LA-UR- 98-4176

Approved for public release; distribution is unlimited.

Title: A Hierarchical Statistic Methodology for Advanced Memory System Evaluation

Author(s): Xian-Je Sun, Louisiana State University
Dongmei He, Louisiana State University
Kirk W. Cameron, CIC-19
Yong Luo, CIC-19

Submitted to: 13th International Parallel Processing Symposium
April 12-16, 1999
San Juan, Puerto Rico

Los Alamos
NATIONAL LABORATORY

DISCLAIMER

A Hierarchical Statistic Methodology for Advanced Memory System Evaluation

Xian-He Sun[†*] *Dongmei He*[†] *Kirk W. Cameron*[†‡] *Yong Luo*[‡]

[†]*Department of Computer Science*
Louisiana State University
Baton Rouge, LA 70803-4020
{sun,dhe}@bit.csc.lsu.edu

[‡]*Mail Stop B256*
Los Alamos National Laboratory
Los Alamos, New Mexico 87545
{kirk,yongl}@lanl.gov

Abstract

Advances in technology have resulted in a widening of the gap between computing speed and memory access time. Data access time has become increasingly important for computer system design. Various hierarchical memory architectures have been developed. The performance of these advanced memory systems, however, varies with applications and problem sizes. How to reach an optimal cost/performance design eludes researchers still. In this study, we introduce an evaluation methodology for advanced memory systems. This methodology is based on statistical factorial analysis and performance scalability analysis. It is two fold: it first determines the impact of memory systems and application programs toward overall performance; it also identifies the bottleneck in a memory hierarchy and provides cost/performance comparisons via scalability analysis. Different memory systems can be compared in terms of mean performance or scalability over a range of codes and problem sizes. Experimental testing has been performed extensively on the Department of Energy's Accelerated Strategic Computing Initiative (ASCI) machines and benchmarks available at the Los Alamos National Laboratory to validate this newly proposed methodology. Experimental and analytical results show this methodology is simple and effective. It is a practical tool for memory system evaluation and design. Its extension to general architectural evaluation and parallel computer systems are possible and should be further explored.

Key Words:
Performance Evaluation, Advanced Memory System, Computer Architecture, Statistical Method, Scalability, Benchmarking

[*]This author was supported in part by NSF under grant ASC-9720215, by LSU under 1998 COR award, and by Louisiana Education Quality SupportFund.

1 Introduction

Memory speed improvement has not kept up with the pace of that of processor speeds. While processor speeds have been increasing by at least 70 percent annually, the DRAM latency has improved only 7 percent per year [1]. Various advanced memory systems have been developed to manage the increasingly wide disparity between central processing unit (CPU) speed and data access speed. An advanced, hierarchical memory system has become a necessity of high-performance computers. Performance and scalability of these modern computers are more dependent on the hierarchical memory systems than the peak CPU rate employed.

Performance evaluation of today's hierarchical memory systems, however, is very challenging. There are a variety of means by which the impact of memory latency on computer performance can be diminished by the computer architecture, as described in [2]. For example, the architecture can tolerate long latencies, by increasing the memory hierarchy complexity (with more cache units) or by increasing the level of concurrency available in memory operations. Computer architectures can also reduce memory latency by using faster processor-memory interconnection networks, supporting out-of-order execution, and allowing outstanding memory accesses to overlap computation; this helps because DRAM speed is only one component of overall memory latency. There are also a variety of techniques programmers can employ to diminish memory latency effects. Programmer-related strategies require algorithmic innovation to tolerate memory latency by taking advantage of data locality, and to reduce memory stall time by eliminating superfluous load and store operations. These architectural and programmatic techniques are both machine and application dependent. How to apply them in practice is still state-of-the-art. In addition, the performance and optimization requirements vary with problem sizes. Scalability needs to be considered.

Based on statistical factorial analysis and performance scalability analysis, in this paper we propose a methodology for examining the effectiveness of both hardware and software memory latency hiding techniques of a memory system. This methodology consists of four levels of evaluation. For a set of codes and a set of machines, we first determine the effect of code, machine, and code-machine interaction on performance respectively. If a main or interaction effect exists, then, in the second level of evaluation, the code and/or machine is classified based on certain criteria to determine the cause of the effect. The first two levels of evaluation are designed to detect the characteristics of codes and their influence on different memory systems. They are based on average performances over the ranges of problem size interested in. The last two levels of evaluation determine the performance variation when problem sizes scale up and are based on scalability analysis which is a new approach for memory system evaluation. The concept of memory scalability is formally introduced. Level three evaluation is the scalability evaluation of the underlying memory system for a give code. Level four evaluation conducts a more detailed examination on the component contribution of the memory system toward the final scalability. The combination of the four levels of evaluation makes the proposed methodology adaptive, effective, and more appropriate for advance memory systems

than existing methodologies.

The Silicon Graphics Inc. (SGI) Origin2000 system and a previous SGI architecture, the PowerChallenge system, have been used as the test-bed to illustrate the newly proposed methodology. The single-processor performance (in terms of cycles per instruction,*cpi*) of the two machines are compared and analyzed. Evaluation results given by this methodology are confirmed by measured results and by a previously-reported performance model. The comparison of these two machines is of particular interest because they both use the same compute node, a 200-MHz MIPS R10000 processor [3] [4], but the memory subsystems of the two architectures are vastly different. Although improvement in the Origin memory network has important consequences for system-wide, multiprocessor performance, Origin single-node performance benefits as well.

We use a benchmark set consisting of applications from the Los Alamos portion of the Accelerated Strategic Computing Initiative (ASCI) workload. The intention of ASCI is to accelerate the development of tera-scale numerical computing capabilities in order to allow effective achievement of massively-parallel, 3-D coupled-physics simulations. However, the long history of DOE programmatic computing on computer architectures is vastly different from those envisioned as part of the ASCI program. Significant algorithmic modification as described above may be required to reach acceptable performance. Thus, it is important that ASCI code developers have some advance information regarding the extent to which cache-based architectures and cache-friendly programming styles will affect the performance of their applications. In order to provide this information in this paper we use codes that incorporate both the legacy style of coding and a locality-improved style. In some cases both styles are available for the same application.

This paper is organized as follows. In Section 2, we present the test environment. The machines and codes used in our experimental testing are briefly described. In Section 3, we introduce the scalability concept and the statistical methods needed for the hierarchical methodology. Terminologies needed are also given in this section. The four level statistical methodology for memory system evaluation is introduced in Section 4. Case study is conducted in Section 5 to apply the proposed methodology on the testing environment. All the four levels of evaluation are applied to examine and compare the five codes on the two machines given in the testing environment. Measured results and an existing analytical model are used to confirm the newly proposed methodology. Finally, section 6 summarizes the work.

2 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 methodology proposed in this study is general. It can be applied to any machine and set of applications.

2.1 Machine Description

The PowerChallenge is an SMP architecture that employs a central bus to interconnect memories and processors [3]. The bus bandwidth (1.2 Gbytes/sec) 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 [4]. 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 4 processors per router.

The processing elements of both the Origin 2000 and PowerChallenge systems use a 200MHz MIPS R10000 microprocessor. The processor is a 4-way super-scalar architecture which implements a number of innovations to reduce pipeline stalls due to data starvation and control flow [4]. 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 [5] and were a necessity for this study. The most common instructions typically have one- or two-clock latencies.

While 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 an 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 4 routers, so that the longest memory access is about 201 clocks (1005 ns) which is close to the uniform latency of the PowerChallenge.

In addition, improvements in the number of outstanding loads that can be queued by the memory system were made. 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 up to 2MB L2 cache while a CPU of Origin 2000 system always has a L2 cache of 4MB.

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 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 collected and used 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 realistic representation of the actual processor performance in a real environment under real conditions.

2.2 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 now proceeding along rows and columns instead of the diagonal, recursion now 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. HYDRO-T 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 timestep. 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.

3 Background and Terminology

Some background knowledge of scalability and statistics is needed for understanding the proposal factorial evaluation methodology. We introduce the memory scalability concept and three statistical methods in this section. These statistical methods are not new. The combination of these statistical methods with scalability analysis and its application in memory system evaluation are new.

3.1 Terminology

Except *cpi*, all the following terminologies are general terms used in statistics [6, 7].

1. *cpi* (Cycle Per Instruction)
   cpi measures the average number of computing cycles used for executing one instruction. Speed, a widely used performance metric, is defined as work divided by time. For scientific computing, speed is often measured in terms of MFLOPS (Million of FLoating-point Operations Per Second). If work is given in terms of instructions and time is given in terms of computing cycles, *cpi* is the reciprocal of speed. We choose *cpi* as the preferred measurement in our study since nonfloating-point operations are an important concern in memory evaluation, and the number of computing cycles consumed is a more accurate measurement than execution time when memory system performance can be separated from that of computing elements.

2. Multiple treatment factors
   Many factors influence the performance of a computer system, however an experimental design with a large number of factors may not be the best approach toward an understanding of performance. A first step of modeling should be to find the factors with significant impact and to reduce the number of factors to be examined. In our experimental design, we use two factor factorial design. Problem size and machine are the two factors used for scalability study and code and machine are the two factors used in data reference pattern study. Each factor has multiple levels. For our design, for instance, the machine factor has two levels, PowerChallenge and Origin2000; the code has five levels, HEAT, HYDRO, SWEEP, DSWEEP, and HYDROT.

3. Factorial experiment
   An experiment that has each combination of all factor levels applied to experimental units is called factorial experiment. An entity that is used for the experiment is called an experimental unit. For example, PowerChallenge and HEAT, one combination of the different levels of the code and machine factors, is an experimental unit.

4. Cell
   Cell refers to the measurement made to an experimental unit. The value of *cpi* measured on PowerChallenge and HEAT could be considered a cell. A cell may include an observation.

5. Main effects

   Main effects are the differences in the mean response across the levels of each factor when viewed individually. For instance, code and machine are two main effects for our study,

6. Interactions effects

   Interactions effects are differences or inconsistencies of the main effect responses for one factor across levels of one or more of the other factors. In our experimental design, both code and machine may have effects on the experimental units. If code influences the performance of a machine, or, vice versa, then interaction effects exist.

3.2 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.

Definition 1 *The* `average unit speed` *(or average speed, in short) is the achieved speed of the given computing system divided by p, the number of processors.*

The *isospeed scalability* has been formally defined in [8] as the ability to maintain the average speed in parallel processing when the number of processors increases.

Definition 2 *(*`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 2, isospeed scalability maintains average speed via increases in problem size. Intuitively, a more scalable code-machine combination should lead to an increased average speed for a given problem size and vice versa. This intuition may not be generally true due to memory or other hardware limitations. A definition of *data (problem size) scalable* is introduced in [9, 10] for parallel processing. Following the same concept, Definition 3 gives a definition of data scalable for memory systems of single node sequential computing.

Definition 3 *(*`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 speed than that of code-memory combination 2 and their speed difference increase 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 speed difference decrease when problem size scales up.*

As we discussed in Section 3.1, *cpi* is a more appropriate measurement for memory system evaluations. Definition 4 gives an equivalent definition of data scalable in terms of *cpi*.

Definition 4 *(*`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 increase 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 decrease when problem size scales up.*

Data scalable is a complement of 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.

3.3 The Two-Factor Factorial Experiment

We arbitrarily label A and C as the two factors used in the Two-Factor Factorial Experiment. Assume factor A has a levels and factor C has c levels, which is referred to as an $a \cdot c$ factorial experiment, and assume there are n independent samples replicated for each of the $a \cdot c$ possible factor-level combinations; we then have a randomized experimental design with $a \cdot c$ treatments and $a \cdot c \cdot n$ observed values of the response variable. The linear model for the corresponding two-factor factorial experiment is

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

where

$$y_{ijk}, k = 1, 2, \ldots, n, \text{ are k-th observed value}$$

of the response variable Y for the cell defined by the definition of the *i-th* level of factor A and the *j-th* level of factor C;

$$\mu \text{ is the reference value, which is usually called the "grand" or overall mean;}$$

$$\alpha_i, i = 1, 2, ..., a, \text{ are main effects of factor } A,$$

they are the difference in the mean response between the subpopulation comprising the *i-th* level of factor A and the reference value μ;

$$\gamma_j, j = 1, 2, ..., c, \text{ are main effects of factor } C,$$

they are the difference in the mean response between the subpopulation comprising the *j-th* level of factor C and the reference value μ;

$$(\alpha \cdot \gamma)_{ij}, i = 1, 2, ..., a, j = 1, 2, ..., c, \text{ are interaction effects of factor } A \text{ and } C,$$

they are the difference between the mean response in the subpopulation defined by the combination of the A_i and C_j factor levels; and finally

$$\epsilon_{ijk}, i = 1, 2, ..., a, j = 1, 2, ..., c, k = 1, 2, \ldots, n, \text{ are random errors}$$

representing the variation among observations that have been subjected to the same factor level combinations. ϵ_{ijk} are the values of a random variable having an approximately normal distribution with mean zero and variance σ^2. Determining the main and interaction effects of a two-factor factorial experiment involves four steps. First, the hypotheses of interested effects should be established. For the memory system study, we are only concerned with the existence of the effects. We have

$$H_0 : \alpha_i = 0, \text{ main effect A (assume main effect A does not exist).} \quad (2)$$

$$H_0 : \gamma_j = 0, \text{ main effect C (assume main effect C does not exist).} \quad (3)$$

$$H_0 : (\alpha\gamma)_{ij} = 0, \text{ for all i and j, interaction of factor } A \text{ and factor } C \quad (4)$$

$$\text{(assume interaction effect does not exist).} \quad (5)$$

Second, compute the main and interaction effects based on measured data and the linear model, equation (1). Third, compare the computed main and interaction effects with the null hypotheses. In our study, the comparison is to compare computed effects' values with the zero value. Finally, the probabilities of correctness of the null hypotheses are calculated by the F distribution function [6, 7]. In statistical factorial analysis, less than 5% is usually used to reject a null hypotheses. For instance, if the probability of main effect A is less than 5%, then the null hypothesis equation (2) will be rejected. That means main A effect is not zero, and main effect A exists. Otherwise, the value of main effect is zero and main effect A does not exist. Main effect C and the interaction effect $A \cdot C$ are evaluated similarly.

The factorial analysis of a factorial experiment is the analysis of variance. For brevity, we use a simple example to illustrate the analysis process. Assume that we have codes HEAT and DSWEEP run on the two machines PowerChallenge and Origin2000. Therefore, there are 4 possible combinations in these two factor factorial experiment: HEAT on PowerChallenge; HEAT on Origin2000; DSWEEP on PowerChallenge; DSWEEP on Origin2000. The measurements made on these four combinations is *cpi*. Let HEAT and DSWEEP be the level 1 and level 2 code, and let PowerChallenge and Origin2000 be the level 1 and level 2 machines, respectively. Table 1 shows a hypothetical data set arrangement.

In Table 1, y is the dependent variable. The first two indices of y's subscript represent the level of code and machine respectively. For instance, as listed, y_{111} is 1.4014525 and y_{112} is 3.12857184. The average of these two cells, the cell mean, represented by $\overline{y}_{11.}$, is 2.2649935. In general, $\overline{y}_{ij.}$ is the

Table 1. A Sample Data Set

machine	code		code means
	HEAT	DSWEEP	$\overline{y}_{i..}$
PowerChallenge	1.40141525	1.02674004	
	3.12857184	1.88990128	1.8616571
CellMeans $\overline{y}_{ij.}$	2.2649935	1.4583207	
Origin2000	0.847917292	0.90590811	
	1.352394111	1.409128055	1.1288369
CellMeans $\overline{y}_{ij.}$	1.1001557	1.1575181	
Machine Means $\overline{y}_{.j.}$	1.6825746	1.3079194	$\overline{y}_{...} = 1.495247$

cell means for y_{ijk}, $k = 1, 2, ..., n$. The same explanation applies to code DSWEEP and machine Origin2000. The average of the cell means in the same column is the machine means represented by $\overline{y}_{.j.}$. The average of the cell means in the same row is the code means represented by $\overline{y}_{i..}$. At last, the overall mean is represented by $\overline{y}_{...}$. Table 2 shows the components needed to be computed for a two-factor factorial experiment.

Table 2. Mean Effects Table

Source	DF	SS	MS	F
Between Cells	$a \cdot c - 1$	$SSCells$	$MSCells$	$MSCells/MSW$
Factor A	*a-1*	SSA	MSA	MSA/MSW
Factor C	*c-1*	SSC	MSC	MSC/MSW
Interaction $A \cdot C$	*(a-1)(c-1)*	$SSAC$	$MSAC$	$MSAC/MSW$
Within Cells(Error)	$a \cdot c(n-1)$	SSW	MSW	
Total	$a \cdot c \cdot n - 1$	TSS		

In Table 2, DF is the degree of freedom. Degree of freedom is the number of observations minus the number of the equations (restrictions) in the experiment. SS is the Sum of Squares and MS (Mean Square) is the sum of square divided by the degree of freedom associated with it. F is the value judging the correctness of a null hypothesis. *Between Cells* refers computation made at the factor levels. In our case, the SS and MS are computed in both code and machine level accordingly. That means SS and MS of *Between Cells* are computed by means over observations with overall mean. Here are the equations:

$$SSCells = n \sum_{ij} (\overline{y}_{ij.} - \overline{y}_{...})^2, \tag{6}$$

$$MSCells = SSCells/(a \cdot c - 1). \tag{7}$$

For factor A, the SS and MS are computed as

$$SSA = c \cdot n \sum_i (\overline{y}_{i..} - \overline{y}_{...})^2, \tag{8}$$

$$MSA = SSA/(a-1). \tag{9}$$

For factor C, the SS and MS are computed as

$$SSC = a \cdot n \sum_j (\overline{y}_{.j.} - \overline{y}_{...})^2, \tag{10}$$

$$MSC = SSC/(c-1). \tag{11}$$

For interaction $A \cdot C$, the SS and MS are computed as

$$SSAC = SSCells - SSA - SSC, \tag{12}$$

$$MSAC = SSAC/(a-1)(c-1). \tag{13}$$

Note that $SSCells$ is the sum of $SSAC$, SSA, and SSC.

Within Cells (the *Error* term) can be computed as

$$SSW = TSS - SSCells, \tag{14}$$

$$MSW = \frac{SSW}{a \cdot c \cdot (n-1)}. \tag{15}$$

Total SS (TSS) is computed in the following equation:

$$TSS = \sum_{ijk} (y_{ijk} - \overline{y}_{...})^2 \tag{16}$$

F values are computed by MS's divided by MSW accordingly. It is the mean square of *Error*.

3.4 Contrast and Post Hoc Comparisons

Many statistical methods exist for classification and grouping. We use two known classification methods in our study. They are Contrast and Post Hoc Comparisons. A contrast is a linear function of means whose coefficients add to zero. In contrast method, *t test* is used to judge the null hypotheses [6]. We use an example to explain the contrast method.

For the experimental environment described in Section 2, suppose we would like to compare whether two machines have the same effect on these codes, with measured *cpi* over these codes on these two machines. Then the null hypothesis is:

$$H_0 : L = \mu_1 - \mu_2$$

where μ_1 is the average of *cpi* over the codes on PowerChallenge and μ_2 is the average of *cpi* over the codes on Origin2000. The *t test* is defined as

$$t = \frac{\sum \alpha_i \overline{y}_{i..}}{\sqrt{\frac{MSW}{n \sum \alpha_i^2}}}, \tag{17}$$

where α_i, $i = 1, 2$, are the coefficients of L. In the null hypothesis defined above,

$$\sum \alpha_i = 1 - 1 = 0 \tag{18}$$

$\overline{y}_{i..}$ is the sample value of μ_i. MSW is the *Error* term in Table 2 and n is the number of observations. The evaluation of *t test* is similar to that of F test in judging null hypotheses. If the probability of t value is less than 0.05 then the null hypothesis is rejected, otherwise you cannot reject the null hypothesis.

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 *t test*. LSD is the easiest one in rejecting a null hypothesis; Duncans is less easy than LSD; Tukey is less easy than Duncans; and Scheffe is the most difficult one in rejecting a null hypothesis. We have used LSD in our experiment.

3.5 Regression Method for Scalability Testing

A regression method has been used by Lyon et. al. to evaluate the scalability of parallel processing [11]. With some modification, here we extend the regression method to data scalability of memory systems. Again, we use a simple example to illustrate the regression method. In 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 Origin2000 as machine level 2. Following the regression method, we need to assign a value to each of the code and machine levels. Conventionally, these values are small integers. Assign level 1 to -1 and level 2 to value 1 for both size and machine level accordingly, we have the index table, Table 3.

In Table 3, X_c is the indicator variable for code; X_m is the indicator variable for machine; $I_{c,m}$ is the indicator variable for interaction. If μ is a constant term, then we have a regression model:

$$cpi = \mu + \beta_c X_c + \beta_m X_m + \beta_{c,m} I_{c,m} \tag{19}$$

Table 3. The Index Table of a Regression Experiment

X_c	X_m	$I_{c,m}$	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

Substitute the values in Table 3 into equation (19), we have

$$a = \mu - \beta_c - \beta_m + \beta_{c,m} \tag{20}$$

$$b = \mu - \beta_c - \beta_m - \beta_{c,m} \tag{21}$$

$$c = \mu - \beta_c + \beta_m - \beta_{c,m} \tag{22}$$

$$d = \mu + \beta_c + \beta_m + \beta_{c,m} \tag{23}$$

solving these equations, we have

$$\mu = \frac{a+b+c+d}{4} \tag{24}$$

$$\beta_c = \frac{-a+b-c+d}{4} \tag{25}$$

$$\beta_m = \frac{-a-b+c+d}{4} \tag{26}$$

$$\beta_{c,m} = \frac{a-b-c+d}{4} \tag{27}$$

The term $\beta_{c,m}$ is the interaction effect. It is tested by *t test* to see whether the interaction effect exists. The null hypothesis tested here is:

$$H_0 : \beta_{c,m} = 0 \tag{28}$$

If the probability of t value is less than 0.05, then $\beta_{c,m} < 0$ leads to the conclusion that the code is more scalable on the level 2 machine than on the level 1 machine; $\beta_{c,m} = 0$ leads to the conclusion that the code has the same scalability on the level 1 and 2 machine; otherwise, $\beta_{c,m} > 0$ 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 which is greater than one for each of the two factors.

4 A Methodology for Hierarchical Memory Systems

We have developed a hierarchical evaluation methodology for advanced memory systems based on the knowledge introduced in Section 3. This methodology consists of four levels of evaluation. All of the four levels of evaluation are based on two-factor factorial statistical methods. 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 predicting the performance when problem size scales up and to suggest further memory system improvements.

4.1 Level One Evaluation: Main Effect

Level one evaluation uses the two-factor factorial experiment (see Section 3.3) to find the effects of code and machine. Using the two factors code and machine, it detects the overall effect of code, machine, and their interaction on the final performance. The dependent variable for the two-factor factorial design is *cpi*. The random samples for each of the code-machine level combination are chosen from different problem sizes within the interested problem size range. So, the effect comparison is based on the mean performance over different problem sizes. If code effect exists, we conclude that the codes have different memory reference patterns which diverge memory access time. When 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 of the effects that exist deserve further investigation to identify the source or sources.

Based on Section 3.3, the result of the two-factor factorial experiment can be given in the format as shown in Table 2. This result table can be generated by *SAS* procedure *PROC GLM* [12]. To be self-complete, the algorithm to compute these needed parameters is listed below.

Algorithm of Main Effects

0) Compute Cell Means $\overline{y}_{ij.}$, Machine Means $\overline{y}_{.j.}$, Code Means $\overline{y}_{i..}$ and Overall Mean $\overline{y}_{...}$

1) Compute TSS, $SScells$, SSW, SSA, SSC, and $SSAC$

2) Compute all the degrees of freedom, such as $a-1$, $c-1$, $a \cdot c-1$, and so on.

3) Compute $MSCells$, MSA, MSC, $MSAC$, and MSW

4) Get F values by using SS divided by the degree of freedom

4.2 Level Two Evaluation: Code/Machine Classification

Level one evaluation detects the overall effect 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. The key technique to single out outstanding contributors is to find the relative performance of a code/machine with that of others. Statistical classification methods provide a means to group code/machine based on their relative performance.

The Contrast method and Post Hoc comparisons introduced in Section 3.4 are classical statistical methods for classification. We have used the contrast method and all the four Post Hoc methods in our study. These methods have different classification criteria.

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 under the interested 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.

Algorithm of Contrast Method

0) Repeat the same steps for constructing Table 2

1) Compute $\sum \alpha_i \overline{y}_{i..}$ and $\sum \alpha_i^2$

2) Compute

$$t = \frac{\sum \alpha_i \overline{y}_{i..}}{\sqrt{\frac{MSW}{n \sum \alpha_i^2}}}$$

3) Judge the testing null hypothesis by using the probability of t *value*.

4.3 Level Three Evaluation: Scalability Comparison

Both level one and level two evaluation evaluate the performance over a set of codes and machines. The third step of our evaluation methodology is individual evaluation for outliers. It compares the data scalabilities of a given code on different machines. As shown in [8], scalability itself is not a

measurement of parallel processing gain. It is a factor that contributes to the ability of a system to deliver the expected performance. Level two evaluation has grouped codes based on their average performance over the range of problem sizes. Data scalability measures the performance variation when problem size scales up. Memory scalability evaluation is a new approach. It evaluates the ability of a memory system in handling large data sizes. The same or a better initial performance combined with a better scalability guarantees a code will have a better performance when problem size scales up. A code with a smaller initial *cpi* and a 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 3.5. The two factors 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. SAS procedure PROC REG [12] can be used to determine results of the regression method. The algorithm for the statistical scalability evaluation is given below.

Algorithm of Scalability Comparison

0) Assign a value for each of the factor levels and construct the index table

1) Substituting values in the index table to equation

$$cpi = \mu + \beta_c X_c + \beta_m X_m + \beta_{c,m} I_{c,m}$$

2) Solve the linear system generated in Step 1.

3) Judge the term $\beta_{c,m}$ by the probability of t *value*.

4.4 Level Four Evaluation: Memory Hierarchy

As discussed in the previous section, the performance of a code may vary with problem size and the 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 four evaluation compares the performance variation of primary components of the underlying memory systems. Combined with the level two 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 four evaluation is the same as that of level three evaluation, except for the dependent variables. The actual design of level four evaluation varies with the underlying memory structure. As discussed in Section 2, the memory hierarchy of SGI PowerChallenge and Origin2000 has four primary components: $L1$ cache, $L2$ cache, *outstanding cache misses*, and main memory. $L1$, $L2$ 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.

The same SAS procedure, *PROC REG*, can be used in level four evaluation as it is used in the scalability evaluation. As shown by the algorithm given below, the inputs of the SAS procedure are different for level three and four evaluations.

Algorithm for Memory Structure Evaluation

0) Assign a value to each of the factor levels and construct the index table

1) Substituting values in the index table into equations

$$L1 = \mu + \beta_c X_c + \beta_m X_m + \beta_{c,m} I_{c,m},$$

and

$$L2 = \mu + \beta_c X_c + \beta_m X_m + \beta_{c,m} I_{c,m},$$

separately.

2) Solve the two linear system generated in Step 1 individually.

3) Judge the term $\beta_{c,m}$ by the probability of t *value*.

4) Determine the performance variation of each of the three primary components.

The two systems generated and solved in Step 1 and 2 are for our experimental environment. In general, if there are m components that need to be compared, m systems will be generated and solved in Step 1 and 2 respectively.

5 Evaluation of SGI PowerChallenge and Origin2000

To verify the feasibility and correctness, we have applied the four level evaluation methodology to the computing environment discussed in Section 2. All four levels of evaluation have been used to evaluate these ASCI machines and benchmarks. Experimental results show that this newly proposed methodology 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 Origin2000, are assigned machine level 1 and level 2, respectively. The five codes, HEAT, HYDRO, SWEEP, DSWEEP, and HYDROT, are assigned a level value of 1, 2, 3, 4, 5, respectively. We have used the SAS solving environment [12] through out 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.

5.1 Main and Interaction Effects

The relationship between code and machine is first investigated. To catch the mean relationship over the interested 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 3.3 is used to find the effects. The GLM procedure of SAS is used to carry the two-factor factorial experiment for level one evaluation. Table 4 and 5 shows results from GLM.

Table 4. Class Level Information

Class	Levels	Values
MACHINE	2	1 2
CODE	5	1 2 3 4 5

Number of observations in data set = 113

Table 5. Mean Effects Table

Dependent Variable:	cpi				
		Sum of	Mean		
Source	DF	Squares	Square	F Value	$Pr > F$
Model	9	112.5410006	12.5045556	27.44	0.0001
Error	103	46.9436516	0.4557636		
Corrected Total	112	159.4846523			
	R-Square	C.V.	Root MSE	CPI Mean	
	0.705654	34.64445	0.675103	1.948661	
Source	DF	Type I SS	Mean Square	F Value	$Pr > F$
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

Table 4 lists the GLM model class level information. Table 5 is the mean effects table (see

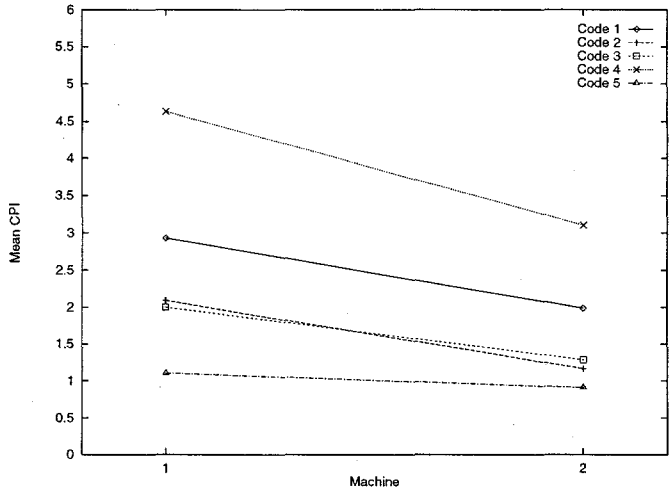

Figure 1. Machine Mean Distribution

Table 2) of the factorial experiment. It consists of two sectors separated by the double-line. The upper table is for overall effect and the lower table is for individual effects. Look at row four of Table 5. The F value is 27.44 and the probability of F ($Pr > F$) is 0.0001. The probability of F is less than 0.05. The hypothesis of overall-effect does not exist is rejected. This means that code or machine effects exist. The lower table is a continuation of the upper table to locate the potential effects. Look at row two of the lower table. The probability of F is $0.0001 < 0.05$, which suggests that 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. 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 Figure 1, the code performance crosses over the two machines between code 2 and code 3. This line crossing indicates the existence of interaction effect of machine and code. It confirms the results given by Contrast method (see Table 5). However, code 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, 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, statistically, code 2 and 3 are the same (see Table 7). The two lines between code 2 and code 3, therefore, statistically are merged to one line.

Figure 2 plots the codes performance over the two machines. We can see that machine 2 always outperforms machine 1. Machine effect exists. Based on two-factor factorial mechanisms the GLM procedure systematically finds the main and interaction effects, which sometimes can be determined easily through visual display, and other times cannot.

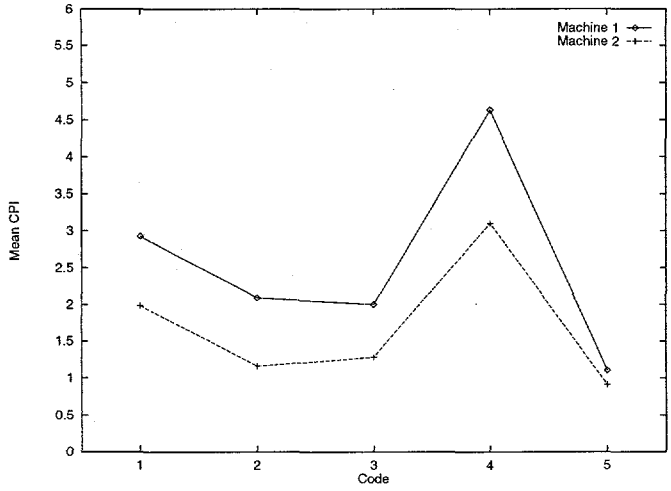

Figure 2. Code Mean Distribution

5.2 Code and Machine Classification

The codes and machines have been classified based on the Contrast and Post Hoc comparisons introduced in Section 3.4 and 4.2. The Contrast procedure of SAS is used for the Contrast comparison. The result of the pairwise code/machine Contrast comparison is given in Table 6 below.

Table 6. 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
Machine1 vs. Machine2	1	19.78987372	19.78987372	43.42	0.0001

In Table 6, except at row nine, all the probability of rejection is less than 0.05. Code HYDRO and SWEEP are in the same group. They have similar performance variations caused possibly by the computational 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. Table 7 and 8 gives the result of the code and machine classification respectively. From Table 7 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 to 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.

Table 7. LSD Post Hoc Comparison for Code

T Grouping	Mean	N	CODD
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)

As shown in Table 8, PowerChallenge and Origin2000 are classified into two different groups. The distance between the two groups is larger than 0.2522 (least significant difference = 0.2522 *cpi*). The Origin2000 is always better than PowerChallenge for the set of codes under consideration. This result again matches that of Contrast comparison well.

Table 8. LSD Post-Hoc Comparison for machines

T Grouping	Mean	N	MACHINE
A	2.3217	54	1 (PowerChallenge)
B	1.65552	59	2 (Origin2000)

5.3 Scalability Comparison

Using the regression method discussed in Section 3.5 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 two evalu-

ation 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. The procedure PROG REG of SAS is used for the scalability comparison. The response variable is *cpi*. Table 9 is generated by PROG REG for the scalability comparison of HEAT over problem size range [50,100].

Table 9. Scalability Comparison of HEAT

		Parameter	Standard	T for H0:	
Variable	DF	Estimate	Error	Parameter=0	$Prob > \|T\|$
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

In Table 9, the "INTAC" stands for INTerACtion effect. Recall that the probability to test whether an interaction is zero is 0.05. At the 0.0001 level (see last column of Table 9), the hypothesis of zero effect has been rejected. The interaction effect exists. The parameter estimate of "INTAC" is 0.0795, which means that the term $\beta_{c,m}$ is positive (see equation (19)) and the performance difference of the two machines decreases with problem size. PowerChallenge is more scalable than Origin2000 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 ratio 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, for HYDRO, the parameter estimate is $-0.050885 < 0$, which means that the performance difference between the two machines increases with problem size. Origin2000 has a better scalability than PowerChallenge for HYDRO. The scalability improvement may be due to Origin2000'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. The probabilities for rejecting zero interaction effects for these codes are larger than 0.05. Our no-effect hypotheses stands. The more advanced memory system of Origin2000 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 10 lists results generated by PROG REG for scalability analysis of SWEEP. From Table 10, the probability level of interaction effect is 0.2216, which is greater than 0.05. Therefore,

$\beta_{c,m} = 0$ 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.

Table 10. Scalability Comparison of SWEEP

		Parameter	Standard	T for H0:	
Variable	DF	Estimate	Error	$Parameter = 0$	$Prob > \|T\|$
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

5.4 Evaluation of Memory Components

As discussed in Section 2, 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 four 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 L1 and L2 hit ratio are known.

Table 11 and 12 show the analysis table for L1 and L2 hit-ratio variation of HEAT. We can see from Table 11, the probability level of "INTAC" is $0.3156 > 0.05$. 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 12, code-machine interaction effect exists ($\alpha = 0.001 < 0.05$) and the effect is negative ($\beta_{c,m} = -0.005011 < 0$). 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 Origin2000 goes down relative to PowerChallenge, when problem size scales up. As we know from Section 5.3, HEAT has a better scalability on PowerChallenge than on Origin2000. The relative L2 hit-ratio decrease explains the smaller scalability of Origin2000.

Recall that the underlying SGI PowerChallenge and Origin2000 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 of the five codes under study. HEAT has demonstrated how the regression method can be used repeatedly for different components of a memory system. For the rest of the codes, we will not list the results for L1 cache since it does not contribute to performance variations.

Table 13 is the L2 hit-ratio analysis table for HYDRO. As given in Table 13, the hypothesis

Table 11. L1 Hit-Ratio Comparison for HEAT

Variable	DF	Parameter Estimate	Standard Error	T for H0: $Parameter = 0$	$Prob > \|T\|$
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 12. L2 Hit-Ratio Comparison for HEAT

Variable	DF	Parameter Estimate	Standard Error	T for H0: $Parameter = 0$	$Prob > \|T\|$
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

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 5.3, HYDRO-Origin2000 has a better scalability than HYDRO-PowerChallenge. This scalability increase is not due to the larger L2 cache of Origin2000 as shown by the cache hit ratios across machines. It is due to the outstanding cache misses ability and faster main memory access time supported by Origin2000. Combined with an existing empirical model [13], a detailed analysis is given in [14] to understand the performance.

Table 13. L2 Hit-Ratio Comparison for HYDRO

Variable	DF	Parameter Estimate	Standard Error	T for H0: $Parameter = 0$	$Prob > \|T\|$
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

Table 14 and 15 are the analysis table for L2 cache comparison for SWEEP and DSWEEP respectively. By Table 14, interaction effect exists for SWEEP and the effect is negative. The L2 hit ratio of SWEEP on Origin2000 becomes relatively smaller compared with that of PowerChallenge when problem size scales up. Since SWEEP has the same scalability on these two machines, the

main memory contribution and/or the outstanding cache miss ratio must be improved on Origin2000 when problem size scales up. The outstanding cache-miss principle and faster main memory access also work well for SWEEP when problem size is large [14].

Table 14. L2 Hit-Ratio Comparison for SWEEP

		Parameter	Standard	T for H0:	
Variable	DF	Estimate	Error	$Parameter = 0$	$Prob > \|T\|$
INTERCEP	1	0.826199	0.00302764	272.886	0.0001
CODE	1	-0.013206	0.00110554	-11.945	0.0001
MEMORY	1	0.042485	0.00302764	14.032	0.0001
INTAC	1	-0.003833	0.00110554	-3.467	0.0047

Like HYDRO, DSWEEP maintains a constant L2 hit-ratio difference on the two machines. DSWEEP has the same scalability on the two machines. When L1, L2 hit-ratio difference remain unchanged, the difference of main memory contribution toward the final performance is also unchanged [14]. Therefore, we can conclude that DSWEEP's outstanding cache-miss ratio does not vary with problem size.

Table 15. L2 Hit-Ratio Comparison for DSWEEP

		Parameter	Standard	T for H0:	
Variable	DF	Estimate	Error	$Parameter = 0$	$Prob > \|T\|$
INTERCEP	1	0.810041	0.00804876	100.642	0.0001
CODE	1	-0.032839	0.00402438	-8.160	0.0001
MEMORY	1	0.077890	0.00804876	9.677	0.0001
INTAC	1	0.002200	0.00402438	0.547	0.5966

Finally, Table 16 lists the L2 hit-ratio comparison for HYDROT. HYDROT has the same effect as SWEEP. Its L2 hit-ratio difference remains the same and has the same scalability on the two machines, as given in the previous section. Like SWEEP, HYDROT's outstanding cache-miss ratio does not change with problem size.

The four-level evaluation methodology proposed in Section 4 has been applied to analyze the performance of two ASCI machines and five benchmarks available at Los Alamos National Laboratory. In the level one 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 two 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 inherented memory reference/computation patterns, the Origin2000

Table 16. L2 Hit-Ratio Comparison for HYDROT

Variable	DF	Parameter Estimate	Standard Error	T for H0: $Parameter = 0$	$Prob > \|T\|$
INTERCEP	1	0.918992	0.00278790	329.636	0.0001
CODE	1	-0.006044	0.00062339	-9.695	0.0001
MEMORY	1	0.023656	0.00278790	8.485	0.0001
INTAC	1	-0.002718	0.00062339	-4.360	0.0002

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 Origin2000, by level three 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 four 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 Origin2000 have played an important role in performance improvement. This is especially true for HYDRO and SWEEP.

6 Conclusions

We have proposed a hierarchical statistic methodology for memory system evaluation. This newly proposed methodology is developed based on three conventional statistical techniques and the concept of scalability analysis. It is robust, general, systematic, and automatic. It is built upon solid mathematic foundations, does not require detailed understanding of the underlying memory systems, is equipped with multiple levels of evaluation for an adaptive study, and is supported by practical algorithms such that the evaluation can be carried out automatically. This statistical methodology is different with queuing theory based probability methods and curve-fitting based regression methods, where the former requires an appropriate modeling of the memory system and the latter is short on information for advance memory structures. This methodology consists of four levels of evaluation. The first two levels find memory reference patterns and their influence on different memory systems. The second two levels determine the performance variation when problem sizes scale up. The last two levels of evaluation are based on scalability analysis which is a new approach for memory system evaluation. The combination of the four levels of evaluation

makes the proposed methodology unique and more appropriate for advance memory systems than existing methodologies.

Unlike many existing statistic and stochastic methods, the newly proposed methodology is not designed to determine parameters of a pre-assumed performance model. Instead, it is built on the approach of relative performance comparison, which is one of the most important concerns in architectural design and algorithmic development, and is set to reach a balance of simplicity and effectiveness. The methodology compares the relative impact of codes, machines, codes and machines, and components of machines toward the final performance. It also compares the relative performance variation when problem sizes scale up, in terms of scalability. It is a post evaluation methodology. This newly proposed method can be used collectively with existing empirical and analytical models for quantitatively assessing the contribution of low-level system components toward the final performance.

We have focused on advanced memory systems for sequential processing in this study due to our current research interests. The statistical techniques and scalability analysis mechanisms used in this study, however, are general. With moderate modifications, they can be extended to other machine architectural evaluations and to parallel computers. The applicability and extendibility of the proposed statistical methodology needs to be further explored in future research.

References

[1] S. McKee, R. L;emle, K.L.Wright, W. Wulf, M. Salinas, J. Aylor, and A. Batson, "Smarter memory: Improving bandwidth for stramed references," *IEEE Computer*, no. 7, pp. 54–63, 1998.

[2] D. Burger, J. Goodman, and A. Kagi, "The declining effectiveness of dynamic caching for general-purpose microprocessors." Tech. Report CS-TR-95-1261, Jan. 1995.

[3] MIPS Technologies, Inc., "R10000 microprocessor product overview." MIPS Product Preview, 1995.

[4] K. Yeager, "The MIPS R10000 superscalar microprocessor," *IEEE Micro*, pp. 28–40, Apr. 1996.

[5] M. Zagha, B. Larson, S. Turner, and M. Itzkowitz, "Performance analysis using the mips r10000 performance counters," in *Proc. of Supercomputing'96*, Nov. 1996.

[6] R. Freund and W. Wilson, *Statistical Methodes.* Academic Press, Inc, 1997.

[7] R. Jain, *The Art of Computer System Performance Analysis.* John Wiley & Sons, 1991.

[8] X.-H. Sun and D. Rover, "Scalability of parallel algorithm-machine combinations," *IEEE Transactions on Parallel and Distributed Systems*, pp. 599–613, June 1994.

[9] X.-H. Sun and L. Ni, "Scalable problems and memory-bounded speedup," *J. of Parallel and Distributed Computing*, vol. 19, pp. 27–37, Sept. 1993.

[10] X.-H. Sun, "Scalability versus execution time in scalable systems." Louisiana State Uiversity, Computer Science TR-97-003, 1997.

[11] G. Lyon, R. Kacker, and A. Linz, "A scalability test for parallel code," *SOFTWARE: Practice and Experience*, vol. 25, pp. 1299–1314, Dec. 1995.

[12] SAS Institute Inc., *SAS User's Guide.* SAS Institute Inc., 1996.

[13] O. M. Lubeck, Y. Luo, H. Wasserman, and F. Bassetti, "An empirical hierarchical memory model based on hardware performance counters," in *Proceeding of PDPTA '98*, July 1998.

[14] X.-H. Sun, K. Cameron, D. He, and Y. Luo, "A memory-centric characterization of ASCI applications via a combined approach of statistical and empirical analysis." in preparation, 1998.