectively. The output of these timing statements forms the basis for simulating the computational time required by each basic block. The second stage, machine benchmarking, involves executing the instrumented version of the application along with a reliable MPI benchmarking utility (e.g. Intel MPI Benchmark Utility/ MPPTest) on the target architecture. A filesystem i/o benchmark might also be required. The output of these benchmarks which is a series of computational and communication timings are analysed to produce input to the simulator. The third stage simulates application control flow attributing respective timings to each basic block and i/o operations including MPI.

11.2.2 WARPP Simulator

Simulation in WARPP requires four inputs: (i) a simulation script capturing the control flow and event structure of the application; (ii) a set of "global" values which capture the respective timing of each computational event; (iii) the machine's network model; and (iv) an i/o model.

The discrete event simulator is written in Java to aid portability and ensure that results are repeatable between runs and installations. The simulator executes a set of "virtual processors", each being an abstract representation of a physical processing element. During execution of the model script, control swaps between one of the virtual processors and handlers in the simulator which process the events being generated. The virtual processor executes the control flow of the model, halting when an event

is reached and passing control back to the simulator. Special handlers are responsible for processing i/o and network communications. These check that both sender and receiver has posted event details (i.e. message size, tag, etc.) and that these agree. The communication time is computed from the message transmission time on the branch of network identified from the machine's topology map. The simulator can stall virtual processors for which the communication events are not ready, and records the waiting time.

12 Performance Analysis Tools

These tools analyse rather than model performance and are mostly based on execution traces. Most of this information was taken from the SciDac-3 PERI Web site `http://www.peri-scidac.org/perci/tools`. Many of these tools can be used during the application model building phase described above.

Paradyn and DynInst:

HPCToolkit: An open source suite of multi-platform tools for profile based performance analysis of applications. `http://www.hipersoft.rice.edu/hpctoolkit/`

IPM: Integrated Performance Monitoring, a low overhead infrastructure combining several instruments as described in Section 5. See `http://ipm-hpc.sourceforge.net/`

KOJAK: A trace based performance analysis tool for parallel applications supporting MPI, OpenMP, SHMEM, and combinations thereof. Includes instrumentation, post-processing of performance data and result presentation. `http://icl.cs.utk.edu/kojak/`

mpiP: A lightweight profiling library for MPI applications. `http://sourceforge.net/projects/mpip`

PAPI: Performance Application Programming Interface – provides a cross platform interface to the hardware performance counters found in most modern microprocessors. `http://icl.cs.utk.edu/papi/`

Paraver: A very powerful performance visualization and analysis tool based on traces that can be used to analyse any information that is expressed on its input trace format. `http://www.bsc.es/plantillaF.php?cat_id=52`

PDT: Program Database Toolkit – a framework for analysing source code written in several programming languages and for making rich program knowledge accessible to developers of static and dynamic analysis tools. `http://www.cs.uoregon.edu/research/pdt`

SvPablo: A graphical performance analysis environment for performance tuning and visualization. Supports both interactive and automatic source code instrumentation. `http://www.renci.org/projects/pablo.php`

TAU: A portable profiling and tracing toolkit for performance analysis of parallel programs written in Fortran, C, C++, Java and Python. `http://www.cs.uoregon.edu/research/tau`

VAMPIR: Visualisation and Analysis of MPI Resources – a commercial post-mortem trace visualisation tool. `http://www.vampir.eu`

13 Conclusions

Most approaches for performance modelling and prediction of parallel and distributed applications are currently of limited use to support performance oriented software engineering and computational science for the following reasons: (1) the use of a notation that is not based on widely accepted standards; and (2) the requirement that the software engineer has a thorough understanding of the underlying performance modelling technique. Some approaches aim to bridge this gap between the performance modeling and the software engineering by incorporating UML [36].

Few approaches are able to cope with anything more than small programs such as matrix vector multiplication. There are several reasons for this lack of scalability: (1) a very complex code analysis is used during the workload modelling that does not scale up to the size and complexity of real world applications; (2) a detailed machine model is used that is so slow that makes the simulation of such applications intractable; or (3) the simulator requires very large resources, typically memory, that may not be available. Some approaches have addressed this issue by using model simplification techniques, combination of mathematical modelling with discrete event simulation, using a simple machine simulation model or a parallelisation of the simulator.

Based on the above survey we have chosen the following tools for further evaluation: Performance Prophet; POSE; WARPP.

References

[1] V. Adve, R. Bagrodia, J. Browne, E. Deelman, A. Dubeb, 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 (2000) 1027-48

[2] V. Adve, R. Bagrodia, E. Deelman, T. Phan and R. Sakellariou *Compiler-Supported Simulation of Highly Scalable Parallel Applications* Proc. SC'99 (ACM, Portland, 1999)

[3] A. Alexandrov, M. Ionescu, K. Schauser and C. Scheiman *LogGP: Incorporating Long Messages into the LogP Model for Parallel Computation* J. Parallel and Distributed Computing 44:1 (1997) 71-9

[4] R. Aversa, B. di Martino, M. Rak, S. Venticinque and U. Villano *Performance Prediction through Simulation of a Hybrid MPI/ OpenMP Application* Parallel Computing 31 (2005) 1013-33

[5] R.J. Allan, R.R. Ward and M.C. Goodman *Survey of Parallel Performance Tools and Debuggers* Technical Report (CLRC Daresbury Laboratory, 1999)

[6] R.J. Allan, M.F. Guest, P. Sherwood, I.J. Bush, A.G. Sunderland, Y.F. Hu and K. Maguire *Parallel Application Software on High Performance Computers.*
Performance Modelling for Optimisation of Whole Codes and Dependencies on Parallel Computer Architectural Parameters Technical Report (UK HPCI Initiative, April 2002)

[7] R. Bagrodia, S. Docy and A. Kahn *Parallel Simulation of Parallel File Systems and I/O Programs* Proc. SuperComputing'97 (ACM, San Jose, 1997)

[8] R. Bagrodia, E. Deeljman, S. Docy, T. Phan *Performance Prediction of Large Parallel Applications using Parallel Simulations* ACM Sigplan Notices 34 (1999) 151-62

[9] Rob H. Bisseling *Parallel Scientific Computation: A Structured Approach using BSP and MPI* (Oxford University Press, March 2004) 324pp. ISBN 0-19-852939-2

[10] D. Burger and T. Austin *The SimpleScalar Tool Set, Version 2.0* Technical Report CS-TR-1997-1342 (University of Wisconsin-Madison, 1997)

[11] H. Burkhart et al. *Basel Algorithm Classification Scheme* Technical Report 93-3 (University of Basel, 1993)

[12] J. Cao, D. Kerbyson, E. Papaefstathiou and G.R. Nudd *Performance Modelling of Parallel and Distributed Computing using PACE* IPCCC-2000, IEEE Int. Performance Computing and Communications Conf. (Feb'2000) 485-92

[13] M. Casas, R.M. Badia and J. Labarta *Prediction of Behaviour of MPI Applications* IEEE Int. Conf. on Cluster Computing (2008) 242-51. 978-1-4224-2640-9

[14] M.J. Clement and M.J. Quinn *Automated Performance Prediction for Scalable Parallel Computing* Parallel Computing 23 (1997) 1405-20

[15] M.J. Clement and M.J. Quinn *Multivariate statistical techniques for parallel performance prediction* HICSS'95 p446

[16] 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 and Practice of Parallel Programming" (1993) 1-12

[17] A. Dunlop, E. Hernandez, O. Naim, T. Hey and D. Nicole *A Toolkit for Optimising Parallel Performance* Lecture Notes in Computer Science (Springer, 1995) vol 919, 548-53 ISBN 978-3-540-59393-5

[18] P. Fortier and H. Michel *Computer Systems Performance Evaluation and Prediction* (Digital Press, 2003) ISBN 1-55558-260-5

[19] R. Fujimoto *Parallel and Distributed Simulation Systems* (John Willey, 2000)

[20] S. Girona and J. Labarta *Sensitivity of Performance Prediction of Message Passing Programs* Proc. Int. Conf. on Parallel and Distributed Processing Techniques and Applications (1999)

[21] S.D. Hammond, G.R. Mudalige, J.A. Smith, S.A. Jarvis, J.A. Herdman and A. Vadgama *WARPP – a Toolkit for Simulating High-performance Parallel Scientific Codes* Proc. 2nd ACM Int. Conf. on Simulation Tools and Techniques (Rome, 2009) preprint

[22] R.W. Hockney *Computational Similarity* Concurrency: Practice and Experience 7 (1995) 147-66

[23] R.W. Hockney and E.A. Carmona *Comparison of Communications on the Intel iPSC/860 and Touchstone Delta* Parallel Computing 18 (1992) 1067-72.

[24] R.W. Hockney and C. Jesshope *Parallel Computers* and *Parallel Computers-2* (Adam Hilger 1981; 2nd edition IOPP 1988)

[25] R.W. Hockney *The Science of Computer Benchmarking* (SIAM, 1996)

[26] C. Hughes, V. Pai, P. Ranganathan and S. Adve *RSIM: Simulating Shared-Memory Multiprocessors with ILP Processors* IEEE Computer, 35:2 (February 2002) 40-9

[27] S.A. Jarvis, D.P. Spooner, H.N. Lim C. Keung, J. Cao, S. Saini and G.R. Nudd *Performance prediction and its use in parallel and distributed computing systems* Future Generation Computer Systems 22 (2006) 745-54

[28] L. Kale and S. Krishnan *Charm++: Parallel Programming with Message Driven Objects* In "Parallel Programming using C++" (MIT Press, 1996)

[29] D. Kerbyson, H. Alme, A. Hoisie, F. Petrini, H. Wasserman and M. Gittings *Predictive Performance and Scalability Modeling of a Large-scale Application* Proc ACM/ IEEE conference on Supercomputing SC'2001 (ACM, Denver, November 2001)

[30] D.J. Kerbyson, A. Hoisie and H.J. Wasserman *Use of Predictive Performance Modelling during large-scale System Installation* In Proc PACT-SPDSEC'02 (Charlottesville, August 2002)

[31] D.J. Kerbyson, A. Hoisie and H.J. Wasserman *Modelling the Performance of large scale Systems* Keynote paper, UK Performance Engineering Workshop (July 2003)

[32] D.J. Kerbyson, A. Hoisie and H.J. Wasserman *Use of Predictive Performance Modelling during large-scale System Installation* Parallel Processing Letters 15:4 (World Scientific, December 2005) 387-96

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

[34] D.J. Lilja *Measuring Computer Performence: A Practitioner's Guide* (CUP, 2000) 278pp ISBN 978-0521641050

[35] G.R. Nudd, D. Kerbyson, E. Papaefstathiou, S. Perry, J. Harper and D. Wilcox *PACE: A toolset for the performance prediction of parallel and distributed systems* Int. J. High Performance Computing Applications 14:3 (2000) 228-51.

[36] S. Pllana and T. Fahringer *UML Based Modeling of Performance Oriented Parallel and Distributed Applications* Proc. 2002 Winter Simulation Conference (IEEE, San Diego, December 2002)

[37] S. Pllana and T. Fahringer *Performance Prophet: A Performance Modeling and Prediction Tool for Parallel and Distributed Programs* Proc. 2005 Int. Conf. on Parallel Processing, ICPP-05. (IEEE, Oslo, June 2005) 509-16. DOI 10.1109/ICPPW.2005.72

[38] S. Pllana, I. Brandic and S. Benkner *Performance Modeling and Prediction of Parallel and Distributed Computing Systems: a Survey of the State of the Art* Proc. 1st Int. Conf. on Complex, Intelligent and Software Intensive Systems (IEEE, 2007) 0-7695-2823-6

[39] S. Prakash, A. Kahn, S. Docy *COMPASS: a Component-Based Parallel System Simulatror* (UCLA) `http://pcl.cs.ucla.edu/projects/mpisim/`

[40] Y. Wen and G.C. Fox *Performance Prediction for Large Scale Parallel Systems* PDPTA Conference 1999

[41] T. Wilmarth, G. Zheng, E. Bohm, Y. Mehta, N. Choudhury, P. Jagadishprasad and L. Kale *Performance Prediction using Simulation of Large-scale Interconnection Networks in POSE* Proc. 2005 Workshop on Principles of Advanced and Disctributed Simulation, PADS. (IEEE, Monterey, June 2005)

[42] G. Zheng, G. Kakulapati and L. Kale *BigSim: A Parallel Simulator for Performance Prediction of Extremely Large Parallel Machines* Proc. 18th Int. Parallel and Distributed Processing Symposium, IPDPS 2004 (IEEE, Santa Fe, April 2004)

[43] G. Zheng et al. *BigSim* `http://charm.cs.uiuc.edu/research/bluegene`

[44] dHPF Compiler. Department of Computer Science, Rice University, `http://www.cs.rice.edu/dsystem/dhpf`

[45] J. Giménez et al. *Dimemas: Performance Prediction for Message Passing Applications* `http://www.cepba.upc.edu`

[46] *Performance Prophet* University of Vienna `http://www.par.univie.ac.at/project/prophet`

[47] *RSIM: Rice Simulator for ILP Multiprocessors* (RSIM) `http://rsim.cs.uiuc.edu/rsim`

[48] *VAMPIR: Visualisation and Analysis of MPI Resources* `http://www.vampir.eu`