IPERF : A Framework for Automatic Construction of Performance Prediction Models

Chung-Hsing Hsu and Ulrich Kremer*

Department of Computer Science
Rutgers University

Abstract

Performance prediction models at the source code level are crucial in optimizing compilers, programming environments, and performance debugging tools. For each performance prediction task, minimal-cost models are needed that deliver the required accuracy. Current techniques to derive the desired models are *ad hoc*. This paper discusses a new framework for automatic construction of cost-effective performance prediction models for different target systems at the program source level. A target system consists of a target optimizing compiler, a target operating system, and a target architecture with a multi-level memory hierarchy. Preliminary results for a small computation kernel on a set of twelve target systems indicate the effectiveness of the proposed framework.

1 Motivation

Performance prediction models[1] are needed in the source code optimization process. The ideal model to support an optimization transformation is *accurate* and *cost-effective*. An accurate model ranks the optimization alternatives correctly according to their expected performance benefits on a target system. Since a more detailed performance model is typically more expensive to compute, the ideal performance model is also *minimal*, i.e., may ignore target system components that do not significantly contribute to distinguish the optimization alternatives. Unfortunately, finding the best performance model for an optimization task is extremely hard, since it may require an in-depth understanding of all components of the target system, including other optimizations performed by the target compiler and their interactions with the features of an advanced operating system and modern computer architecture. In fact, finding a model may be even impossible, since a model with the required accuracy and cost-effectiveness may not exist. What is needed are tools to support the difficult task of finding an adequate performance model.

The need for software support for finding accurate and cost-effective performance models has become even more apparent in the light of recent trends in architecture and language design. For example, the success of Intel's and Hewlett Packard's new IA-64 architecture (Merced) will depend on the ability of optimizing compilers to take advantage of the parallelism provided by the machine, putting an even greater burden on the compiler to achieve efficient program execution than for current superscalar architectures. Another new challenge for optimizing compilers are dynamic compilation strategies for languages such as Java. Dynamic optimizations are performed at program execution time and therefore have more severe compile time constraints than static compilation systems.

In this paper, we propose a framework, called IPERF (Intelligent PERFormance Prediction), for *automatic* construction of performance models for a requested trade-off between prediction accuracy and cost. The main idea of the framework is to provide a data base of performance models of individual components of a target system, and heuristics that are able to select the best combination of the component models that, for a given optimization task, will result in an accurate and cost-effective overall model. Particular implementations based on the IPERF framework may represent different component models in their data bases or use different sets of model search heuristics. The main contribution of the framework is its infrastructure and strategy to

*C-H. Hsu, email: chunghsu@cs.rutgers.edu; U. Kremer (*corresponding author*), email: uli@cs.rutgers.edu, phone: (732) 445-4974; address: Department of Computer Science, Hill Center, Busch Campus, Rutgers University, Piscataway, NJ 08855

[1]Abbreviated as performance models or models throughout the paper.

find performance models of a requested accuracy and cost-effectiveness. The framework is motivated by the observations that (1) performance models for the same prediction task may be different for different target systems, (2) using the most accurate model to support a compiler optimization may not be necessary or even possible due to compile-time constraints, and (3) advanced optimization transformations and architecture features may actually make the performance models simpler, not harder. In other words, the complexity of the performance model is not necessarily increasing with the complexity of the target system. For instance, in the presence of aggressive data locality optimizations, the desired performance model may not have to distinguish cache misses and hits, i.e., may assume that all data references have the same cost.

The IPERF framework is based on the assumption that the observed program performance, i.e., its execution time, can be attributed to different components, each of which represents a different aspect of the target system. Each such aspect is called a *performance factor*. Examples of performance factors on a target system include CPU performance, cache performance, and network performance. In the context of this paper, a target system is a target compiler consisting of a set of optimization passes, a target operating system, and a target architecture with a multi-level memory hierarchy. In IPERF , each performance factor has an associated set of models. The goal is to construct a linear combination of *terms*, each of which is a model for a particular performance factor, to satisfy the user's precision and cost requirements.

The IPERF framework has a data base of performance factors and models. Heuristics will be used to systematically search through the data base, using the analysis of the source program and the description of the target system. These heuristics select appropriate performance factors and models, and report satisfying models if they exist. User supplied models for a performance factor may be inserted into the data base on request, allowing the evaluation of the model as part of a performance prediction model for the entire system.

Performance prediction in the context of a specific optimization has been extensively discussed in the literature. Examples include finding the best loop order of a loop nest to take advantage of the target machines memory hierarchy [8, 7, 20, 15], or determining an efficient computation and data mapping onto a multiprocessor system [10, 2, 5, 9, 13]. Nearly all published models are given and are different in terms of their cost and accuracy. It is hard to compare their efficiency and effectiveness across different optimizations. The IPERF framework provides a platform to evaluate them automatically.

More systematic approaches of constructing performance models have been investigated by a small group of researchers. Brewer's *auto-calibration toolkit* constructs from a list of user provided models an additive model through linear regression over performance profiles. The cost of the model is reduced by repeatedly eliminating statistically insignificant terms in the additive model [4, 3]. The toolkit has no explicit cost metric and relies on the user to provide the models. Crovella and LeBlanc developed the *lost cycles toolkit* that constructs models for a fixed set of performance factors individually through linear regression, and then sum them up [14]. The effectiveness of the toolkit depends on the independence and completeness of these performance factors. Note that Brewer's toolkit and Crovella and LeBlanc's toolkit include the automatic generation to profiling codes, while IPERF relies on the user to provide the performance profile of a good quality. Saavedra-Barrera developed a *micro-benchmarking* framework that constructs the additive model from a fixed set of more than hundred terms, each of which is derived by linear regression of an appropriate set of micro-benchmarks [17]. The portability of this framework is limited. However, Saavedra-Barrera's framework does consider the optimization passes implicitly, by reducing the cost of each term. The IPERF framework proposes explicit models for compilation passes. The effectiveness of explicit models for compilation passes has been illustrated by our previous work on performance prediction for automatic data layout [13], and by other researchers, such as Ko-Yang Wang in the context of instruction scheduling for superscalar architectures [19]. More recently, Emer and Gloy have investigated generic algorithms to generate new models for the purpose of branch prediction [6]. Instead of working with a predefined model space, their method may generate new performance factors.

The paper is organized as follows. Section 2 presents the overall structure of the proposed framework. Then the framework is instantiated and illustrated in Section 3 through a a simple benchmark set on twelve different target systems. The future work is discussed in Section 4.

2 IPERF : The Framework

The proposed framework, called IPERF (Intelligent PERFormance Prediction), takes as input a target system specification, a benchmark set of program representations with their actual performance measure-

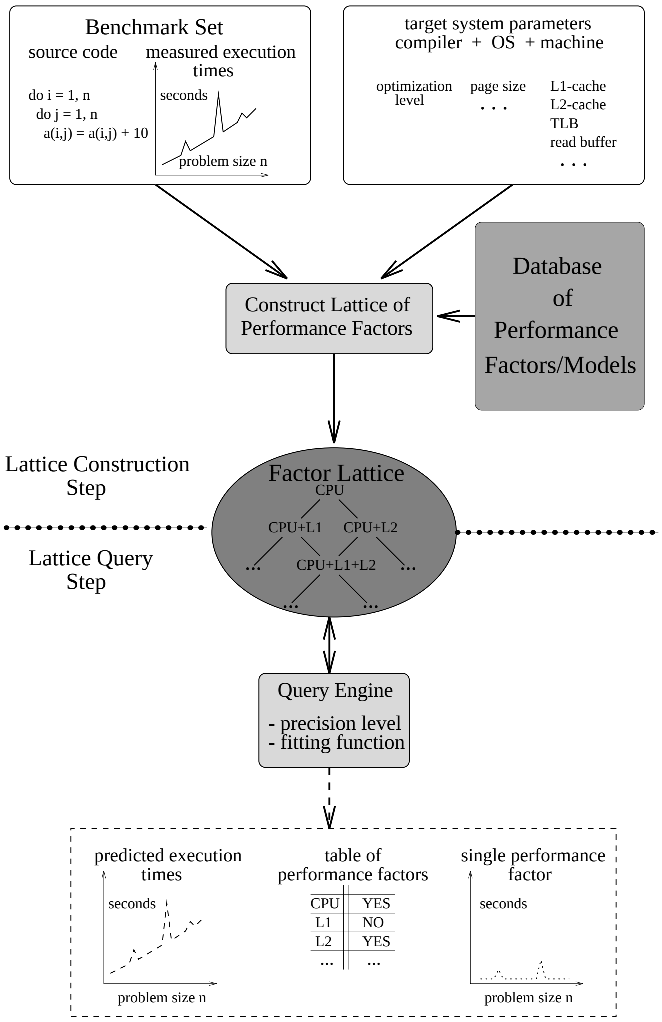

Figure 1: Overview of IPERF .

ments or performance ranking, and a set of precision and cost requirements. Based on an analysis of the source code and the specification of the target system, the framework will first determine a lattice of performance factors, typically containing only a subset of the performance factors represented in the data base. Once the lattice has been constructed, the precision and cost requirements will direct the query engine to search for the minimal performance model that satisfies the requested precision level. The search process may consider only a subset of the data base's models for each performance factor in the lattice. The output is the minimal-cost performance model that approximates the measured behavior of the benchmark set within the specified precision and cost constraints, assuming that such a model can be found. The constructed performance model may be presented in different ways, for instance, as a graph of the predicted performance or as a table of the performance factors of the model. In addition, the predicted performance of a single performance factor or subset of performance factors may be displayed. Figure 1 shows the overall structure of IPERF .

The framework uses a data base of performance factors and models for different aspects of the target system. There are compiler, operating system, and machine architecture performance factors. The data base may contain several models with different accuracy/cost trade-offs for each performance factor. Performance factors are classified as either *compositional* or *numerical*. Models for compositional factors map program representations into other program representations, possibly lowering the level of program abstraction. In contrast, numerical models map program representations into numerical values. Examples of compositional performance factors are compiler optimization passes that perform program transformations such as loop interchange, register allocation, or instruction scheduling. Numerical performance factors may include cost models for computation, TLB misses or hits, and the target machine's memory hierarchy. The proposed framework assumes that all compiler models are compositional, and all operating system and machine architecture models are numerical. In addition, the framework assumes that the set of all potential performance factors are known for a given target system, including all performed target compiler optimizations and their relative orders, and operating system and target machine characteristics such as page size, cache sizes, cache architecture, number of functional units etc. The final generated model may not consider all performance factors. Figure 2 shows a sample IPERF data base of performance factors and their associated models. The data base is organized as a tree, where nodes represent performance factors, and edges represent a refinement relation between factors. If a performance factor is selected, it can either be represented by a model directly associated with its node, or by a linear combination of models for its children in the tree. For instance, an enumeration of all possible models for factor CACHE is shown in Figure 3.

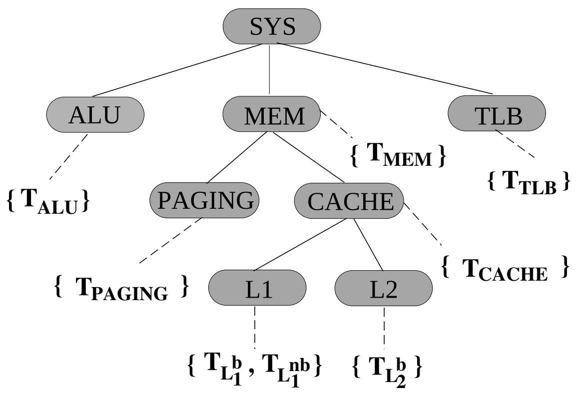

Figure 2: Structure of a sample IPERF data base.

The subset relation over possible linear combinations of performance factors can be represented by a canonical subset lattice. The lattice defines a partial order with respect to the precision of models for subsets of performance factors. An example relation is shown in Figure 4. The sample subset of performance factors consisting of computation (ALU), virtual address translation (TLB), and the memory hierarchy (MEM) forms a canonical subset lattice of all possible linear combinations. If modeling each performance factor is assumed to be of unit cost, the

Models

no.		cost
1	$c_0 * \mathbf{T_{L_1^b}}$	1
2	$c_0 * \mathbf{T_{L_1^{nb}}}$	1
3	$c_0 * \mathbf{T_{L_2^b}}$	1
4	$c_0 * \mathbf{T_{CACHE}}$	1
5	$c_0 * \mathbf{T_{L_1^b}} + c_1 * \mathbf{T_{L_2^b}}$	2
6	$c_0 * \mathbf{T_{L_1^{nb}}} + c_1 * \mathbf{T_{L_2^b}}$	2

Figure 3: All possible models for factor CACHE.

lattice also represents the cost relation between the different models, with the most precise model being also the most expensive. since it considers all performance factors. Note that in practice, this unit cost assumption may not hold.

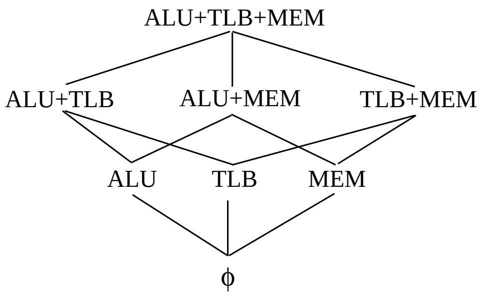

Figure 4: Canonical precision and cost subset lattice of three sample performance factors.

The framework requires that all generated numerical performance model consist of linear combinations of performance models of different numerical performance factors. Models for a single numerical performance factor may be non-linear.

A core component of the proposed framework is the search algorithm that exploits the lattice structures of the factors in the data base, and selects a subset of performance factors and their models that best fits the user requested accuracy/cost constraints. It is important to note that the particular classification of a compiler, operating system, or machine characteristic as a separate performance factor or a model parameter for some other performance factor is not part of the framework, but rather part of its implementation. The proposed framework consists of the following three major steps:

1. Pruning of possible performance factors and

their models based on an analysis of the characteristics of the benchmark set.

2. Searching the lattice of performance factors and their models, and evaluating each resulting performance model by comparing predicted values against measured data in the benchmark set.

3. Reporting single or set of performance factors and their models that best fits the user requested accuracy/cost trade-off. A failure is reported if no such models can be found.

3 Experiments

A prototype system [2] of the framework has been evaluated for a simple benchmark set and twelve different target systems. The prototype evaluation was performed based on a combination of automated and hand-simulated steps. The benchmark set contained programs represented at the source level, and the measurements were actual execution times rather than more abstract ranking functions. With the prototype system, we were able to

- determine different performance models for different precision requirements for the *same* target system.
- determine different performance models across different target systems for the *same* precision requirement.
- discover a deficiency of the cache model for one target architecture. This discovery led to the design of a new cache model. The system was also used to evaluate the benefits of the updated cache model in the context of other performance factors.
- discover a performance factor within the operating system by failing to produce a performance model of the requested precision.

Input/Output A small computation kernel (shown in Figure 5) with non-unit stride memory accesses is used to illustrate the design and functionality of the IPERF framework. Although the computation kernel is simple, understanding the actual performance behavior on a set of different target systems turns out to be nontrivial. The kernel consists of a doubly nested loop that steps through a double precision array of size $n*n$ with stride n, and increments each element by 1.0.

[2]The prototype system is a particular, rather limited *implementation* of the IPERF framework and has been designed to illustrate the feasibility of the framework.

```
     double precision a(n,n)
     do 10 i=1,n
        do 10 j=1,n
10         a(i,j) = a(i,j) + 1.0
```

Figure 5: Small example kernel (non-unit stride)

We assume that a user needs to determine a performance model that describes the performance behavior of the example kernel for different array sizes n on a specific target architecture with a specific prediction accuracy requirement. In addition, the user wants to understand the cost/precision tradeoffs of different performance models. The current IPERF system expects as input a set of measured execution time data points as a function of a single parameter, in our example case the array size represented by variable n. In addition, the source code of the measured kernel, a specification of the target system used, and the performance prediction requirements are input to the tool. It is the responsibility of the user to provide a data set that exhibits the performance behavior that the user is trying to understand. Therefore, the choice of such a "representative" data set is outside the scope of IPERF .

In order to illustrate the IPERF system, we measured the execution times of the example kernel for the set of data points $\{n | \; n = 8i, 1 \leq i \leq 100\}$ on twelve different target systems. For the experiments, a target system consisted of a *(target compiler / target machine)* pair, where the compiler was SUN's SPARC compiler version 4.0 using three different optimization levels (`-none`, `-O3`, and `-fast -depend`), and one out of four possible SUN SPARC–based target machines (SuperSPARC–I and II, UltraSPARC–I and II). Optimization level `-O3` performs classical local and global optimizations and level `-fast -depend` performs in addition data dependence analysis and loop restructuring optimizations such as loop interchange. In all cases, the operating system was Solaris 2.5.1. The resulting measured execution times are shown in Figure 6.

Figure 6 shows significantly different performance behavior for most of the target systems. The exact description of the method used to obtain the measured performance numbers can be found in [12]. The IPERF prototype does not consider the characteristics of the target compiler in any of its performance models. However, the target machines characteristics as listed in Figure 7 are considered. In particular, an IPERF user may be interested in understanding the

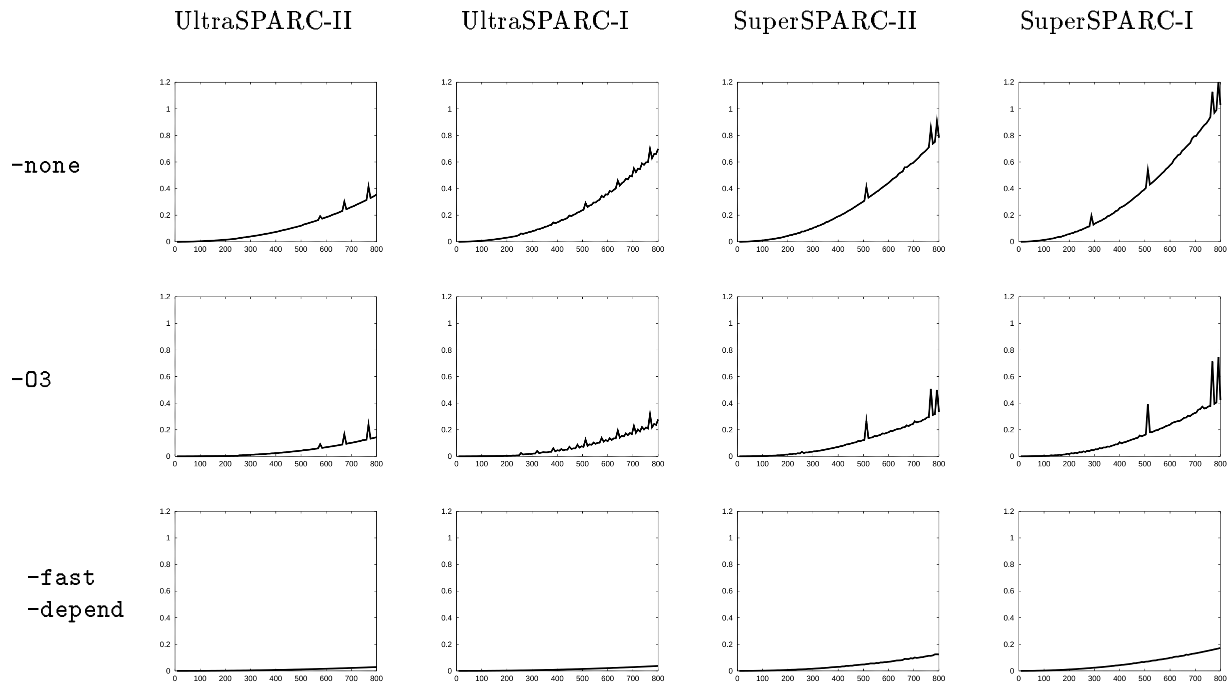

Figure 6: Measured execution times (in seconds) on twelve different target systems

performance peaks at different problem sizes n for the target compilers using the `-none` and `-O3` optimization levels.

Factor Lattice The IPERF prototype system considers performance factors and models for computation (ALU), virtual address translation (TLB), and memory hierarchy (MEM) as shown in Figure 2. All performance factors are numerical, i.e., produce a numerical cost value. The IPERF system will select a performance model that is a linear combination of the models for selected performance factors. Models for individual performance factors may be non-linear or linear combinations of non-linear models.

In the preselection step, the factor lattice is constructed based on the factors in the data base and the specific program characteristics of the benchmark set and the features of the target system. For instance, if the target system does not contain an L_2 cache, there is no need to include a model for this hardware feature. Similarly, the benchmark sets may not exhibit characteristics for which there are models in the data base. In our example case, models for cross reference conflict misses or temporal locality are not required since the example kernel contains only a single array and the loop does not have any temporal reuse. In the prototype IPERF system, the preselection of factors { ALU, TLB, L1, L2 } was done by hand. The associated models are derived from the literature except the model for the nonblocking cache. The model is new and its definition was the result of the preliminary experiments with IPERF . Figure 8 shows the different models used for our example benchmark set. The constants c_i represent the coefficients in a linear combination of the different components of the models. The actual values of the coefficients are determined during the fitting process. A detailed discussion of these models can be found in [12].

Model Search and Evaluation

The IPERF prototype system does not contain any automatic, "intelligent" strategy to search the performance model lattice. In the experiments described below, a reasonable subset of performance factors and models was selected by hand. Each of the resulting models were evaluated automatically, using a tool that performs data *fitting* for additive models. The tool fits a candidate performance model against the data points in the benchmark sets by computing a coefficient for each additive model component such that the user specified error criterion (relative or absolute error) is minimized. Note that (1) coefficients are not interpreted as absolute timing costs, but as the relative importance among performance factors, and

	UltraSPARC-x		SuperSPARC-x	
	Ultra-II	**Ultra-I**	**Super-II**	**Super-I**
TLB	D:64/8KB/64		D:64/4KB/64	U:64/4KB/64
L1-cache	D:512/32B:16B/1		D:128/32B/4	
	non-blocking		blocking	
	virtually indexed and physically tagged			
L2-cache	U:32768/64B/1	U:8192/64B/1	U:8192/128B:32B/1	
	blocking			
	physically indexed and physically tagged			

Cache Spec = *Type*:*Sets*/*BlockSize*:*SubBlockSize*/*Associativity*
Type: D (data), U (unified)
In TLB, *BlockSize* indicates page size

Figure 7: Hardware characteristics of four SPARC-based machines.

(2) for some cases, coefficients satisfying the user's request do not exist. It the latter case, either the precision request is too strong, or the chosen performance factors and their models are not adequate to model the observed behavior.

In the IPERF prototype system, two fitting methods are available, each appropriate for a different error criteria, namely ordinary least-square fitting (OLS) and weighted least-square fitting (WLS) [16]. OLS is better suited for expressing constraints on absolute prediction errors, while WLS deals better with the relative prediction errors constraints. The prototype system is *qualitative* in the sense that it determines *which* system components may explain a particular performance behavior, but not necessarily *how much* of the observed behavior can be attributed to each selected component. In other words, the prototype system performs a sensitivity analysis with respect to the different target system components.

Queries All experiments were based on the evaluation of nine models. Each model was assigned a cost defined as the number of component models, assuming that each component model has unit cost. Future work will include an investigation of more realistic cost models.

For the *same* target system, IPERF selected different performance models depending on the user requested error criterion and accuracy/cost trade-off. Figure 9 shows the model evaluations for the target system consisting of the UltraSPARC-I and `-O3` compiler option. The benchmark set of measured execution times has 100 data points total, and is shown in Figure 6. For the largest measured data point ($n = 800$), the execution time was around 300ms (milliseconds). Each table entry in Figure 9(a) is the number of predicted data points that are within the requested accuracy for the given model. Based on this information, IPERF generates different performance models for requests such as "*Determine minimal-cost model for which x% of the predicted data points are within prediction accuracy constraint y*". The accuracy constraint specifies an upper bound on the relative or absolute error. If multiple models satisfy a request, a lowest-cost model is selected. Figure 9(b) and Figure 9(c) show sample sets of requests and the models returned by IPERF . For example, a model query "*minimal-cost model that predicts 85% of the data points in the benchmark set within an absolute error of 10ms*" will result in model 6, shown as the entry in the 85% column and 10ms row in Figure 9(b). Different requests lead to significantly different performance models. Note that with a single exception, model 6 is the most accurate model. Since it is also the most expensive, it does not have the requested accuracy/cost trade-off in many cases.

Across *different* target systems, different models were generated for the same error criterion and requested accuracy/cost trade-off. Figure 10 shows the selected minimal-cost models for six sample queries. In cases of multiple minimal cost models with the highest precision, both models were listed. For example, a model query "*minimal cost model that approximates as many benchmark data points as possible within an absolute error of 10ms*" will result in model 1 for Ultra-II/`-O3`, model 6 for Ultra-I/`-O3`, and model 8 for Super-I and II with `-O3` compiler option, as shown in the second row of Figure 10 marked $AE \leq 10$.

Models	definition	remarks
$\mathbf{T_{ALU}}$	$c_0 + c_1 * n + c_2 * n^2$	n : loop bound, assumes unit cost per iteration
$\mathbf{T_{TLB}}$	$\begin{cases} 1 \cdot \frac{n^2}{P} & \text{if } n^2 \leq E \cdot P \\ n \cdot \frac{n^2}{P} & \text{otherwise} \end{cases}$	E : #TLB entries, P : page size [11, 7] $L2$: model for L_2 cache
$\mathbf{T_{L^b}}$	$\begin{cases} \frac{n^2}{ss} & \text{if } \frac{n}{D} \leq a \\ n^2 & \text{if } \frac{n}{D} \geq a+1 \\ \frac{n}{ss}[n + (ss-1) * (a+1) * (n - a * D)] & \text{otherwise} \end{cases}$ $\boxed{\textbf{blocking cache model}}$	$D = \min\{d > 0 \mid \lfloor \frac{n*d}{l} \rfloor \equiv_{mod(S)} 0\}$ d : reuse distance in iterations a : associativity, S : #sets, ss : subblock size [18, 21]
$\mathbf{T_{L^{nb}}}$	$\begin{cases} n * \frac{n}{D_{L1}} & \text{if } \frac{n}{D_{L1}} > a_{L1} \text{ and } \frac{n}{D_{L2}} \leq a_{L2} \\ 0 & \text{otherwise} \end{cases}$ $\boxed{\textbf{nonblocking cache model}}$	D_{Li} : parameter D for L_i a_{Li} : associativity of L_i

Figure 8: Performance models used in IPERF prototype system.

Summary The IPERF prototype system was able to identify different performance models for different accuracy/cost trade-offs. The resulting models are qualitative in the sense that the coefficients returned by the fitting algorithm do not represent the actual execution time contribution of a model component to the overall performance behavior. For instance, in Figure 9(b), model 9 is selected as the best model for a request. This model does not have ALU as a performance factor, although the computation in our example program contributes to the actual execution time. If each data TLB access results in a miss, the computation costs can be summarized as part of the TLB costs since it is linear in the number of TLB misses. In other words, fitting provides a form of "shape" analysis, and two performance factors with a similar shape may be substituted for each other during linear fitting. However, qualitative shape analysis can give significant insights into the system components that appear to determine the observed performance.

A failure to find a model of a requested precision means that a system component significant for the overall system performance is not appropriately modeled. We discovered such a model deficiency during the evaluation of models for the UltraSPARC-I with optimization level `-O3`. For absolute error requests, we initially did not have a non-blocking cache model for the L_1 cache (L_1^{nb}). By developing a new model for non-blocking caches, prediction accuracy was improved by up to 31% over the corresponding model based on a blocking cache model (see models 6 and 7 in Figure 9(a)).

A similar success story can be reported for the UltraSPARC-II system with `-none` and `-O3` compiler options. Here the failure to find an efficient model leads to the discovery of a performance effect due to the algorithm used to translate virtual to physical address translation in the Solaris operating system. By changing the default setting of the page coloring parameter, the behavior of the resulting target system became more predictable.

Finally, it is important to note that advanced hardware features and software optimizations can result in target system performance behavior that can be modeled more easily. In our example program, the cache behavior has a significant impact on the overall program performance. Transformations such as loop interchange can improve spatial locality significantly and thereby reduce the performance impact of cache misses. This explains why the target systems with `-fast -depend` optimization can be predicted with high precision with only an ALU performance model (see Figure 10). Hardware features reducing conflict misses are also able to improve the predictability of a target system.

4 Future Work

More experiments are needed to validate the effectiveness of our proposed framework. In order to allow

Models			Absolute Error (AE)			Relative Error (RE)		
no.		cost	$\leq$ 5ms	$\leq$ 10ms	$\leq$ 15ms	$\leq$ 10%	$\leq$ 20%	$\leq$ 30%
1	ALU	1	58	74	87	1	10	26
2	ALU+TLB	2	57	77	84	81	90	96
3	ALU+L_1^{nb}	2	64	83	96	1	11	23
4	ALU+L_2^b	2	60	79	93	3	12	28
5	ALU+L_1^{nb}+L_2^b	3	81	85	89	3	11	24
6	ALU+TLB+L_1^{nb}+L_2^b	4	88	98	99	86	94	97
7	ALU+TLB+L_1^b+L_2^b	4	67	79	94	87	94	97
8	ALU+TLB+L_2^b	3	60	82	92	83	93	97
9	TLB+L_1^{nb}+L_2^b	3	70	92	97	76	89	93

(a) Model precision of different requests for same target system (-O3,UltraSPARC-I)

AE	% of predicted data points				
	95%	90%	85%	80%	75%
$\leq 5ms$	-	-	**6**	**5**	**5**
$\leq 10ms$	**6**	**9**	**9**	**3**	**3**
$\leq 15ms$	**3**	**3**	**1**	**1**	**1**

(b) Results of absolute error queries

RE	% of predicted data points				
	95%	90%	85%	80%	75%
$\leq$ 10%	-	-	**7**	**2**	**2**
$\leq$ 20%	-	**2**	**2**	**2**	**2**
$\leq$ 30%	-	**2**	**2**	**2**	**2**

(c) Results of relative error queries

Figure 9: Minimal-cost models satisfying different requests for single target system.

Queries	UltraSPARC-x						SuperSPARC-x					
	Ultra-II			**Ultra-I**			**Super-II**			**Super-I**		
	-none	-O3	-f/d	-none	-O3	-f/d	-none	-O3	-f/d	-none	-O3	-f/d
$AE \leq 5ms$	**3** (90)†	**3** (90)	**1** (100)	**6** (88)	**6** (88)	**1** (100)	**4** (84)	**4** (78)	**1** (100)	**8** (76)	**8** (78)	**1** (100)
$AE \leq 10ms$	**3** (97)	**1** (97)	**1** (100)	**5** (98)	**6** (98)	**1** (100)	**4** (87)	**8** (87)	**1** (100)	**4** (90)	**8** (84)	**1** (100)
$AE \leq 15ms$	**1** (98)	**1** (98)	**1** (100)	**5** (99)	**6** (99)	**1** (100)	**8** (98)	**4** (94)	**1** (100)	**4** (93)	**4** (88)	**1** (100)
$RE \leq$ 10%	**8** (97)	**6, 7** (95)	**1** (100)	**8** (99)	**7** (87)	**1** (100)	**8** (99)	**6** (93)	**1** (100)	**8** (98)	**6 , 7** (77)	**1** (99)
$RE \leq$ 20%	**6** (99)	**2** (96)	**1** (100)	**2** (100)	**6, 7** (94)	**1** (100)	**4** (100)	**6** (98)	**1** (100)	**4** (99)	**8** (97)	**1** (100)
$RE \leq$ 30%	**2, 3** (100)	**6** (97)	**1** (100)	**1** (100)	**8** (97)	**1** (100)	**1** (100)	**8** (99)	**1** (100)	**1** (99)	**8** (99)	**1** (100)

† Model precision in terms of number of predicted data points (out of 100) within the requested accuracy.

Figure 10: Minimal-cost models satisfying same query for different target systems.

experimentation for larger program kernels and whole programs, we are planning to implement a fully automatic IPERF prototype system based on the SUIF compiler infrastructure that is currently being extended as part of the NSF/DARPA National Compiler Infrastructure Project [1]. SUIF contains frontends for different languages such as Fortran, C, C++, and Java. The set of target machines for the proposed system will span across several current and future architectures.

We are currently investigating different model lattice construction and search strategies for compositional and numerical models. In addition, new methods are being developed that allow the extension of the current techniques to produce quantitative performance models.

References

[1] National Compiler Infrastructure (NCI) project. Overview available online at http://www-suif.stanford.edu/suif/nci/index.html., Co-funded by NSF/DARPA, 1998.

[2] J. Anderson and M. Lam. Global optimizations for parallelism and locality on scalable parallel machines. In *Proceedings of the SIGPLAN '93 Conference on Programming Language Design and Implementation*, pages 112–125, Albuquerque, NM, June 1993.

[3] E. A. Brewer. High-level optimization via automated statistical modeling. In Jeanne Ferrante and David Padua, editors, *5th ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming*, pages 80–91, New York, NY, USA, August 1995. ACM Press.

[4] Eric Allen Brewer. *Portable High-Performancei Supercomputing: High-Level Platform-Dependent Optimizations.* PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, September 1994.

[5] S. Chatterjee, J.R. Gilbert, R. Schreiber, and S-H. Teng. Optimal evaluation of array expressions on massively parallel machines. *ACM Transactions on Programming Languages and Systems*, 17(1):123–156, January 1995.

[6] Joel Emer and Nickolas Gloy. A language for describing predictors and its application to automatic synthesis. In *24th International Symposium on Computer Architecture*, pages 304–314, 1997.

[7] J. Ferrante, V. Sarkar, and W. Thrash. On estimating and enhancing cache effectiveness. In *1991 Workshop on Languages and Compilers for Parallel Computing*, pages 328–343, 1991.

[8] D. Gannon, W. Jalby, and K. Gallivan. Strategies for cache and local memory management by global program transformation. *Journal of Parallel and Distributed Computing*, 5(5):587–616, October 1988.

[9] J. Garcia, E. Ayguadé, and J. Labarta. Dynamic data distribution with control flow analysis. In *Proceedings of Supercomputing '96*, Pittsburgh, PA, November 1996. , available at URL http://www.supercomp.org/sc96/proceedings.

[10] M. Gupta and P. Banerjee. Compile-time estimation of communication costs on multicomputers. In *Proceedings of the 6th International Parallel Processing Symposium*, Beverly Hills, CA, March 1992.

[11] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann, San Mateo, California, second edition, 1996.

[12] C-H. Hsu and U. Kremer. A framework for qualitative performance prediction. Technical Report LCSR-TR98-363, Department of Computer Science, Rutgers University, July 1998.

[13] K. Kennedy and U. Kremer. Automatic data layout for distributed memory machines. *ACM Transactions on Programming Languages and Systems*, 1998. To appear.

[14] Wagner Meira, Jr. *Understanding Parallel Program Performance Using Cause-Effect Analysis.* PhD thesis, Department of Computer Science, University of Rochester, August 1997. TR-663.

[15] K. S. M^c^Kinley, S. Carr, and C.-W. Tseng. Improving data locality with loop transformations. *ACM Transactions on Programming Languages and Systems*, 18(4):424–453, July 1996.

[16] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, Cambridge, 2 edition, 1992.

[17] Rafael H. Saavedra-Barrera. *CPU Performance Evaluation and Execution Time Prediction Using Narrow Spectrum Benchmarking.* PhD thesis, U.C. Berkeley, February 1992. UCB/CSD-92-684.

[18] O. Temam, C. Fricker, and W. Jalby. Cache interference phenomena. In *Proceedings of the Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 261–271, New York, NY, USA, May 1994. ACM Press.

[19] K-Y. Wang. Precise compile-time performance prediction for superscalar-based computers. In *Proceedings of the SIGPLAN '94 Conference on Programming Language Design and Implementation*, Orlando, FL, June 1994.

[20] M. E. Wolf and M. Lam. A data locality optimizing algorithm. In *Proceedings of the SIGPLAN '91 Conference on Programming Language Design and Implementation*, Toronto, Canada, June 1991.

[21] Michael J. Wolfe. *High Performance Compilers for Parallel Computing.* Addison-Wesley Co., 1996.