Cache Performance Analysis Under Multi-Parameters Using SimpleScalar Toolset

Department of Computer Science

Abhijit Kumar
Amity School of Engineering and Technology, India
abhijitkaran@hotmail.com

Himanshu Sati
Amity School of Engineering and Technology, India
himanshu.aset@gmail.com

Arunit Gupta
Amity School of Engineering and Technology, India
arunitgupta@gmail.com

Sandeep Kumar
Amity School of Engineering and Technology, India
sandy.aset@gmail.com

Saurabh Tandan
Amity School of Engineering and Technology, India
mailsaurabh14@gmail.com

Abstract— Cache is of significant importance in total system performance of computer architecture. To provide references for the computer architecture designer, we put forward the relationship between the cache performance and cache parameters. Firstly, we described the main software simulation method, and analyzed various factors of effects on cache performance, and then carried out the simulation experiment of cache performance based on SimpleScalar tools set and SPEC2000 benchmark suite. We compared the effects on cache performance when cache size, cache block size and association are varied separately, and then we analyzed the experimental results in detail. The result is basically in keeping with the cache principle of architecture. At last we gave the reasonable cache parameters. The software platforms used are SimpleScalar 3.0, Ubuntu 12.04 and SPEC2000. We used SimpleScalar and SPEC2000 as our main simulation tools and data sets and conclusions.

Keywords- Miss Rate, Cache Size, Block Size, Association

I. *INTRODUCTION*

A CPU cache is a cache used by the central processing unit of a computer to reduce the average time to access memory. The cache is a smaller, faster memory which stores copies of the data from the most frequently used main memory locations. As long as most memory accesses are cached memory locations, the average latency of memory accesses will be closer to the cache latency than to the latency of main memory.

Cache is an important effect factor in reducing average memory access time. Having cache in a system reduces the increasing performance gap between main memory and processor. We thought that better understanding of the cache would help us in developing efficient programs. Our goal is to analyse cache performance by taking various parameters into account. The proportion of accesses that result in a cache hit is known as the hit rate, and can be a measure of the effectiveness of the cache for a given program or algorithm. By analysing cache performance, we mean to analyse miss rate. Miss rate can be reduced by increasing temporal and spatial localities of cache blocks.

Software simulation can be divided into function simulation and performance simulation. Function simulation target simulates hardware function by software methods, and performance simulation target provides an effective research platform that can be implemented for easy evaluation performance of all kinds of design proposal before hardware implementation. Then the design bottleneck can be found and improved upon.

A. *Objective*

- To analyze cache performance by taking various parameters into account.
- By analyzing cache performance, we mean to analyze Miss Rate which can be reduced increasing temporal and spatial localities of cache blocks.
- To plot the statistics of cache performance in EXCEL.

B. *Problem Description*

There are many factors that can affect cache performance, such as cache size, cache block (line) size, application program, replacement algorithm, association, etc. In our project, we analyzed three main factors related with simulation as follows:

1) *Cache size*: In general, larger the cache size, lower the miss rate and better will be the performance. But increasing the cache size to a certain degree introduces overhead.
2) *Cache block size*: Larger cache block size exploits spatial locality and reduces the miss rate. This is valid only up to a certain extent because as the cache block size exceeds a certain point, number of lines in a set reduces (spatial locality decreases), thereby reducing the hit rate.
3) *Associativity*: Increasing associativity decreases the conflict miss rate. But increasing the associativity after a certain point comes at a significant cost. Higher associativity is expensive to implement and hard to make fast. Higher associativity also increases the hit time and miss penalty.
4) *Cache Miss*: A cache miss refers to a failed attempt to read or write a piece of data in the cache, which results in a main memory access with much longer latency. There are three kinds of cache misses: instruction read miss, data read miss, and data write miss. In order to lower cache miss rate, a great deal of analysis has been done on cache behavior in an attempt to find the best combination of size, associativity, block size, and so on.

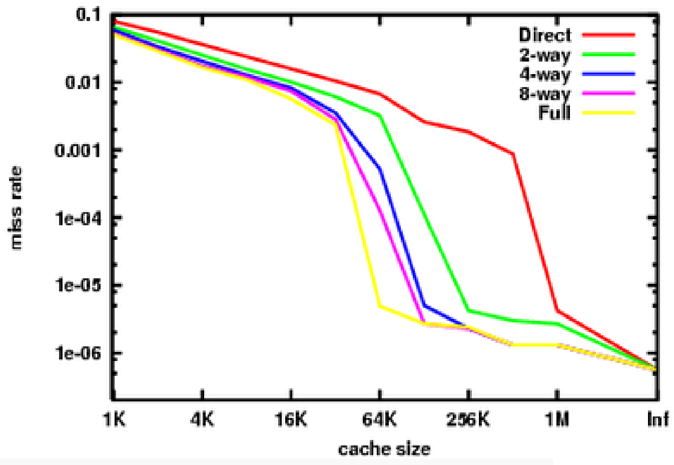

Figure 1: Miss rate versus cache size on the Integer portion of SPEC CPU2000

The graph to the right summarizes the cache performance seen on the Integer portion of the SPEC CPU2000 benchmarks. These benchmarks are intended to represent the kind of workload that an engineering workstation computer might see on any given day. The reader should keep in mind that finding benchmarks which are even usefully representative of many programs has been very difficult, and there will always be important programs with very different behavior than what is shown here.

5) Locality of reference: Locality of reference, also known as the locality principle, reveals that through analysis of data locations, referenced in a short period of time, clusters can often be relatively predictable. Important special cases are temporal,
spatial, equidistant, and branch locality.

Temporal locality: if at one point in time a particular memory location is referenced, then it is likely that the same location will be referenced again in the near future. There is a temporal proximity between the adjacent references to the same memory location. In this case it is common to make efforts to store a copy of the referenced data in special memory storage, which can be accessed faster. Temporal locality is a very special case of the spatial locality, namely when the prospective location is identical to the present location.

Spatial locality: if a particular memory location is referenced at a particular time, then it is likely that nearby memory locations will be referenced in the near future. In this case it is common to attempt to guess the size and shape of the area around the current reference for which it is worthwhile to prepare faster access.

II. PROPOSED WORK

We know how important it is to analyse cache performance under various parameters. But due to complexity of hardware implementation, analysing performance using software simulation is a good choice. We varied Data L1 cache parameters to study the performance of both Data L1 and Unified L2 cache on alpha Instruction Set Architecture (ISA). L1 cache parameters were cache size, cache block size and associativity. Number of sets in a cache is given by

$$\text{Number of sets}(S) = \frac{\text{Cache size}(C)}{(\text{Block size}(B) * \text{ Number of Lines }(E))}$$

Where Block size ‘B’ is in bytes.

In our project, cache performance was interpreted by considering the miss rate. In order to analyse the cache performance, we used SimpleScalar as the simulation tool. SimpleScalar is an extensive and architectural research toolset. With this tool set, the user can simulate real programs using fast execution-driven simulation. We needed some benchmark (or an application) to analyse the cache performance. Of many benchmarks available, we used SPEC 2000 benchmarks for alpha ISA. We analysed cache performance for four sets of integer benchmarks. The integer benchmarks were gcc, gzip, perlbmk and vpr.

The input cache parameters to the simulator were: type of data cache (L1 or L2), number of sets, block size, associativity and replacement algorithm.

Replacement algorithms that can be used are Least Recently Used (LRU) (by typing ‘l’ in place of replacement policy), FIFO (‘f’) and Random replacement (‘r’).

We will plot the statistics of cache performance in EXCEL and analyse the result.

A . Sample Program Analysis

The sample program shown below calculates the sum of the elements of a two dimensional array by accessing the elements in row-wise fashion.

```
void main(void)
{
        Int
a[4][8]={{1,2,3,4,5,6,7,8},{6,7,8,9,5,6,7,8
},{10,9,8,7,5,6,7,8}, {5,4,3,2,5,6,7,8}};
        int i, j, sum = 0;
        for (i = 0; i < 4; i++)
                for (j = 0; j < 8; j++)
                        sum += a[i][j];
 }
```

The elements can be accessed column-wise by interchanging the loop variables i and j. The elements of the array can be accessed with stride=2 and stride=3 by incrementing both i and j by 2 for stride=2 and 3 for stride=3.

As shown in the graph below, maximum performance is obtained when the elements of the array are accessed row-wise since the spatial locality is higher. When we access the elements column-wise, performance decreases since spatial locality decreases. When we try to access elements row-wise in an order different from what is stored in the memory, spatial locality decreases thereby increasing the miss rate.

The graph below shows the miss rate of L1 cache versus the type of access of the array elements.

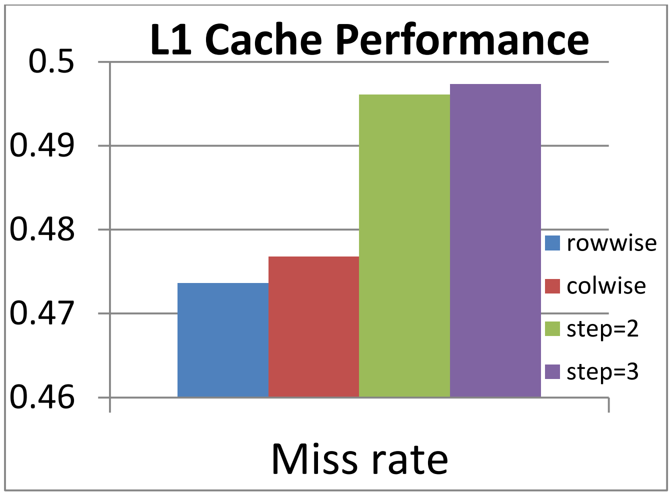

Figure 2: L1 miss rate varied vs access of array.

III. *Tools Used:*

A. SimpleScalar Tool Set

SimpleScalar is developed by Todd Austin, it is a performance simulation tools set based on processor architecture. And it is a well-known architecture simulator. While user programs, target application programs, operating system could run on processor performance simulator developed by SimpleScalar tools set. At present, the architecture of object instruction sets that SimpleScalar support is PISA, Alpha, Power PC, X86. We include five execution- driven processor simulators in the release. They range from an extremely fast functional simulator to a detailed, out-of-order issue, superscalar processor simulator that supports nonblocking caches and speculative execution.

The five distribution files in the directory (which are symbolic links to the files containing the latest version of the tools) are:
• simplesim.tar.gz
• simpleutils.tar.gz
• simpletools.tar.gz
• simplebench.big.tar.gz
• simplebench.little.tar.gz

We have four main simulators in SimpleScalar toolset namely Functional simulator, Profiling simulator, Cache simulator and Out-of-order processor timing simulator.

1) *sim-fast* **:**Functional simulator called sim-fast performs only functional simulation in the same order the instructions are executed without instruction checking or existence of cache memory.
2) *sim-safe:* A different version of *sim-fast* called sim-safe which performs functional simulation checking for correct alignment of each memory reference.
3) *sim-profile :*Simulator sim-profile provides detailed profiles on instruction classes and addresses, text symbols, memory accesses, branches, and data segment symbols.
4) *sim-cache and sim-cheetah:* The simulators ideal for fast simulation of architectures that include cache memory are sim-cache and sim-cheetah**.**

5) *sim-outorder:* The most complicated and detailed simulator in the tool set is sim-outorder. This simulator runs for days if large input datasets are given in the SPEC 2000 benchmark. Hence using an appropriate method, smaller datasets are generated. This method is incorporated in the several SPEC 2000 benchmarks when using detailed execution-driven simulators like sim-outorder so that we get reasonably comparable simulation times.

B) SPEC 2000 Binaries:

SPEC is an acronym for the Standard Performance Evaluation Corporation. SPEC is a non-profit organization composed of computer vendors, systems integrators, universities, research organizations, publishers and consultants whose goal is to establish, maintain and endorse a standardized set of relevant benchmarks for computer systems. It's all test programs come from real application. SPEC could compare performance of high-strength calculation among all kinds of hard architecture. Although no one set of tests can fully characterize overall system performance, SPEC believes that the user community will benefit from an objective series of tests which can serve as a common reference point.

The SPEC 2000 benchmarks are divided into floating-point and integer categories. There are twelve integer benchmark programs (CINT 2000), written in C, with the exception of 252.eon, which is written in C++. There are fourteen floating-point benchmark programs (CFP 2000), six are written in Fortran 77 (F77), four are written in Fortran 90 (F90), and the remaining four are written in C.

A few of the programs from SPEC 95 have been carried over into the new SPEC 2000 suite, specifically, gcc, perl, and vortex from the integer programs, and swim, mgrid, and applu from the floating-point programs. However, these programs have been modified for the SPEC 2000 suite. Consequently, the results produced by a program with the same name are not comparable across the two benchmark sets. More detailed information about the

SPEC 2000 benchmark suite can be obtained.

IV. *RESULT ANALYSIS*

A. Cache Size :

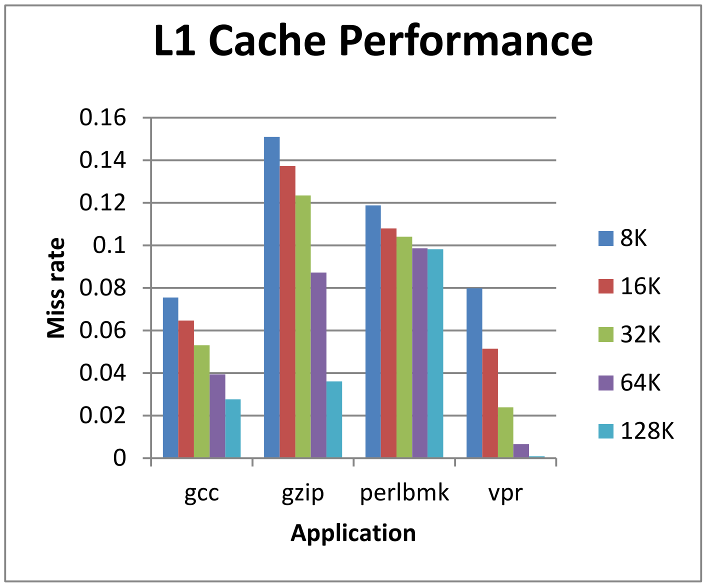

Figure 3: L1 Cache performance by varying L1 cache size

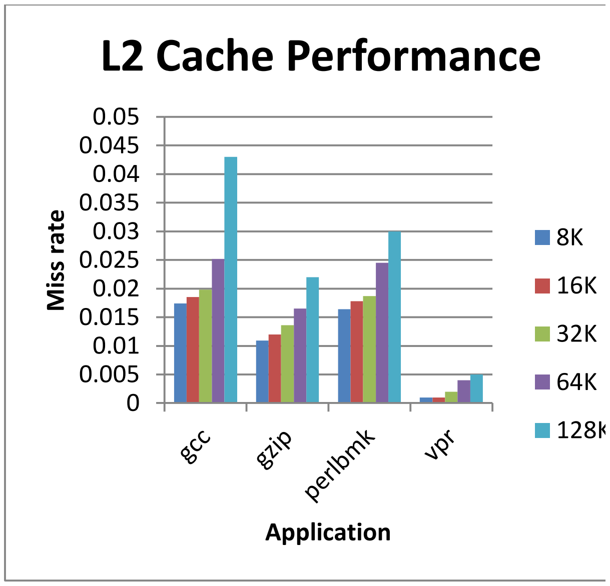

Figure 4: L2 Cache Performance by varying L1 cache size

1) *Parameters Varied:* This experiment compared the cache performance when L1 size is changed. The L2 size fixed for 256K, meanwhile L1 size is 8K, 16K, 32K, 64K and 128K. respectively. The test sets of experiment are four programs of SPEC: perl, vpr, gzip and gcc.

2) *Simulation and analysis:* As per results generated by simulator it can be seen that L1 miss-rate of all programs are high when L1 size is 8K. With the L1 size increases, the L1 miss-rate is decreased. It’s reasonable because of L1 could load more data; the hit rate decreases naturally, therefore the L1 miss-rate decreases. There is phenomenon that the cache performance promotes significantly when L1 size is increased at the beginning of a period of time. But the speed of the performance promotes will slow down when L1 increases to a certain size. Moreover, cache performance is different when use different program. For example, compress application needs a great deal of data for computation, it access cache and memory more frequently, therefore, the access number and miss-rate is high. And gcc is the C compiler under Linux, it access cache and memory much lesser, so the access number and miss-rate is low.

B. Block Size:

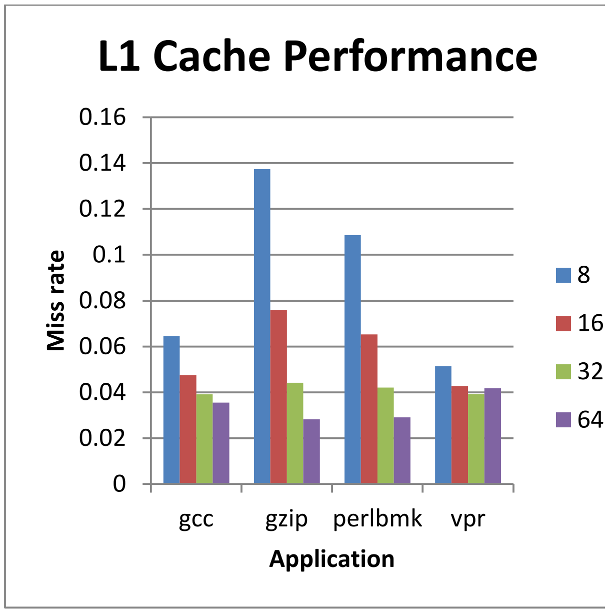

Figure 5: L1 Cache Performance when varying L1 block size

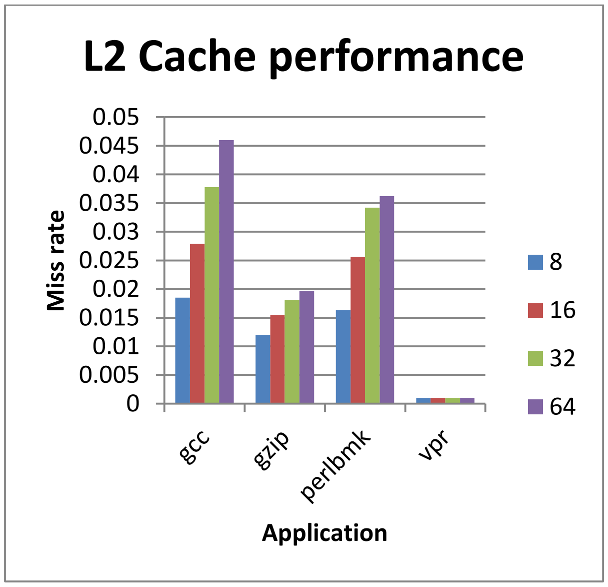

Figure 6: L2 Cache Performance when varying L1 block size

1) *Parameters Varied:* We simulated the cache performance while block size is changed and cache size unchanged. We take L1 as an example with Parameters as follows: L1 size fixed for 16K, L1 block size is 8B, 16B, 32B and 64B respectively, and L2 size fixed for 256K, block size is 256B.

2) *Simulation and analysis:* As per results generated by simulator it can be seen that the L1 miss-rate is decreased significantly. L1 can load more data when L1 block size increased. Based on principle of locality, as the program runs it always concentrate in certain area, therefore, the possibility of finding data is increased, thus the miss-rate is decrease. In the case of L2, with L1 size increased, the L2 access number will decrease gradually, thus increasing its miss rate.
The reason is the possibility of access to L2 is decreased when L1 block size is increased. At the same time, if we can't find the data in L1 the possibility of finding data in L2 is decreased, which led to the miss-rate increase. From the above simulation, we can see cache size has great effect on performance. So choosing a suitable block size will enhance performance. In currency hardware platform, the suitable L1 block size is from 32B to 64B, which will result in better performance.

C. *Associativity*

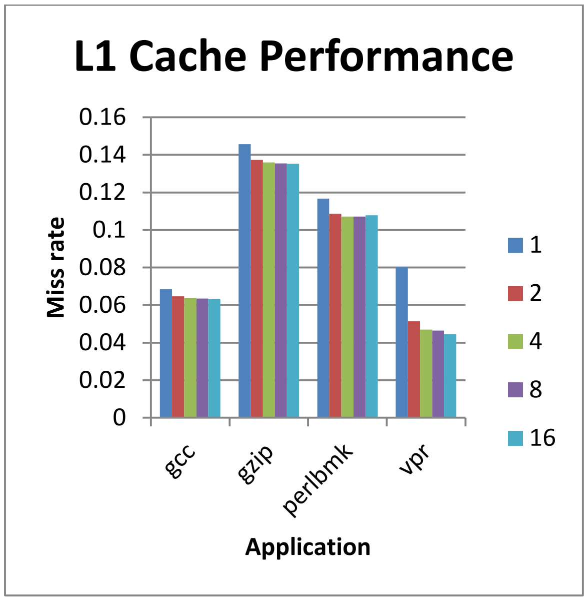

Figure 7: L1 Cache Performance when varying L1 line size

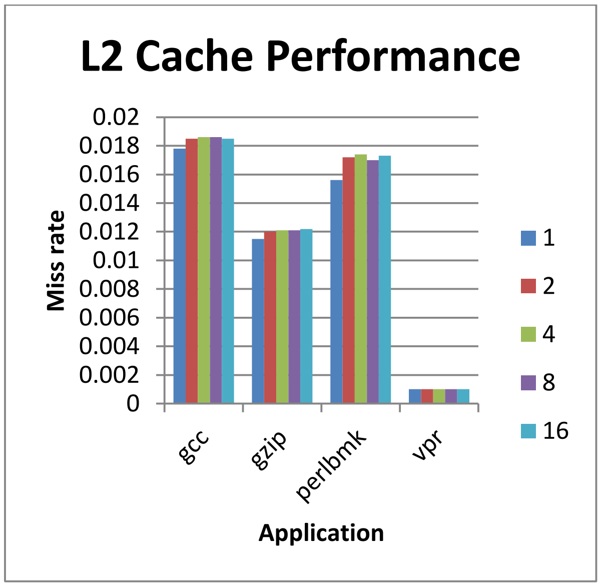

Figure 8: L2 Cache Performance when varying L1 line size

1) *Parameters Varied:* We simulate the cache performance when association is changed. Again, we take L1 as our example. The parameters are as follow: L1 size fixed for 16K and L2 size fixed for 256K. The association is 1, 2, 4, 8, and 16 respectively.

2) *Simulation and analysis:* As per results generated by simulator it can be seen that L1 miss-rate is decreased significantly in some range when association increased. Each memory block has more cache block that can mirror as the association increased. Thus, two memory block can be loaded to two cache block simultaneously when it mirror to one cache block (shake phenomenon). It can use the cache space more efficiently. Combine with locality principle, the possibility of finding data is increase and miss-rate decrease. But as association increased to a certain degree, then shake phenomenon disappeared. All needed data can be loaded to cache conveniently, so the miss-rate don't change. The possibility of finding data in L2 is decreased due to L1 miss-rate decreased. But when beyond a certain range, the L1 miss-rate becomes constant, and corresponding the L2 miss-rate is also constant. We there conclude that cache association has some effect on performance, therefore it is important to set suitable association. In currency hardware platform, association is from 2 to 4, which will attain a better performance.

V. *CONCLUSION AND FUTURE SCOPE*

Simulation and analysis the main factors of effect cache performance in detail. The simulation result indicated that cache performance is in direct proportion with the cache size, cache block size and association. The experimental data are basically keeping with the cache theory of architecture except for some small different in parameters. There are many reasons for this, such as benchmark programs aren't enough, the parameters aren't optimized, and the limitation of simulation platform.

The future works are to analyze more factors of effect on cache performance, construct more full-scale simulation platform and research improved methods. More in-depth analysis in the relation between cache parameters and cache performance. Therefore, we could provide more valuable references for architecture designer

REFERENCES

[1] Haifeng Ma, Xin Yue and Jingfeng Song, "Cache Performance Simulation and Analysis under Multi-parameters", 2010 International Conference on Machine Vision and Human-machine Interface (MVHI), pp. 37-40, 2010.

[2] Randal E. Bryant, David R. O'Hallaron, "Computer Systems A Programmer's Perspective", pp. 298-321, November 2001.

[3] Waseem Ahmad, Enrico Ng, "A Quantitative/Qualitative Study for Optimal Parameter Selection of a Superscalar Processor using SimpleScalar", Computer Sciences Technical Report, 2004.

[4] AJ KleinOsowski, John Flynn, Nancy Meares, and David J. Lilja, "Adapting the SPEC 2000 Benchmark Suite for Simulation-Based Computer Architecture Research", In Proceedings of the International Conference on Computer Design, 2000.

[5] H. Al-Zoubi, A. Milenkovic, M.Milenkovic. "Performance Evaluation of cache Replacement Policies for the SPEC CPU2000 Benchmark Suite". In Proc. of the 42nd ACM Southeast Conf,April 2004.