A Statistical-Empirical Hybrid Approach to Hierarchical Memory Analysis

Xian-He Sun[1] and Kirk W. Cameron[2]

[1] Illinois Institute of Technology, Chicago IL 60616, USA
[2] Los Alamos National Laboratory, Los Alamos NM 87544, USA

Abstract. A hybrid approach that utilizes both statistical techniques and empirical methods seeks to provide more information about the performance of an application. In this paper, we present a general approach to creating hybrid models of this type. We show that for the scientific applications of interest, the scaled performance is somewhat predictable due to the regular characteristics of the measured codes. Furthermore, the resulting method encourages streamlined performance evaluation by determining which analysis steps may provide further insight to code performance.

1 Introduction

Recently statistics have provided reduction techniques for simulated data in the context of single microprocessor performance [1, 2]. Recent work has also focused on regressive techniques for studying scalability and variations in like architectures statistically with promising but somewhat limited results [3]. Generally speaking, if we were to combine the strength of such comparisons with a strong empirical or analytical technique, we could conceivably provide more information furthering the usefulness of the original model. A detailed representation of the empirical memory modeling technique we will incorporate in our hybrid approach can be found in [4].

2 The Hybrid Approach

2.1 The Hybrid Approach: Level 1

We will use *cpi*, cycles per instruction, to compare the achievable instruction-level parallelism (ILP) of particular code-machine combinations. We feel that great insight can be gathered into application and architecture performance if we break down *cpi* into contributing pieces. Following [5] and [6], we initially break *cpi* down into two parts corresponding to the pipeline and memory *cpi*.

$$cpi = cpi_{pipeline} + cpi_{memory} \tag{1}$$

Level one of the hybrid approach focuses on using two-factor factorial experiments to identify the combinations that show differences in performance that

warrant further investigation. Following the statistical analysis method in [3], we identify codes and machines as observations to be used in the two-factor factorial experiments. Once all measurements have been obtained, we can perform the experiments for the factors code and machine. Using statistical methods with the help of the SAS statistical tool, we gather results relating to the variations present among codes, machines and their interactions. We accomplish this via a series of hypothesis experiments where statistically we determine whether or not a hypothesis is true or false. This is the essence of the two-factor factorial experiment. This allows us to identify within a certain tolerance, the differences among code-machine combinations.

Hypothesis: Overall effect does not exist. For this experiment, the dependent variable is the overall average *cpi* measured across codes for the machines. With these parameters, disproving the hypothesis indicates that in fact, differences between the architectures for these codes exist. If this hypothesis is not disproved, then we believe with some certainty, that there are no statistical differences among the two architectures for these codes. If this hypothesis is rejected, then the next three hypotheses should be visited.

Hypothesis: Code effect does not exist. For this experiment, the dependent variable is the $cpi_{pipeline}$ term from the decoupled *cpi* of Equation 1. In practice, this term is experimentally measured when using the empirical model. If the hypothesis holds in this experiment, no difference is observed statistically for these codes on these machines at the pipeline level. Conversely, if the hypothesis is rejected, code effect does exist indicating differences at the pipeline level for this application on these architectures. In the empirical model context, if this occurs, further analysis of the $cpi_{pipeline}$ term is warranted.

Hypothesis: Machine effect does not exist. For this experiment, the dependent variable is the cpi_{memory} term from the decoupled *cpi* of Equation 1. This term can be derived experimentally as well. If the hypothesis holds in this experiment than no discernible difference between these machines statistically is apparent for these codes. Otherwise, rejecting this hypothesis indicates machine effect does exist. In the case of the empirical memory model, this warrants further investigation since it implies variations in the memory performance across code-architecture combinations.

Hypothesis: Machine-code interaction does not exist. For this experiment, the dependent variable is overall *cpi* measured across individual codes and individual machines. If this hypothesis is held, then no machine-code interaction effects are apparent statistically. Otherwise, rejecting the hypothesis begs for further investigation of the individual codes and machines to determine why machine-code interaction changes the performance across machines. Such performance differences indicate that codes behave differently across different machines in an unexpected way, hence requiring further investigation.

2.2 The Hybrid Approach: Level 2

If code effect exists, study $cpi_{pipeline}$. This indicates fundamental differences at the on-chip architectural level. The empirical memory model does not provide insight to such performance differences, treating $cpi_{pipeline}$ as a black box. Another model, such as that found in [7] could be used to provide more insight to performance variations for such a code.

If machine effect exists, study cpi_{memory}. If machine effect exists, statistical variations are present between different codes at the memory hierarchy level across machines. This is exactly the purpose of the empirical memory model: to analyze contributions to performance from the memory hierarchy. At this point, the statistical method has provided an easy method for determining when further analysis using the memory model is necessary. This requires a more detailed look at the decoupled *cpi* in Equation 1.

Latency hiding techniques such as out-of-order execution and outstanding memory loads increase performance. We can no longer calculate overall *cpi* as simply the dot-product of maximum latencies T_i at each i level of the hierarchy and the associated number of hits to level i, h_i. We require the average latency incurred at each level in the hierarchy, t_i. Furthermore, if we define a term that expresses the ratio of average latencies to maximum latencies in terms of *cpi* we can express overall *cpi* in the following form:

$$cpi = cpi_{pipeline} + (1 - m_0) \sum_{i=2}^{nlevels} h_i * T_i \tag{2}$$

It is obvious that this is another representation of Equation 1. m_0 is formally defined as one minus the ratio of the average memory access time to the maximum memory access time:

$$m_0 = 1 - \frac{\sum_{i=2}^{nlevels} h_i * t_i}{\sum_{i=2}^{nlevels} h_i * T_i}. \tag{3}$$

m_0 quantifies the amount of overlap achieved by a processor that overlaps memory accesses with execution. $(1 - m_0)$ is the portion of time spent incurring the maximum latency.

The above equations would indicate that m_0 reflects the performance variations in *cpi* when $cpi_{pipeline}$ is constant. Calculating m_0 is costly since it requires a least square fitting first to obtain each t_i term. By applying the statistical method and through direct observation, we have isolated the conditions under which it is worthwhile to calculate the terms of Equation 2. For conditions where machine effect exists, m_0 will provide useful insight to the performance of the memory latency hiding effects mentioned. We can also use m_0 statistically to describe the scalability of a code in regard to how predictable the performance is as problem size increases. We can use other variations on the original statistical method to study the variations of m_0. This is somewhat less costly than determining m_0 for each problem size and machine combination. Nonetheless,

actually calculating m_0 values provides validation to the conclusions obtained using this technique. If m_0 values show no statistical variations or are constant as problem sizes increase, performance scales predictably and m_0 can be used for performance prediction of problem sizes not measured. If m_0 values fluctuate statistically or are not constant as problem size increases, performance does not scale predictably and cannot be used for performance prediction.

m_0 values across machines can also provide insight into performance. If statistical differences across machines for the same problem are non-existent or if $m_0 - m_0'$ is constant as problem size increases, where each m_0 represents measurements for the same code over different machines, then the memory design differences make no difference for the codes being measured.

If machine-code interaction exists, study *cpi*. This corresponds to the fourth hypothesis. If machine-code effect exists, statistical variations are present when machine-code interactions occur. This indicates further study of the resulting *cpi* is necessary since there exist unexplained performance variations. This scenario is outside the scope of the hybrid method, but exactly what the statistical method [3] was intended to help analyze. Further focus on particular code and architecture combinations should be carried out using the statistical method.

3 Case Study

3.1 Architecture Descriptions

Single processor hierarchical memory performance is of general interest to the scientific community. For this reason, we focus on a testbed consisting of an SMP UMA architecture and a DSM NUMA architecture that share the same processing elements (the MIPS R10000 microprocessor) but differ in the implementation of the memory hierarchy. The PowerChallenge SMP and the Origin 2000 DSM machines offer state-of-the-art performance with differing implementations of the memory hierarchy.

The 200MHz MIPS R10000 microprocessor is a 4-way, out-of-order, superscalar architecture [8]. Two programmable performance counters track a number of events [9] on this chip and were a necessity for this study. 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 on PowerChallenge is equipped with up to 2MB L2 cache while a CPU of Origin 2000 system always has a L2 cache of 4MB.

In our context, we are only concerned with memory hierarchy performance for a dedicated single processor. As mentioned, the PowerChallenge and Origin 2000 differ primarily in hierarchy implementation and we will not consider shared memory contributions to performance loss since all experiments have been conducted on dedicated single processors without contention for resources.

3.2 ASCI Representative Workloads

Four applications that form the building blocks for many nuclear physics simulations were used in this study. A performance comparison of the Origin and PowerChallenge architectures has been done using these codes [10] along with a detailed discussion of the codes themselves. In the interest of space, we provide only a very brief description of each code.

SWEEP is a three-dimensional discrete-ordinate transport solver. HYDRO is a two-dimensional explicit Lagrangian hydrodynamics code. HYDROT is a version of HYDRO in which most of the arrays have been transposed so that access is now largely unit-stride. HEAT solves the implicit diffusion PDE using a conjugate gradient solver for a single timestep.

3.3 Hybrid Analysis

We now apply the hybrid method to draw conclusions regarding our codes. We should note that some of the statistical steps involved can be performed by simple inspection at times. For simple cases this can be effective, but generally simple inspection will not allow quantification of the statistical variance among observations. For this reason, we utilize statistical methods in our results. Inspection should certainly be used whenever the confidence of conclusions is high. We will not present the actual numerical results of applying statistical methods to our measurements due to restrictions on space. We will however provide the general conclusions obtained via these methods, such as whether or not a hypothesis is rejected. The observations used in our experiments include various measurements for the codes mentioned at varying problem sizes. All codes were measured on both machines using the same compiled executable to avoid differences and with the following problem size constraints: HEAT [50,100], HYDRO [50,300], SWEEP [50,200], and HYDROT [50,300].

Level 1 Results For the first hypothesis, "overall effect does not exist," we use level one of the original statistical model. A straight-forward two-factor factorial experiment shows that in fact the hypothesis is rejected. This indicates further study is warranted and so, we continue with the next 3 hypotheses. Using $cpi_{pipeline}$ as the dependent variable, the two-factor factorial experiment is performed over all codes and machines to determine whether or not code effect exists. Since identical executables are used over the two machines, no variations are observed for $cpi_{pipeline}$ values over the measured codes. This is expected as the case study was prepared to focus on memory hierarchy differences. Thus the hypothesis holds, and no further study of $cpi_{pipeline}$ is warranted for these code-machine combinations.

Next, we wish to test the hypothesis "machine effect does not exist". We perform the two-factor factorial experiment using cpi_{memory}. The results show variations for the performance of cpi_{memory} across the two machines. This will require further analysis in level two of the hybrid model. Not rejecting the hypothesis would have indicated that our codes perform similarly across machines.

The third hypothesis asks whether "machine-code interaction exists". In fact, performing the two-factor factorial experiment, shows that machine-code interaction is present since we reject the hypothesis. This will have to be addressed in level two of the hybrid model as well.

Level 2 Results Now that we have addressed each of the hypothesis warranted by rejection of the "overall effect" hypothesis, we must further analyze anomalies uncovered (i.e. each rejected hypothesis). We have identified code effect existence in level 1. It is necessary to analyze the m_0 term of Equation 2. Statistical results and general inspection show strong variations with problem size in HYDRO on the Origin 2000. Less fluctuations, although significant occur for the same code on the PowerChallenge. This indicates that unpredictable variations are present in the memory performance for HYDRO. As problem size scales, the m_0 term fluctuates indicating memory accesses do not achieve a steady state to allow performance prediction for larger problem sizes. Performing the somewhat costly linear fitting required by the empirical model supports the conclusions as shown in Figures 1 and 2. In these figures, problem size represents the $y - axis$ and calculated m_0 values have been plotted. The scalability of HYDRO is in question since the rate at which latency overlap contributes to performance fluctuates.

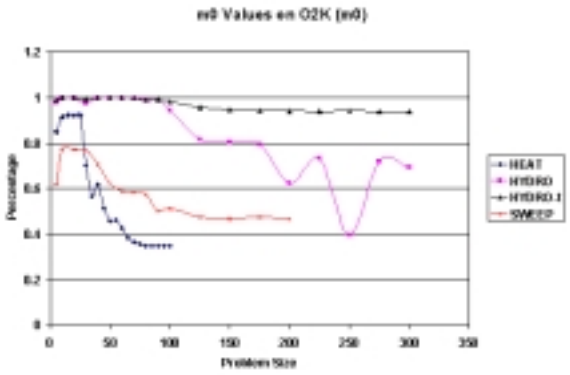

Fig. 1. m_0 values calculated on the Origin 2000.

On the other hand HEAT, HYDROT, and SWEEP show indications of predictability on the PowerChallenge. Statististical analysis of m_0 for problem sizes achieving some indication of steady state (greater than 50 for these codes - necessary to compensate for cold misses, counter accuracy, etc.) reveals little variance in m_0. For problem sizes [50,100], [75,300], and [50,200] respectively, m_0 is close to constant indicating the percentage of contribution to overlapped performance is steady. This is indicative of a code that both scales well and is somewhat predictable in nature over these machines. For these same codes on the Origin

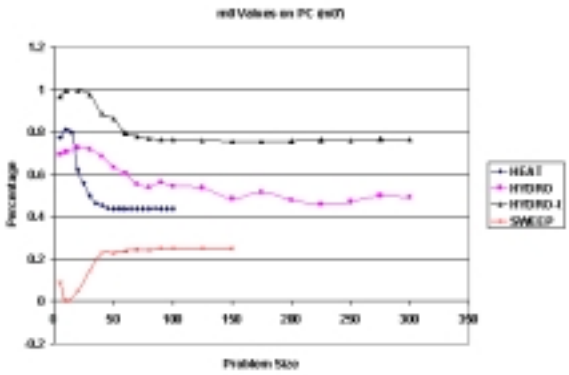

Fig. 2. m_0 values calculated on the PowerChallenge.

2000, larger problem sizes are necessary to achieve little variance in m_0. Respectively, this occurs at sizes of [75,100], [100,300], and [100,200]. The shift this time is due to the cache size difference on the Origin 2000. It takes larger problem sizes to achieve the steady state of memory behavior with respect to the latency tolerating features previously mentioned. For both machines, these three codes exhibit predictable behavior and generally good scalability.

For two codes, HEAT and HYDROT, the fluctuations in the differences between m_0 values are minimal. This can be confirmed visually in a figure not presented in this paper due to space. Such results indicate scaling between machines for these two codes over these two machines is somewhat predictable as well. HYDRO and SWEEP show larger amounts of variance for differences in m_0 values conversely. The scalability across the two machines for these codes should be analyzed further.

Finally, we must address the rejected hypothesis of machine-code interaction. Identifying this characteristic is suitable for analysis by level 2 of the original statistical method since it is not clear whether the memory architecture influence is the sole contributor to such performance variance. The statistical method refined for individual code performance [3], shows that the variance is caused by performance variations in 2 codes. Further investigation reveals that these two codes are statistically the same, allowing us to discount this rejected hypothesis.

4 Conclusions and Future Work

We have shown that the hybrid approach provides a useful analysis technique for performance evaluation of scientific codes. The technique provides insight previously not available to the stand-alone statistical method and the empirical memory model. Results indicate that 3 of the 4 codes measured show promising signs of scaled predictability. We further show that scaled performance of latency overlap is good for these same three codes. Further extensions to multi-processors

and other empirical/analytical models are future directions of this research. The authors wish to thank the referees for their suggestions regarding earlier versions of this paper. The first author was supported in part by NSF under grants ASC-9720215 and CCR-9972251.

References

1. R. Carl and J. E. Smith, *Modeling superscalar processors via statistical simulation*, Workshop on Performance Analysis and its Impact on Design (PAID), Barcelona, Spain, 1998.
2. D. B. Noonburg and J. P. Shen, *A framework for statistical modeling of superscalar processor performance*, 3rd International Symposium on High Performance Computer Architecture, San Antonio, TX, 1997.
3. X. -H. Sun, D. He, K. W. Cameron, and Y. Luo, *A Factorial Performance Evaluation for Hierarchical Memory Systems*, Proceedings of IPPS/SPDP 1999, April, 1999.
4. Y. Luo, O. M. Lubeck, H. Wasserman, F. Bassetti and K. W. Cameron, *Development and Validation of a Hierarchical Memory Model Incorporating CPU- and Memory-operation Overlap*, Proceedings of WOSP '98, October, 1998.
5. D. Patterson and J. Hennessy, *Computer Architecture: A Quantitative Approach*, Prentice Hall, pp.35-39 ,1998.
6. P. G. Emma, *Understanding some simple processor-performance limits*, IBM Journal of Research and Development, vol. 41, 1997.
7. K. Cameron, and Y. Luo, *Instruction-level microprocessor modeling of scientific applications*, Lecture Notes in Computer Science 1615, pp. 29–40, May 1999.
8. K. C. Yeager, The MIPS R10000 Superscalar Microprocessor, IEEE Micro, April, 1996, pp. 28–40.
9. M. Zagha, B. Larson, S. Turner, and M. Itzkowitz, Performance Analysis Using the MIPS R10000 Performance Counters, Proc. Supercomputing '96, IEEE Computer Society, Los Alamitos, CA, 1996.
10. Y. Luo, O. M. Lubeck, and H. J. Wasserman, *Preliminary Performance Study of the SGI Origin2000*, Los Alamos National Laboratory Unclassified Release LAUR 97-334, 1997.