ELSEVIER

Performance Evaluation 45 (2001) 1–18

PERFORMANCE EVALUATION
An International Journal

www.elsevier.com/locate/peva

Adaptive multivariate regression for advanced memory system evaluation: application and experience

Xian-He Sun[a,*,1], Dongmei He[a], Kirk W. Cameron[b], Yong Luo[b]

[a] *Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616, USA*
[b] *Los Alamos National Laboratory, Mail Stop B256, Los Alamos, NM 87545, USA*

Received 2 October 1998; received in revised form 24 October 2000

Abstract

Recent advances in latency hiding techniques have made performance evaluation of memory hierarchies a more difficult task. Applications compiled for a particular architecture may be executed on vastly different memory hierarchy implementations. There is a need for performance analysis techniques that provide methods for understanding the interaction between applications and a given memory hierarchy. In this paper, we present a statistical approach to performance analysis of advanced memory hierarchy implementations. The method involves the utilization of previously available statistical analysis techniques coupled with scalability analysis. The result is a novel step-wise approach to understanding the hierarchical memory performance of scientific applications. We apply the method to several scientific applications of interest to the accelerated strategic computing initiative (ASCI) over the SGI machines PowerChallenge and Origin 2000. Results indicate some codes are statistically identical in memory performance, while others vary greatly. Furthermore, some codes do not take advantage of the performance enhancements to the memory system found in the Origin 2000. © 2001 Elsevier Science B.V. All rights reserved.

Keywords: Performance evaluation; Advanced memory system; Computer architecture; Statistical method; Scalability; Benchmarking

1. Introduction

Hierarchical memory systems are complex entities. Application developers typically create a single code that will run on a single microprocessor platform, but many different implemented memory systems. Cache sizes will vary, latencies will be different, and the ability to perform work while waiting on memory access will quantifiably differ. For these reasons, it would be helpful if we could evaluate how well certain codes take advantage of certain characteristics of the underlying design for a memory system. Some codes may benefit from larger caches, others from lower latencies, and still others may not benefit from these

[*] Corresponding author. Tel.: +1-312-567-5260; fax: +1-312-567-5067.
E-mail addresses: sun@cs.iit.edu (X.-H. Sun), kirk@lanl.gov (K.W. Cameron), yongl@lanl.gov (Y. Luo).
[1] This author was supported in part by NSF under grant ASC-9720215 and CCR-9972251, and by ONR under the PET program.

differences at all. It is also interesting to observe the scaled performance of such codes. From such studies we can determine whether or not codes behave regularly or as expected as they scale computationally.

In this paper, we present a novel use of existing statistical methods coupled with scalability analysis in the interest of classifying and analyzing application performance. We present a four-level statistical analysis technique that allows both statistical classification and step-wise refinement of code analysis. Results include the ability to gauge scaled performance and the isolation of performance impact due to on-chip latency hiding techniques for the codes and platforms studied. The approach itself is somewhat general in nature for statistical code classification and memory scalability analysis. Results show the method is both feasible and useful for analyzing the performance of scientific codes generally across different memory hierarchies.

This paper is organized as follows. We present related work followed by a limited discussion of the statistical methods necessary for our model. Next, we present the model itself in the context of hierarchical memory evaluation. Experimental results are provided by applying the statistical approach on the Origin 2000 and PowerChallenge systems from SGI. We close with overall conclusions of the paper and a discussion of future directions for this research.

2. Related work

Complex superscalar microprocessors found in today's marketplace provide special registers for monitoring on-chip events without impacting the performance of the architecture or applications. These performance monitors provide data previously not available to the general performance community without the aid of simulators or special hardware. Results provided by these counters can be used successfully in performance analysis provided techniques that take advantage of monitored events are identified and utilized. Empirical and analytical models have been introduced that take advantage of these counts to provide explanation of application performance. Empirical fitting can be used to infer the performance attributable to latency overlap provided on-chip. Analytical models of on-chip performance provide insight to architecturally constrained limitations at the instruction level [2].

Other models utilize information provided at the system level or via simple timings. These can provide some information as to the performance of applications, but are limited by the information provided to them. Multidimensional curve fitting provides methods for characterizing architectures generally in the same fashion as empirical fitting [5,8]. An exhaustive mean value approach can be used provided extensive measurements of the system can be obtained [1]. The problem here is that, at least for now, only a simulator can provide such information.

The statistical method in this paper focuses primarily on analyzing the statistical characteristics of cpi (cycles per instruction) for the codes measured. The approach is multileveled since we focus on different averages and code–machine combinations in a certain order to focus on performance differences among the platforms analyzed. We also incorporate memory scalability analysis to isolate performance differences as problem sizes (computationally speaking) scale up. By focusing on cpi, we are actually measuring the different levels of ILP (instruction-level parallelism) achieved by the architecture for each code. Different memory hierarchies implemented for the same microprocessor will impact the achieved cpi. Hence, the statistical method is presented to isolate the impact of such differences on average and as applications scale.

There have been many attempts to understand and model performance of code and machine combinations at the instruction level. Some of the earliest work was accomplished by Peuto and Shustek

[13]. These methods primarily use trace-driven simulation or direct timed measurements for model input. Trace-driven simulation approaches are limited to analyzing generally smaller applications since simulations are orders of magnitude slower than actual runs. Due to the large amount of information provided from simulations, statistical reduction methods have been introduced to sift through the enormous amount of data generated [4]. Methods such as micro-benchmarking [14] focus on measuring performance of individual code segments while characterizing codes for the number of occurrences of each micro-code segment. Iterative performance bounding [11], isolates the performance of singular contributions like compiler and scheduler on code. None of these approaches or others utilize performance monitors, and most have been made obsolete by the inherent parallelism of current microprocessors.

There also have been many attempts to understand and model performance at application and machine levels [5,18]. Ein-Dor and Feldmesser [5] studied CPU performance based on the *relative performance* metric, which is defined as a dimensionless number calibrated in terms of IBM 370/158 model on the basis of vendor claims, user experience, and information supplied by independent consultants. They used regression fitting to determine the coefficients of their hardware-parameter-based linear model and used the model to predict the *relative performance* of new CPUs. Other non-statistical curve fitting approaches have been successfully presented as well [8,13]. They differ from Ein-Dor and Feldmesser primarily in derivation of the coefficients and the defined model parameters.

Our statistical approach consists of four levels of evaluation. The last two levels of evaluation are designed for scalability study, which is measured in terms of cpi variations when problem size increases. The comparison of relative cpi variation is not related to the term *relative performance* as used by Ein-Dor and Feldmesser. There is no traditional "curve fitting for prediction" in our study. In fact, a wide cpi variation indicates that none of the traditional prediction method can predict performance accurately for the underlying code–machine combinations without considering the factor of problem size. An application of the scalability study is to identify such "unpredictable" code–machine combinations.

More recently, researchers have been turning to statistical analysis for trace-reduction methods as mentioned previously. Still others have begun using statistical sampling methods on trace-driven simulations to analyze performance directly rather than simply providing concise data [3]. Again, none of these approaches utilize performance monitors nor the statistical approach of our model. To the best of our knowledge, no current methods combine the use of performance counters with scalability, and the two-factor factorial experiments and regression methods utilized in our framework.

3. Background and terminology

3.1. Memory scalability

A goal of high performance computing is to solve large problems fast. Considering both execution time and problem size, what we seek from parallel processing is *speed*, which is defined as work divided by time. The average unit speed is a good measure of parallel processing. It measures the computation performed in each processor per second. Since our approach is in the context of single processor performance, we wish to redefine our parallel processing concept of speed into the confines of a single processor in terms of cpi. We begin with the parallel processing definitions. The result is a new definition of memory scalability in terms of one processor scaled over varying computational problem sizes.

The *isospeed scalability* has been formally defined in [18] as the ability to maintain the average speed in parallel processing when the number of processors increases, where average speed is defined as the achieved speed of the given computing system divided by p, the number of processors.

Definition 1 (Isospeed scalability). A code–machine combination is scalable if the achieved average speed of the code on the given machine can remain constant with increasing numbers of processors, provided the problem size can be increased with the system size.

By Definition 1, isospeed scalability maintains average speed via increases in problem size. Intuitively, a more parallelizable code should lead to a better scalability on a give machine and vice versa. This intuition may not be generally true due to memory or other hardware limitations. Following the same concept, Definition 2 gives a definition of data scalable for memory systems of single node sequential computing in terms of cpi [17].

Definition 2 (Data scalable for single system). We say code–memory combination 1 is better (data) scalable than code–memory combination 2, if code–memory combination 1 has a better initial cpi than that of code–memory combination 2 and their cpi difference increases when problem size scales up, or if code–memory combination 2 has a better initial speed than that of code–memory combination 1 and their cpi difference decreases when problem size scales up.

Data scalable for a single system (or processor) is a complement to isospeed scalability for parallel processing. It measures the hardware/software constraints of serial computing when problem size scales up, where the most likely constraint of sequential computing is the limitation of memory capacity. Evaluating and characterizing the performance of a single memory system is the focus of this study.

3.2. The two-factor experiment

In our methodology, we use cpi as a gauge of performance. From our previous discussion related to speed, we claim cpi is the inverse of speed and an appropriate measurement in memory hierarchy analysis since effects on memory access times (in cycles) will cause changes in measured cpi values. In our two-factor factorial experiments, code is one factor and machine is another. Code has five factor levels (HEAT, HYDRO, SWEEP, DSWEEP, HYDROT), machine has two factor levels (PowerChallenge, Origin 2000), and problem size has a factor level for each different size, seen later. This forms the following linear model for the corresponding two-factor factorial experiment:

$$y_{ijk} = \mu + \alpha_i + \gamma_j + (\alpha \cdot \gamma)_{ij} + \epsilon_{ijk}, \tag{1}$$

where y_{ijk}, $k = 1, 2, \dots, n$, are the kth observed value of cpi, μ the reference value, which is usually called the "grand" or overall mean, α_i, $i = 1, 2, \dots, a$, the main effects of factor code, γ_j, $j = 1, 2, \dots, c$, the main effects of factor machine, $(\alpha \cdot \gamma)_{ij}$, $i = 1, 2, \dots, a$, $j = 1, 2, \dots, c$, the interaction effects of factors code and machine, and ϵ_{ijk}, $i = 1, 2, \dots, a$, $j = 1, 2, \dots, c$, $k = 1, 2, \dots, n$, are the random errors.

The hypotheses of interesting effects are established as: $H_0 : \alpha_i = 0$ (assume main effect A does not exist), $H_0 : \gamma_j = 0$ (assume main effect C does not exist), and $H_0 : (\alpha\gamma)_{ij} = 0$ (assume no interaction effect exist). The probabilities of correctness of the null hypotheses are calculated by the F distribution function [6,7]. In statistical analysis, a significance level of 5% is usually used to test a null hypotheses.

The analysis of a factorial experiment is called the analysis of variance (ANOVA). A simple example is given in [17] to illustrate this analysis process.

3.3. Contrast and post hoc comparisons

Many statistical methods exist for classification and grouping. We use two well-known classification methods in our study to show corroborating results, namely contrast and post hoc comparisons. A contrast is a linear function of means whose coefficients add to zero. In the contrast method, a t-test is used to judge the null hypotheses [6]. An example is given to explain the contrast method in [17]. The evaluation of the t-test is similar to that of the F-test in judging null hypotheses. If the probability of the t-value is less than 0.05, then the null hypothesis is rejected; otherwise the null hypothesis cannot be rejected.

When the factors and their levels are not defined in a manner that allows the use of preplanned comparisons, a post hoc comparison procedure would be more appropriate. The post hoc comparisons, namely the LSD, Tukey, Duncans, and Scheffe comparison [6], are similar to the above t-test contrast method. The differences are that these methods have their own criteria to determine the “significant difference” for the t-test.

3.4. Regression method for scalability testing

A regression method has been used by Lyon et al. [10] to evaluate the scalability of parallel processing. With some modification, we extend the regression method to data scalability of memory systems. We use a simple example to illustrate the regression method. In a scalability study, the two factors are problem size and machine, and the experiment is for a given code on different machines. Assuming we are interested in testing the scalability of code HEAT which has a (problem) size level 1 and 2 with problem size 25 and 50, respectively, we set the PowerChallenge as machine level 1 and Origin 2000 as machine level 2. Following the regression method, we need to assign a value to each of the code and machine levels. Denoting code level 1 by -1 and level 2 by 1 for both size and machine level accordingly, we have the index table, Table 1.

In Table 1, $X_{\rm c}$ is the indicator variable for code, $X_{\rm m}$ the indicator variable for machine, $I_{\rm c,m}$ the indicator variable for interaction. If μ is a constant term, then we have the following regression model:

$$\mathrm{cpi} = \mu + \beta_{\rm c} X_{\rm c} + \beta_{\rm m} X_{\rm m} + \beta_{\rm c,m} I_{\rm c,m} + \epsilon. \tag{2}$$

The term $\beta_{\rm c,m}$ is the interaction effect. It is tested by a t-test to see whether the interaction effect is significant. The null hypothesis tested here is $H_0 : \beta_{\rm c,m} = 0$. If the probability of the resulting t-value is less than 0.05, then $\beta_{\rm c,m} < 0$ leads to the conclusion that the code is more scalable on the level 2 machine

Table 1
The index table of a regression experiment

$X_{\rm c}$	$X_{\rm m}$	$I_{\rm c,m}$	cpi	cpi actual
-1	-1	$+1$	a	1.233678
$+1$	-1	-1	b	0.900876
-1	$+1$	-1	c	1.112349
$+1$	$+1$	$+1$	d	1.387690

than on the level 1 machine; $\beta_{\mathrm{c,m}} = 0$ leads to the conclusion that the code has the same scalability on the level 1 and level 2 machines; otherwise, $\beta_{\mathrm{c,m}} > 0$ leads to the conclusion that the code is more scalable on the level 1 machine than on the level 2 machine. Comparing the data scalable concept given in Section 3, we can see that the regression method provides a relative scalability comparison of a code on two different machines. As shown in the example, the relative comparison is in terms of the size level used in the problem size factor. In general, the relative scalability is a function of the size and the number of size levels of the problem size factor. For an appropriate experimental design, the problem sizes tested should be chosen from an appropriate range which represents the actual usage.

For simplicity, we have used a two-level experiment to illustrate the regression method for scalability evaluation. However, the regression method is general. It can be applied to any number of levels greater than one for each of the factors as will be shown in our experimental results.

4. Evaluation of hierarchical memory systems

We propose an adaptive evaluation method for advanced memory systems based on the standard statistical methodology summarized in Section 3. This method consists of four levels of evaluation. While the first two levels of the methodology focus on the mean performance over problem sizes, the last two level evaluations show the performance variation when problem size increases. The combination of these four levels of evaluations provides a feasible solution for suggesting performance when problem size scales up and for further memory system improvements.

4.1. Level 1 evaluation: main effect

Level 1 evaluation uses the two-factor factorial experiment (see Section 3.2) to find the effects of code and machine. Using these factors, it detects the overall effect of code, machine, and their interaction on the final performance. The dependent variable for the design is cpi. The random samples for each of the code–machine level combinations are chosen from different problem sizes within the range of interest. So, the effective comparison is based on the mean performance over different problem sizes. If a code effect exists, we conclude that the codes have different memory reference patterns which diverge memory access time. When a machine effect exists, the memory system difference on the machines does make a difference in performance. Finally, when code–machine interaction effects exist, the memory system difference has a different impact on different memory reference patterns. Notice that all these effects are overall effects of codes and machines. Any significant effect deserves further investigation to identify the source or sources.

4.2. Level 2 evaluation: code/machine classification

Level 1 evaluation detects the overall effects of code, machine, and their interaction on performance. When these effects exist, we would like to know the contribution of each code/machine toward the effects, and to identify the outstanding code/machine for more detailed study. Statistical classification methods provide a means to group code/machine based on their performance. We no longer use overall means for code and machine, we look at mean performance on a code by code basis.

The contrast method and post hoc comparisons introduced in Section 3.3 are used in our study. These comparison methods will be applied pairwisely. For instance, for code classification under our experimental environment, HEAT has to be compared with all the other codes, namely HYDROT, HYDRO, SWEEP, and DSWEEP; HYDROT is compared with the rest of the codes, namely HYDRO, SWEEP, and DSWEEP; HYDRO is compared with SWEEP and DSWEEP; and finally, SWEEP is compared with DSWEEP. In general, there are $a!$ comparisons for a factor with a levels. If two machines belong to the same category, then statistically they are the same for the set of codes and the given range of problem sizes. If two codes belong to two different categories, then they have different memory reference/computation patterns. A good general purpose machine should not deliver a wide cpi distribution among codes.

4.3. Level 3 evaluation: scalability comparison

Both level 1 and level 2 evaluation consider performance over a set of codes and machines. The third step of our evaluation method is individual detection of outliers. It compares the data scalabilities of a given code for varying problem sizes on different machines. Scalability is a factor that contributes to the ability of a system to deliver the expected performance. Memory scalability evaluation is a new approach. It evaluates the ability of a memory system to handle large data sizes. The same or better initial performance combined with better scalability guarantees a code will have better performance when problem size scales up. A code with a smaller initial cpi and better scalability has the potential to become superior as problem size scales up.

The basic statistical method for memory scalability evaluation is the regression method given in Section 3.4. The two factors (independent regression variables) are problem size and machine. The regression method does not measure data scalability directly for which a formal quantitative definition of scalability is required. Instead, it gives a statistical relative comparison of two or more machines for a given code. Problem size increase may change the performance of a code–machine combination. This change varies with code, machine, and code–machine combination. It forms the base of scalability comparison. Using cpi as the measurement with the same code on two different machines, if the interaction of the two variations is negative, then the second machine has a better scalability; if the interaction of the two variations is zero, then the two machines have the same scalability; otherwise, the first machine has a better scalability.

Ideally, different scalability benchmarks can be used for different scalability applications. For instance, a performance benchmark can be used to evaluate a memory subsystem's ability in maintaining performance when problem size increases; or a variation benchmark can be used to determine the predictability of a code–machine combination. In general, however, benchmark development is a challenging issue on itself. There is no widely accepted benchmark for scalability study at this time.

4.4. Level 4 evaluation: memory hierarchy

As discussed in the previous section, the performance of a code may vary with problem size, and this variation is different over different memory architectures. The last step of our evaluation methodology is designed to locate memory components which cause the variation. Level 4 evaluation compares the performance variation of primary components of the underlying memory systems. It is necessary for the performance monitors to provide information in this context. In other words, we will use cache hit measurements to determine relative memory performance. Our method is general for testing the variance of any measurable quantity for further evaluation of a system. Combined with the level 2 evaluation, this evaluation determines the ability of each memory component in handling different memory reference patterns and suggests possible improvements at the component level.

The basic statistical method used in level 4 evaluation is the same as that of level 3 evaluation, except for the dependent variables. The actual design of level 4 evaluation varies with the underlying memory structure. As discussed in Section 5, the memory hierarchy of SGI PowerChallenge and Origin 2000 has four primary components: L1 cache, L2 cache, *outstanding cache misses*, and main memory. L1, and L2 hit-ratio can be derived using hardware counters provided on-board the SGI microprocessor. For this reason, we choose L1 and L2 as the dependent variables.

5. Test environment

Two machines and a set of five benchmarking codes are used throughout our study to illustrate the method and to verify the correctness. These machines and benchmarks are described below. While our discussion is focused on a particular environment, the factorial method used in this study is general. It can be applied to any machine and set of applications.

5.1. Machine description

The PowerChallenge is an SMP architecture that employs a central bus to interconnect memories and processors [12]. The bus bandwidth (1.2 Gbps) does not scale with more processors. Cache coherence is maintained through a snoopy bus protocol which broadcasts cache information to all processors connected to the bus. The Origin 2000, on the other hand, is a distributed shared memory (DSM) architecture which uses a switch interconnect that improves scalability by providing interconnect bandwidth proportional to the number of processors and memory modules [19]. Coherence is maintained by a distributed directory-based scheme. Each router in the hypercube topology connects two nodes to the network. Each node contains two processing elements and one local memory unit. A 128-processor system, for example, consists of a fifth-degree hypercube with four processors per router.

The processing elements of both the Origin 2000 and PowerChallenge systems use a 200 MHz MIPS R10000 microprocessor. The processor is a four-way super-scalar architecture which implements a number of innovations to reduce pipeline stalls due to data starvation and control flow [19]. For example, instructions are initially decoded in order, but are executed out-of-order. Also, speculative instruction fetch is employed after branches. Register renaming minimizes data dependencies between floating-point and fixed-point unit instructions. Logical destination register numbers are mapped to the 64 integer and 64 floating-point physical registers during execution. The two programmable performance counters track a number of events and were a necessity for this study. The most common instructions typically have one- or two-clock latencies.

Whereas the processing elements of the PowerChallenge and Origin 2000 systems are identical, there are major differences in the memory architecture and corresponding performance of the two systems. The PowerChallenge is a UMA architecture with a latency of 205 clocks (1025 ns). Latencies to the memory modules of the Origin 2000 system, on the other hand, depend on the network distance from the issuing processor to the destination memory node. Accesses issued to local memory take about 80 clocks (400 ns), while latencies to remote nodes are the local memory time plus 33 clocks for an off-node reference plus 22 clock periods (CP; 110 ns) for each network router traversed. In the case of a 32 processor machine, the maximum distance is four routers, so that the longest memory access is about 201 clocks (1005 ns) which is close to the constant latency of the PowerChallenge.

In addition, improvements were made in the number of outstanding loads that can be queued by the memory system. Even though the R10000 processor is able to sustain four outstanding primary cache misses, external queues in the memory system of the PowerChallenge limited the actual number to less than two. In the Origin 2000, the full capability of four outstanding misses is possible. The L2 cache sizes of these two systems are also different. A processor of PowerChallenge can be equipped with up to 2 MB L2 cache, while a CPU of Origin 2000 system always has a L2 cache of 4 MB.

As evident, these SGI machines provide a unique performance evaluation environment since the architectures employ identical microprocessors, but differ significantly in the design of the memory subsystems. The particular differences, namely L2 cache size, main memory latency, and number of outstanding misses, allow this statistical factorial study to unveil the relative performance impact of the memory subsystem. We intend to focus on single processor execution and use identical executables across machines to eliminate software differences. All data was captured using on chip performance counters provided for the MIPS R10000 microprocessor. This method of data collection, as opposed to simulation or other similar methods, provides a non-intrusive representation of the processor performance under real conditions.

5.2. Code description

The following codes were used in the factorial experiment design:

- SWEEP and DSWEEP are both three-dimensional discrete-ordinate transport solvers that differ in their implementations. In both versions, the main part of the computation consists of a balance loop in which particle flux out of a cell in three Cartesian directions is updated based on the fluxes into that cell and on other quantities such as local sources, cross-section data, and geometric factors. The cell-to-cell flux dependence implies a recursive wavefront structure. In the DSWEEP implementation, the mesh is swept using diagonal planes which enable the balance loop to be vectorized. In this version, gather/scatter operations must be used to obtain local source and cross-sectional values. In the second implementation, namely SWEEP, a "line sweep" is accomplished involving separately nested, quadrant, angle, and spatial-dimension loops. There are no gather/scatter operations, all accesses are now unit-stride, and memory traffic is significantly reduced through "scalarization" of some array quantities. However, with the balance loop proceeding along rows and columns instead of the diagonal, recursion prohibits complete vectorization.
- HYDRO is a two-dimensional explicit Lagrangian hydrodynamics code based on an algorithm by W.D. Schulz. HYDRO is representative of a large class of codes in use at the laboratory. The code is 100% vectorizable. An important characteristic of the code is that most arrays are accessed with a stride equal to the length of one dimension of the grid. HYDROT is a version of HYDRO in which most of the arrays have been transposed so that access is now largely unit-stride. A problem size of N implies N^2 grid points.
- HEAT solves the implicit diffusion PDE using a conjugate gradient solver for a single time-step. The code was written originally for the CRAY T3D using SHMEM. The key aspect of HEAT is that its grid structure and data access methods are designed to support one type of adaptive mesh refinement (AMR) mechanism, although the benchmark code as supplied does not currently handle anything other than a single-level AMR grid (i.e., the coarse regular level-1 grid only). A problem size of N implies N^3 grid points.

6. Evaluation of SGI PowerChallenge and Origin 2000

To verify the feasibility and correctness of our method, we apply this method to the computing environment discussed in Section 5. All four levels of evaluation have been used to evaluate these ASCI machines and benchmarks. Experimental results show that this proposed adaptive statistical method is feasible and effective. To illustrate the implementation procedure and to demonstrate the evaluation results, the experimental results are presented and discussed in this section. In our experimental testing, the two machines, PowerChallenge and Origin 2000, are denoted as machine level 1 and level 2, respectively. The five codes, HEAT, HYDRO, SWEEP, DSWEEP, and HYDROT, are denoted as 1, 2, 3, 4, and 5, respectively. We have used SAS [15] throughout the experimental evaluation.

The problem sizes used in the experiment range from $N = 50$ to memory/time constraints. The corresponding range for the codes are: HEAT = [50, 100], HYDRO = [50, 300], SWEEP = [50, 200], DSWEEP = [50, 200], HYDROT = [50, 300]. All the experimental data are measured from single node sequential executions using SGI hardware performance counters.

6.1. Main and interaction effects

The relationship between code and machine is first investigated. To catch the mean relationship over the range of problem sizes, replicate measurements have been taken for different problem sizes for a given experimental unit. The two-factor factorial experiment introduced in Section 3.2 is used to find the effects using an initial hypothesis that no effect (or significant statistical differences) exists in the current experiment. The GLM procedure of SAS is used to analyze the two-factor factorial experiment for level 1 evaluation. Table 2 shows results from GLM.

Table 2 is the mean effects table of the factorial experiment. It consists of two sectors, Part A and Part B. Part A is for overall effect, and Part B is for individual effects. Look at row 2 of Table 2. The F-value is 27.44 and the probability of $F(\mathrm{Pr} > F)$ is 0.0001, which is less than 0.05. The hypothesis

Table 2
Mean effects table

Source	DF	Sum of squares	Mean square	F-value	Pr > F
Part A					
Model	9	112.5410006	12.5045556	27.44	0.0001
Error	103	46.9436516	0.4557636		
Corrected total	112	159.4846523			
Dependent variable	cpi				
R^2	0.705654				
CV	34.6444				
Root MSE	0.675103				
CPI mean	1.948661				
Source	DF	Type I SS	Mean square	F-value	Pr > F
Part B					
Machine	1	14.39563307	14.39563307	31.59	0.0001
Code	4	93.17895152	23.29473788	51.11	0.0001
Machine × code	4	4.96641604	1.24160401	2.72	0.0334

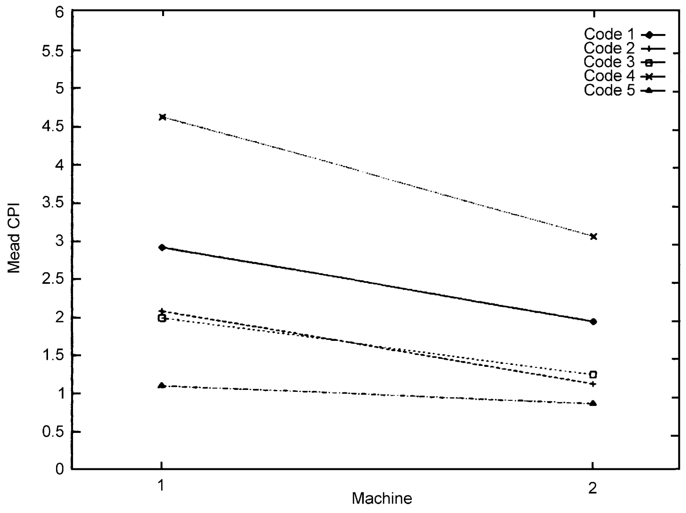

Fig. 1. Machine mean distribution.

that an overall-effect does not exist is rejected. This means that code or machine effects exist. Part B is a continuation of Part A to locate the potential effects. Look at row 2 of Part B. The probability of F is $0.0001 < 0.05$, which suggests that a machine main effect exists. The same conclusion can be drawn for code. For machine and code interaction, the probability of F is 0.0334, which is again smaller than 0.05. An interaction effect for code and machine also exists. Evaluation should be continued to understand these effects.

The mean effect analysis can be explained visually. As depicted in Fig. 1, the code performance crosses over the two machines between code 2 and code 3. This line crossing indicates the existence of an interaction effect of machine and code. It confirms the results given by the contrast method (see Table 3). However, codes 2 and 3 have very similar performances on the two machines. If we can take code 2 and 3 as one code through classification, then there is no code performance crossing over the two machines and,

Table 3
Contrast method for pairwise comparison

Contrast	DF	Contrast SS	Mean square	F-value	Pr > F
HEAT vs. DSWEEP	1	18.73737434	18.73737434	41.11	0.0001
HEAT vs. SWEEP	1	6.48938939	6.48938939	14.24	0.0003
HEAT vs. HYDRO	1	8.44857266	8.44857266	18.54	0.0001
HEAT vs. HYDROT	1	25.87993484	25.87993484	56.78	0.0001
DSWEEP vs. SWEEP	1	42.24375672	42.24375672	92.69	0.0001
DSWEEP vs. HYDRO	1	51.96661369	51.96661369	114.02	0.0001
DSWEEP vs. HYDROT	1	84.81327756	84.81327756	186.09	0.0001
SWEEP vs. HYDRO	1	0.00268119	0.00268119	0.01	0.9390
SWEEP vs. HYDROT	1	4.41163307	4.41163307	9.68	0.0024
HYDRO vs. HYDROT	1	5.40337655	5.40337655	11.86	0.0008
Machine 1 vs. Machine 2	1	19.78987372	19.78987372	43.42	0.0001

Table 4
LSD post hoc comparison for code

T grouping	Mean	*N*	Code
A	3.7324	17	4 (DSWEEP)
B	2.4568	22	1 (HEAT)
C	1.6287	28	2 (HYDRO)
C	1.6048	18	3 (SWEEP)
D	1.0074	28	5 (HYDROT)

therefore, no interaction effect for machine and code. Classification of code and machine is important for understanding measured performances. In fact, based on our level 2 evaluation, codes 2 and 3 are statistically the same (see Table 4). The two lines between code 2 and code 3, therefore, statistically are merged to one line.

Fig. 2 plots the codes performance over the two machines. We can see that machine 2 always outperforms machine 1, so a machine effect does exist. Based on two-factor factorial mechanisms the GLM procedure systematically finds the main and interaction effects, which sometimes, but not always, can be determined easily through visual display.

6.2. Code and machine classification

The codes and machines have been classified based on the contrast and post hoc comparisons introduced in Section 3.3. The contrast procedure of SAS is used for the contrast comparison. The result of the pairwise code/machine contrast comparison is given in Table 3.

In Table 3, all the probabilities of rejection are less than 0.05, except at row 9. Code HYDRO and SWEEP are in the same group. They have similar performance variations caused possibly by the computational

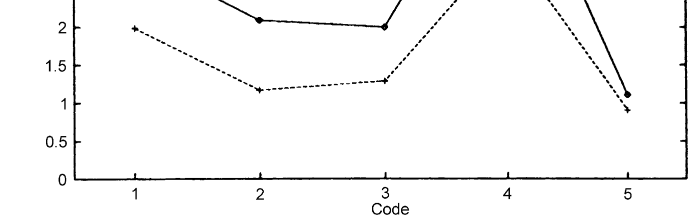

Fig. 2. Code mean distribution.

Table 5
LSD post hoc comparison for machines

T grouping	Mean	N	Machine
A	2.3217	54	1 (PowerChallenge)
B	1.65552	59	2 (Origin 2000)

pattern and/or the data reference pattern. All other codes, namely HEAT, DSWEEP, and HYDROT, have their own signatures. They each belong to different groups. The two machines are also in two different groups.

The LSD procedure of post hoc comparison is also applied to classify the sets of codes and machines. Tables 4 and 5 give the result of the code and machine classification, respectively. From Table 4, we can see that HEAT belongs to group B; DSWEEP belongs to group A; HYDROT belongs to group D; and HYDRO and SWEEP belong to group C. The result is the same as that of contrast comparison. In the post hoc comparison, the grouping distance used is 0.4072. The groups are ordered according to their mean cpi values. The group with the highest cpi value (worst in performance) is listed first. The group with the second highest cpi value is listed second, and so on. It is interesting to note the implications of these simple results for code classification. We observe that with the exception of HYDRO and SWEEP, each code has a unique performance variation pattern that warrants further investigation. As will be shown, these unique patterns can be further broken down into individual effects contributed by differences in the memory hierarchy in this particular test environment. These patterns directly contribute to the inherent scalable performance across machines for these particular codes. As shown in Table 5, PowerChallenge and Origin 2000 are classified into two different groups. The distance between the two groups is larger than 0.2522 (least significant difference = 0.2522 cpi). The Origin 2000 is always better than PowerChallenge for the set of codes under consideration. This result again matches that of contrast comparison.

6.3. Scalability comparison

Using the regression method discussed in Sections 3.4 and 4.3, we have conducted scalability comparisons on all of the five codes over the two machines. Recall that this third step in our methodology compares the data scalabilities of a given code on different machines, whereas the level 2 evaluation grouped codes based on their average performance over the range of problem sizes. As we discussed in the previous section, a better memory system should lead to a smaller cpi, and a more scalable memory system should have a smaller cpi increase, or no cpi increase at all, as problem size scales up. Table 6 is generated by PROG REG of SAS for the scalability comparison of HEAT over problem size range [50, 100]. The response variable is cpi.

Table 6
Scalability comparison of HEAT

Variable	DF	Parameter estimate	Standard error	T for H_0 parameter = 0	Pr > $\|T\|$
Intercep	1	2.453200	0.05065942	48.425	0.0001
Code	1	0.077618	0.01601992	4.845	0.0001
Memory	1	−0.468297	0.05065942	−9.244	0.0001
INTAC	1	0.079500	0.01601992	4.963	0.0001

Table 7
Scalability comparison of SWEEP

Variable	DF	Parameter estimate	Standard error	T for H_0 parameter $= 0$	Pr $> \|T\|$
Intercep	1	1.613494	0.02647227	60.950	0.0001
Code	1	0.049352	0.00966631	5.106	0.0003
Memory	1	−0.390073	0.02647227	−14.735	0.0001
INTAC	1	0.012463	0.00966631	1.289	0.2216

In Table 6, “INTAC” stands for INTerACtion effect. At the 0.0001 level (see the last column of Table 6), the hypothesis of zero effect has been rejected, so an interaction effect exists. The parameter estimate of “INTAC” is 0.0795, which means that the term $\beta_{\mathrm{c,m}}$ is positive (see Eq. (2)) and the performance difference of the two machines decreases with problem size. PowerChallenge is more scalable than Origin 2000 over the range of problem sizes. This reduction in difference is very reasonable. When problem size increases into main memory, the advantage of having a larger L2 cache fades away. The performances of the two machines, therefore, become closer. Different codes have different memory access/computing ratios and have different memory reference patterns. Some codes have good locality, some do not. Some memory reference patterns can take advantage of the underlying memory support, some cannot. These factors and others give codes different scalabilities on different memory systems. While the resulting table is not shown, HYDRO has an INTAC probability level of 0.0111 indicating interaction effects exist for HYDRO. Unlike HEAT, HYDROs parameter estimate is negative −0.050885, which means that the performance difference between the two machines increases with problem size. Origin 2000 has a better scalability than PowerChallenge for HYDRO. The scalability improvement may be due to Origin 2000’s larger L2 cache or hardware support in handling cache misses or faster memory access time. The results of code SWEEP, DSWEEP and HYDROT are different. Our no-effect hypotheses stands. The more advanced memory system of Origin 2000 does not improve the performance difference of these three codes when problem sizes scale up. The relative performances over the two machines remain unchanged.

Table 7 lists results generated by PROG REG for scalability analysis of SWEEP. From Table 7, the probability level of interaction effect is 0.2216. Therefore, $\beta_{\mathrm{c,m}} = 0$ and SWEEP has the same scalability on the two machines. For DSWEEP and HYDROT, the probability level of interaction effect is 0.3002 and 0.2799, respectively.

6.4. *Evaluation of memory components*

As discussed in Section 5, the memory systems of the SGI machines consist of four primary components: L1 cache, L2 cache, outstanding cache misses, and main memory. In the level 4 evaluation, we examine the role of the four components in scalability variation. The same regression method used in scalability study is used here. We use SAS procedure PROC REG to evaluate the relative performance of L1 and L2 cache independently. The response variable is the cache hit-ratio of L1 and L2 accordingly. The cache hit-ratios of L1 and L2 are independent of each other and can be used as independent variables. Outstanding cache misses cannot be measured. However, based on the scalability comparison given in the previous section, its role in performance variation can be estimated when the variations of the L1 and L2 hit-ratios are known.

Table 8
L1 hit-ratio comparison for HEAT

Variable	DF	Parameter estimate	Standard error	T for H_0 parameter = 0	Pr > $\|T\|$
Intercep	1	0.818304	0.00039150	2090.171	0.0001
Code	1	0.000086900	0.00012380	0.702	0.4917
Memory	1	−0.000289	0.00039150	−0.738	0.4699
INTAC	1	0.000128	0.00012380	1.032	0.3156

Table 9
L2 hit-ratio comparison for HEAT

Variable	DF	Parameter estimate	Standard error	T for H_0 parameter = 0	Pr > $\|T\|$
Intercep	1	0.766496	0.00267152	286.914	0.0001
Code	1	−0.004971	0.00084481	−5.884	0.0001
Memory	1	0.015196	0.00267152	5.688	0.0001
INTAC	1	−0.005011	0.00084481	−5.931	0.0001

Tables 8 and 9 show the analysis table for L1 and L2 hit-ratio variation of HEAT. We can see from Table 8 that the probability level of "INTAC" is $0.3156 > 0.05$. The zero effect hypothesis is true for L1 hit-ratio of HEAT. HEAT has a constant L1 hit-ratio difference over the two machines. By Table 9, code–machine interaction effect exists ($\alpha = 0.001 < 0.05$) and the effect is negative ($\beta_{\mathrm{c,m}} = -0.005011 < 0$). In practice, we prefer a smaller cpi and a larger hit-ratio. The negative effect means that the L2 hit-ratio difference of HEAT on Origin 2000 goes down relative to PowerChallenge, when problem size scales up. As we know from Section 6.3, HEAT has a better scalability on PowerChallenge than on Origin 2000. The relative L2 hit-ratio decrease explains the smaller scalability of Origin 2000.

Recall that the underlying SGI PowerChallenge and Origin 2000 machine have the same CPU and the same L1 cache. It is no surprise that the relative L1 hit-ratio does not change for all five codes. HEAT has demonstrated how the regression method can be used repeatedly for different components of a memory system.

Table 10 is the L2 hit-ratio analysis table for HYDRO. As given in Table 10, the null hypothesis of interaction is accepted. The hit-ratio differences of HYDRO remain the same for the SGI machines when problem size scales up. As analyzed in Section 6.3, HYDRO-Origin 2000 has a better scalability than HYDRO-PowerChallenge. This scalability increase is not due to the larger L2 cache of Origin 2000 as shown by the cache hit-ratios across machines. It is due to the outstanding cache misses ability and faster

Table 10
L2 hit-ratio comparison for HYDRO

Variable	DF	Parameter estimate	Standard error	T for H_0 parameter = 0	Pr > $\|T\|$
Intercep	1	0.911569	0.00944229	96.541	0.0001
Code	1	−0.011458	0.00211136	−5.427	0.0001
Memory	1	0.046284	0.00944229	4.902	0.0001
INTAC	1	0.003901	0.00211136	1.847	0.0771

main memory access time supported by Origin 2000. Combined with an existing empirical model [9], a detailed analysis is given in [16] to understand the performance.

Hit-ratio comparison and component level evaluation of SWEEP, DSWEEP, and HYDROT are also conducted and can be found in [17]. We have applied the four-level evaluation proposed in Section 4 to analyze the performance of two ASCI machines and five benchmarks available at Los Alamos National Laboratory. In the level 1 evaluation, we have found that both code and machine effects exist. Performance varies with codes and machines. Continued from the first level evaluation, in level 2 evaluation, the codes and machines have been classified into four and two groups, respectively, based on their performance. This classification shows that, while the codes have a wide distribution in performance due to their inherent memory reference/computation patterns, the Origin 2000 definitely outperforms PowerChallenge on all the codes. It is interesting to note that, despite the fact that all the codes had a better performance on Origin 2000, by level 3 evaluation these codes have different relative performance variations over the two machines when problem size scales up. When problem size becomes large, the performance difference of HEAT on these two machines becomes smaller; the performance difference of HYDRO on these two machines becomes larger; while the differences of the other three codes remain unchanged. Obtaining the variation in relative performance is important for benchmarking and other performance comparisons. For instance, the scalability analysis shows that the relative performance of HEAT and HYDRO are more likely to vary with problem size than the other three codes. A more detailed evaluation, the level 4 evaluation, has found the causes of the scalability difference over the codes. In addition to a larger L2 cache capacity, the four outstandings for cache misses and the faster main memory access supported by Origin 2000 have played an important role in performance improvement. This is especially true for HYDRO and SWEEP.

7. Conclusions

In this paper, we have presented a novel approach to performance evaluation of advanced hierarchical memory systems. While statistical methods for trace-driven reduction with some analysis exist, our approach has more in common with other empirical and analytical approaches to performance analysis. We have shown, however, that no methods of this type exist that utilize performance counters and take into account latency hiding techniques. Since we utilize counts provided from performance monitors, we stipulate the architecture of interest must provide cycle, instruction, and cache hit measurements. Otherwise, the method can be generally applied to systems of the same underlying architecture with different implementations of the memory hierarchy.

Levels 1 and 2 of our model identified the inherent differences among our machines and codes, and statistically classified our machines and codes, respectively. Of particular interest was the result that SWEEP and HYDRO are statistically similar in the memory usage patterns and hence can be classified accordingly. The rest of the codes were statistically dissimilar enough to warrant further investigation following the pair-wise comparison of level 2. Level 3 analysis of the memory scalability of our code–machine combinations is not necessarily intuitive. In fact, the scaled performance of HYDRO is the only code that performed better (as problem size increased) on the more advanced memory system of the Origin 2000. SWEEP, DSWEEP, and HYDROT showed no discernible scaled performance gain between the two machines, while HEAT scaled better on the PowerChallenge. This underscores the fact that performance differences between hierarchy implementations are extremely code dependent while sometimes counterintuitive.

In the final level 4, the statistical approach evaluated the individual characteristic differences between the hierarchies by analyzing the calculated hit-rates obtained via the performance monitors. Since the L1 cache was identical across machines, performance differences were not attributable to L1 cache differences. We use the observed statistical differences in L2 cache hit rates to explain the counterintuitive notions given by the level 3 analysis. We showed HEAT was more scalable on the PowerChallenge in level 3. Level 4 indicated this was not due to the L2 cache leaving only the combined difference in the memory access time and outstanding misutilization. The better scalability of the Origin 2000 for HYDRO was also not due to differences in the L2 caches. This would indicate these are also attributable to outstanding misutilization and possibly memory access time to some extent.

Since the complexity of memory systems continuously increases, we believe the statistical method presented herein will provide useful conclusions regarding memory performance differences such as whether or not code utilizes architectural enhancements. Furthermore, we believe the statistical method will become even more useful to future generations of memory hierarchies. We have shown the usefulness of these results and discussed the novelty of our approach. Future improvements to this technique include incorporation of shared memory attributes, extension to different architectural characteristics, and tight coupling with other empirical approaches.

References

[1] D. Albonesi, I. Koren, A mean value analysis multiprocessor model incorporating superscalar processors and latency tolerating techniques, Int. J. Parallel Program. 24 (3) (1996) 235–263.
[2] K.W. Cameron, Y. Luo, Instruction-level microprocessor modeling of scientific applications, Lecture Notes Comput. Sci., Vol. 1615, Springer, Berlin, 1999, pp. 29–40.
[3] R. Carl, J.E. Smith, Modeling superscalar processors via statistical simulation, in: Proceedings of the Workshop on Performance Analysis and its Impact on Design (PAID), Barcelona, Spain, 1998.
[4] P. Crowley, J.-L. Baer, On the use of trace sampling for architectural studies of desktop applications, in: Proceedings of the SIGMETRICS'99, Atlanta, GA, 1999.
[5] P. Ein-Dor, J. Feldmesser, Attributes of the performance of central processing units: a relative performance prediction model, Commun. ACM 30 (4) (1987) 308–317.
[6] R. Freund, W. Wilson, Statistical Methods, Academic Press, New York, 1997.
[7] R. Jain, The Art of Computer System Performance Analysis, Wiley, New York, 1991.
[8] U. Krishnaswamy, I.D. Scherson, Micro-architecture evaluation using performance vectors, in: Proceedings of the SIGMETRICS'96, Philadelphia, PA, 1996.
[9] O.M. Lubeck, Y. Luo, H. Wasserman, F. Bassetti, An empirical hierarchical memory model based on hardware performance counters, in: Proceedings of the PDPTA'98, July 1998, pp. 386–393.
[10] G. Lyon, R. Kacker, A. Linz, A scalability test for parallel code, Softw. Pract. Exper. 25 (12) (1995) 1299–1314.
[11] W. Mangione-Smith, T.P. Shih, S.G. Abraham, E.S. Davidson, Approaching a machine-application bound in delivered performance on scientific code, in: IEEE Proceedings on Computer Performance Analysis (special issue), 1993, pp. 1166–1178.
[12] MIPS Technologies, Inc., R10000 microprocessor product overview, MIPS Product Preview, 1995.
[13] B. Peuto, L. Shustek, An instruction-timing model of CPU performance, in: Proceedings of the Fourth Annual Symposium on Computer Architecture, March 1977, pp. 165–178.
[14] R.H. Saavedra, A.J. Smith, E. Miya, Machine characterization based on an abstract high-level language machine, IEEE Trans. Comput. 38 (1989) 1659–1679.
[15] SAS User's Guide, SAS Institute, Inc., Cary, NC, 1996.
[16] X.-H. Sun, K. Cameron, A statistical–empirical hybrid approach to hierarchical memory analysis, Lecture Notes Comput. Sci., Vol. 1900, Springer, Berlin, 2000, pp. 141–148.

[17] X.-H. Sun, D. He, K. Cameron, Y. Luo, An adaptive statistical methodology for advanced memory systems evaluation, Los Alamos National Laboratory Unclassified Technical Report (LAUR) No. 98-4176, 1998.
[18] X.-H. Sun, D. Rover, Scalability of parallel algorithm–machine combinations, IEEE Trans. Parallel Distrib. Syst. 5 (6) (1994) 599–613.
[19] K. Yeager, The MIPS R10000 superscalar microprocessor, IEEE Micro 16 (2) (April 1996) 28–40.

Xian-He Sun received his Ph.D. degrees in computer science from Michigan State University. He was a Staff Scientist at ICASE, NASA Langley Research Center and was a Tenured Associate Professor in the Computer Science Department at Louisiana State University (LSU). Currently, he is an Associate Professor and the Director of the Scalable Computing Software Laboratory in the Computer Science Department at Illinois Institute of Technology (IIT) and a Guest Faculty at the Argonne National Laboratory. Dr. Sun's research interests include parallel and distributed processing, software system, performance evaluation, and scientific computing. He has published intensively in the field and his research has been supported by DoD, DoE, NASA, NSF, and other government agencies. He is a senior member of IEEE, a member of ACM, New York Academy of Science, PHI KAPPA PHI, a partner of the Esprit IV APART (automatic performance analysis: resources and tools) working group, and has served and is serving as the Chairman or on the Program Committee for a number of international conferences and workshops. He received the ONR and ASEE Certificate of Recognition Award in 1999.

Dongmei He received her B.S. degree in Chemistry from Chengdu Science and Technology University, China. She got both her Master's degree in Applied Statistics and Master's degree in Computer Science in 1998 from Louisiana State University. Specializing in SAS programming, she is a System Analyst in the Computing Center at Louisiana State University. Prior to that, she was a Computer Programmer with USAencies.

Kirk W. Cameron is an Assistant Professor in the Department of Computer Sciences at the Florida Institute of Technology. His research interests include performance analysis and prediction, computer architecture, and parallel and distributed computing. Cameron holds his Ph.D. recently in Computer Science from Louisiana State University. His thesis topic was application modeling using performance monitors. Portions of this work were accomplished, while funded as a graduate research assistant at the Los Alamos National Laboratory in Los Alamos, NM.

Yong Luo is currently a Senior Software Engineer of Server and Workstation Architecture Performance Group at Intel Corporation. Before joining Intel in 1999, Yong was a technical staff member of Los Alamos National Laboratory. Yong received his Ph.D. in Electrical Engineering from the University of British Columbia in 1994. Yong started his parallel computing and performance analysis career in 1991 and was awarded Fujitsu (Canada) Award in 1994. Yong's research interest includes performance analysis/modeling, architecture research, digital signal processing and numeric algorithms.