

## **Prophesy: Automating the Modeling Process**\*

Valerie Taylor, Xingfu Wu, Jonathan Geisler, Xin Li, Zhiling Lan  
Department of Electrical and Computer Engineering  
Northwestern University, Evanston IL 60208  
Email: {wuxf,taylor}@ece.nwu.edu

Mark Hereld, Ivan R. Judson, Rick Stevens  
Mathematics and Computer Science Division  
Argonne National Laboratory, Argonne, IL 60439

### **ABSTRACT**

Performance models provide significant insight into the performance relationships between an application and the system, either parallel or distributed, used for execution. The development of models often requires significant time, sometimes in the range of months, to develop; this is especially the case for detailed models. This paper presents our approach to reducing the time required for model development. We present the concept of an automated model builder within the Prophesy infrastructure, which also includes automated instrumentation and extensive databases for archiving the performance data. In particular, we focus on the automation of the development of analytical performance models. The concepts include the automation of some well-established techniques, such as curve fitting, and a new technique that develops models as a composition of other models of core components or kernels in the application.

### **1. INTRODUCTION**

Performance models provide significant insight into the performance relationships between an application and the system used for execution. Such models can be analytical models, simulation-based models or statistical models. Each category of models can require significant time to develop, for which the time increases proportionally with the increase in level of detail. For example, to develop a good analytical model of a small kernel, such as the FFT, one needs to analyze the code, perform experiments to validate the model and refine the model as dictated by the validation process. Completion of this sequence of steps can require weeks or even months for a detailed model with system features exposed. As systems become more complex, as is the case with distributed systems, models are critical to understanding the performance relationships and identifying methods for improvement. Distributed systems are becoming available through programs such as the NASA Information Power Grid [JG99], the Alliance [AL], the National Partnership for Advanced Computational Infrastructure [NP], and the European Grid Effort [EG]. This paper describes our effort to automate the process of developing analytical models for parallel and distributed applications, so as to significantly decrease the time required for model development. We are developing an infrastructure, called Prophesy, which automates the

---

\* This research was supported in part by the National Science Foundation under NSF NGS grant EIA-9974960, a grant from NASA Ames and an NSF ITR grant.

performance analysis and modeling processes. By its design, Prophecy is a community tool, driven by databases that can be easily expanded via incremental effort from researchers in the community.

The Prophecy framework consists of three major components: data collection, data analysis, and three central databases, as illustrated in Figure 1. The data collection component focuses on the automatic instrumentation of codes at the level of basic blocks, procedures, or loops. The default mode consists of instrumenting the entire code at the level of loops and procedures. A user can specify that the code be instrumented at different levels of granularity or manually insert directives for the instrumenting tool to instrument specific segments of code. The resultant performance data is automatically placed in the performance database. This data is used by the data analysis component to produce an analytical performance model at the level of granularity specified by the user, or answer queries about best implementation of a given function. The models are developed based upon performance data from the performance database, model templates from the template database, and system characteristics from the systems database. These models can be used to predict the performance of the application under different system configurations. The Prophecy interface uses web technology to enable users from anywhere to access the performance data, add performance data, or utilize the automated instrumentation and modeling processes.

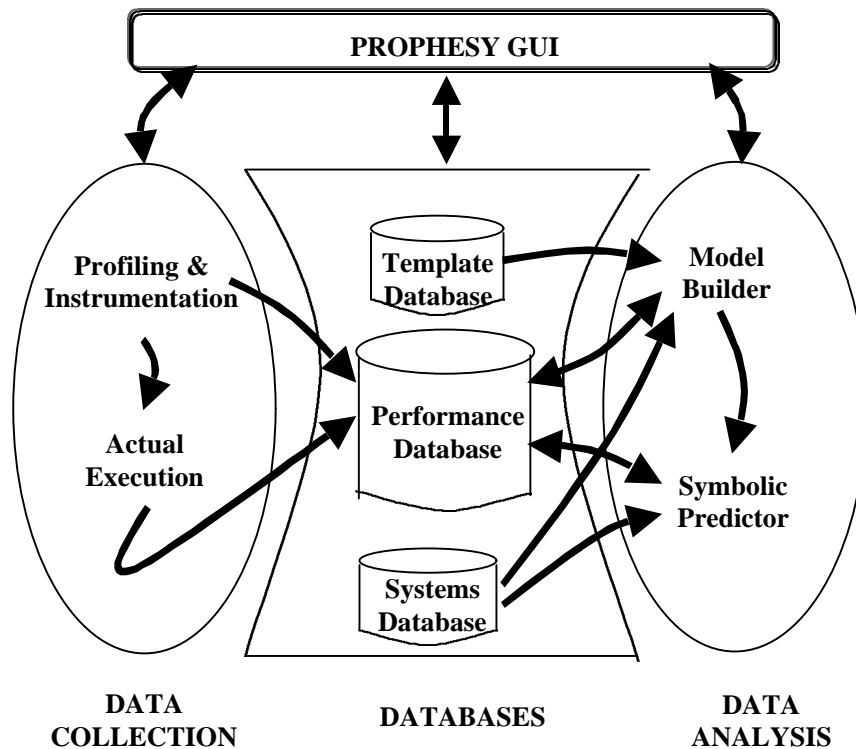


Figure 1. Prophecy framework

In this paper we focus on the data analysis component, in particular the model builder. Specifically, we focus on the automation of the development of analytical models. The modeling concepts include the automation of some well-established techniques, such as curve fitting, and a new technique that develops models as a composition of other models of the core components or kernels of an application. We present examples illustrating these different techniques. By having the modeling process automated, one can explore different models with ease.

The remainder of this paper is organized as follows. In Section 2 we discuss related work in the area of automated models. Section 3 presents some background of Prophecy as it relates to the automated modeling component followed by the details of this component in Section 4. Several examples of the automated modeling techniques are presented in Section 5.

## 2. RELATED WORK

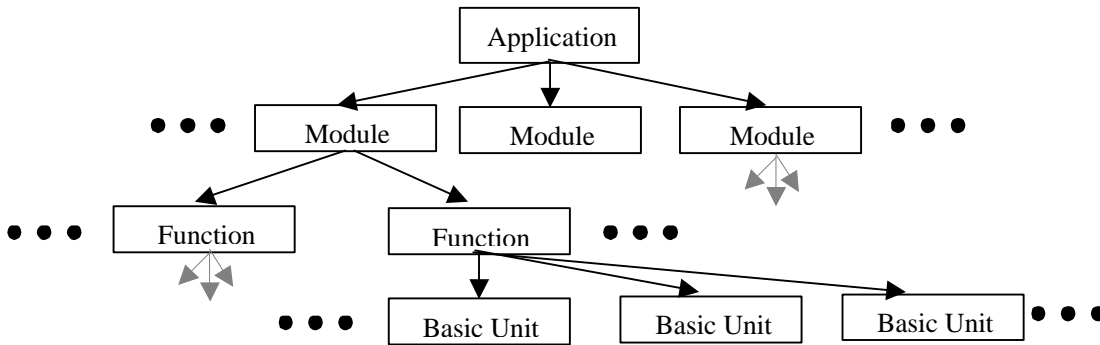
Significant work has been done in the area of automated performance analysis, but very little work with automating the process of developing models. While there is a very rich body of work related to prediction techniques, we focus on the automated methodologies for parallel and distributed applications. Dimemas [HS99] is a trace driven simulation that predicts the performance of message passing programs. Dimemas uses traces generated by VAMPIRtrace, an instrumented MPI library and API. The target architecture is characterized by a few parameters: the relative processor speed, linear communication model, and a simple model of network contention. In contrast, our focus is on the analytical models.

Liang and Tripathi [LT00] have developed a prediction method that is applicable to parallel applications. The method is based on a modified mean value analysis using iterative approach. Farhinger [FA96] also focuses on prediction of parallel program, with a focus on loops. His method, P3T, counts the number of loop iterations as a basis for estimating performance; the problem is generalized as computing the number of integer solutions to a set of inequalities. Davidson et. al. [AD98] have developed a modeling technique called MACS, which gives a lower bound on the computation time on a given machine. The technique uses the peak floating-point performance of a machine ( $M$ ) independent of the application, the essential operation in the compiler-generated schedule of the application workload. One can combine the MACS bounds with linear communication models to derive an overall performance bound for the application executed on a target machine. Mak and Lundstrom [ML90] developed a method for which the parallel computation is modeled as a directed acyclic task graph and the system is modeled as service center in a queuing network. Using these two models, the method uses an iterative algorithm to develop a performance prediction. Saavedra-Barrera and Smith [SS89] use the time required for a set of abstract operations on a given machine and the frequency counts of these operations in a program to estimate performance. All of these methods focus on predicting performance in contrast to automated model development, which is the focus of Prophecy.

Performance analysis environments, in particular PACE [KH96] and POEMS [PO], are being developed. These environments focus on performance predication. PACE represents the application, computational resource requirement and communication patterns in their CHIP<sup>3</sup> language. The CHIP<sup>3</sup> scripts are compiled and evaluated to generate a performance prediction very quickly. POEMS evaluates the end-to-end performance of a problem solving environment, consisting of application software, runtime and operating system software and hardware architecture. The analytical models with POEMS include deterministic task graph analysis, LogP [CK93] and LoGPC [MF98] models. These models are generally coarse grain, representing asymptotic performance. In contrast, the focus of our work is on detailed, analytical model development. Further, Prophecy complements the PACE and POEMS environments by providing a framework for developing models that can be added to their various libraries.

### 3. PROPHECY: BACKGROUND

In this section we present some details about Prophecy as it relates to the model builder component. See [TX01, XT01] for detailed information about the Prophecy database and automated instrumentation. Prophecy assumes that applications can be decomposed into modules, which can be further decomposed into functions that can be decomposed into basic units in a hierarchical manner as depicted in Figure 2. Viewing applications as this hierarchical composition of components is the basis for the composition algebra modeling technique we present in Section 4.



**Figure 2. Hierarchical structure of an application.**

Each component in the above structure has the following meaning:

- **Application:** refers to the complete large-scale application.
- **Modules:** refer to the various files that comprise the application; it is assumed that the application designer uses some modularity in the application design.
- **Functions:** refer to the different function routines that may be contained in a given module. Users will be asked to associate a "pure function" name with their given function where appropriate. For example, a user may identify their function "genfft" as the pure function FFT. Pure functions are widely used functions such as conjugate gradient or gaussian elimination.
- **Basic Units:** refer to a code segment that may be of smaller granularity than a function but higher granularity than a basic block. For example, a segment of nested loops would be considered one basic unit.

The Prophecy database also has a hierarchical organization, consistent with the hierarchical structure of the applications. The schema given below includes all three databases given in Figure 1: performance database, system models database and template database. The entities in the database are organized into four areas: application information, executable information, run information and performance statistics. Descriptions of these four areas are given below.

- **Application Information:** includes one entity that gives the application name, version number, a short description, and owner information and password (such that only the owner can modify or add data for a given application). Data is placed into this entity when a new application is being developed.

- **Executable Information:** includes all of the entities related to generating an executable of an application. These entities include details about compilers, libraries (compile time and run time) and the control flow. Data is placed into this entity when a new executable is being used.
- **Run Information:** includes all of the entities related to running an executable, which includes the system information and inputs used for execution. *This system may be a single processor, single parallel machine or distributed system.* Data is placed into these entities for each run of a given executable. Further, detailed information about different systems (e.g., processor performance, node memory subsystem, operating system, etc.) is contained in this area.
- **Performance Statistics Information:** includes all of the entities related to the raw performance data collected during execution. Performance statistics are collected for all levels of the application hierarchy.

#### 4. PROPHECY: AUTOMATED MODELING

Prophesy's automated modeling component allows models to be developed easily and stores relevant modeling information in the database for later use. This latter feature results in a modeling process that improves over time. Currently, Prophesy supports the following modeling techniques: curve fitting, parameterization, and composition methods. The first two methods are well-established techniques, for which Prophesy facilitates the methods via automation. The last method, composition methods, is enabled by Prophesy because of the significant amount of archival data and the automation of the modeling process. Details about the Prophesy support for each method are given below.

##### 4.1 Curve Fitting

Curve Fitting is a method that uses optimization techniques, e.g., least squares, to develop a model. For this method, Prophesy uses the empirical data found in the database and the computational complexity of the application to generate the model. The computational complexity is determined by the complexity of the dominating pure function in the application. The complexities of the different pure functions are stored in the Prophesy database. The empirical data to be used for the fit is determined by the user via the Prophesy data query facility. Then Matlab is used to generate the resultant model. The advantage of this method is the ease for which the analytical model is generated; the disadvantage is the lack of exposure of system terms versus application terms. The models resulting from curve fitting are generally a function of some input parameters of the application; the system performance (e.g., operation execution time or communication performance) is lumped into the coefficients determined by curve fitting. Hence, models resulting from curve fitting can be used to explore application scalability but not different system configurations.

##### 4.2 Parameterization Method

Parameterization is a method that combines manual analysis of the code with system performance measurements. The manual analysis entails hand-counting the number of different operations in the code. It is assumed that this type of analysis is done on kernels or functions that are generally in the range of 100 lines of code or less. With Prophesy, the manual analysis is used to produce an analytical equation with terms grouped together such that the equation is a function



### 4.3 Composition Method

The composition modeling technique focuses on how to effectively represent the performance of an application in terms of its component kernels or functions. Hence, this technique focuses on the level of functions, for which most applications have only a few major functions; examples of functions include FFT or matrix-vector multiplication. The major obstacle to developing performance models of an application is the lack of knowledge about the performance relationships between the different functions that compose an application. Further, it is necessary to capture this relationship as a coefficient that can be used in an equation. For example, assume we have an application that is composed of *kernelA* followed by *kernelB*, and the application iterates on a loop that contains these two functions until some condition is satisfied. Also assume that we have manually analyzed these two functions such that we have *modelA* and *modelB*, analytical models of the two kernels. A *composition algebra* would provide the coefficients of the relationships between the two kernels, represented below where T is the execution time of the application:

$$T = \mathbf{h} \text{ modelA} + \mathbf{k} \text{ modelB}$$

where coefficients **h** and **k** represents the performance relation between the two kernels that identifies how they should be combined to reflect the performance of the application.

Our approach to quantifying this relation is based upon our previous work on *coupling* [GE99]. Performance *coupling*,  $C_{ij}$ , refers to the effect that kernel *i* has on kernel *j* in relation to running each kernel in isolation. The two kernels can correspond to adjacent kernels in the control flow of the application or kernels that are some distance apart. We categorized the coupling into one of three sets based upon the value of  $C_{ij}$ :

- $C_{ij} = 1$  indicates no interaction between the two kernels.
- $C_{ij} < 1$  indicates constructive interaction resulting from the effective sharing of some resource between the two kernels; this type of coupling results in performance gains in the full application.
- $C_{ij} > 1$  indicates destructive interaction resulting from conflicts or the two kernels interfering with each other; this type of coupling results in performance losses in the full application.

The values for the coefficients are generated in the following manner. First, three measurements must be taken as described below:

- $P_i$  is the performance of kernel *i* alone,
- $P_j$  is the performance of kernel *j* alone
- $P_{ij}$  is the performance of kernel *i* followed by kernel *j*

The coefficient,  $C_{ij}$  is equal to the ratio of the measured performance of the pair of functions to the expected performance resulting from combining the isolated performance of each function. This performance may be cache misses, which was the focus of previous work, or execution time, which is the focus of current work with Prophecy. The coupling coefficient has the following equation:

$$C_{ij} = \frac{P_{ij}}{P_i + P_j}$$

In our previous work we used coupling to identify parts of the application that required performance improvement [GE99, GT99]. In particular, the coupling value provided insight into where further algorithm and code implementation work was needed to improve performance. The performance improvement focused on improving the reuse of data between kernels. The goal was to produce transformations that result in minimal coupling values. The coupling value was used to produce transformations that resulted in performance improvements for the CG and BT benchmarks in the NPB suite and one of the SPECjava benchmarks. Additional work has been done with the coupling values to provide performance improvements with multithreaded applications executing on the Cray MTA machine.

We extend our previous work on coupling to use it to quantify the performance relationship between the kernels to represent the performance of an application. The values for  $C_{ij}$  are stored for each pure function pair in the database and used to determine how to combine the parameterized models of these functions. As mentioned previously, the number of pairs is bounded as we believe the number of kernels or pure functions is very small. Performance models are generated based upon the control flow information about the pure functions in the application. The coefficients of the different kernel models are generated as a weighted sum of the coupling values. This is illustrated with the example below.

Assume that an application has three kernels (A, B, C) that are executed together in one loop. Further assume we have the coupling values for the kernel pairs ( $C_{AB}$ ,  $C_{BC}$ ,  $C_{AC}$ ) and the parameterized models for A, B, C (given as  $E_A$ ,  $E_B$ , and  $E_C$ ). The equation for the application is given as:

$$T = \mathbf{h} E_A + \mathbf{k} E_B + \mathbf{y} E_C$$

The coefficients of the models have the following equations :

$$\mathbf{h} = [(C_{AB} * P_{AB}) + (C_{AC} * P_{AC})] / (P_{AB} + P_{AC})$$

$$\mathbf{k} = [(C_{AB} * P_{AB}) + (C_{BC} * P_{BC})] / (P_{AB} + P_{BC})$$

$$\mathbf{y} = [(C_{AC} * P_{AC}) + (C_{BC} * P_{BC})] / (P_{AC} + P_{BC})$$

The use of the weighted averages of the coupling values was validated with synthetic benchmarks consisting of array accesses that allowed us to vary the resultant coupling values. The use of the weighted average assumes, that when the coupling values were generated, the number of times a kernel is executed is fixed. Further, it is assumed that the different runs needed to generate all of the coupling values have the same input. We illustrate this technique with a detailed example in the following section.

#### 4. EXAMPLES

This section provides examples that illustrate the three modeling techniques currently support by Prophesy. The examples utilize a matrix-matrix multiplication kernel and an FFT from the NPB suite [BH95].

## 4.1 Curve Fitting Method

To illustrate the curve fitting method, we use a matrix-matrix multiplication kernel executed on a SGI Origin 2000. The kernel was executed for different problem sizes, with the matrix rank ranging between 100 and 1000. Figure 4 compares the experimental results with those of the model generated using curve fitting to the execution times for the matrix rank between 100 and 800; the modeling technique used the fact that this kernel has cubic complexity in terms of the matrix rank. We used the large range since this would be indicative of a one using the largest amount of data available for curve fitting. The results indicate that this methodology provides very good modeling of the data. These models can be used to predict the execution time of the machines for a given matrix rank. For the case where one uses the data from different machines to generate models using curve fitting, the resulting models can be used to provide feedback to the user on a recommendation for the existing machine that would perform best for the given the implementation.

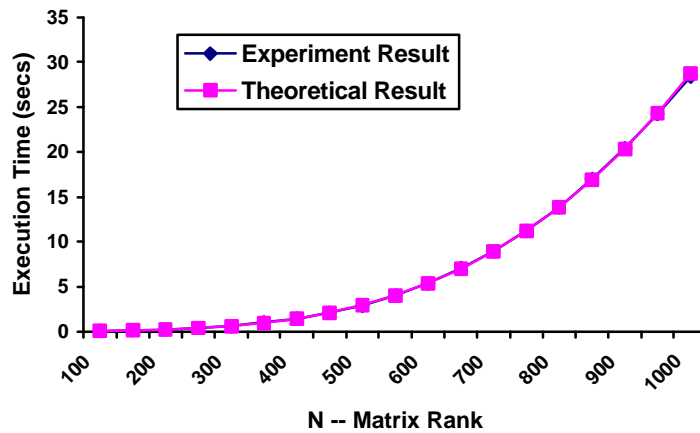


Figure 4. Curve fitting for SGI Origin using 8 processors.

## 4.2 Parameterization Method

In this example, we illustrate the use of Prophecy's parameterization method option to generate the performance model of the same matrix-matrix multiply kernel for the SGI Origin at NU. This method uses information found in the systems and template databases. Recall from the previous example that a cubic template is used (i.e., the template being  $\alpha_1 N^3 + \alpha_2 N^2 + \alpha_3 N + \alpha_4$ , where N is the rank of the matrix), as reflected in the template database; this database also contains the scripts used to generate the coefficients for template. These scripts were determined manually by detailed analysis of the code segments. The values for  $\alpha_1$ ,  $\alpha_2$ ,  $\alpha_3$  and  $\alpha_4$  are determined by entries found in the systems database for the SGI Origin at NU. In particular, the parameterization modeling method uses the entries for the communication times for the MPI broadcast, send and receive operations and well as the entries for some core computations to generate the values for  $\alpha_1$ ,  $\alpha_2$ ,  $\alpha_3$  and  $\alpha_4$ . The communication values were obtained using the MPPTTEST performance test suit

[MP99]. For the core computations, the method uses the entries for performing the following operations: (1) integer add ( $a + b$ ) for array indexing, (2) integer multiply add ( $a + b * c$ ) for multi-dimensional array indexing and (3) floating-point multiply add ( $a + b * c$ ).

The results of using the automated parameterization modeling with Prophecy, along with a comparison to the empirical data, are given in Figure 5. The analytical model generated with Prophecy is very good in predicting the performance of the matrix-matrix multiply kernel on the SGI Origin.

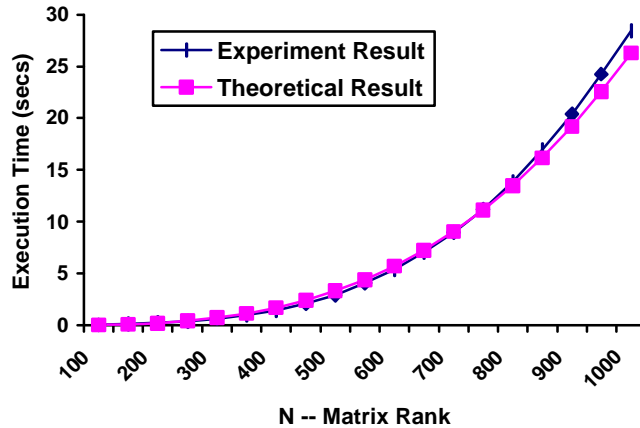


Figure 5: Parametric modeling for the SGI Origin, using 8 processors.

### 4.3 Coupling Method

To illustrate the coupling modeling method we use the FFT from the NPB suite. This benchmark contains three major functions:

- **cffts1**: performs fft on the first dimension, with blocking on the second dimension
- **cffts2**: performs fft on the second and third dimensions, with blocking on the second dimension
- **transpose**: performs a transpose along a given dimension

These three functions are used to move forward and then reverse through the butterfly network to generate the FFT. This example will focus on the reverse computation and its execution on the 8-processor SGI Origin at Northwestern University. The call sequence consists of a loop performing the following four steps for each iteration:

1. cffts1
2. transpose
3. cffts2
4. cffts1

First, performance models were generated manually for each of the three kernels and stored in the template database. These models consisted of parameterization models that were validated using Prophecy. Next, the coupling values were generated for each pair of the three kernels. These values are given below in Table 1.

Kernel Pair	Coupling Value
cffts1 $\leftrightarrow$ transpose	1.00
tranpose $\leftrightarrow$ cffts2	1.00
cffts2 $\leftrightarrow$ cffts1	1.00
cffts1 $\leftrightarrow$ cffts1	1.00

**Table 1: Coupling values for FFT kernel pairs**

Using the values in Table 1 , we obtained the following analytical model:

$$T_{\text{reverse\_fft}} = 1.0 E_{\text{cffts1}} + 1.0 E_{\text{cffts2}} + 1.0 E_{\text{transpose}} + 1.0 E_{\text{cffts1}}$$

where  $E_{\text{cffts1}}$ ,  $E_{\text{cffts2}}$  and  $E_{\text{transpose}}$  represent the parameterized models of the respective functions. The results from using this model as compared to the experimental data are given below in Table 3 for the three datasets that come with the NPB suit, described in Table 2, executed on 2, 4, and 8 processors of the SGI Origin. The results indicate that this model is a good indicator of performance. It should be noted that the NPB FFT has very little code between the function calls, resulting in a very good model. Furture work will focus on codes with significant code between major functions or kernels.

FFT Data Set	Size
S	64 x 64 x 64
W	128 x 128 x 32
A	256 x 256 x 128

**Table 2: Data sets used with the NPB FFT.**

Data Set	2 Processors		4 Processors		8 Processors	
	Exper.	Theor.	Exper.	Theor.	Exper.	Theor.
S	4.813s	5.2951s	2.4015s	2.6478s	1.2276s	1.324s
W	11.3602s	10.9507s	5.5614s	5.4754s	2.811s	2.7379s
A	208.4397s	198.2968s	103.4339s	99.1485s	52.8934	49.5744s

**Table 3: Experimental versus theoretical results for the NPB FFT.**

## 5. SUMMARY

This paper presents our approach to automating the process of developing analytical performance. We present the concepts of the automated model builder within the Prophecy infrastructure, which also includes automated instrumentation and extensive databases for archiving the performance data. In particular, we focus on the automation of the development of analytical performance models. The concepts include the automation of two well-established techniques, curve fitting and parameterization, and a new technique that develops models as a composition of other models of core components or kernels in the application. We provided examples illustrating how Prophecy automates each of the three techniques. It is our belief that Prophecy can be used to generate a composition algebra that identifies how the performance of individual kernels should be composed to represent the application performance.

Future work with this project includes advertising to different application communities to get the database populated with the performance data. This data will be used to further explore the composition technique. We are also developing techniques for effectively including the code between kernels in an application. The Prophecy system is available for use at [prophecy.mcs.anl.gov](http://prophecy.mcs.anl.gov).

## ACKNOWLEDGEMENTS

The authors would like to acknowledge the Center on Parallel and Distributed Computing at Northwestern University for the use of the SGI Origin used with the case studies.

## References

- [AD98] Gheith A. Abandah and Edward S. Davidson, Characterizing Distributed Shared Memory Performance: A Case Study of the Convex SPP1000, *IEEE Transactions on Parallel and Distributed Systems* 9(2): 206-216 (1998).
- [AL] The Alliance, <http://www.alliance.uiuc.edu/>.
- [BD00] S. Browne, J. Dongarra, N. Garner, K. London, and P. Mucci, A Scalable Cross-Platform Infrastructure for Application Performance Tuning Using Hardware Counters. In *Proc. SC 2000*, November 2000.
- [BH95] D. Bailey, T. Harris, et al., *The NAS Parallel Benchmarks*, Tech. Report NAS-95-020, Dec. 1995. See also <http://science.nas.nasa.gov/Software/NPB/>.
- [BW96] F. Berman, R. Wolski, S. Figueira, J. Schopf and G. Shao, Application-level scheduling on distributed heterogeneous networks. In *Proc. of Supercomputing*, 1996.
- [CF98] K. Czajkowski, I. Foster, N. Karonis, et. al., A resource management architecture for metasystems. *Lecture Notes on Computer Science*, Springer-Verlag, D. Feitelson and L. Rudolph, editors, 1998.
- [CF99] K. Czajkowski, I. Foster and C. Kesselman, Resource co-allocation in computational grids. In *Proc. IEEE International Symposium on High Performance Distributed Computing*, 1999.
- [CK93] D. Culler, R. Karp, D. Patterson, et al., LogP: Towards a realistic model of parallel computation. In *Proc. of 4<sup>th</sup> ACM SIGPLAN Conf. on Para. Prog. Pract. and Exp.*, 1993.
- [CO] The Condor Project, <http://www.cs.wisc.edu/condor/>.
- [EG] The European Grid, <http://www.ggf1.nl/>.

- [FA96] Thomas Fahringer, Toward Symbolic Performance Prediction of Parallel Programs, *IEEE Proc. of the 1996 International Parallel Processing Symposium*, pages 474-478, Honolulu, Hawaii, April 15 - 19, 1996.
- [GE99] J. Geisler, *Performance Coupling: A Methodology for Analyzing Application Performance using Kernel Performance*, MS Thesis, Northwestern University, March 1999.
- [GL94] W. Gropp, E. Lusk and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, MIT Press, Cambridge, MA, 1994.
- [GT99] J. Geisler and V. Taylor, Performance coupling: A methodology for predicting application performance using kernel performance, In *Proc. of the 9<sup>th</sup> SIAM Conference on Parallel Processing for Scientific Computing*, March 1999.
- [GL] The Globus Project, <http://www.globus.org/>.
- [HB94] R. Hockney and M. Berry, *Public International Benchmarks for Parallel Computers*, PARKBENCH Committee: Report-1, February 7, 1994.
- [HS99] Hans-Christian Hoppe and Sergi Girona, "Performance Prediction with Dimemas," *EuroTools Workshop on Performance Measurement and Prediction HPCN'99*, Amsterdam, The Netherlands.
- [JG99] W. Johnston, D. Gannon and B. Nitzberg, Grids as production computing environments: the engineering aspects of NASA's Information Power Grid. In *Proc. International Symposium on High Performance Distributed Computing*, 1999.
- [KH96] D. Kerbyson, J. Harper, et al., PACE: A toolset to investigate and predict performance in parallel systems. In *Proc. of European Parallel Tools Meeting*, Oct. 1996.
- [KR82] D. R. Kincaid, J. R. Respass, D. M. Young, and R. G. Grimes, Itpack 2c: A Fortran package for solving large sparse linear systems by adaptive accelerated iterative methods, *ACM Tran. On Mathematical Software* 8, 1982, 302-322.
- [LA93] B. H. LaRose, *The Development and Implementation of a Performance Database Server*, Master Thesis, University of Tennessee at Knoxville, August 1993.
- [LT00] De-Ron Liang and Satish K. Tripathi, On Performance Prediction of Parallel Computations with Precedent Constraints, *IEEE Transactions on Parallel and Distributed Systems* May 2000 (Vol. 11, No. 5), pp. 491-508.
- [LE] The Legion Project, <http://www.cs.virginia.edu/~legion/>.
- [MA99] *Matlab: The Language of Technical Computing*, The Math Works, Inc., 1999, Natick, MA.
- [MC95] B. P. Miller, M. D. Callaghan, et al., The Paradyn parallel performance measurement tools, *IEEE Computer*, Vol. 28 (11), Nov. 1995.
- [MF98] C. A. Moritz, and M. I. Frank, LoGPC: modeling network contention in message-passing programs. In *Proc. of Intern. Conf. on Meas. and modeling of computer systems*, June 1998.
- [ML90] Victor. W. Mak, Stephen F. Lundstrom, Predicting Performance of Parallel Computations, *IEEE Traction on Parallel and Distributed Systems*, Vol. 1, No. 3, July 1990.
- [MP99] MPPTTEST --- Measuring MPI Performance, <http://www.mcs.anl.gov/mpi/mpptest> .
- [NP] The National Partnership for Advanced Computational Infrastructure, <http://www.npaci.edu/>
- [PO] POEMS Performance Environment, <http://www.cs.utexas.edu/users/poems>.
- [RA93] D. A. Reed, R. A. Aydt, et al., Scalable performance analysis: The Pablo performance analysis environment. In *Proc. of the Scalable Parallel Libraries Conference*, Oct. 1993.
- [S99] W. Smith, Resource Management in Metacomputing Environments, *PhD Thesis*, Northwestern University, December 1999.
- [SG93] S. R. Sarukkai, and D. Gannon, SIEVE: A performance debugging environment for parallel program, *Journal of Parallel and Distributed Computing* 18, 1993, 147-168.
- [SN88] R. Snodgrass, A relational approach to monitoring complex systems, *ACM Transactions on Computer Systems*, Vol. 6, No. 2, May 1988, 157-196.

- [SS] SpeedShop User's Guide, DocumentNumber 007-3311-003, Silicon Graphics, Inc.
- [SS89] Rafael H. Saavedra-Barrera, Alan Jay Smith and Eugene Miya, Machine Characterization Based on an Abstract High-Level Language Machine, *IEEE Transactions on Computers* 38(12): 1659-1679 (1989).
- [TR95] V. Taylor, A. Ranade, and D. Messerschmitt, SPAR: A new architecture for large finite element computations, *IEEE Tran. on Computers*, 44(4), April 1995, 531-545.
- [TX01] V. Taylor, X. Wu, J. Geisler, Z. Lan, X. Li, R. Stevens, M. Hereld, I. Judson, "Design and Development of Prophecy Performance Database for Distributed Scientific Applications," *Proceedings of the SIAM Conference on Parallel Processing, March 2001*.
- [XT01] X. Wu, V. Taylor, R. Stevens, "Design and Implementation of Prophecy Automatic Instrumentation and Data Entry System", to appear in the *Parallel and Distributed Computing and Systems Conference (PDCS2001)*.
- [YS95] J. C. Yan, S. R. Sarukkai, and P. Mehra, Performance measurement, visualization and modeling of parallel and distributed programs using the AIMS toolkit, *Software Practice and Experience*, Vol. 25 (4), April 1995.