# Performance Modeling, Analysis and Simulation of Scientific Applications: a Case Study

T. Fahringer[1], N. Mazzocca[2], M. Rak[2], S. Pllana[1], U. Villano[3], G. Madsen[4]
[1]*Institute for Software Science, University of Vienna, Liechtensteinstrasse 22, 1090, Vienna, Austria*
tf@par.univie.ac.at , pllana@par.univie.ac.at
[2]*DII, Seconda Università di Napoli, via Roma 29, 81031 Aversa(CE), Italy*
n.mazzocca@unina.it, massimiliano.rak@unina2.it
[3]*Universita' del Sannio, Dipartimento di Ingegneria, C.so Garibaldi 107, 82100 Benevento, Italy*
villano@unisannio.it
[4]*Institute of Physical and Theoretical Chemistry, Technical University Vienna, Getreidemarkt 9/156,A-1060 Vienna, Austria*
gmadsen@theochem.tuwien.ac.at

**Abstract**

*This paper presents a high-level approach for assessing the performance behavior of complex scientific applications running on a high-performance system through simulation. The proposed methodology relies on high-level descriptions of both application and system. The application is described in MetaPL, an XML-based description language, and the system is modeled and simulated by using HeSSE, an extensible distributed heterogeneous system (DHS) simulator. This modeling technique is applied to the performance analysis of a real-world scientific application (LAPW0), running on a cluster of SMP nodes. Extendibility features of both MetaPL and HeSSE were largely used, developing extensions for the MetaPL language and new components for the simulator. The paper closes with the validation of the performance model, obtained through the comparison of the predicted performance results with measurements on test runs of the application on the real system.*

## 1 Introduction

The development of efficient programs for high performance systems is an error-prone and time-consuming process, which may involve many cycles of code editing, compiling, executing and performance analysis. The adoption of off-the-shelf and/or commercial components for building low-cost computing platforms (e.g., Beowulf clusters) may lead to good price/ performance ratio, but introduces a higher level of complexity in the system architecture. Attaining good performance on such systems is not decidedly a simple task. Additional problems may complicate software development and tuning. For example, the system in its full scale may not be available at testing time. Many applications do not scale linearly with the number of nodes, and so a more powerful cluster configuration (e.g., more nodes) may not improve the application performance. The execution of measurement sessions may be complex and time-consuming. In light of all the above, the availability of predicted performance figures can be very helpful.

Software performance analysis is becoming increasingly more important as new computing architectures and programming paradigms emerge. Software Performance Engineering (SPE) promotes the idea of applying modeling and performance analysis at early development stages. Unfortunately, this approach and the corresponding tools (e.g., SPEED [13]) are not commonly adopted for scientific application development. Most of existing tools that support performance analysis of scientific applications are limited to specific architectures or programming paradigms. P3T [15] is a performance estimator for mostly-regular HPF (High Performance Fortran) programs, but it covers only partially the totality of message-passing codes. PAMELA [4] utilizes a high-level language for resource utilization description and compiles it into a symbolic function that can be evaluated fairly easily, but its prediction accuracy is not satisfactory. DiMeMas [5] is a heterogeneous system simulator that suffers from the accuracy loss due to oversimplified network models; furthermore, it supports only message-passing paradigms.

In this paper we present a methodology that can substantially alleviate the problem linked to the analysis of large, complex scientific application performance behavior on a high performance system. Our methodology hinges on high-level descriptions of both application and system. The application is described in MetaPL

[9][10][11], an XML-based description language. The system is modeled and simulated using HeSSE [7][8], which is an extensible distributed heterogeneous system (DHS) simulator. We demonstrate the validity of our methodology and tools by analyzing the performance behavior of a real-world application (LAPW0 [2]) on a high performance cluster. This task required the development of suitable extensions for MetaPL and HeSSE, both of which have an open structure that can be easily extended through custom language- or paradigm-based notations (MetaPL) or simulation components (HeSSE).

Our methodology can be successfully applied in some well-defined cases:

- the application is a scientific program (low number of different execution paths);
- the source code is available, but it is too complex to manage/to be understood;
- the application execution time is too long for obtaining a complete performance characterization by exhaustive sets of direct measurements.

The paper is organized as follows: an overview of the MetaPL description language, the proposed modeling methodology and the tools adopted are presented in the next section. Section 3 describes the extensions developed to meet for modeling and simulating the LAPW0 application on the target cluster. The modeling of application and system is described in Section 4. Section 5 deals with the model validation. Finally, some concluding remarks are made and future work is outlined in Section 6.

# 2 Overview

In this section we briefly describe the methodology for performance evaluation of scientific applications and the tools accordingly adopted. In addition, we give a short description of the application and of the cluster architecture, which in this paper will be used as a case study to illustrate and to validate the proposed methodology.

## 2.1 DHS Modeling and Simulation with HeSSE

The tool used to model and simulate the target computer architecture is HeSSE (*He*terogeneous *S*ystem *S*imulator *E*nvironment), a simulation tool that can reproduce the performance behavior of a given distributed heterogeneous system (DHS) for a given application, under different computing and network load conditions [7][8]. One of the distinctive features of HeSSE is the adoption of a compositional modeling approach. DHS are modeled as sets of interconnected components; each component reproduces the performance behavior of a section of the complete system at a given level of detail.

A component is an object that can reproduce the temporal and functional behavior of a part of the whole system. Components can be *passive*, if they offer services to other components, or *active*, if they use services offered by other components.

Physical-level performance, such as that resulting from a given processor architecture, is in general modeled with simple analytical models or with integral (i.e., not punctual) behavioral simulation. For example, the use of a processor to execute instruction sequences is modeled as the total time spent processing, without considering the execution flow behavior on a per-instruction basis.

Applications are usually active components, whose behavior is described through traces. A trace is a file, which contains a sequence of events or actions that is to be simulated. For example, a typical HeSSE trace file for parallel MPI applications is a sequence of CPU bursts and message passing primitives. A trace represents a single and specific execution of a parallel/distributed program.

## 2.2 MetaPL Application Modeling

MetaPL is an XML-based language for the description of distributed/parallel programs. The main characteristic of MetaPL[9] is easy extensibility, obtained exploiting XML distinctive characteristics. The goal of MetaPL is to supply a unified description notation for parallel/distributed programming, which has always been characterized by the use of a wide set of programming paradigms and models. MetaPL is provided with a filtering mechanism that allows the generation of graphical representations of a program, such as UML Diagrams [12], or automatic documentation. This mechanism is also employed to generate traces in the format required for HeSSE simulation.

The language is composed by a very simple *core* and *language extensions*. The *core* defines elements for description of control-flow (e.g. loop, switches, etc.), simple task management instructions (e.g. spawn, exit, wait) and *code blocks*, which encapsulate sections of code. A code block can be annotated with timing information or with a cost function. A cost function defines a relationship between the input data dimension and the code execution time. *Language extensions* define all paradigm-, language-, model- specific primitives. For example communication primitives for message-passing (e.g. send, receive, broadcast, etc.) are defined through language extensions.

In the rest of the paper we will refer to an instruction description construct as a *command*. The MetaPL language is discussed in more detail in [9][10][11].

The main feature of the MetaPL language is the *Filter system* for *View* generations. A *View* is a less complete and more specific representation of the program, such as an UML Diagram or a specific execution trace. A *Filter* is composed by the description of the target View, and the XSLT transformation needed to obtain it from a MetaPL description.

## 2.3 Methodology

Our methodology is based on high-level application description with MetaPL, and system modeling and simulation with HeSSE. The MetaPL filter system translates the high-level application description into HeSSE traces.

The basic steps of our methodology are:

- target system and application preliminary analysis;
- system detailed modeling;
- system performance analysis and prediction.

The first step involves the analysis of the specific problem and system, the second step the model development, tuning and validation, and the third one the collection and analysis of the simulation results.

**2.3.1 Preliminary analysis.** The objective of this step is to define the main system features (as far as performance is concerned) that have to been simulated for successful performance prediction. This means understanding which hardware and software components of the system are needed for simulation.

As extendibility is one of the main features of both HeSSE and MetaPL, it is possible to develop new simulator components or language extensions and filters if needed.

The analysis step involves the definition of the objectives, the wished accuracy level, and the selection and development of extensions for MetaPL and HeSSE that may be possibly required.

**2.3.2 System modeling.** The objective of system modeling is to obtain a usable and valid model of both application and system. The modeling phase can be divided into following steps:

- modeling of the application in MetaPL;
- construction of the HeSSE system model;
- tuning and validation of the simulation model.

The MetaPL modeling phase involves the description of the application in terms of code blocks and communication primitives. The MetaPL filter system translates the MetaPL description into traces for simulation. The HeSSE modeling phase involves the definition of the actual DHS configurations. This is actually done by means of configuration files. HeSSE and MetaPL models are related: the MetaPL output traces are the input for HeSSE simulation. So they have to be build together.

The task of describing the application with MetaPL can be divided in two phases:

- application description;
- cost function definition.

The first step aims to obtain the MetaPL description, and the second one to annotate this description with the required timing information.

To validate the timing information some test cases should be executed, gathering information about the software code regions that correspond to MetaPL code blocks. A suitable test selection and validation procedure has to be defined to obtain the right values. The obtained simulation model needs to be tuned and validated by comparing it to the real system.

**2.3.3 System Performance Analysis and Prediction.** Once the simulation model is completely defined and validated, it can be used to obtain the required performance information about the application and the system. This usually entails the use of further simulation sessions to gather the data, and to analyze the performance behavior of the target system.

In particular, the presented methodology is suitable for the performance analysis of scientific programs. This kind of programs is usually characterized by a behavior that is very similar in several successive executions. In other words, the program has a limited number of possible execution paths (often only one), all of which share the same (or very similar) performance behavior. Of course, even in such cases, the program behavior (as far as performance is concerned) may depend on some input data parameters, such as the dimension of a matrix. This is managed at the interface between MetaPL and HeSSE. The MetaPL trace filter just asks the user the actual values of such parameters, and uses them to generate traces for simulation that are bound to the supplied values.

## 2.4 Target Application and System.

To illustrate and validate our methodology we will analyze here the performance behavior of a real-world parallel application on a cluster of SMP nodes. The selected application is LAPW0 [2], developed at the Institute of Physical and Theoretical Chemistry, TU Vienna, Austria.

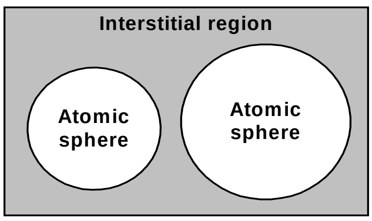


**Figure 1. Partitioning of the Unit Cell into Atomic Spheres and an Interstitial Region.**

The Linearized Augmented Plane Wave (LAPW) method is among the most accurate methods for performing electronic structure evaluation for crystals. The unit cell of a crystal is divided into non-overlapping atomic spheres (centered at the atomic sites) and an interstitial region (see Fig. 1).

The LAPW0 application, which is part of the Wien97 package [2], calculates the *effective potential*. The LAPW0 application code is written in FORTRAN90 and MPI [3].

The Gescher cluster consists of a front end and two subclusters, which are interconnected via a 100Mbs Fast Ethernet switch. The front end contains two 400MHz Pentium II processors. The first subcluster consists of eight nodes, each having two 400MHz Pentium II processors. The nodes of this cluster are interconnected via 100Mbs switch. The second subcluster consists of sixteen nodes, each having four 700MHz Pentium III processors, which are interconnected via both Fast Ethernet and Myrinet switches. The Gescher cluster is located at Institute for Software Science, University of Vienna. Our simulation target is the second subcluster, with Ethernet as network layer.

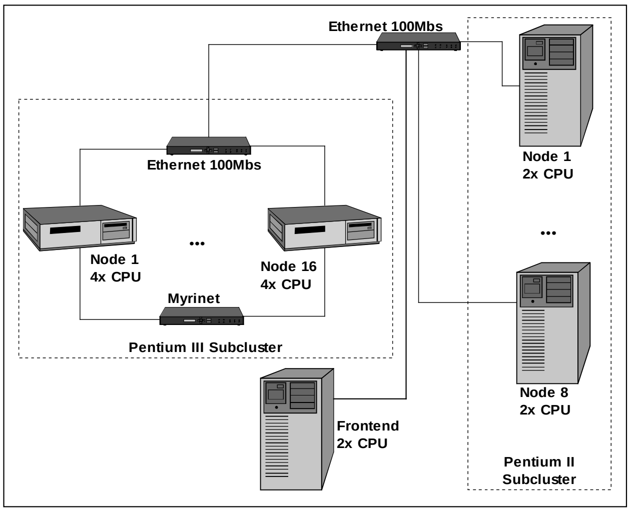


**Figure 2. The Gescher Architecture**

# 3 The Extension of MetaPL and HeSSE

Both MetaPL and HeSSE can be extended to meet the needs of a specific description/analysis task. We have developed new components for HeSSE to model and simulate the specific architecture of the Gescher cluster, namely:

- an *Ethernet Switch* component that reproduces the specific features of the target cluster.
- an *SMP Node* component to model the cluster multiprocessor nodes.

The run-time support layer in HeSSE is managed by a generic Message Passing system, which is most typically used to simulate PVM applications. Some modifications were needed to model the MPI layer. In particular, further message-passing extensions were developed to support the collective communication primitives used by the LAPW0 application.

## 3.1 The Switch Component

A switch connects two or more LANs through its network interfaces; it maintains a table, named *address table*, consisting of two columns; the first column contains a layer 2 address (MAC address), while the second column contains a network interface ID. The address tables are built dynamically: each time the switch receives a frame from a network interface, it updates the table with the source MAC address and the network ID. A detailed description of the mechanism adopted to build the table is out of scope here.

The simulation model adopted is based on the solution proposed by Korcyl et al. [6]. They have defined a parameterized model that is useful for modeling every kind of switch, using parameters that can be found by direct measurement or consulting the component data-sheets.

## 3.2 SMP Node Simulation

Processing nodes are modeled in HeSSE through the *Node* Component. The *Node* is essentially an interface to three internal components, which simulate processor (*CPU*), pre-emptive scheduler (*Scheduler*) and operating system kernel behavior (*Kernel*), respectively. Among other things, the *Node* provides the processes with a function that makes it possible to create a new process and primitives for process synchronization. The interested reader may find details about the N*ode* simulation model in [7][8].

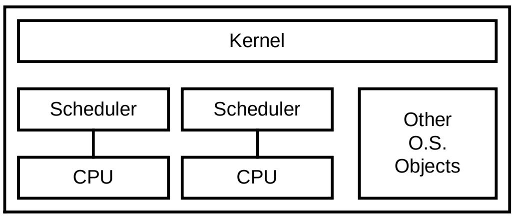


**Figure 3. The SMP Node Component Architecture**

SMP nodes are composed of more than one processor, typically two or four, which share a common memory. Processes and threads can be executed in parallel on the different CPUs, and are synchronized using OS primitives. Our simulation model provides replicated Scheduler/CPU pair in the *SMP Node* Component, as shown in Figure 3. To manage the presence of different processes (and threads) and their parallel execution, when a new process is created a round-robin algorithm is adopted to define the pair to which it will be assigned. After creation, the process never migrates from one CPU to another. All the single-processor *Node* component synchronization primitives can be used in the *SMP Node* Component.

### 3.3 MetaPL MPI Language and Filter Extension

LAPW0 uses essentially two MPI collective communication primitives, *MPIBCast* and *MPIAllReduce*. The first one uses a simple binary tree algorithm for implementing the broadcast, whereas the second one is implemented by sending all processes' contributions to the root process, and broadcasting from the root the reduced data.

MetaPL language extensions and filters for message passing primitives are already available, so we needed to develop only an extension for the above mentioned collective communication primitives, and a MetaPL filter performing the translation of the collective primitives into basic Send/Receive primitives.

## 4 Application and System Modeling

In this section we will describe the modeling of the LAPW0 application with MetaPL and of the Gescher cluster with HeSSE.

The LAPW0 application consists of 100 file modules (a module is a file containing source code). The modeling procedure aims to individuate the more relevant (from performance point of view) code sections. We will call these code sections "fundamental units". According to the above definition, a fundamental unit can be a sequence of computation steps, communication operations and input/output operations. Both computation steps and input/output operations are modeled as MetaPL *CodeBlocks*, accounting for generic CPU bursts. Communication primitives are instead modeled by suitable MetaPL commands.

The application source code was analyzed in cooperation with the research group that originally developed Lapw0, in order to identify the fundamental units. These turned out to be mostly top-level functions, or sections of code containing time-expensive loops. The choice of the fundamental units was verified by profiling the code. The profiling data showed that the chosen fundamental units account for more than 99% of the total application execution time. Table 1 summarizes the fundamental units individuated, together with a brief description and their timing model, which will be discussed in the next section.

The cluster configuration to be simulated will be modeled in HeSSE as a configuration file. In each configuration a SMP Node Component represents the used nodes. The MPI Layer is modeled by the MPS (Message Passing System) components, daemon (MPSD) and shared data (MPS Data). Details about these components can be found in the HeSSE documentation [8]. Each process is modeled by a MPS_Apps component, which reads the process behavior from a trace file. The Network layer is modeled using the Ethernet NICs components and the Switch component.

The number of Nodes and the number of processors univocally characterize each configuration adopted, under the assumption that the number of processes is equal to the number of processors, and that the same number of processor is used in every node.

### 4.1 Cost Function Symbolic Modeling

In this section we describe the choice of the cost functions for the MetaPL *CodeBlocks* corresponding to the code fundamental units.

Figure 4. illustrates the basic steps for building the cost functions. The first step is to instrument the parts of the application code that correspond to code blocks. Then the test to be executed is chosen. The tests are defined by the system configuration and the application input parameters.

After each execution, the measured data are stored in a performance database. After performing all the experiments, the data stored in the database are used to build the cost functions by regression.

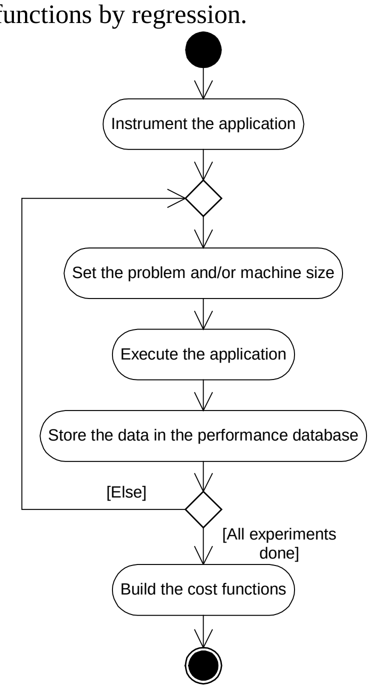


**Figure 4. Cost Functions Building Process**

To define the tests to be used, we have to classify the application inputs and their relationship with machine configurations, needed by the MetaPL filter system for the simulation trace file generation.

The application analysis leads to the identification of a fairly small parameter set. The regression analysis showed that each *CodeBlock* can be modeled as a linear function of only one or two of these parameters. Table 1 summarizes the obtained cost functions and their parameter dependencies.

| Cost Function | Description |
|---|---|
| MPIinit (NNOD, NID)<br>= 210 + 112 (NNOD – NID) | Initialize MPI Environment (1) |
| Loop250 (NAT)<br>= 0,7525 + 0,1595NAT | Loop 250 |
| ReadTCD (NAT)<br>= 385,4866 + 317,9183NAT | Read TCD |
| CalcMPM (PNAT)<br>= 3312,9855 + 7718,3527PNAT | Calculate Multipolmoments |
| CalcCIP (PNAT)<br>= 390,0921 + 9876,4525PNAT | Calculate Coulomb interstitial potential |
| CalcPWCP<br>= 66,846 | Calculate plane wave coulomb potential |
| CalcCRMT (PNAT)<br>= 460,2772 + 10192,637PNAT | Calculate coulomb radius muffin tin |
| Output1 (PNAT)<br>= 6,6405 – 0,03257PNAT | First part of output |
| CalcXCPIS (PNAT)<br>= 63,0062 + 1940,8488PNAT | Calculate XC potential inside spheres |
| CalcIXCP<br>= 5503,6346 | Calculate interstitial XC potential |
| Output2 (NPE)<br>= -2107,9096 + 1286,034NPE | Second part of output |
| FFT (NAT)<br>= 110,6438 + 203,6609NAT | FFT, Call REAN0 and REAN3 |
| CalcTE (PNAT)<br>= -0,28996 + 18,0251PNAT | Calculate total energy |
| SFT<br>= 954,3173 | Call REAN4 |
| DefVTOT (PNAT)<br>= -0,5586 + 0,8016PNAT | Define total potential VTOT |
| Output3 (PNAT)<br>= 344,2876 + 39,29298PNAT | Third part of output |
| MPIterm (NAT, NPE)<br>= 220,65NAT + 60,65NPE + 1012,71 | Terminate MPI Environment (2) |

- NAT is the number of atoms.
- PNAT is the number of atoms per process.
- NPE is the number of processes. One process is mapped to one CPU.
- NNOD is the number of nodes. One node may have one or more CPU's.
- NID is the ID of the node. NID = 0, .. , NNOD – 1.

1. The value of MPIinit for processes within one SMP node is identical.
2. If MYID>0 then MPIterm ~ 1ms.

**Table 1. Cost Functions Description**

### 4.2 Model Utilization

The modeling process lead to the definition of a model composed of:

- a MetaPL application description;
- a set of HeSSE configuration files ;

This model can be used as shown in Figure 5. The simulation process takes place in two phases: specialization of the model to a test case, and test simulation.

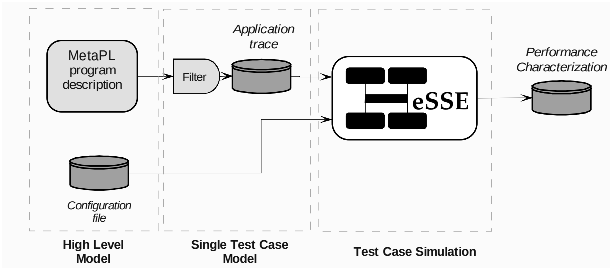


**Figure 5. Complex Model Utilization**

The specialization of the generic model corresponds to choosing the system configuration and defining the model input parameters (and accordingly, cost function values).

## 5 Model Tuning and Validation

The model shown in the previous section needs to be tuned to the real system timing features and to be validated. The long LAPW0 application execution time limits the number of measurements that can be done on the system. So the objective is to perform tuning and validation using fewer data values as possible.

### 5.1 Tuning

This procedure relies on the analyzer expertise and cannot be easily formalized. Many techniques exist for parameter evaluation, such as simulated annealing, but the low number of parameters we actually need lead us to not automatize this procedure.

For brevity's sake, below system configurations will be univocally identified through strings with the format N<number of nodes>p<number of CPUs>. For example, the string N4p16 represents the configuration with four nodes, each with four CPUs.

As nodes are homogeneous (the multiple CPU have identical processing power), we can use a simple configuration (e.g., N2p2) to characterize the single-node performance timing. The parameter values measured on the N2p2 configuration have been adopted for all nodes and for all configurations. Network parameters (such as the switch internal bandwidth) can instead be obtained from the available technical documentation.

### 5.2 Model Validation

To validate the model we have performed simulations and compared the execution time *Ts* obtained by simulation to the execution time obtained by direct measurement *Tm*, for the previously described set of input sizes and machine sizes.

The results of our simulations and measurement runs are shown in Table 2 and represented graphically in Figure 6. and Figure 7.

| Configuration | NA=8 | NA=16 | NA=32 | NA=64 |
|---|---|---|---|---|
| N2p2 | 0,3214 | 0,254 | 0,199 | 0,178 |
| N4p4 | 0,0104 | 0,421 | 0,367 | 0,342 |
| N8p8 | 1,759 | 0,610 | 0,610 | 0,5750 |
| N12p12 | 1,264 | 0,173 | 0,858 | 1,075 |
| N16p16 | 0,594 | 1,779 | 0,496 | 0,800 |
| N1p4 | 0,579 | 0,515 | 0,526 | 0,547 |
| N2p8 | 0,373 | 0,0471 | 0,566 | 0,699 |
| N3p12 | 0,214 | 0,4217 | 1,057 | 1,197 |
| N4p16 | 1,781 | 1,7210 | 0,177 | 0,499 |
| N8p32 | 0,826 | 0,792 | 0,301 | 1,035 |

**Table 2. Simulation Results**

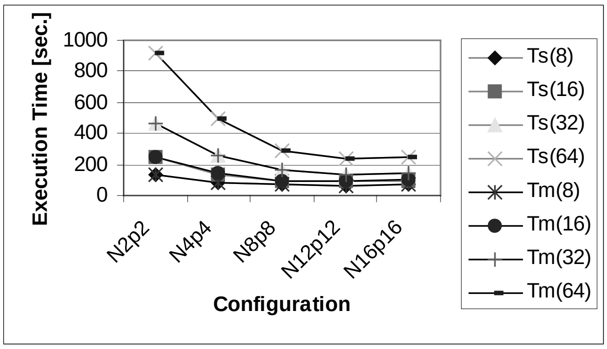


**Figure 6. Simulation and Measurement Results on Monoprocessor Nodes**

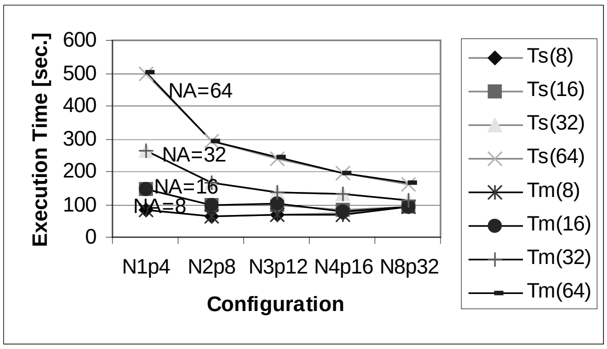


**Figure 7. Simulation and Measurement Results on SMP Nodes**

The presented results show that the relative error is less than 5% for all test cases. It is important to note that for the selected test cases a single execution of the program on the cluster requires from 15 minutes to 1 hour of running time, whereas the simulation takes less than 30 seconds, independently of the application input parameters.

The adoption of cost functions let the users to make complex analysis of the application and the system in just a few minutes. The LAPW0 application scales poorly due to load imbalance and large overheads due to loss of parallelism, data movement, and synchronization (see Figure 6. and Figure 7. ). Our methodology allows fast performance evaluation of LAPW0 for different problem and machine sizes. This makes it possible, for example, to select the best machine size for a given input dimension.

# 6 Conclusions

We have presented a methodology that can be adopted to analyze scientific application performance behavior on a high performance system. The approach has been illustrated by a study involving the execution of a complex parallel application on a SMP cluster.

The LAPW0 application was described with MetaPL, and the SMP cluster was modeled with HeSSE, resulting in an easy-to-use simulation model. The obtained model has been validated with a fixed set of measurements. The validation procedure shows that the model is valid for the chosen system configuration and can be adopted to gather performance information about the program execution on the cluster in a very fast way, without time-expensive execution of the real application on the real system.

The proposed technique is interesting because the tests needed to define and validate the model can be reproduced in a limited amount of time, while a complete analysis of the application using direct measurement needs long execution times, possibly days of computation.

In the future, we intend to validate our methodology for different applications and computer architectures. We also plan to investigate the possibility to improve the performance of an application by using the system and application model. Particularly promising seems the use of MetaPL along with the simulator to predict the impact of possible code changes without actually rewriting and executing the code.

# 7 References


[1] R. Aversa, A. Mazzeo, N. Mazzocca and U. Villano. Heterogeneous System Performance Prediction and Analysis using PS, *IEEE Concurrency*, Vol. 6, July-Sept. 1998, pp. 20-29.

[2] P. Blaha, K. Schwarz and J. Luitz. WIEN97, Full-potential, linearized augmented plane wave package for calculating crystal properties. Institute of Technical Electrochemistry, Vienna University of Technology, Vienna, Austria, 1999.

[3] W. Gropp et al. A high performance, portable implementation of the MPI message passing interface standard, *Parallel Computing*, vol. 22, n. 6, Sep. 1996, pp. 789-828.

[4] A. Gemund. Performance Prediction of Parallel Processing Systems: The Pamela Methodology, in *Proc. ACM*

*International Conf. On Supercomputing*, Tokoi, ACM, July 1993, pp. 318-327.

[5] S. Girona, J. Labarta and R. Badia. Validation of Dimemas communication model for MPI collective operations. *EuroPVM/MPI'2000*, Balatonfuered, Lake Balaton, Hungary, September 2000.

[6] K. Korcyl, F. Saka, and W. Dobinson. Modeling Large Ethernet Networks using Parameterized Switches, OPNETWORK 2000 Conference, Washington DC, 28 Aug.-1 Sept. 2000, available at: http://nicewww.cern.ch/korcyl/opnetwork/

[7] N. Mazzocca, M. Rak and U. Villano. The Transition from a PVM Program Simulator to a Heterogeneous System Simulator: The HeSSE Project, Recent Advances in Parallel Virtual Machine and Message Passing Interface, in J. Dongarra et al. (eds.), *Lecture Notes in Computer Science*, Vol. 1908, Springer-Verlag, Berlin 2000, pp. 266-273.

[8] N. Mazzocca, M. Rak and U. Villano. Predictive Performance Analysis of Distributed Heterogeneous System with HeSSE, to be published in: Proc. of ISCS 2001, Naples, Italy 2001, pp. 80-93.

[9] N. Mazzocca, M. Rak and U. Villano. MetaPL: a Notation System for Parallel Program Description and Performance Analysis, Parallel Computing Technologies, in Malyshkin. V. (eds.), *Lecture Notes in Computer Science*, Vol. 2127, Springer-Verlag, Berlin 2001. pp. 80-93

[10] N. Mazzocca, M. Rak and U. Villano. Parallel Program Development using the MetaPL Notation System, to be published in Proc. ParCo 2001 Conference, 4-7 September, Naples, Italy, 2001.

[11] N. Mazzocca, M. Rak and U. Villano. The MetaPL approach to the performance analysis of distributed software systems, to be published in Wosp 2002 Proc., July 24-26, 2002, Rome Italy.

[12] S. Pllana and T. Fahringer. On Customizing the UML for Modeling Performance-Oriented Applications. To appear in <<UML>> 2002, Dresden, Germany, September 30 - October 4, 2002.

[13] C. Smith and L. Williams. Performance Engineering Evaluation of Object-Oriented Systems with SPE-ED, in R. Marie, et al. Eds., *Lecture Notes in Computer Science* 1245, Computer Performance Evaluation Modelling Techniques and Tools, Springer, 1997.

[14] H. Truong, T. Fahringer, G. Madsen, A. Malony, H. Moritsch and S. Shende. On using SCALEA for Performance Analysis of Distributed and Parallel Programs. *Supercomputing 2001 Conference (SC2001)*, Denver, Colorado, USA, November 10-16, 2001.

[15] T. Fahringer and A. Pozgaj: P3T+: A Performance Estimator for Distributed and Parallel Programs. *Journal of Scientific Programming*, IOS Press, The Netherlands, 7(1), Nov. 2000.