Automatically Selecting Representative Traces for Simulation Based on Cluster Analysis of Instruction Address Hashes

Yue Luo and Lizy Kurian John
The University of Texas at Austin
Department of Electrical and Computer Engineering
IBM Mentor: Alex Mericas
Processor Performance, Advanced CEC Engineering
IBM Server Group

Abstract

Simulation of standard benchmarks is the most important method for computer architects to evaluate architectural enhancement. However, modern benchmarks usually take prohibitively long time to run in cycle accurate simulator. In this paper we propose a simple method that is based on clustering analysis of instruction address hashes to identify representative simulation intervals. Preliminary results show that the instruction address hash is an acceptable feature to capture the microarchitectural behavior of benchmarks. Future experiment plans are also laid out.

1. Introduction

Simulation of standard benchmarks is the most important method for computer architects to study the characteristics of workloads and to evaluate design tradeoffs. Modern benchmarks are no longer small kernels or synthesized toy programs. They are very close to real world programs and often take a long time to execute. Moreover, modern superscalar microprocessors are becoming increasingly complex; thus simulators modeling the processors are also becoming complex and slow. As a result, running a benchmark on a detailed microarchitecture simulation model can take prohibitively large simulation times. Table 1 shows the number of instructions executed by some SPECint2000 benchmarks with reference dataset and the time to do the complete simulation on a 1GHz Pentium III machine with sim-outorder, the detailed out-of-order superscalar simulator from the SimpleScalar 3.0 tool set. Full system simulation of emerging workloads on complete system simulators such as SimOS also often takes prohibitively large amounts of simulation time.

Table 1. Number of instructions and simulation time of selected SPECint2000 benchmarks with reference data set. The benchmark is shown as program name with input data set name.

Benchmark	Number of instructions (million)	Simulation time (hour)
gcc-166	46917	52.2
bzip2-source	108878	104.9
eon-rushmeier	57870	63.8
gzip-graphic	103706	173.1
vortex-1	118976	110.6
vpr-route	84068	98.9
crafty	191882	221.4

Since running simulation to completion is impractical, a popular practice is to fast forward up to several billion instructions to skip the initialization phase of the program, and then simulate several hundred of millions of instructions in detail. However, analysis has shown that this practice often results in large errors with respect to full simulation [1]. Random sampling is easy to implement, but often faces problems due to cache state loss. Therefore, it is important to develop a methodology to select representative sections of traces to accurately evaluate processor design.

An ideal trace selection scheme should meet the following criteria:

1. The error of metrics (e.g. IPC, cache miss rate) of simulation of selected trace with respect to full simulation is small. This requires the scaled down trace preserve the key characteristics of the execution, such as thread-level and fine-grain parallelism, temporal and spatial locality of the memory references, predictability of branches, operating system activity, distribution of time spent in program segments, etc.
2. The selected trace is a small portion of the full trace so that simulation time can be greatly reduced.
3. The trace selection method is microarchitecture independent. If the selected trace is representative of the full simulation on only a specific hardware configuration,

it is not of much use because simulations are usually employed to study various hardware enhancements.

4. The trace selection method is simple to implement and fast to execute. If trace selection takes as much time as full simulation, we would rather choose the latter.

5. The trace selection method is flexible. It can not only be used with single-process user-code-only simulator (e.g SimpleScalar), but also work with multiprocessor full system simulator (e.g. SimOS).

In this project, we investigate an automatic representative trace selection scheme based on cluster analysis of instruction address hash. We compare the new scheme with published research and explore the design space.

2. Related Work

Ever since the standard industry benchmarks became unwieldy for detailed simulation, researchers have been devising methods to reduce traces and simulation time. Yet no one single method has been adopted as the standard practice in computer architecture research community.

One of the traditional ways to reduce workloads for simulators is to reduce the problem size. MinneSPEC [3] is a set of reduced input set for SPECcpu2000. It reduced the simulation time by 20% to 50%. But it still takes too long to simulate. Also, it requires an understanding of each application in order to produce a reliable, reduced input dataset. But there is no systematic method to design the reduced data set, which is the biggest problem for this approach.

Another means of reducing the runtime cost of processor simulation is time sampling. The sampling points within the program are chosen either randomly or uniformly or non-uniformly. The advantage of time sampling is that it provides good coverage. But cache state loss problem is a difficulty encountered when sampled traces are used. Also, it is difficult to decide on the sample size and skip size. Sampling does not take advantage of the phase behavior commonly observed in program execution [4]. In one phase, the program repeatedly execute the same part of the code and follows similar path every time. Therefore, multiple samples in one phase are usually redundant because they give very close results in all aspects.

A better approach than random sampling is representative sampling, which chooses representative samples that accurately reflect an application's runtime behavior. Representative trace selection is usually based on cluster analysis, a technique widely used in data mining to identify patterns in data. SPEClite [5] uses such technique. First the benchmark is run on an Itanium machine. Performance counters are used to gather event counts every 1 million instructions. Since the processor has 4 event counters, the benchmark is run multiple times to gather the 29 performance metrics. Every 1-million instruction sample interval is represented as a 29-dimension vector. The 29 metrics are not totally independent, so principal component analysis is performed to identify the five principal metrics. Thus the dimension of the vector for each sample interval is reduced to 5. Finally, cluster analysis is conducted on the new vectors. One cluster identified is a group of sample intervals that exhibit similar behavior. The interval that is closest to the cluster centroid is chosen to be the representative sample for all the samples in that cluster. The number of intervals in that cluster is the weight for this sample. The results show that 0.1% representative sampling can be as accurate as 3% uniform sampling. That is 30X reduction in simulation time over uniform sampling. The main disadvantage of SPEClite is its dependence on microarchitecture. The metrics are collected on a specific processor. There is no guarantee that the selected sample intervals are sill representative on a different processor. In addition, it is hard to synchronize the sample intervals between measurement on real hardware and simulator. This will become unachievable when server workloads are studied. Every execution of the same multi-threaded program on a machine will be different.

SimPoint [1] is another representative sampling method. It runs ATOM instrumented binary, which is much faster than cycle-accurate simulator (sim-outorder), to gather Basic Block Vectors (BBV) for each 100 million-instruction interval. A basic block is a section of code that is executed from start to finish with one entry and one exit. They use the frequencies with which basic blocks are executed as the metric to compare different sections of the application's execution to one another. A Basic Block Vector is a single dimensional array, where there is a single element in the array for each static basic block in the program. The dimension of BBV (static basic block count) ranges from 2,756 to 102,038. Clustering method become ineffective at such high dimensionality. So random projection is used to reduce the dimension to 15. And then cluster analysis is performed to come up with the representative samples and weight for each sample. SimPoint has the main advantage of microarchitecture independence. It guarantees that the highly executed part of code (hotspots) will be sampled. However, SimPoint suffers from two disadvantages. First, BBV is difficult to get. Second, the basic block concept does not scale to full system simulation of servers. When interrupt happens, it may break a basic block. Also, it is hard to separate the instruction streams of different processes running the same code.

R-metric [2] is a metric proposed by Iyengar et al to measure the representativeness of the reduced trace. A basic block that is qualified by the branching history of length k and by the preceding (n-1) qualified basic blocks is called a fully-qualified basic block ω with parameters n and k. R-metric R(n, k) is defined as:

$$R(n,k) = \frac{1}{|\tau|} \times \sum_{\omega} (scaled_{T,\tau}(\omega) - count_{\tau}(\omega)) \times size(\omega)$$

$\forall$ fully qualified basic blocks ω with parameters n, k such that

$$(scaled_{T,\tau}(\omega) - count_{T}(\omega)) \geq 0$$

where

$count_{\tau}(\omega)$ is the number of occurrences of qualified basic block w in the reduced trace τ

$scaled_{T,\tau}(\omega)$ is the expected scaled count for in a representative trace of size |τ|

$size(\omega)$ is the number of instructions in ω

R-metric is more rigorous than BBV because it incorporates not only the execution frequency of each basic block, but also the path of the execution (i.e. how one basic block is reached). Iyengar et al [2] also gives an algorithm to calculate the reduced trace while retaining the representativeness (minimizing the R-metric). Because path information is taken into consideration, the number of fully qualified basic blocks is at least one order larger than the number of basic blocks. This makes the algorithm's time complexity very high. As a result R-metric is used at IBM to evaluate the representativeness of the reduced trace, but how to come up with such a reduced trace still relies on trial-and-error and the expertise of the performance engineer.

3. Algorithm

We use instruction address hash to represent one trace interval. And then cluster analysis is conducted to select the representative trace intervals. Hash vector HV is a one-dimension array of N elements initialized to 0. HashFun() is a hashing function. It takes an address as input and generates an integer in the range of [0, N-1]. When an instruction at address *addr* is executed, the operation done on HV is HV[HashFun(addr)]++. The result is a hash vector for each trace interval. Then cluster analysis is performed on all the hash vectors to select the representative intervals.

The underlying idea of instruction address hash is similar to BBV. Both assume that the execution frequency of every static instruction has a major impact on the execution behavior of the code, and that the microarchitecture metrics such as IPC, cache miss rate are decided by execution behavior. Yet, instruction address hash shows two major advantages over BBV. First, it is much easier to obtain than BBV. It is very complicated and expensive to compute basic block-based parameters. Functional simulators such as sim-safe in SimpleScalar, or ARIA, an IBM internal tool, can run much faster than cycle accurate simulators. With one pass of functional simulation, we can get all the hash vectors. Second, the concept of instruction address hash scales to full system simulation. As long as there are instructions executing, we can easily get the hash vectors, whereas the meaning of BBV is not clear if multiple processes are running with system interrupts coming in to break a basic block from time to time.

4. Preliminary Results and Future Plans

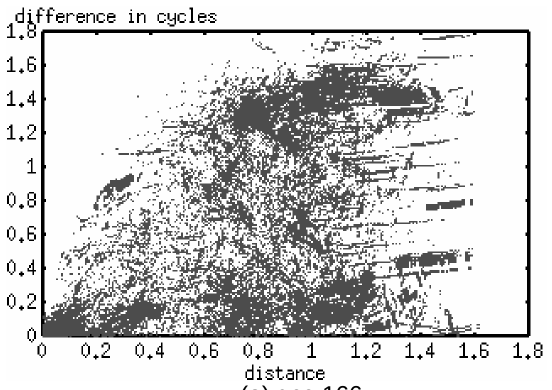
Establishing validity of the proposed scheme requires extensive experimentation. Preliminary results are presented here. We simulated selected SPECint2000 benchmarks with SimpleScalar tool set. Statistics are collected for every 100 million-instruction interval as in SimPoint. With the sim-outorder simulator, microarchitecture metrics such as IPC, cache miss rate, branch misprediction rate are collected for each interval. The hash vector of each interval is collected with sim-safe, the functional simulator.

4.1 Relation between instruction address hash and performance metrics

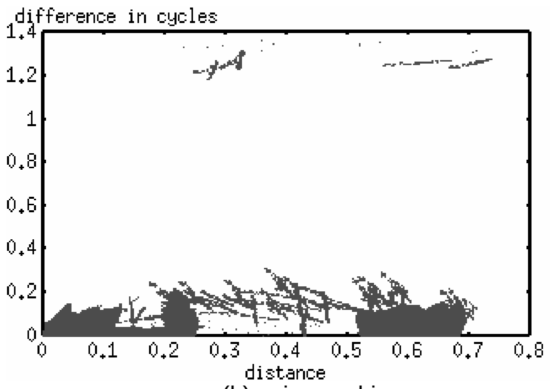
First we wish to decide if instruction address hash is a feature representative of the behavior of the execution interval. A good representative feature must have the following property: If two intervals show very similar features, the performance metrics should also be very close. In this experiment, the dissimilarity of hash vectors is measured by their Manhattan distance. The vector is first normalized so that the sum of all the N elements equals to 1. The Manhattan distance of two vectors a and b is computed as

$$ManhattanDist(a,b) = \sum_{i=0}^{N-1} | a_i - b_i |$$

The distance has a range of [0,2]. If two vectors are the same, they bear a distance of 0. The more dissimilar the two vectors are, the larger the distance is. Figure 1 shows the relationship between the Manhattan distance and relative cycle difference between intervals. The x-coordinate of a data point is the distance between the hash vectors of two intervals and the y-coordinate is the relative difference in cycles between the two intervals. The data points close to the origin show that if the distance of hash vectors of two intervals is small, the difference in cycles is also small. Figure 2 shows the relationship between average distance and average relative difference in cycles. Each data point in Figure 2 is the result of the moving average of the data points in Figure 1. It clearly shows the distinct characteristic of the benchmarks. For gcc-166, there is a large range of distance, indicating varying code execution frequency. It also shows a large variation of CPI in different parts of the code. Though gzip-graphic, also executes different part of the code with different frequency, the CPI does not change much at different portions of the code. On the other hand, vpr-route does not exhibit large variation in code frequency, but its CPI range is relative large indicating drastically different microarchitectural behavior at different parts of the code. Finally, there is crafty, which does not change much in either code frequency or CPI. However different the benchmarks may be, each line starts from the origin, and the difference in cycles increases with the distance of hash vectors. Therefore, instruction address hash appears to be an acceptable feature to describe and distinguish trace segments.



(a) gcc-166



(b) gzip-graphic

Figure 1. Correlation between distance of hash vectors and difference in cycles (Manhattan distance, 16 dimensions)

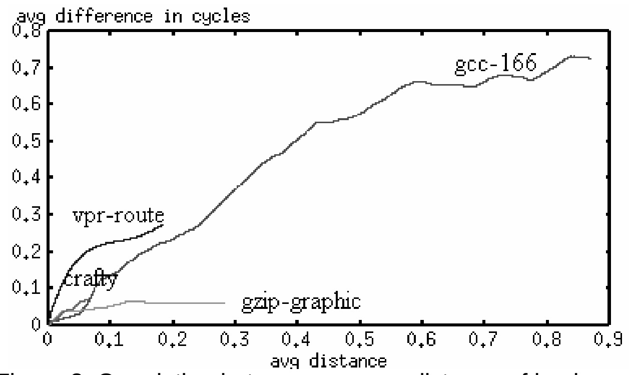


Figure 2. Correlation between average distance of hash vectors and average difference in cycles (Manhattan distance, 16 dimensions)

4.2 Number of simulation intervals

The number of intervals in the reduced trace corresponds to the number of clusters because we select exactly one interval from every cluster to represent all intervals in the cluster. How to choose the number of clusters (k) is a difficult question and there is no agreed-upon method in the data mining research community. SimPoint chooses the k to maximize Bayesian Information Criterion (BIC) score. However, all such methods in data mining try to find the natural clusters in the data, so they avoid splitting a cluster if there is no clear cluster structure within it. Our goal is different: we try to minimize the distance between intervals inside a cluster. Therefore, when the average distance is large, we have to split the cluster to ensure small error in performance metrics even if the data points (intervals) in the cluster do not show any sub cluster structure. Figure 2 can give us some guidance on choosing the number of simulation intervals. For gcc, a large number of intervals have to be simulated to reduce the final error, because the CPI is very different from interval to interval. On the other hand, a couple of intervals may be enough for crafty because it does not show much variation. Figure 2 is drawn after full detailed simulation of the benchmarks. Therefore it is impractical to use it as a guidance in real work. We are looking into the vast body of sampling theory for solving this problem.

4.3 R-metric

R-metric is a very rigorous metric to evaluate the representativeness of the reduced trace. It is used inside IBM to test the quality of the reduced trace. We plan to investigate how the instruction address hash relates to R-metric. We also want to explore the method to combine R-metric with cluster analysis to automatically select simulation intervals that show small R-metrics with respect to the full simulation.

4.4 Other Clustering Algorithms

There are many clustering algorithms as a result of active research in the data mining community. Both SPEClite and SimPoint use k-means method. K-means algorithm is easy to implement and exhibits low time complexity. However, it is known to be susceptible to outliers. In addition, it does not directly give the representative interval of each cluster. It uses the centroid of a cluster to represent the cluster. A centroid does not correspond to any real data point, so after cluster analysis the interval closest to the centroid is chosen to represent the cluster. On the other hand, another clustering algorithm, k-medoids, directly selects one data point (medoid) to represent a cluster. K-medoid methods are very robust to the existence of outliers. In addition, clusters found by k-medoid methods do not depend on the order in which the objects are examined. We plan to try CLARANS method [5], which is a recent k-medoid method with relatively low time complexity. We want to investigate the impact of the different clustering techniques on the final simulation error.

4.5 Dimensionality

BBV has up to hundreds of thousands of dimensions. At such high dimension, clustering algorithms suffer from the so-called “curses of dimensionality” if simple Euclidean distance is used. SimPoints uses random projection to reduce the dimension to 15. It projects the original high dimension data to a randomly selected 15-dimension space. The projection matrix is filled by random value between –1 and +1. Hashing is similar to the random projection used in SimPoint. Both methods try to reduce the dimensionality of the data. But hashing is much simpler. It is equivalent to randomly put +1 in the projection matrix and it does the matrix multiplication on the fly. Obviously, when the dimension is over-reduced, two data points (intervals) that is separable in the original high dimension space may become too close. Therefore, the quality of clustering will suffer. Since researchers in data mining have proposed using cosine distance to do clustering directly at high dimensionality, we plan to investigate the impact of dimension reduction and search for the optimal dimensionality.

4.6 Distance Measurement

For any clustering technique, the choice of distance to measure the dissimilarity of two intervals is very important. In the above example, we used Manhattan distance after the sum of elements of every hash vector is normalized to 1. SPEClite and SimPoint use Euclidean distance after the dimension is reduced to 5 or 15. Cosine distance is a newly proposed distance metric, which has been successfully applied to cluster analysis of web text [6]. We plan to investigate the difference of Manhattan distance and cosine distance.

Acknowledgements: This research is supported by the IBM Center for Advanced Studies. It is also partly supported by the National Science Foundation under grant 0113105.

References

[1] T. Sherwood, E. Perelman, G. Hamerly, B Calder, “Automatically Characterizing Large Scale Program Behavior”, Proceedings of ASPLOS X, 2002.
[2] V. S. Iyengar, L. H. Trevillyan, P. Bose, "Representative Traces for Processor Models with Infinite Cache", Proceedings of HPCA-2, 1996.
[3] A. KleinOsowski, D. J. Lilja, “MinneSPEC: A New SPEC Benchmark Workload for Siumulation-Based Computer Architecture Research”, Computer Architecture Letters, vol 1, number 2, 2002.
[4] J. Cook, R. L. Oliver, E. E. Johnson, “Examining Performance Differences in Workload Execution Phases”, Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization, 2001
[5] R. T. Ng, J. Han, “Efficient and Effective Clustering Methods for Spatial Data Mining”, Proceedings of the 20th VLDB Conference, 1994
[6] A.Strehl, J. Ghosh and R. Mooney, "Impact of Similarity Measures on Web-page Clustering", in Proc. AAAI workshop on AI for Web Search, K. Bollacker (Ed), TR WS-00-01, AAAI Press, July 2000, pp. 58-64.