Performance of One's Complement Caches*

Qing Yang, Sridhar Adina and T. Sun
Dept. of Elec. & Computer Engineering
University of Rhode Island
Kingston, RI 02881
e-mail: qyang@ele.uri.edu

ABSTRACT

On-chip caches are common place in today's commercial microprocessors to reduce average memory access latency. These on-chip caches generally have low associativity and small cache sizes. Cache line conflicts are the main source of cache misses which are essential to overall system performance. This paper introduces an innovative design for on-chip data caches of microprocessors, called one's complement cache. While binary complement numbers have been successfully used in designing arithmetic units, to the best of our knowledge, no one has ever considered using such complement numbers in cache memory designs. This paper will show that such complement numbers help greatly in reducing cache misses in a data cache thereby improving data cache performance. By means of parallel computation of cache addresses and memory addresses, the new design does not increase critical hit time of cache accesses. Cache misses caused by line interferences are reduced by means of evenly distributing data items referenced by program loops across all sets in a cache. Even distribution of data in the cache is achieved by making the number of sets in the cache a prime or an odd number thereby the chance of related data being mapped to a same set is small. Trace-driven simulations are used to evaluate the performance of the new design. Performance results on benchmarks show that the new design improves cache performance significanly with negligible additional hardware cost.

Index Terms: *Cache Memory, Memory conflicts, Memory Hierarchy, Performance Evaluation.*

1 Introduction

With the rapid advances in device technology and increased speed gap between processor and memory, effectiveness of memory system design becomes essential to the performance of a computer system. Many existing commercial computers utilize memory hierarchies consisting of cache memories to minimize memory latency. It has become a common practice to implement microprocessors having several kilobytes or tens of kilobytes on-chip instruction and/or data cache memories. Since the speed disparity between an on-chip cache and an off-chip memory is large, the performance of such computers are very sensitive to the miss ratio of the on-chip cache.

Cache misses can be generally classified into three categories [1]: compulsory, capacity, and conflicts. The compulsory misses are the misses in the initial loading of data. The capacity misses are due to the size limitation of a cache to hold data in a working set of a running process. The last category, conflict misses result from cache line interferences. A conventional cache usually consists of 2^s sets for some integer s. Each set contains one or several (d) cache lines or blocks. We call the number of cache lines in a set, d, the degree of associativity. If $d = 1$, the cache is called direct-mapped cache. A block (line) of memory data can be placed in the cache only in one (for direct-mapped cache) or d (for d-way set associative cache) cache locations. If more than d blocks of data referenced by a program are mapped into a same set, a cache line interference occurs. As a result, some useful data may be replaced giving rise to a high miss ratio. Most microcomputers that are available today have a relatively small degree of associativity or direct-mapped cache because of its advantages of easy implementation and fast cache access[2]. Therefore, if data are unevenly mapped among the available cache sets, cache line conflicts can pose a severe performance limitation on the cache memory. A recent study on blocked matrix multiply algorithm [3] shows that the cache line interference

*This research is supported in part by National Science Foundation under Grants No. MIP-9208041 and MIP-9505601

misses in a direct-mapped cache increase drastically and dominate cache misses after the fraction of a 16K-word cache being used exceeds 3%.

In this paper, we propose a novel cache design that distributes data evenly across all sets in the cache. The new design, referred to as *one's complement cache*, is particularly suitable for microprocessors as an on-chip data cache. The primary idea of the design is that memory data are mapped into cache lines according to a one's complement number. The cache is indexed using the one's complement form of the data address rather than the original binary form. Complement forms of binary numbers have been successfully used in computer designs (e.g. arithmetic units) for quite a long time. Because of the nice properties of the complement numbers, they helped greatly in designing simple and efficient arithmetic units. However, to the best of our knowledge, no one has ever attempted to use a complement number in designing memory systems. For the first time, we utilize a one's complement number to design cache memory for the purpose of reducing cache misses. It turns out that the one's complement cache has a great performance potential. We will show that the one's complement cache minimizes data cache misses caused by line interferences. Such minimization is achieved by evenly distributing cached data across all sets in the cache. We evaluate the performance of our new design by means of trace driven simulation. A set of application programs from SPEC92 benchmarks is simulated on the one's complement cache and the conventional set-associative cache. Simulation results show that the one's complement cache reduces miss ratios of the conventional set-associative cache significantly for the benchmark programs considered.

The paper is organized as follows. The next section presents the details of the new design. Section 3 describes the simulation methodology and benchmark characteristics. We evaluate and compare the performance of one's complement cache and the conventional cache in Section 4. Section 5 discusses related work in the field and Section 6 concludes the paper.

2 One's Complement Cache

It is a known fact that two cache lines conflict with each other in the cache if they fall into the same congruence class modulo the number of sets in the cache. The main idea of our design is simple. We make the number of sets of a cache a prime number or an odd number. As a result, for any given set of memory accesses we try to maximize the number of different congruence classes and minimize the number of data items that fall into the same congruence class for a given cache size.

2.1 Implementation Details

The new cache organization generally consists of $2^s - 1$ sets for some integer s. Each set in a cache has a unique identification number in one's complement form [1] ranging from 0 through $2^s - 2$ which are called set number, hence the name *"one's complement cache"*. There are two different 0's: positive 0 (s 0's in binary form) and negative 0 (s 1's in binary form, i.e. $2^s - 1 = 0$). Both of these two 0's represent one logical data and one physical cache location. Each set number in the one's complement form represents a unique remainder resulting from a memory address divided by $2^s - 1$, i.e. memory address *modulo* $2^s - 1$. Therefore, given a line address A of a memory data and a cache that has $2^s - 1$ sets, the cache mapping function is defined as

$$A \ modulo \ (2^s - 1); \quad for \ some \ integer \ s, \tag{1}$$

instead of A *modulo* 2^s. The memory block with starting address A is then placed in set number A *modulo* $(2^s - 1)$.

Each memory address, as in conventional cache-based computer system[2], is partitioned into three fields: $l = log_2(L)$ bits of byte address in a line (offset); s bits of index; and the remaining *tag* bits of tag. The access logic of the one's complement cache consists of three components: data memory, tag memory, and matching logic. As in a set-associative cache, the data memory contains a set of address decoders and cached data; the tag memory stores tags corresponding to the cached lines; and the matching logic checks if the tag in an issued address matches the tag in the cache. The cache lookup process is exactly the same as in the set-associative cache and hence takes the same amount of time as in the set-associative cache. A data item in the cache is identified by using the index field of a memory address (in one's complement) that activates a set where the requested data is potentially located.

[1] Actually, an s-bit one's complement number has the range from $-2^{(s-1)} - 1$ to $2^{s-1} - 1$. But the sign bit has no significance in this context. We consider the sign bit as one more binary bit. e.g. instead of calling 1101 as -2, we call it 13 which is the remainder of the number modulo 15. The main reason why we name it one's complement is that all arithmetic operations performed on the index fields are done in one's complement arithmetic as explained shortly.

However, the index field used to access the data memory is not just a subfield of the original address word issued by the processor since the modulus for cache mapping is not a power of 2 any more. It is the residue of the line address modulo $2^s - 1$. How efficient this modular operation can be done is essential to the cache performance. In the following, we show how the one's complement index can be derived conveniently.

For generating a cache address, the tag field and the word (offset) field are the same as that for the memory address. It is only the index field (s bits) that needs to be calculated in a one's complement form. Since the mapping function is defined as an integer modulo $2^s - 1$, arithmetic operations are greatly simplified by noting that $2^s \ mod \ (2^s - 1) = 1$. Consider any two integers represented in an s-bit binary form. If we add the two numbers we obtain an $(s + 1)$-bit sum with the most significant bit being carry bit. But $2^s \equiv 1 \ mod \ (2^s - 1)$. Thus, addition modulo $(2^s - 1)$ is performed simply by using a conventional full binary adder of s bits and by folding the most significant carry bit output back into the least significant carry bit input. Thus arithmetic operations modulo $2^s - 1$ are equivalent to the familiar s-bit one's complement arithmetic and hence we call it one's complement cache. Suppose that the binary representation of the index of address A contains i s-bit subfields: $A_1, \cdots A_i$. Then we have

$$A \ mod \ (2^s - 1) = A_1 + 2^s A_2 + 2^{2s} A_3 + \cdots + 2^{(i-1)s} A_i \ mod \ (2^s - 1).$$

But

$$2^s = 1 \ mod \ (2^s - 1).$$

Therefore, reduction of A modulo $2^s - 1$ can be done very easily by the following addition

$$A \ mod \ (2^s - 1) = \sum_{k=1}^{k=i} A_k. \tag{2}$$

Therefore, only additions are needed to calculate the index field of a cache address. It is important to note that such one's complement address calculation does not increase the critical path length of a processor since it can be done in parallel with the normal address calculation as evidenced in the following discussion.

Figure 1 shows the block diagram of the address computation logic for a RISC processor. We take the MIPS R2000/R3000 [9] as an example to describe the details of the design. The execution of a single R2000 instruction consists of five pipelined stages: **IF, RD, ALU, MEM** and **WB**. Memory operations are performed by **Load/Store** instructions which are I-type instructions containing a base-register name and a 16-bit immediate displacement. The only addressing mode directly supported is base register plus 16-bit displacement. At the **ALU** stage, memory addresses are calculated using the ALU unit for all **Load/Store** instructions. It is at this stage where one's complement index for cache access is computed. The 16-bit displacement and the source register (base register) are converted to one's complement form by performing addition between the designated index field and the remaining high order bits. The two sums are added again to obtain the final index field for cache access. These operations are done in parallel with other normal machine operations. Because these additions are performed on an s-bit field which is a portion of a memory word, this process should take no longer than the normal address calculation process. Two addresses are therefore generated concurrently: one for cache access and the other for memory access in case of a cache miss.

The additional hardware cost as a result of this new cache design includes a few full adders, a few registers and some control logic as shown in Figure 1. Such additional cost will be a small fraction of a cache memory system. Even a small percentage of performance improvement can justify such insignificant hardware cost. As will be evidenced latter in this paper, we expect significant performance improvements over the set-associative cache with the same degree of associativity.

In a microprocessor that support a virtual memory system, the cache design presented above can be directly applied if the cache is accessed before address translation is done, i.e. the cache is accessed using virtual addresses. Virtually addressed cache requires special attention to keep data consistency or to avoid synonyms problem. The new cache organization is also directly applicable to processors that carry out cache accesses and address translations concurrently as the case of DEC's Alpha processors. Depending on whether physical address or virtual address are used for index and tag, there are 4 different types of caches. A good discussion of various cache types and their advantages and disadvantages can be found in [10]. If the cache is physically addressed, the address calculation will be done after the address translation. Thus, the physical address of a memory reference will go through the address computation unit shown in Figure 1 and lengthen the cache access time. As a result of this address computation process, two addresses are generated, a cache address in one's complement form and the original memory address, see Figure 1. If a cache miss occurs, a memory access is initiated and the data fetched from the memory is placed in the cache using the one's complement cache address.

For a multiple level cache hierarchy, it is possible that the second level cache could replace a line which is still in the first level cache provided that the second level cache is implemented using the conventional cache. As a result, "inclusion" property may not be guaranteed. If the "inclusion" property is essential to a design, the second level cache has to be a one's complement cache with a larger capacity and with the modulus being a prime number.

3 Simulation

In order to quantitatively evaluate the performance potential of the new cache design, trace-driven simulation experiments have been carried out. The MIPS Pixie and XSIM tools have been used to generate address traces to feed to the Dinero [11] cache simulator. Some minor changes have been made in the Dinero simulator in order to simulate the one's complement cache organization. Benchmark programs are first compiled and run on DECstations that contain the MIPS R3000 processors running version 4.2a of the DEC Ultrix operating system. Version 2.0 of the C compiler is used to compile C programs while version 3.6.20 FORTRAN compiler is used to compile floating point programs that are written in FORTRAN.

In addition to the above on-the-fly trace driven simulation, we have also carried out execution-driven simulation based on IBM3090 architectures. The execution driven simulation simulates program executions at the assembly instruction level of the IBM 3090VF. The IBM 3090 FORTRAN V2 compiler is used to compile the Fortran programs. Due to the large register set of IBM3090 and its significant difference in architecture from that of the MIPS R3000, we may notice some differences in cache miss ratios on the two simulation environments for the same benchmark program. We are, however, interested only in the relative difference in cache performance between the one's complement cache and the traditional cache design on one simulation environment.

The benchmark programs chosen in this paper for our performance evaluation are selected from the SPEC92 benchmarks that have been considered to be such an important measure of CPU performance that some machine designers and compiler writers are parameterizing their designs to maximize SPEC benchmark performance. SPEC benchmarks consist of a selection of nontrivial programs particularly suitable for intersystem comparisons. Due to their realistic nature and acceptable portability, SPEC benchmarks have been widely used for benchmarking purpose.

There are totally 20 programs in the entire SPEC92 benchmark suite including both integer and floating point programs [12]. To avoid time consuming exhaustive simulations, we have made a guided selection of the programs that are more informative for our comparison purpose. Since our main objective here is to show that the new cache design can minimize cache misses caused by cache line conflicts, we have selected the set of benchmark programs that have high cache miss ratios. Other programs in the benchmarks that have less than 1% miss ratio are not considered here. However, it should be pointed out that the programs which show good cache performance in conventional set-associative cache will show as good or better cache performance in the one's complement cache for obvious reasons. To verify this fact, we started our simulation experiments by running all those programs written in C in the SPEC92 benchmarks. During our initial experiments, we found out that some of the programs show very good cache performance (such as Alvinn, Ear, Espresso, Sc, Xlisp) on both conventional cache and the one's complement cache designs. For programs Alvinn, Sc, and Xlisp, the performance difference between the two cache organizations is hardly noticeable. For Ear and Espresso, the performance difference between the two cache organization is not significant as shown in Table 1 and Table 2.

We therefore selected those programs that have relatively high miss ratios. It has been shown in [12] that several integer and floating point programs exhibit poor cache performance including Gcc, Compress, Eqntott, Su2cor, Hydro2d, Swm256, Tomcatv and NASA7. These programs and their trace statistics in our simulations are shown in Table 3.

In order to verify the correctness of our simulation process and gain more confidence in our results, we have compared our simulation results on conventional cache organizations with previously published simulation results on the same benchmark programs. In particular, we compared our simulation results of the selected SPEC92 programs under the conventional cache organizations with the results reported in [12]. For all the cache parameters that we simulated, the absolute difference between our results and the results in [12] in terms of miss ratios is always less than 2%. About 70% of our results are within 10% relative difference compared with the results in [12]. Less than 5% of results exceed 20% relative difference compared with the results in [12]. The small differences can mainly be attributed to the fact that we used a later version of operating system and the Fortran compiler. The operating system used in our experiment is version 4.2a of the DEC ultrix as opposed to version 4.1 of the DEC ultrix, and the Fortran compiler used here is version 3.6.20 while version 2.1 was used in [12]. As indicated in [12], different compilers will result in different cache miss ratios.

4 Performance Results

We present numerical results from our simulation experiments in this section. Cache miss ratios are used as the performance measure in our following discussions. It is observed in our experiments that miss ratios of instruction caches are very small for the cache configurations considered and the performance difference between the 1's complement cache and the conventional cache is hardly noticeable. The reason why the one's complement cache does not show much performance improvement over the conventional caches is the following. Memory references to data may have a stride which is the address difference between two consecutive memory accesses. If the stride is not relative prime to the number of sets, a sequence of memory accesses with the stride may result in cache line conflicts. The one's complement cache tries to reduce such conflicts by using an odd number of sets. Memory references to instructions, on the other hand, are usually sequential with an exception of loops. Stride accesses do not present in instruction caches. Therefore, the number of sets (modulus) in an instruction cache does not play as much a role as in a data cache. We therefore concentrate our evaluation on data cache only.

Figure 2 shows the cache miss ratio of program GCC as a function of cache size. The degree of associativity is assumed to be one for both the conventional cache and the one's complement cache. It is shown in [12] that different cache line sizes have little effect on cache performance for this program. Similar observations are obtained in our experiments as shown in Table 4. We therefore fix the cache line size at 16 bytes. As the cache size increases from 4 Kbytes to 32 Kbytes, the miss ratio of both cache organizations reduces. However, the cache miss ratio of the one's complement cache is constantly lower than the conventional cache. For all the cache sizes considered in this figure, the cache miss ratios of the one's complement cache are about half of the miss ratios of the conventional direct-mapped cache. When we increase the degree of associativity for both caches, similar performance improvements are observed as shown in Figure 3 with the degree of associativity increased to 2.

There are two unusual observations from the above two figures. First of all, as the cache size increases we would expect that the cache line conflicts should reduce. Therefore, by common belief the performance improvement of the one's complement cache over the conventional cache would decrease with the increase of cache size. However, our experiments show the opposite. The reason to this phenomenon is the following. As the cache size increases, the miss ratios of both cache organizations drop as shown in the two figures. This drop in miss ratios is caused by both reduction of capacity misses and reduction of conflict misses. In case of a conventional cache design, both capacity miss and conflict miss reductions are noticeable while in case of the one's complement cache we only see capacity miss reduction. As a result, the absolute change in miss ratio while changing cache size from 4KB to 32KB (see Figure 2) is about 8% (6% in Figure 3) in case of the conventional cache, whereas it is about 4% (3% in Figure 3) in case of one's complement cache. However, as the effects of capacity misses go away the conflict misses dominate the cache misses. Therefore, the relative performance difference between the two cache designs is getting larger when a majority of cache misses are caused by cache line conflicts.

Secondly, when we increase the associativity from 1 to 2, one would expect less performance improvement. However, our results show that the one's complement cache sustains the similar performance improvements while changing the associativity, (compare Figures 2 and 3). This is another "unusual" phenomenon which may appear to be unreasonable. If we analyze the caching behavior of application programs carefully, it is not difficult to explain this fact. Although increasing associativity results in more locations in a set that data can be placed, the number of sets in the cache is reduced for the same cache size. As a result, for many applications cache line conflicts may keep the same or little change. We will discuss this observation further after we see more experimental results.

Figure 4 shows the data miss ratios of program Compress which is a C program for data compression. The miss ratios are plotted against cache sizes with line size fixed at 16 bytes and associativity 1 since the miss ratio is not very sensitive to cache line size as shown in [12] and our experiment shown in Table 5. As shown in the figure, the one's complement cache has significantly lower miss ratios than the conventional cache. The performance improvement for this program ranges from 80% to 90%. Similarly for associativity 2, the performance improvements ranges from 76% to 88% as shown in Figure 5.

Eqntott is a C program which translates a boolean equation into a truthtable. Table 6 and Figure 6 show the data miss ratios for two different degrees of associativities of program Eqntott. It is shown in the figure that the one's complement cache significantly improves cache performance.

Remarks: It is important to note that all the above three integer programs show fairly high miss ratio on the conventional cache. The numerical results collected by Gee et al. [12] show that high degrees of associativities do not significantly reduce the cache miss ratio of these programs, particularly for the cache parameters considered above. One mistaken conclusion that could be drawn from this phenomenon would be that cache miss ratios of these programs are not caused by cache line conflicts. Our simulation results together with the new cache design

unveiled an important fact which has been unclear before: cache line conflicts exist even in set-associative cache memories with a high degree of associativity. Therefore, it may be misleading to use miss ratio difference between direct-mapped cache and 2-way set-associative cache as a performance indicator to show the percentage of cache line conflicts being removed by a new cache design. Achieving similar performance to the 2-way set-associative cache does not mean that the cache line conflicts are minimum. All above results and the results that follow show significant reduction in cache line conflicts from 2-way set-associative cache.

Data miss ratios of HYDRO2D which is a floating point Fortran program are plotted in Figure 7. Previous simulation results reported in [12] indicate that cache performance of this program is more sensitive to cache line sizes than to the degree of associativity. We therefore plotted the miss ratios as a function of cache line sizes while fixing associativity at 2 and the cache size at 32 Kbytes. From Figure 7, we can see that the miss ratios of both cache organizations drop quickly as the cache line size increases from 16 bytes to 256 bytes. The one's complement cache shows better cache performance for all cache line sizes considered here. The performance gain of the one's complement cache over the conventional cache is 75% for cache line size being 32 bytes and 68% for cache line size being 256 bytes.

Similar to HYDRO2D, cache performance of Swm256 keeps virtually unchancged while the degree of associativity varies from 1 to 8 [12]. But, it changes drastically with the change of cache line size. Miss ratios of this program for different cache line sizes are shown in Figure 8. Still the one's complement cache presents better cache performance than the conventional cache. In other words, cache line conflicts do exist in this program although high degree of associativity does not reduce cache miss ratio. The one's complement cache eliminates the cache line conflicts that can not be reduced by increasing the degree of associativity.

Gee et al. [12] noted in their study that there are some anomalies in their results for the effect of associativity on miss ratio. That is, miss ratio can increase as the associativity increases for certain data reference patterns. Tomcatv is one of such programs that exhibit these anomalies. With the analysis and the experiments presented in this paper, we are able to explain this phenomenon. Figure 9 shows the miss ratios of Tomcatv program as a function of cache line sizes. It can be seen from this figure that cache line conflicts exist in this program because the miss ratios of the one's complement cache are less than half of that of conventional cache. The reason why higher degree of associativity may give high miss ratio in the conventional cache is the following. With the same cache size, increasing the associativity decreases the number of sets in the cache. This reduction in set number may result in more cache line conflicts. However, the one's complement cache prevents this kind of line conflicts from occurring. The result is the reduction by half of the data miss ratio as compared to the conventional cache.

Compared to the above three floating point Fortran programs, cache performance of program Su2cor is more sensitive to cache size change in the medium cache size range. For example, miss ratios of Tomcatv change from 7.9% to 7.3% and miss ratios of Swm256 keep unchanged while cache size changes from 32Kbytes to 256Kbytes for $d = 2$ and $L = 16$ bytes [12]. The miss ratio of Su2cor, on the other hand, changes from 15.9% to 3.9% in the same size range and same other cache parameters. Based on this observation, we plotted cache miss ratios of the Su2cor program versus cache size as shown in Figure 10. Although the shape of the one's complement cache is similar to the conventional cache, miss ratio of the former is much lower than the miss ratio of the latter.

For program NASA7, previous research [12] has shown that cache line size of 64 bytes exhibits the best cache performance for medium cache sizes. By fixing the cache line at 64 bytes, we plotted the cache miss ratios of the NASA7 as a function of cache size as shown in Figure 11. The performance improvement of the one's complement cache over the conventional caches exceeds 90% for all cache sizes considered.

In order to assess exactly to what extent our one's complement cache gets rid of the conflict misses, we listed the miss ratios of fully associative cache along with the miss ratios of the one's complement cache and the conventional caches as shown in Tables 7, 8, and 9. It is observed from the tables that the miss ratios of the one's complement cache are very close to those of the fully associative cache indicating that the one's complement cache removes cache line conflicts very effectively.

5 Related Work

Realizing the importance of reducing cache line conflicts, extensive research has been reported in the literature aiming at minimizing cache line conflicts. The most straightforward approach is to increase the degree of associativity of a set-associative cache so that a data item can be potentially placed in a large number of places thereby decreasing the chance of conflicts. However, increasing the degree of associativity results in complicated hardware for associative data search in the cache and also possibly large cache access time as compared to direct-mapped cache [2]. In addition, in many situations, a high degree of associativity may not help in reducing cache line conflicts. For the

same cache size, increasing associativity results in decreased number of sets that data can be mapped to although several cache lines can be mapped to the same set. As an example, consider 2 alternative designs of an eight-line cache memory: 2-way set-associative and direct-mapped. The 8-line cache size gives rise to 4 sets for 2-way set-associative design and 8 sets for direct-mapped design. Suppose that we want to access a row of a matrix that is stored in column-major and the length of a column in terms of cache lines is an even number. No matter which one of the two designs one wishes to consider, only four or less number of cache lines can be placed in the cache. In other words, the number of line interferences is the same for either case if the row has more than 4 elements. While this example is simple, it demonstrates a potential problem that exists in any existing set-associative cache design except for a fully associative cache.

Several other approaches have been proposed to reduce cache line conflicts. Jouppi presented an interesting idea of adding a so-called victim cache which is a small fully associative cache used to hold data that are removed from a direct-mapped cache [4]. Agarwal and Pudar proposed [5] a column-associative cache in which a different hashing function is dynamically applied to place the data in a different set when presented with conflicting addresses. They have shown that the column-associative cache performs as well as a 2-way set-associative cache while keeping the advantages of a direct-mapped cache. It is also shown in [5] that the column-associative cache compare favorably with victim-cache and hash-rehash cache. But Agarwal and Pudar remarked that the column-associative cache may not be easily extended to high degree of associativity since it is likely to result in high complexity in implementing the design which in turn would adds little to the performance and might even degrade it. Recently, Seznec has presented a very interesting cache design called two-way skewed-associative cache [6]. The main idea behind the skewed-associative cache is that two different mapping functions are used to map data into two different cache banks (note: a set contains two lines, one from each of the two different banks). They claimed that different mapping functions on different banks would not affect performance if the computations of the mapping functions were added to a non critical path. Compared to the same size 2-way set-associative cache, the skewed-associative data cache [6] reduces cache miss ratios of the considered benchmarks from about 2.6% to 2.09% for an 8Kbyte cache and from 4.2% to 3.76% for a 4Kbyte cache.

In spite of many previous efforts in reducing cache line conflicts as discussed above, we believe that the line conflict problem has yet to be solved. Our belief comes from two factors. First of all, most of existing approaches were able to achieve the cache performance close to that of 2-way set-associative cache. It has been shown in [3] and in the matrix example given above that 2-way set-associativity does little in reducing cache line conflicts. Secondly, our experience with vector cache designs [7, 8] as well as our recent simulation experiments indicate that in many application programs significant conflict reduction is possible if cache is designed right. Therefore, in order to eliminate cache line conflicts, we proposed the one's complement cache.

6 Conclusions

In this paper, an innovative cache design for microprocessors has been presented. The main idea behind the new cache design is to map memory data evenly into cache memory to prevent cache line conflicts from occurring. Through parallel computation of cache addresses and memory addresses, the new design does not increase cache hit time. Since the cache address computation is done in one's complement arithmetic, the new cache is called one's complement cache. In terms of reducing cache line conflicts, this design is shown to be very effective. The one's complement cache can resolve cache line conflicts that conventional cache can not resolve by just increasing the degree of set-associativity. Address traces of programs from SPEC92 are used to evaluate the performance of the new design. Numerical results show that the miss ratios of the one's complement data cache for several considered benchmarks are half as much as the miss ratios of conventional set-associative caches.

Acknowledgement

In our simulation, XSim developed by Michael D. Smith in 1990 at Stanford Univ. and the cache simulator called Dinero III developed by Professor Mark D. Hill of University of Wisconsin have been used. The authors would like to thank the reviewers of this paper for providing detailed comments that helped in improving the quality of this paper.

References

[1] J. L. Hennessy and D. A. Patterson, *Computer Architecture, A Quantitative Approach.* Morgan Kaufmann, 1990.

[2] M. D. Hill, "A case for direct-mapped caches," *IEEE Computer*, pp. 25–40, Dec. 1988.

[3] M. S. Lam, E. E. Rothberg, and M. E. Wolf, "The cache performance and optimizations of blocked algorithms," in *Proc. of Arch. Supp. for Prog. Lang. and Opr. Sys.*, pp. 63–74, April 1991.

[4] N. P. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers," in *17th Annu. Symp. on Comput. Arch.*, 1990.

[5] A. Agarwal and S. D. Pudar, "Column-associative caches: a technique for reducing the miss rate of direct-mapped caches," in *The 20th Ann. Int. Symp. on Comp. Arch.*, pp. 179–190, May 1993.

[6] A. Seznec, "A case for two-way skewed-associative caches," in *The 20th Ann. Int. Symp. on Comp. Arch.*, pp. 169–178, May 1993.

[7] Q. Yang and L. W. Yang, "A novel cache design for vector computers," *19th Ann. Int'l Symp. on Computer Architectures*, May 1992. Queensland, Australia.

[8] Q. Yang, "Performance of cache memories for vector computers," *Journal of Parallel and Distributed Computing 19*, pp. 163–178, 1993.

[9] G. Kane, *MIPS RISC Architecture.* Prentice Hall, 1989.

[10] C. E. Wu, Y. Hsu, and Y.-H. Liu, "A quantitative evaluation of cache types for high performance computer systems," *IEEE Tran. on Comput.*, vol. 42, pp. 1154–1162, Oct. 1993.

[11] M. D. Hill, "Dinero cache simulator,". Copyright 1985, 1989, Univ. of Wisconsin.

[12] J. D. Gee, M. D. Hill, D. N. Pnevmatikatos, and A. J. Smith, "Cache performance of the SPEC92 benchmark suite," *IEEE Micro*, pp. 17–27, Aug. 1993.

[13] H. Cheong and A. V. Veidenbaum, "A version control approach to cache coherence," in *Proc. Int. Conf. Supercomputing*, pp. 322–330, June 1989.

[14] S. L. Min and J. L. Bear, "A timestamp-based cache coherence scheme," in *Proc. Int. Conf. Parallel Processing*, pp. 23–32, August 1989.

Cache Line size	16KB cache conventional	1's complement	32KB cache conventional	1's complement
16B	2.74	2.21	2.74	2.21
32B	1.45	1.23	1.44	1.23
64B	0.97	0.81	0.97	0.81
128B	0.49	0.41	0.49	0.40

Table 1: Data cache miss ratio (%) of program Ear (associativity 1)

Cache Line size	16KB cache conventional	1's complement	32KB cache conventional	1's complement
16B	3.57	2.89	1.95	1.47
32B	2.45	1.93	1.44	1.16
64B	1.54	1.18	0.81	0.62
128B	1.13	0.84	0.53	0.46

Table 2: Data cache miss ratio (%) of program Espresso (associativity 2)

Program	Inst. Refer.	Data Refer.	Total
Compress	82,213,437	22,498,112	104,711,549
Eqntott	1,159,645,211	212,709,676	1,372,354,887
Gcc	1,256,362,217	395,172,921	1,651,535,138
Tomcat	1,194,205,124	635,308,445	1,829,513,569
Hydro2d	5,064,358,776	2,172,136,548	7,236,495,324
Su2cor	4,651,222,371	1,891,432,821	6,542,655,192
SWM256	8,917,512,117	2,605,171,450	11,522,683,567
Nasa	6,096,437,722	3,258,043,729	9,354,481,451
Ear	15,931,241,862	3,524,795,221	19,457,037,083
Espresso	2,564,375,602	610,824,656	3,175,200,258

Table 3: Memory reference counts of selected SPEC92 programs.

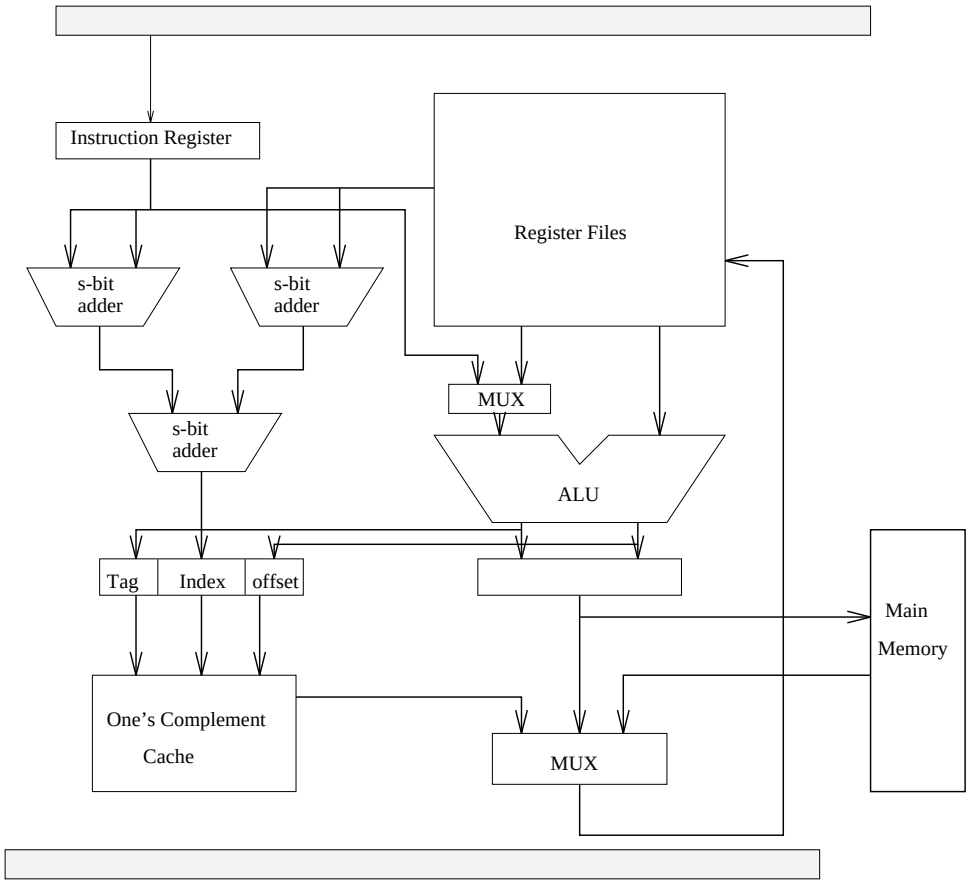

Figure 1: Block diagram of the one's complement cache design.

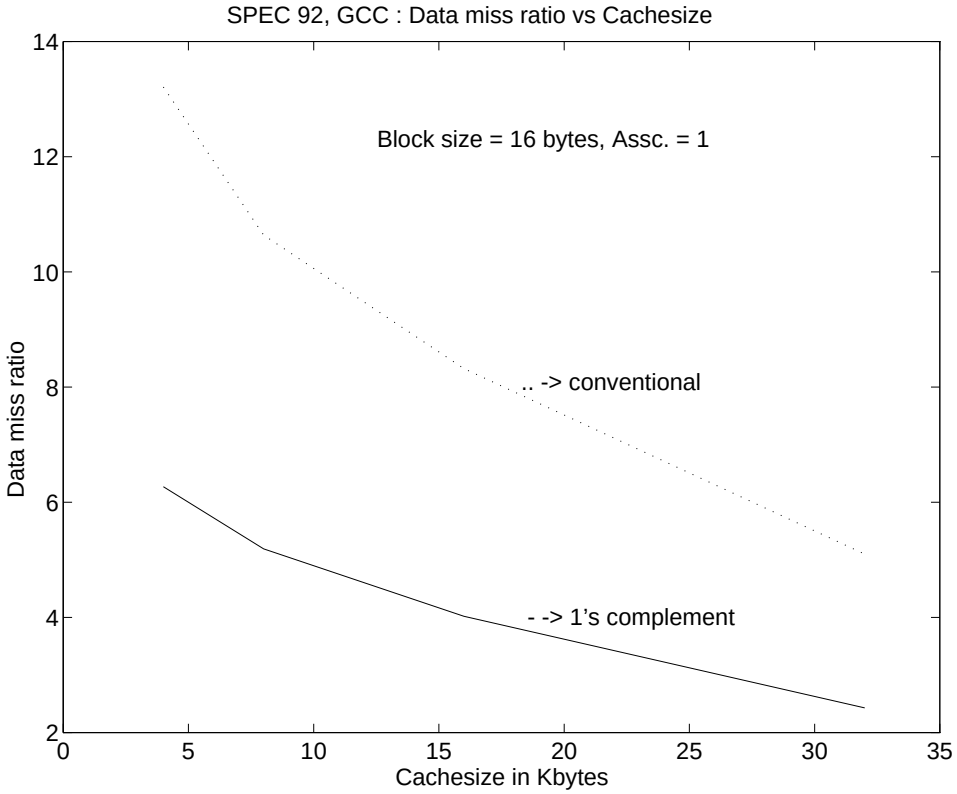

Figure 2: Data miss ratio of GCC vs cache size with the degree of associativity $d = 1$.

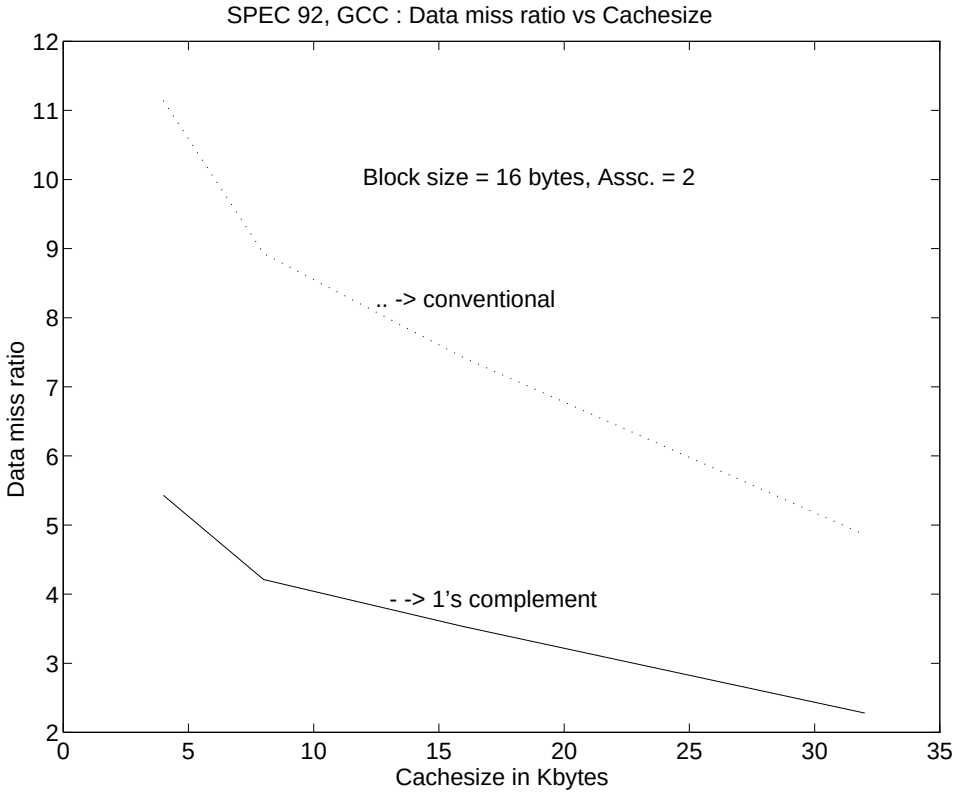

Figure 3: Data miss ratio of GCC vs cache size with the degree of associativity $d = 2$.

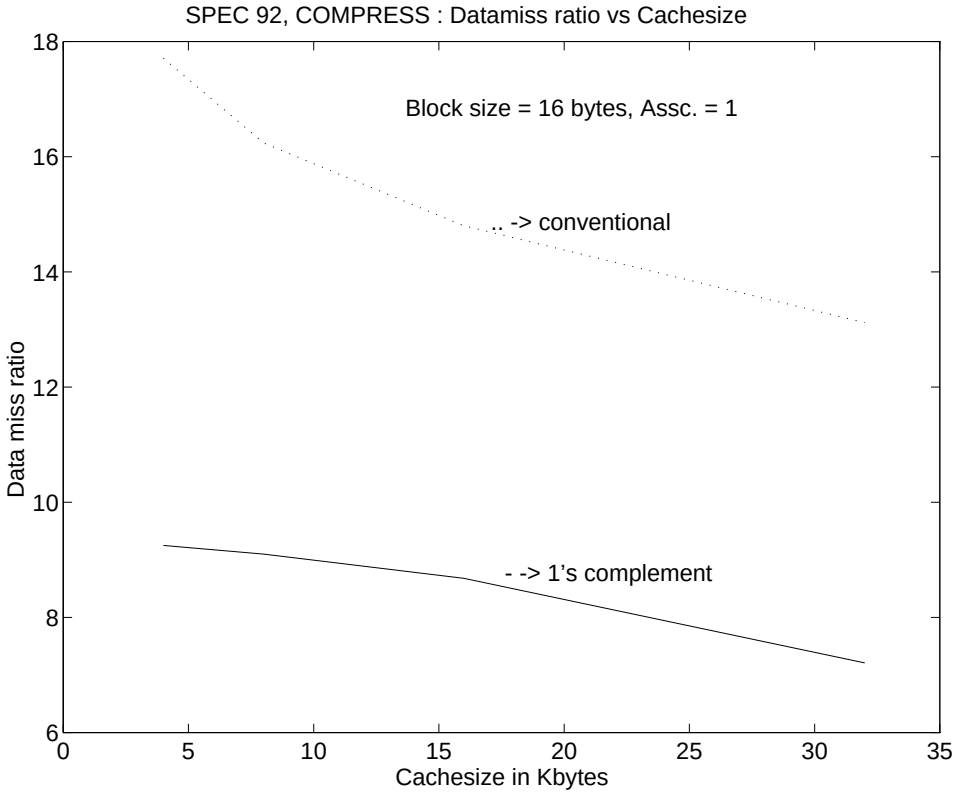

Figure 4: Data miss ratio of COMPRESS vs cache size with the degree of associativity $d = 1$.

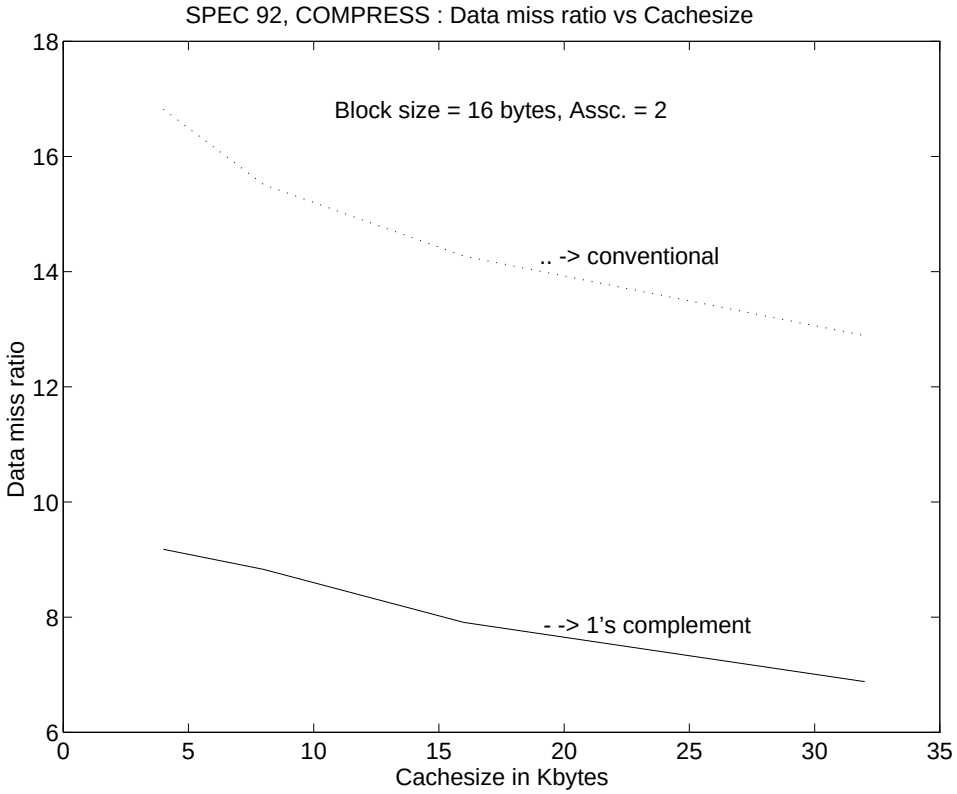

Figure 5: Data miss ratio of COMPRESS vs cache size with the degree of associativity $d = 2$.

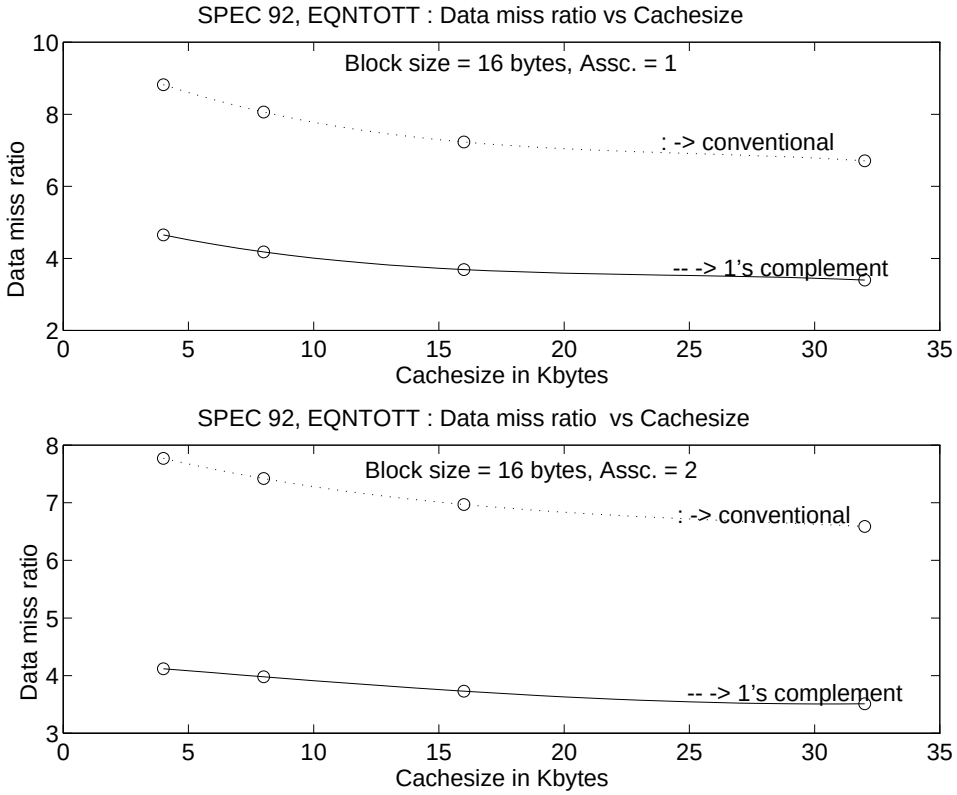

Figure 6: Data miss ratio of EQNTOTT vs cache size.

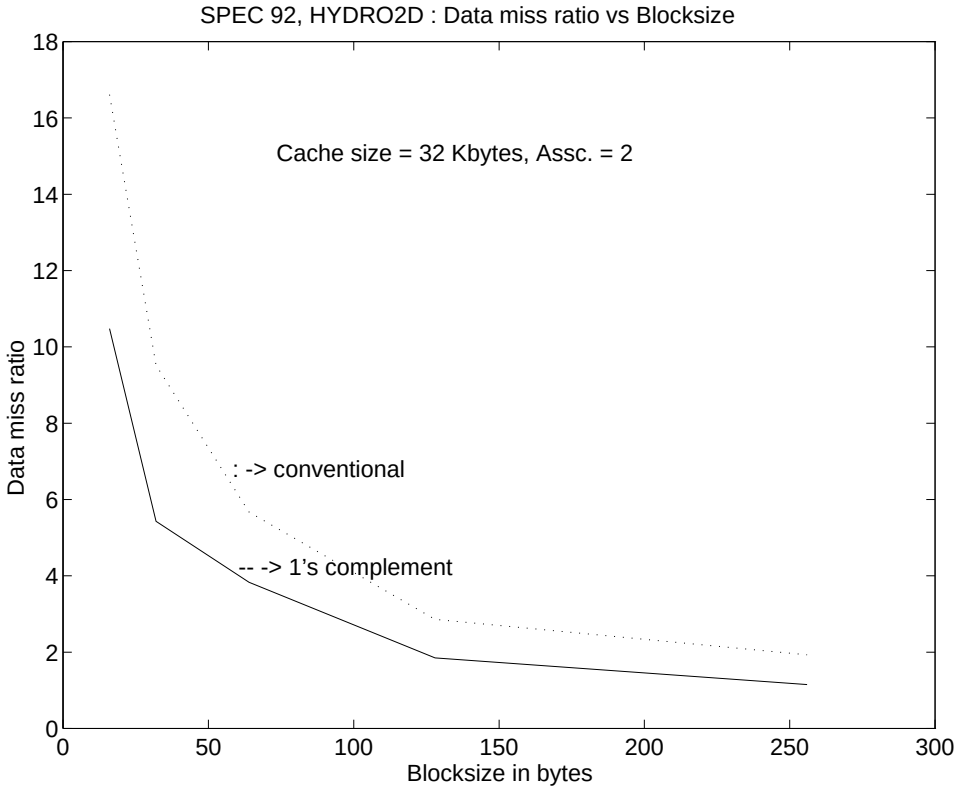

Figure 7: Data miss ratio of HYDRO2D vs cache line size (blocksize).

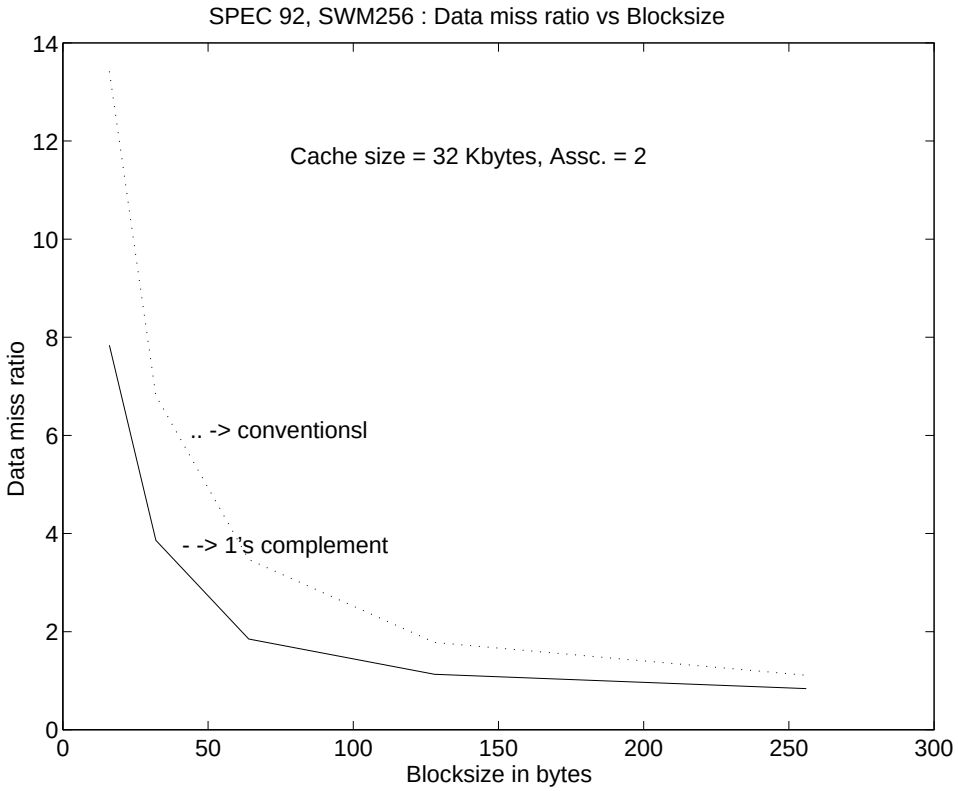

Figure 8: Data miss ratio of SWM256 vs cache line size (blocksize).

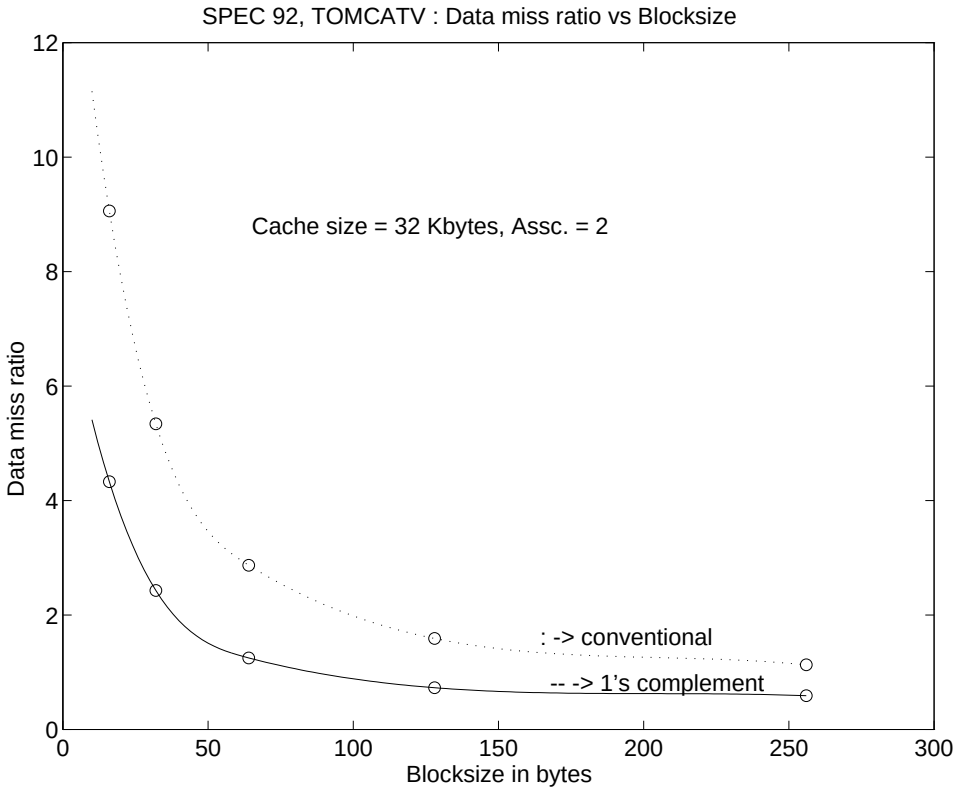

Figure 9: Data miss ratio of TOMCATV vs cache line size (blocksize).

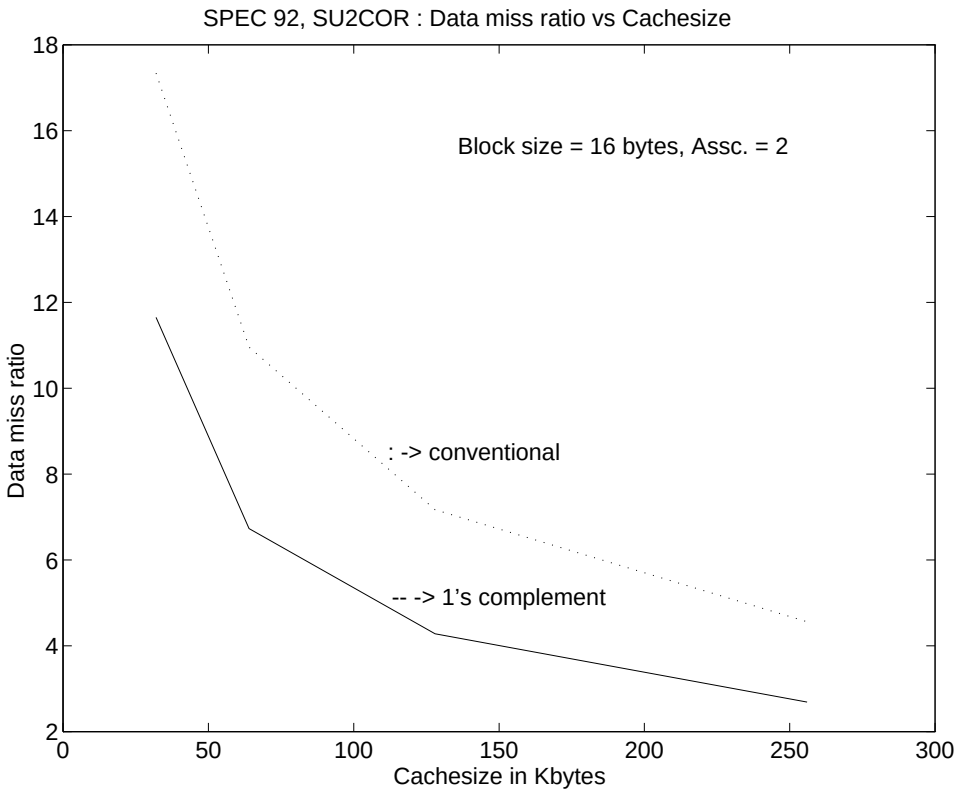

Figure 10: Data miss ratio of SU2COR vs cache size.

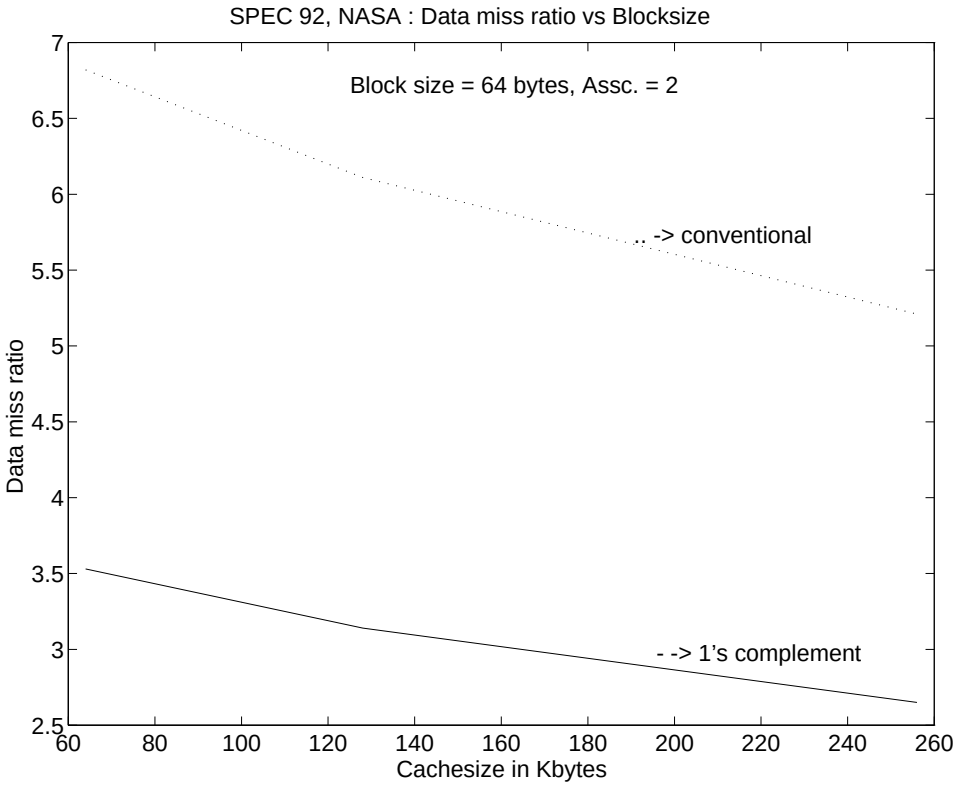

Figure 11: Data miss ratio of NASA7 vs cache size.

Direct-mapped cache:

Cache configuration	Conventional cache	1's complement	abs. difference
4KB cache, 16B line	0.1259	0.0627	0.0632
4KB cache, 32B line	0.1184	0.0599	0.0585
4KB cache, 64B line	0.1300	0.0573	0.0727
8KB cache, 16B line	0.1005	0.0519	0.0486
8Kb cache, 32B line	0.0921	0.0483	0.0438
8KB cache, 64B line	0.1005	0.0462	0.0543
16KB cache, 16B line	0.0796	0.0402	0.0394
16KB cache, 32B line	0.0710	0.0372	0.0338
16KB cache, 64B line	0.0770	0.0359	0.0411
32KB cache, 16B line	0.0420	0.0243	0.0177
32KB cache, 32B line	0.0307	0.0217	0.0090
32KB cache, 64B line	0.0256	0.0211	0.0045

2-way set-associative cache:

Cache configuration	Conventional cache	1's complement	abs. difference
4KB cache, 16B line	0.1072	0.0543	0.0529
4KB cache, 32B line	0.0988	0.0491	0.0497
4KB cache, 64B line	0.0986	0.0474	0.0512
8KB cache, 16B line	0.0851	0.0421	0.0430
8KB cache, 32B line	0.0761	0.0382	0.0379
8KB cache, 64B line	0.0731	0.0363	0.0368
16KB cache, 16B line	0.0693	0.0353	0.0340
16KB cache, 32B line	0.0609	0.0318	0.0291
16KB cache, 64B line	0.0570	0.0299	0.0271
32KB cache, 16B line	0.0328	0.0228	0.0100
32KB cache, 32B line	0.0219	0.0212	0.0007
32KB cache, 64B line	0.0162	0.0160	0.0002

Table 4: Miss ratio comparison between 1's complement cache and existing caches for benchmark "GCC" on MIPS R2000/R3000.

direct-mapped cache:

Cache configuration	Conventional cache	1's complement	abs. difference
4KB cache, 16B line	0.1739	0.0925	0.0814
4KB cache, 32B line	0.1805	0.0952	0.0853
4KB cache, 64B line	0.1894	0.0964	0.0930
8KB cache, 16B line	0.1604	0.0910	0.0694
8KB cache, 32B line	0.1648	0.0932	0.0716
8KB cache, 64B line	0.1712	0.0941	0.0771
16KB cache, 16B line	0.1465	0.0818	0.0647
16KB cache, 32B line	0.1499	0.0836	0.0663
16KB cache, 64B line	0.1549	0.0848	0.0701
32KB cache, 16B line	0.1301	0.0771	0.0530
32KB cache, 32B line	0.1326	0.0788	0.0538
32KB cache, 64B line	0.1367	0.0797	0.0570

2-way set-associative cache:

Cache configuration	Conventional cache	1's complement	abs. difference
4KB cache, 16B line	0.1633	0.0918	0.0715
4KB cache, 32B line	0.1660	0.0943	0.0717
4KB cache, 64B line	0.1710	0.0958	0.0752
8KB cache, 16B line	0.1533	0.0883	0.0650
8KB cache, 32B line	0.1553	0.0901	0.0652
8KB cache, 64B line	0.1591	0.0912	0.0679
16KB cache, 16B line	0.1414	0.0791	0.0623
16KB cache, 32B line	0.1436	0.0817	0.0619
16KB cache, 64B line	0.1472	0.0830	0.0642
32KB cache, 16B line	0.1260	0.0688	0.0572
32KB cache, 32B line	0.1281	0.0711	0.0570
32KB cache, 64B line	0.1317	0.0724	0.0593

Table 5: Miss ratio comparison between 1's complement cache and existing caches for benchmark "*Compress*" on MIPS R2000/R3000

direct-mapped cache:

Cache configuration	Conventional cache	1's complement	abs. difference
4KB cache, 16B line	0.0859	0.0465	0.0394
4KB cache, 32B line	0.0618	0.0372	0.0246
4KB cache, 64B line	0.0618	0.0362	0.0256
8KB cache, 16B line	0.0775	0.0418	0.0357
8KB cache, 32B line	0.0530	0.0361	0.0169
8KB cache, 64B line	0.0481	0.0356	0.0125
16KB cache, 16B line	0.0719	0.0369	0.0350
16KB cache, 32B line	0.0475	0.0342	0.0133
16KB cache, 64B line	0.0402	0.0341	0.0061
32KB cache, 16B line	0.0660	0.0340	0.0326
32KB cache, 32B line	0.0426	0.0313	0.0113
32KB cache, 64B line	0.0347	0.0302	0.0045

2-way set-associative cache:

Cache configuration	Conventional cache	1's complement	abs. difference
4KB cache, 16B line	0.0735	0.0412	0.0323
4KB cache, 32B line	0.0482	0.0336	0.0146
4KB cache, 64B line	0.0391	0.0262	0.0129
8KB cache, 16B line	0.0714	0.0398	0.0316
8KB cache, 32B line	0.0462	0.0314	0.0148
8KB cache, 64B line	0.0362	0.0231	0.0131
16KB cache, 16B line	0.0687	0.0373	0.0314
16KB cache, 32B line	0.0440	0.0297	0.0143
16KB cache, 64B line	0.0343	0.0212	0.0131
32KB cache, 16B line	0.0645	0.0351	0.0294
32KB cache, 32B line	0.0409	0.0292	0.0117
32KB cache, 64B line	0.0319	0.0199	0.0120

Table 6: Miss ratio comparison between 1's complement cache and existing caches for benchmark "*Eqntott*" on MIPS R2000/R3000

programs	*1 way*		*2 way*		*fully associative*
	conv.	*1' comp.*	*conv.*	*1' comp.*	
Doduc	0.0110	0.0099	0.0107	0.0097	0.0094
Fppp	0.0011	0.0008	0.0088	0.0008	0.0009
Hydro2d	0.0923	0.0490	0.0841	0.0489	0.0485
Mdjdp2	0.0133	0.0069	0.0098	0.0068	0.0067
Mdjsp2	0.0094	0.0053	0.0073	0.0052	0.0050
Nasa	0.1610	0.0782	0.1412	0.0770	0.0760
Su2cor	0.0734	0.0510	0.0666	0.0463	0.0461
Swm256	0.0820	0.0466	0.0700	0.0460	0.0438
omcatv	0.0789	0.0413	0.0651	0.0388	0.0362
Wave	0.0206	0.0105	0.0199	0.0105	0.0101

Table 7: Miss ratio comparison for 64KB cache with cache line size of 16 bytes on IBM3090

programs	*1 way*		*2 way*		*fully associative*
	conv.	*1' comp.*	*conv.*	*1' comp.*	
Doduc	0.0102	0.0096	0.0102	0.0095	0.0091
Fppp	0.0007	0.0005	0.0007	0.0005	0.0004
Hydro2d	0.0686	0.0413	0.0621	0.0406	0.0400
Mdjdp2	0.0091	0.0068	0.0084	0.0067	0.0066
Mdjsp2	0.0061	0.0041	0.0058	0.0041	0.0038
Nasa	0.1213	0.0616	0.0962	0.0612	0.0603
Su2cor	0.0650	0.0441	0.0610	0.0436	0.0424
Swm256	0.0615	0.0388	0.0597	0.0384	0.0372
Tomcatv	0.0562	0.0310	0.0511	0.0300	0.0290
Wave	0.0157	0.0099	0.0150	0.0100	0.0092

Table 8: Miss ratio comparison for 64KB cache with cache line size of 32 bytes on IBM3090

programs	*1 way*		*2 way*		*fully associative*
	conv.	*1' comp.*	*conv.*	*1' comp.*	
Doduc	0.0096	0.0088	0.0096	0.0088	0.0082
Fppp	0.0006	0.0004	0.0006	0.0004	0.0003
Hydro2d	0.0654	0.0400	0.0618	0.0392	0.0392
Mdjdp2	0.0073	0.0051	0.0068	0.0050	0.0046
Mdjsp2	0.0045	0.0033	0.0045	0.0033	0.0030
Nasa	0.1004	0.0555	0.0996	0.0550	0.0540
Su2cor	0.0514	0.0396	0.0510	0.0394	0.0386
Swm256	0.0487	0.0300	0.0485	0.0298	0.0295
Tomcatv	0.0416	0.0251	0.0410	0.0250	0.0240
Wave	0.0101	0.0074	0.0095	0.0074	0.0072

Table 9: Miss ratio comparison for 64KB cache with line size of 64 bytes on IBM3090