Design of a Performance Technology Infrastructure to Support the Construction of Responsive Software

E. Papaefstathiou

September 26, 2000

Technical Report
MSR-TR-2000-99

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

Design of a Performance Technology Infrastructure to Support the Construction of Responsive Software

E. Papaefstathiou
Microsoft Research Limited
St. George House, 1 Guildhall Street
Cambridge CB2 3NH, UK
+44 (1223) 744744
efp@microsoft.com

ABSTRACT
The construction of software that meets its performance objectives is a challenging task considering the distributed nature of modern architectures. Although substantial progress is being made in areas of performance technology, a breakthrough in the wider use of prediction tools during the software development process still remains to be seen. Lack of integration and standardization is one of the reasons for the slow adoption of the performance technology. This paper presents the design of a Performance Technology Infrastructure (PTI), an integration environment for hardware models, workload descriptions, and performance analysis tools. PTI includes a workload specification library, a model evaluation engine, and an interface to external hardware models. PTI components interact with XML scripts based on the syntax of predefined schemas that can be extended to include the requirements of new components incorporated into the system. A prototype implementation of a performance prediction tool is also introduced. It includes third party hardware models, a performance specification language, and an analysis tool. Performance predictions for the Sweep3D application running on a PC cluster are obtained and analyzed to demonstrate the tool's capabilities.

Keywords
Performance prediction and analysis, infrastructure, parallel/distributed systems, responsive software.

1. Introduction
Software development is in the process of a paradigm shift from product-oriented to service oriented markets. The web and the wider adoption of distributed architectures promoted a greater focus to some aspects of software features such as maintainability, scalability, and availability. Performance is for these systems a central issue and therefore sophisticated performance technology is required to guide the software development process.

.Predicting software performance is a difficult task that requires considering the complex nature of both software and hardware. Only a small number of tools and techniques are currently available that model realistic workloads [4, 12]. However, none of these tools are used in the development of commercial software. The only exception is Software Performance Engineering [17, 18], an emerging discipline that introduces the incorporation of performance studies into the software development process. Software houses consider performance issues after the source code implementation, when monitoring tools can be employed to measure and analyze possible bottlenecks (e.g. [19]).

Academia has been studying system performance for many decades and a multitude of performance models have been developed for hardware components (e.g. [6]). The majority of these models use different evaluation approaches and have diverse workload requirements. Workload definitions have been extended from statistical based to detailed descriptions that can capture all the aspects of application control flow and interactions. Performance specification languages provide formalism for defining the application behavior in varying level of abstractions (e.g. [13]). These languages can be used to prototype the performance of an application or represent the performance characteristics of the source code in detail. Monitoring and predictive tools generate detailed outputs that can further be processed by advanced visualization tools to allow the identification of bottlenecks and other performance characteristics (e.g. [16]).

Although aspects of performance technology are continuously advancing, a breakthrough in its wider use in the software development still remains to be seen. One of the reasons for this drawback is the lack of an environment that can integrate different hardware models, workload descriptions, and performance analysis tools. This paper presents our work at Microsoft Research to design a Performance Technology Infrastructure (PTI) that aims to address these issues.

PTI provides an integration platform for multiple performance tools and models. The PTI architecture can be extended to include third party hardware models, workload descriptions, and performance analysis tools. It provides an XML based interface between the architecture components and user extensions. A prototype implementation of a predictive tool based on PTI is also presented. It combines a System Area Network (SAN) model, with a CPU model, and a performance specification language. A Mathematica based tool further analyses the results.

The paper is organized as follows: Section 2 introduces the design objectives of the integration environment. Section 3 describes the PTI architecture, the system components, and their interfaces. Section 4 presents the prototype implementation of the predictive performance system based on PTI. Section 5 includes some results and observations obtained by the predictive system from parallel application running on a PC cluster. Finally, Section 6 contains a summary of PTI and discusses further developments.

2. DESIGN OBJECTIVES

The Performance Technology Infrastructure aims to provide an integration platform for multiple performance technologies. The core design objective is to encourage the creation of performance tools that can be incorporated in the software development process. PTI aims to promote the utilization of model technology during the execution of the application running on distributed systems to guide performance related operating system tasks such as load balancing and scheduling.

There are a plethora of performance issues that should be addressed during the lifecycle of the software. The need and the benefits of incorporating performance techniques during the lifecycle have been described in detail in [17]. A performance model should be initially developed at the early design stages to determine which software designs have the potential to meet the performance objectives. In the latter development stages the initial model should be extended to represent the software architecture in more detail.

PTI supports the software lifecycle modeling process by endorsing multiple levels of model abstraction. A particular abstraction level is a combination of two factors: 1. the level of detail of the control flow information, and 2. the type of resource usage associated with each control statement. As will be seen in the next section, PTI has a flexible and dynamic methodology for defining both the control flow and the associated resource usage information. Models can use timing measurements or high-level operation counts (e.g. floating point operations) in the initial development stages. When the source code is available, very detailed models can be developed to encapsulate the behavior and interaction of many system components, such as the memory hierarchy and the operating system overheads.

Performance modeling and analysis may involve specialists with different backgrounds. A hardware model developer has a good understanding of both the operation of the hardware component and modeling techniques. A software developer and a system administrator are the end users of a performance analysis system and typically have no performance expertise. PTI has a modular architecture that shields the different type of model developers, and users, from unnecessary information overload. PTI components use XML to exchange information including the type of resource usage information, configuration and results. XML provides an extensible formalism that promotes interface standardization.

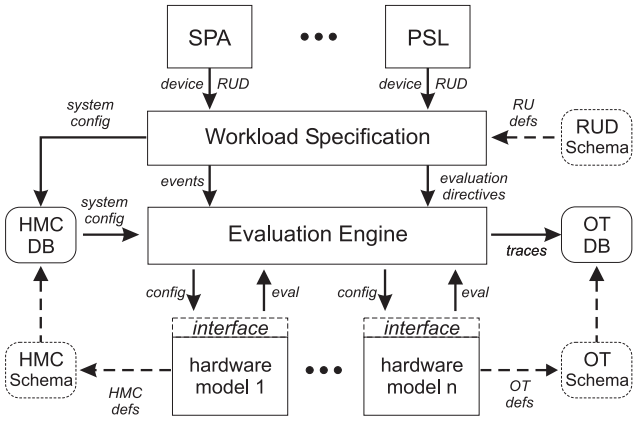
Hardware models, workload specification techniques, and performance analysis tools are available from academia and industry. PTI provides a plug and play approach to the integration of these existing tools. A hardware model interface allows the integration of component models within the infrastructure independently of the model evaluation technology (e.g. analytical, simulation). The resource usage information supported by PTI can be extended to consider the workload required for external hardware model. Existing workload specification tools can be linked to the PTI workload specification library to produce PTI workloads. PTI produces detailed performance outputs that can be further processed to produce performance visualisations and analysis. Filters can convert PTI performance traces to standard monitoring format (e.g. SDDF [16], PICL [8]). The PTI trace framework can be extended to integrate detailed hardware model traces.

3. OVERVIEW OF PERFORMANCE TECHNOLOGY INFRASTRUCTURE

The PTI architecture provides a framework that can be extended to comply with the requirements of a performance study. The next sections present the main components of the architecture and their interactions. The integration of components is realized with XML scripts that provide a standard extensible interface amongst the workloads, hardware models, and performance analysis tools. Examples of component interface scripts demonstrate the flexibility of this approach.

3.1 Architecture

The core architecture includes three components: 1. the Workload Specification Library (WSL), 2. the Evaluation Engine (EE), and 3. the hardware models. An overview of the PTI architecture is shown in Figure 1. The evaluation engine allows the combination of different workload specification and hardware model technologies into a single model.



SPA - Static Performance Analyzer, PSL - Performance Specification Language, RUD - Resource Usage Data, HMC - Hardware & Model Configuration, OT - Output Trace

Figure 1 – Overview of performance technology infrastructure architecture

The workload is composed of a list of hardware or virtual device usage requests. Hardware devices represent system components (e.g. CPU) while virtual devices are typically associated with software libraries. For example, an application can use a message-passing library such as MPI to define a communication pattern. The workload also defines the type of model that should be used to predict the delay for using the device. A single device can be modeled by many underlying hardware models. Depending on the requirements of the performance study, a specific model might be selected for a type of device. For example, the speed of evaluation or accuracy of the model can be two factors that determine the selection of an appropriate hardware model.

The use of a device is configured with Resource Usage Data (RUD). This represents the necessary resources required from the device in order for a task to be performed. For example, a CPU device might be configured with RUDs that include the number of operations that are performed by a computation. The underlying hardware models might employ the same or different RUD definitions. For example, calls to a send function of the MPI virtual device are configured with an RUD that include the source processor, the target processor, and the length of the message. The underlying hardware models might include a detailed simulation of the network or a statistical regression model. Both models can use the same RUD definition.

Workloads are implemented as a sequence of calls to the evaluation engine via the workload specification library. WSL also incorporates library calls that configure hardware models and guide the evaluation process. If required, the user can bypass the automatic evaluation process by alternative evaluation algorithms appropriate for specific workloads. The evaluation process and the available alternatives are presented in Section 3.2.

The evaluation engine combines workload information with the underlying hardware models to predict the performance of the system. The evaluation is structured into stages. Initially, the hardware models are configured to reflect the characteristics of the system. The system and model configuration are determined in the Hardware and Model Configuration (HMC) database. The user can configure the database using the WSL or defining HMC scripts. The second stage is the workload specification where the evaluation engine stores the sequence of device calls and the corresponding RUDs. When the workload definition is completed, the evaluation engine organises the device interactions and calls the appropriate hardware models. As an example of a device interaction, in the case of MPI send call, the source and target processor have an execution dependency. In the last evaluation stage, the EE combines the device model predictions to predict the overall performance of the system. The evaluation process and the detailed operation of the system are captured in the output trace database.

The EE interacts with the hardware models through the Hardware Model Interface (HMI) library. HMI isolates the idiosyncrasies of the models from the evaluation engine operations. The component models can be analytical, statistical, characterization, or simulation. Through HMI the evaluation engine configures the models, passes workload specifications, and evaluates them.

3.2 Evaluation Engine

The evaluation engine (EE) is an event-based system. It uses event lists to represent the system components (processors, processes, threads) and employs events to characterize the use of devices. Interactions of the system components such as synchronizations, communications, and sequencing of shared device usage correspond to event associations. The event-based approach is a flexible technique to capture any type of component associations and it has been used successfully in other generic performance tools [12]. Disadvantages of this approach include slower evaluation times than analytical based approaches and high memory requirements. The Performance Technology Infrastructure is not a simulation engine given that individual device hardware models might use non-simulation techniques to predict delays.

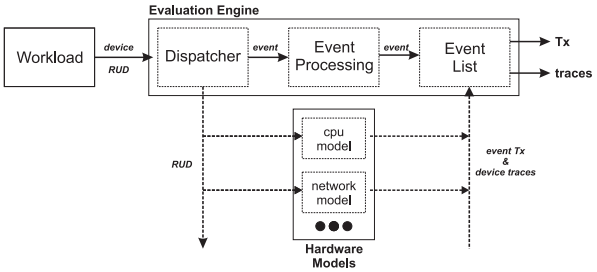


Figure 2 – Evaluation process

The process of evaluating a PTI model and producing performance predictions is demonstrated in Figure 2. Initially the workload specification is processed to produce a series of device usage scenarios and their associated RUDs that are passed to the EE. A dispatcher distributes input, from the workload descriptions, to an event handler and then to the necessary individual hardware models. The event processor is responsible for the construction of an event list for each component of the system. Although the device involved can resolve the type of event, the time required to access the device is still unknown at this stage. The individual hardware model can however produce a predicted execution time (event Tx in Figure 2) of any event passed to it based on its RUDs. The model can pass additional output traces that describe the performance and operation of the device for a specific workload scenario. The resultant prediction and output traces are recorded in the event list. When all device requests have been handled, the evaluation engine processes the event list to produce an overall performance estimate for the execution time (Tx in Figure 2) of the application. The structure of events has been designed to account of any idiosyncrasies of both the workload and underlying hardware models, as shown in Table 1.

Table 1 – Elements of event object

Event Object Element	Source	Description
Event list	workload	Reference to system, processor, or process event list
Device	workload	Reference to hardware device
Model	workload	Reference of model hardware device to support multiple models for device
Resource Usage Data	workload	Input to the hardware device models (XML script)
Inter-event relation	workload	Reference to other event representing component interactions
Duration	hardware model	Duration of event determined from hardware model evaluation
Output traces	hardware model	Detailed hardware model performance traces. (XML script)

The processing of the event list is actually a two-stage operation. The first is in the construction of events, and the second is to re-

solve any ordering dependencies and take into account any contention factors. For example predicting the time for a communication, the traffic on the inter-connection network should be known to calculate channel contention. In addition, messages obviously cannot be received until after they are sent. The exception to this type of evaluation is a computational event that involves a single CPU device - this can be predicted in the first stage evaluation since it does not require interaction with any other events.

The user can control the process to optimize the evaluation time and minimize memory requirements. The user can specify a partial evaluation of the event list in any part of the workload. The EE evaluates the existing events and frees up the memory occupied by the event lists. Applications that include many repetitions of the same workload can be modeled efficiently by using the analytical repetition evaluation option. The user marks the beginning and the end of the workload and the EE evaluates the intervening events once. The workload can then be repeated without re-evaluating the event lists. A fast analytical evaluation algorithm that takes into account the event relationships but does not require involvement of the hardware models, performs this operation.

3.3 Workload Specification

Workloads are specified as library calls to WSL. The EE library includes three types of function calls: 1. declaration of a device usage followed by the related RUD, 2. hardware and model configuration functions, and 3. functions that direct the evaluation process.

A user can either write a program that employs WSL or use a front-end to simplify the workload definition process. Existing workload definition tools can be adjusted to support the PTI. These tools can be organized into the following categories:

- **Static Performance Analyzers (SPAs):** SPAs construct software models that encapsulate the static performance characteristics of applications [5,14]. SPAs translate the application source code to a notation that captures the application control flow and the frequency of operations performed. Dynamic characteristics of the application such as problem size can be incorporated in the workload description as data dependency parameters, specified by the user before the model evaluation.
- **Performance Specification Languages (PSL):** Domain Specific Languages aim to assist the user in determining the performance characteristics of their system [4,13]. They can be employed in the design stage of software development to prototype the performance characteristics of the application. PSLs can also be used in later stages of software development to experiment with new software designs and configurations. For example, a program can use a model of a sequential application created by SPAs to experiment with the performance of numerous parallelisation scenarios.
- **Graphics Design Modeling Tools (GDMT):** GDMT support graphical representation of the software structure and can be used as performance prototyping or analysis tools [18].
- **Monitoring Traces:** Traces created by a past execution of an application can be utilized to perform capacity planning studies. What-if scenarios can be studied for the effects of system component modifications (e.g. increasing the CPU clock speed) [10].

Hardware models have different workload requirements. The characteristic of the workload description is captured in the RUD type definition or schema. RUDs might include a simple list of arguments or a control flow structure. A call to a synchronous MPI send is defined in WSL as a call to the `MPI_send` device followed by an RUD that includes the source processor, the target processor, and the length of the message. The RUD is defined within WSL as an XML script, for example:

```
<!-- RUD for MPI send function -->
<rud device="MPIsend">
<src>1</src><!-- Source processor -->
<trg>10</trg><!-- Target processor -->
<len>1024</len><!— Msg length(bytes) -->
</rud>
```

Other devices might include more complex RUD definitions. For example, a CPU device models a computational segment that includes the control flow of the computation. The model RUD includes the frequency of language operations and the corresponding control flow constructs. In the following example a code segment from the Alternative Direction Implicit (ADI) solution to a Partial Differential Equation (PDE) is modeled with an XML RUD:

```
<!-- C code defined as a CPU RUD
for(i=0; i < MY_SIZE; i++ )
  if( (j >= 2) && ( j != SIZE_J) ) {
    fact = a/b[j-1];
    d[i][j] = d[i][j]-fact*d[i][j-1];
  }
-->
<!-- RUD for CPU model -->
<rud device="CPU">
<loop iter="MY_SIZE" type="clc"
   of="LFOR,CMLL,INLL">
   <compute type="clc" of="2*CMLL,ANDL">
   <case type="clc" of="IFBR">
   <branch exprob="0.9">
     <compute type="clc" of="ARF1,\
        3*ARF2,DFSL,2*TFSL,MFSL,AFSL">
   </branch>
   </case>
</loop>
</rud>
```

The loop tag includes the `iter` attribute that defines the number of loop iterations while the `of` attribute includes the cost for executing the loop header in terms of C language statements. The `type` attribute defines the type of operation frequency counts, C language statement in this case. Similarly the `compute` and `case` represent a computational segment and a conditional statement respectively. The `branch` tag corresponds to a branch of the conditional statement. The `exprob` attribute is the probability of executing the branch. The control flow statements include data and system dependency information. These can be static such as the branch execution probability or dynamic such as the loop

iteration count `MY_SIZE`. Before the evaluation of the model the user specifies the value of the dynamic RUD parameters.

XML offers numerous advantages in the definition and processing of RUDs. It is easy for the hardware model developer to either use an existing RUD definition or create a new type. The complexity of the RUD can be arbitrary, ranging from a simple list of arguments to complex control structures. XML processors simplify the parsing and accessing of XML data. XML provides a framework for defining standard RUDs that can be shared amongst different models and tools.

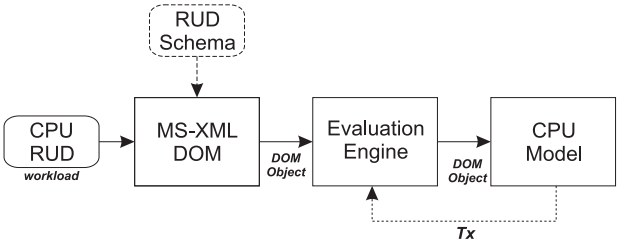


Figure 3 – Example processing of CPU model RUD

PTI uses the MS-XML 2.5 [11] processor to define and process XML tags. RUD definitions are stored in the RUD database, a file written in the XML-Data language. RUDs defined in the workload are initially processed by the XML DOM (Document Object Model) processor and then are passed to the hardware models for further evaluation. XML-DOM parses workload RUDs based on the syntax and rules defined in the RUD schema. The processor builds a tree that correctly represents the structure of the RUD. The hardware model can access and manipulates the elements, values, and attributes of RUDs. An example of the CPU RUD processing is shown in Figure 3.

3.4 Hardware & Model Configuration

PTI provides a dynamic approach for system and model configuration. System component and model configuration is defined in the HMC database. The HMC database is an XML script defined according to the rules specified in the hardware model XML schema. During the initialization of the EE the HMC database is read and a DOM object is created. The EE creates a graph where nodes correspond to computers or networks and edges correspond to network connections. The EE associates each node with an XML tree DOM object that includes the configurations of the node models. EE uses this graph to create event lists and associate workload information to the underlying hardware models. An example of a simple XML configuration tree is:

```
<system name="pc_cluster">
<computer name="pc_node" count="16">
  <!-- CPU Model C Operations -->
  <cpu_clc>
    <DILG>0.043</DILG>
    <IADD>0.127</IADD>
    <!-- Other operation follow -->
  </cpu_clc>
</computer>
<network name="myrinet">
  <ccmod>
    <Nproc>16</Nproc>
  <-- other configuration follow -->
  </ccmod>
</network>

<connect>
  <computer name="pc_node" node="1">
  <network name="myrinet" port="1">
</connect>
<!-- Connection to other nodes ... -->
</system>
```

The script describes a 16-node PC cluster system connected though a Myrinet System Area Network [1]. The computer tag defines the PC nodes and configures a CPU hardware model by defining the cost of C language operations. Similarly the Myrinet network model CCMOD is configured. The `connect` tag specifies a connection of a PC node to a network port. The EE uses this configuration to create the graph in Figure 4.

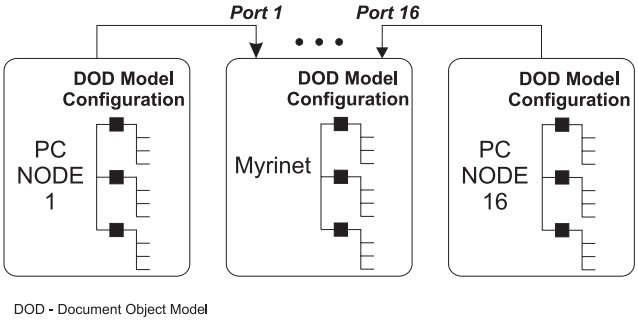


Figure 4 – Hardware & model configuration graph for PC cluster

The user through WSL can change the model and system configuration before or during the evaluation of the model. This feature aims to support models that are evaluated during the execution of an application to guide a performance related task. During the evaluation, system characteristics might change (i.e. network background load). A monitoring sensor can forward the status change to the model and update the model configuration DOM object.

3.5 Performance Traces Post-Processing

The ability to produce predictive traces directly derives from the list of events formed during the model evaluation. EE generates detailed traces that correspond to the duration and interaction of events and their organization into event lists. The process of mapping PTI specific traces to standard trace format (e.g. SDDF) is straightforward.

The output trace format can be extended to include hardware model component specific traces. The traces are stored in XML format and can be further processed by performance analysis tools. The hardware component specific traces can be filtered out for analysis tools that do not support them. The output trace XML schema file includes definitions for the standard PTI traces and definition of the hardware model trace extensions. A detailed presentation of the output trace format is out of the scope of this paper.

4. PROTOTYPE IMPLEMENTATION

An implementation of PTI has been developed for prototyping the performance of MPI parallel applications. The implementation

includes a performance specification language, CHIP^{3}S [13], a hybrid communication model, CCMOD [15], a processor hardware model, and a Mathematica based performance analysis toolset, Figure 5.

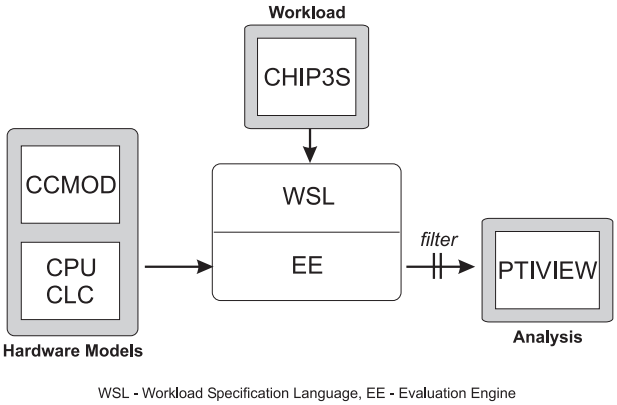


Figure 5 – Prototype PTI implementation

The PTI implementation includes some of the basic features of the architecture including the XML inter-component schema and database processing for RUDs and OTs. It does not currently support dynamic hardware and model configuration. Configurations are hard-wired into the hardware models and EE. Only the basic services for defining workload and processing events are supported from WSP and EE libraries.

The CHIP^{3}S performance specification language is used to describe the performance characteristics of software in the design and implementation stage. The software model includes Software Execution Graphs (SEGs) [17] to describe the control flow and resource usage information of computations. Computations can be defined at different levels of abstraction depending on the stage of software development. In the early design stages the computation flow represents design prototypes in high level of abstraction. In the implementation stages when the source code is available, the computation description models the source code in detail line by line. Communication interactions between processors using the MPI library are straightforward. A CHIP^{3}S compiler translates the performance scripts to C code that includes calls to WSL. The transformation from CHIP^{3}S statements to WSL function calls is easy since there is a one-to-one correspondence between the CHIP^{3}S language and the WSL constructs. CHIP^{3}S is part of the PACE [12] performance modeling and analysis environment and includes additional tools such as static performance analyzers. The application source code can be automatically translated to CHIP^{3}S performance scripts.

CCMOD is a C++ library that can be used to provide performance predictions of communication networks. It models the underlying network topology using detailed workload information (traces), which encapsulates the expected computation/communication requirements of an application. CCMOD is a hybrid model that contains statistical model information as well as simulating the main stages that change the state of the communication network traffic. CCMOD produces communication delay predictions considering the link contention and background loads. The integration of CCMOD into PTI involved the creation of a hardware model interface that converted EE events into workload traces compatible to the expected CCMOD workload. Additionally, the output of CCMOD was integrated into the event duration field. Detailed output information showing the network status during the network model evaluation was embedded into the standard PTI output trace format with the introduction of new XML tags in the OT schema.

The CPU model is a simple C language operation cost analysis. The frequency of C operations is determined by the workload RUDs. A benchmark measures the cost of C operations on a specific system and combines the workload RUDs to determine the computational costs. The model provides a simplistic representation of the CPU performance since it does not consider the memory hierarchy and sophisticated CPU operations (e.g. pipeling). However, it is sufficient to represent a hardware model that operates on single events.

A custom Mathematica performance analysis tool (PTIVIEW) was used to visualize the output traces generated by the PTI prototype. PTIVIEW processes both the standard and CCMOD traces to create visualization representation of the system operation such as time-space diagrams. It also produces a number of overall performance metrics and comparisons of subsequent evaluations varying the workload or system configuration. Since Mathematica does not provide an XML processor the output traces were processed by a filter utility that converted the XML tags to a Mathematica compatible format.

5. RESULTS

The aim of developing the PTI prototype was to prove the concept, not to produce accurate or detailed performance predictions. The system and model configuration was not dynamically defined through the HMC database but was statically included into the hardware model and evaluation engine. A 16 PC cluster connected via a Myrinet SAN switch (M2-OCT-SW8) was modeled. The switch is a multistage ATM (Asynchronous Transfer Mode) network with peak bandwidth of 1.28 Gb/s and latency of 100ns per 8-port switch. CCMOD includes a hardware model description of the Myrinet switch, so no further adjustments are necessary. The PCs are 300 Mhz Pentium IIs with 384 Mb of RAM running Windows 2000.

A test workload was obtained for the Sweep3D application that is part of the ASCI application suite [9]. Sweep3D is a complex benchmark for evaluating wavefront application techniques used to evaluate advanced parallel architectures at the Los Alamos Laboratories. The application uses the HPVM-MPI version 1.9 library from University of Illinois [3] for inter-processor communication. The application source code was analyzed by the PACE static performance analysis tools and produced a CHIP^{3}S script [2]. The script can be configured for a range of problem sizes and processor configurations.

Figure 6 shows a comparison of the predicted versus the measured execution time for various problem and system sizes. The error ranges from 5%, for smaller problem sizes, to 30% for larger configurations. The predicted accuracy is better than the individual hardware model accuracies. The CPU model tends to over predict the computational delays because it does not model the memory hierarchy and the processor internal parallelism. The communication model tends to under-predict the communication delays because it does not model the message link pipelining and the MPI library overheads. The evaluation of the model took place on a Pentium III 450 MHz running Windows 2000. The 16-processor

configuration and the 200x200x200 Sweep3D array size generated 386233 events and it took 3.437 sec to evaluate without generating detailed output traces (112375 events/sec).

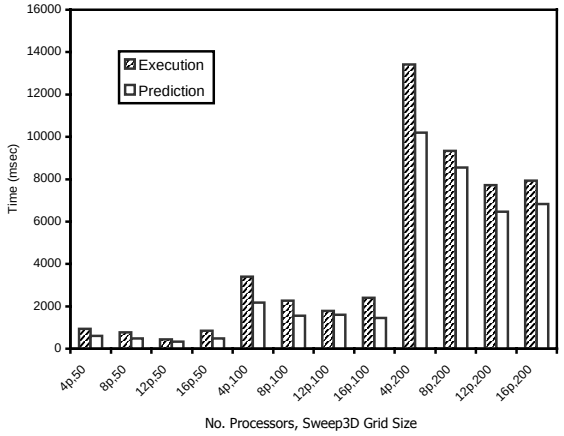


Figure 6 – Predicted and execution times for various system and problem sizes for the Sweepd3D

Figure 7 includes the spacetime diagrams for the Sweep3D application. It includes a time line for each processor and shows four basic types of events (idle, send, receive, and wait). Note that the idle event corresponds to processing while the wait event corresponds to delays due to synchronization. Lines connect the time lines to denote a communication. The diagram includes a part of the overall workload (1%) and it illustrates a typical data parallel grid type communication pattern.

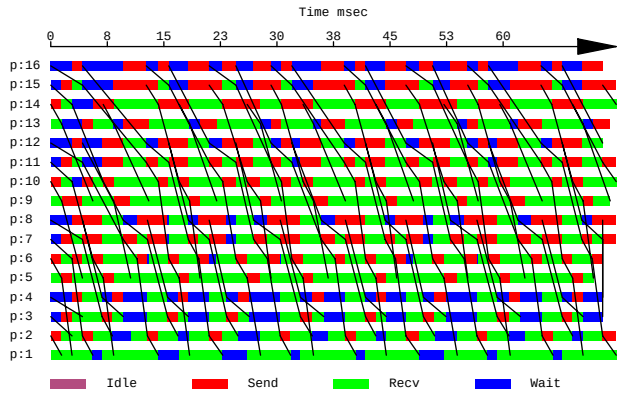


Figure 7 - Timespace diagram for the Sweep3D application

Figure 8 shows two communication stages and the corresponding *Bandwidth Availability (BA)*. For a message that travels though the network links BA is calculated for the link with the highest load. BA is the effective bandwidth of this link considering the link contention and the background load. CCMOD includes extended performance information (e.g. BA) into the standard output trace format by extending the XML output trace schema. PTIVIEW can either ignore the extended traces (e.g. spacetime diagram) or use them for detailed hardware component analysis.

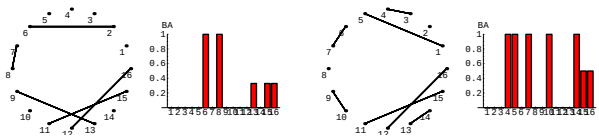


Figure 8 – Two communication stages and corresponding *BA* for the Sweep3D application

Figure 9 shows the knee capacity [7] of the network during the execution of Sweep3D. Knee capacity is the throughput of the network in packets/sec for a message that passes through the link that has the highest load (i.e. is the bottleneck of the system). It is possible in the PC cluster for two different applications to run on different (or the same) PC nodes and share the network. This causes interference in the communication performance of the applications. The knee capacity graph includes the effect of 30% background load on the communication performance of the workload.

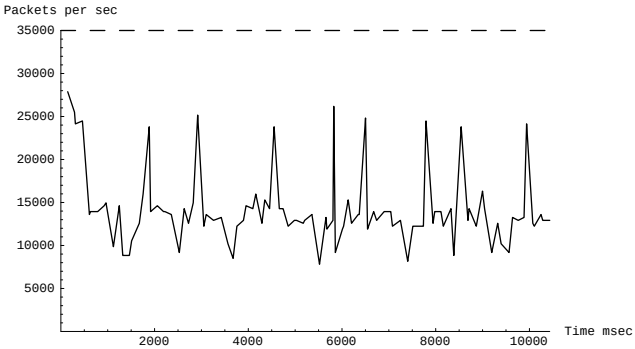


Figure 9 – Sweep3D 200x200x200 array size knee capacity

6. SUMMARY

In this paper, a novel Performance Technology Infrastructure has been presented that aims to provide an integration platform for multiple performance tools and models. The PTI architecture can be extended to include third party hardware models, workload descriptions, and performance analysis tools. It provides an XML based interface between the architecture components and user extensions.

PTI can be used as the underlying framework to develop models that support the software lifecycle. This is possible because it provides dynamic and flexible definition of control flow and related resource usage information. In the initial stages of software development models can be of high level of abstraction aiming to provide initial insight to the performance of the application. In the latter stages of development the models can be refined to represent in more detail the workloads.

Another design objective of the PTI architecture is to shield the end user and the performance specialists from irrelevant parts of the system. An overview of the use of PTI components and interfaces for four type of users is shown in Table 2. The user categories that correspond to the table columns are: workload specification tool developers, hardware model developers, end users, and performance analysts. It is clear that with the exception of the

hardware model developer the other users have a limited exposure to the system components. Existing hardware models can be incorporated within PTI by the use of the EE interface and the definition of XML schemas.

Table 2 – Users of PTI components

PTI Components	Users			
	Workload Developer	Hardware Model Developer	End User	Performance Analyst
Workload Specification Library	✔		✔	
Evaluation Engine		✔		
Resource Usage Database	✔	✔	✔	
Output Trace Database		✔	✔	✔
H&M Configuration Database		✔		
Resource Usage Schema		✔		
Output Trace Schema		✔		
H&M Configuration Schema		✔		

A prototype implementation of a performance prediction system using PTI has been presented. The system includes a performance specification language (CHIP^{3}S), a CPU hardware model, a communication model (CCMOD), and a Mathematica based performance analysis system. A case study of the performance of the Sweep3D application was also presented.

PTI is currently in the early development stages. The prototype implementation presented here is a proof of concept. The success of PTI as a valuable performance technology framework is yet to be proven. A full version of the PTI implementation is under way. A number of performance tools have been scheduled to be developed based on PTI including a static performance analyser as part of the compiler infrastructure.

6. ACKNOWLEDGEMENTS

I would like to thank Dr. Darren Kerbyson, Mr. Junwei Cao and Prof G.R. Nudd from the University of Warwick for providing the Sweep3D workload specification. I would also like to thank Dr. Jonathan Hardwick for his suggestions.

7. BIBLIOGRAPHY

[1] N.J. Boden, D. Cohen, et.al., "Myrinet: A Gigabit-per-Second Local Area Network", *IEEE Micro* 15(1), 1995, pp. 29-36.

[2] J. Cao, D.J. Kerbyson, E. Papaefstathiou, and G.R. Nudd, "Performance Modelling of Parallel Distributed Computing Using PACE", in: *IEEE International Performance Computing and Communication Conference (IPCCC-200)*, Phoenix, USA, February 2000.

[3] A.Chien, M. Lauria, et.al, "Design and Evaluation of an HPVM-based Windows NT Supercomputer", *The International Journal of High-Performance Computing Applications*, Vol. 13(3), 1999, pp. 201-219.

[4] E. Deelman, A. Dube, A. Hoisie, Y. Luo, R.L. Oliver, D. Sundaram-Stukel, H. Wasserman, V.S. Adve, R. Bagrodia, J.C. Browne, E. Houstis, O. Lubeck, J. Rice, P.J. Teller, and M.K. Vernon, "POEMS: End-to-end Performance Design of Large Parallel Adaptive Computational Systems", in: *Proceedings of the 1st International Workshop on Software and Performance*, pp. 18-30, 1998.

[5] T. Fahringer, "Estimating and Optimizing Performance for Parallel Programs", *IEEE Computer*, Vol. 28(11), pp. 47-56, 1995.

[6] J.S. Harper, D.J. Kerbyson, and G.R. Nudd, "Analytical Modeling of Set-Associative Cache Behavior", *IEEE Trans. on Computers*, Vol. 48(10), pp. 1009-1024, 1999.

[7] R. Jain, *The Art of Computer Systems Performance Analysis*, John Wiley & Sons, pp.38.

[8] M.T. Heath, "Recent Developments and Case Studies in Performance Visualization Using ParaGraph", in: *Proceedings of the Workshop on Performance Measurement and Visualization of Parallel Systems*, 1992.

[9] K.R. Koch, R.S. Baker, and R.E. Alcouffe, "Solution of the First Order Form of the 3-D Discrete Ordinates Equation on a Massively Parallel Processor", *Trans. of the American Nuclear Society* 65(108), 1992.

[10] J. Labarta, S. Girona, and T. Cortes, "Analyzing scheduling policies using Dimemas", *Parallel Computing* Vol. 23(1-2), pp. 23-34, 1997.

[11] MS XML 2.5 Software Development Kit, Microsoft Corporation, http://msdn.microsoft.com/xml/default.asp.

[12] G.R. Nudd, D.J. Kerbyson, E. Papaefstathiou, S.C. Perry, J.S. Harper, and D.V. Wilcox, "PACE – A Toolset for the Performance Prediction of Parallel and Distributed Systems", to appear in Inter. Journal of High Performance Computing Applications, Sage Science Press, 2000.

[13] E. Papaefstathiou et al., "An overview of the CHIP^{3}S performance prediction toolset for parallel systems", in *Proc. of 8th ISCA Int. Conf. on Parallel and Distributed Computing Systems*, pp. 527-533, 1995.

[14] M. Parashar and S. Hariri, “Compile-Time Performance Prediction of HPF/Fortran 90D”, *IEEE Parallel & Distributed Technology*, Vol. 4(1), pp. 57-73.

[15] E. Papaefstathiou and D.J. Kerbyson, “Predicting Communication Delays of Detailed Application Workloads”, submitted to the 13th International Conference on Parallel and Distributed Computing (PDCS-2000).

[16] D.A. Reed, et al., “Scalable Performance Analysis: The Pablo Analysis Environment”, in: *Proc. Scalable Parallel Libraries Conf.*, IEEE Computer Society, 1993.

[17] C.U. Smith, “Performance Engineering of Software Systems”, Addison Wesley, 1990.

[18] C.U. Smith and L.G. Williams, “Performance Engineering Evaluation of Object Oriented Systems with SPE*ED”, in: *Computer Performance Evaluation: Modelling Techniques and Tools*, No. 1245, (R.Marie, et.al. eds.), Springer-Verlag, Berlin, 1997.

[19] VTune Performance Enhacement Environment Manual, Intel Corporation, 1999.