A One's Complement Cache Memory *

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

ABSTRACT

Most of today's microprocessors have an on-chip cache to reduce average memory access latency. These on-chip caches generally have low associativity and small sizes. Cache line conflicts are the main source of cache misses which are essential to overall system performance. This paper introduces an innovative, conflict-free cache design, called one's complement cache. By means of parallel computation of cache addresses and memory addresses of data, the new design does not increase critical hit time of cache accesses. Cache misses caused by line interferences are minimized by means of evenly distributing data items referenced by program loops across all sets in a cache. Evenly 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 a set of programs from SPEC92 benchmarks show that the new design improves cache performance over the conventional set-associative cache by about 100% with negligibly additional hardware cost.

1 Introduction

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. Conventional caches 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 associative cache) cache locations. If more than d blocks of data referenced by a program are mapped into a same set, 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 relatively small degree of associativity or direct-mapped cache because of its advantages of easy implementation and fast cache access etc.[2]. Therefore, if data are unevenly mapped among the available cache sets, cache line conflicts can pose a severe performance limitation to the cache memory. A recent study on blocked matrix multiply algorithm [3] shows that the cache line interference misses in a direct-mapped cache increase drastically and dominate cache misses after the fraction of a 16K-word cache being used exceeds 3%.

1.1 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, 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. 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 inter-

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

esting 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.

1.2 Source of Conflicts

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 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 explore the possibility of eliminating cache line conflicts, we made an effort to analyze where line conflicts come from.

It is a well known fact that loops constitute the main portion of any computer program. Data accesses inside these loops usually have some regularities. Such regularities can be characterized by the distance between consecutive memory references to a data array. One typical example is memory references inside a loop to a row vector of a two-dimensional matrix stored in a column-major. The distance is the column length of the matrix. As described in the example given above, if the column length in terms of cache line is not relative prime to the number of sets in the cache, conflicts may occur. We say such conflicts are the results of uneven distribution of data in the cache. Many application programs such as FFT, LUD, PERFECT Club benchmarks, SPEC92 exhibit the similar behavior as evidenced by our detailed analysis of data reference patterns with respect to cache organizations and our simulation experiments on the benchmarks as evidenced later in this paper.

1.3 Contribution of This Paper

In this paper, we propose a novel conflict-free cache design that distributes cached data evenly across all sets in the cache, i.e. prevent cache line conflicts based on general caching behavior of programs rather than trying to do something when conflicts occur. The new design, referred to as *one's complement cache*, uses a uniform conflict-free mapping function without adding any additional computation time for different mapping functions. It also allows the flexibility of chosing different degree of associativity in cache design. The primary idea of the design is that memory data are mapped into cache lines according to a prime or an odd number in the form of $2^s - 1$. In other words, the addresses used to access cache memory are in one's complement form rather than the direct binary number.

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 performs about 100% better than the conventional set-associative cache with the same degree of associativity.

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 concludes the paper.

2 One's Complement Cache

In this section, we present the new design of the conflict-free cache. In order to understand where cache line conflicts come from so that a right cache design can be made, we first carry out a thorough analysis on the source of cache line conflicts. Based on our analysis, we will present the one's complement cache design which eliminates conflicts. The trick to make such cache work is to make sure the address computation for cache accesses adds no additional delay to the critical cache hit time.

2.1 Analysis of Cache Line Conflicts

In our following analysis, we assume that the considered cache has $S = 2^s$ set. Each set is assumed to have d cache lines or the degree of associativity is represented by d. Each cache line consists of L bytes which is called line size. The cache size is represented by C which is equal to $dL2^s$ bytes. As discussed in the introduction, any computer program contains loops that form the most time-consuming part. PROFILing of large number of benchmarks [9] has shown that almost all of the time-consuming loop nests contain at least three level loops. Majority of these nested loops involve only one major array which is usually two-dimensional or three-dimensional with a small size in the third dimension[9].

Amdahl's law tells us that one principle in computer design is to make common case faster. We therefore analyze the data access patterns of general loops.

Consider the following most frequent loops in a program [9]:

For $k = 1$ to N DO
  For $j = 1$ to N DO
    For $i = 1$ to N DO
      $A[\cdots] = \cdots + A[a_1 i + b_1 j + c_1 k + d_1,$
      $a_2 i + b_2 j + c_2 k + d_2],$
    END
  END
END

where i, j, k, are loop variables and a, b, and c are constant integers. Consider iterations of the innermost loop with different values of i for a given loop instance of j and k. As a result of the loop iterations, a sequence of memory references are issued to the following memory locations:

$$B + iD, \quad for \quad i = 1, 2, \cdots, N, \tag{1}$$

where B and D are constant with respect to i but functions of j, k, a, b, c and array size N. If we consider only the memory references to different cache lines since we are interested in line interferences, these memory locations are mapped into cache locations

$$B + iD' \ mod \ 2^s, \quad for \quad i = 1, 2, \cdots, N', \tag{2}$$

in a set-associative cache with 2^s sets.

For the purpose of clarity, we define several terms as follows.

Definition 2.1 *A cache line* **contention** *is said to occur between two data items when the two cache lines containing respectively the two data items are mapped to the same cache set.*

A cache line contention may or may not result in a conflict miss depending on the degree of associativity and the number of free line locations in a set.

Definition 2.2 *A cache line* **conflict** *is said to occur between two data items when a cache line contention between the two data items results in a necessary replacement of one of the two contending cache lines.*

Example 2.1 *Consider a 2-way set associative cache with 8 cache lines, i.e. there are 4 sets in the cache. Suppose a processor issues a sequence of memory references to six data items, x_i, in memory locations $2Li$, for $i = 1, \cdots, 6$. Cache line contentions among x_1, x_3, x_5 and among x_2, x_4, x_6 occur. Cache line conflicts occur between x_1 and x_5, and between x_2 and x_6 if LRU replacement algorithm is assumed.*

Let $GCD(x, y)$ be the greatest common divisor of x and y. The following theorem can be easily proved.

Theorem 2.1 *In a S-line direct-mapped cache, a sequence of N consecutive memory references to data blocks (lines) located in $B+Di$ $(i = 1, \cdots, N)$ results in exactly $N - \frac{S}{GCD(D,S)}$ cache line conflicts.*

Proof: According to the mapping function in a set-associative cache, data in block addresses $B + Di$ for $i = 1, \cdots, N$ are mapped into cache locations:

$$B + Di \ mod \ S, \quad for \quad i = 1, \cdots, N, \tag{3}$$

The number of different sets that this sequence of references spans in the cache is the same as the return number defined in [10, 11]. This return number is given by $S/GCD(D, S)$. Starting from the $S/GCD(D, S) + 1$st reference, repetition starts. That is, a cache line is mapped to the set that hold a data referenced previously in the reference sequence. All $N - \frac{S}{GCD(D,S)}$ references in the sequence will similarly be mapped to previously mapped sets. Since each set contains only one cache line in the direct-mapped cache, $N - \frac{S}{GCD(D,S)}$ lines of data that are previously mapped into the $S/GCD(D, S)$ sets will be replaced. Therefore, there will be $N - \frac{S}{GCD(D,S)}$ cache line conflicts. □

An immediate extension of the above theorem is the following corollary.

Corollary 2.1 *If $GCD(D, S) = 1$, i.e. D and S are relative prime, a sequence of any S consecutive memory references to data items located in $B+Di$ $(i = 1, \cdots, N)$ is conflict-free in a S-line direct-mapped cache*

Similarly, for high degree associative cache, we have the following theorem that can be proved in the same way as above.

Theorem 2.2 *In a d-way set-associative cache with S sets, a sequence of N consecutive memory references to data items located in $B + Di$ $(i = 1, \cdots, N)$ results in exactly $N - \frac{d*S}{GCD(D,S)}$ cache line conflicts.*

2.2 Implementation

From the analysis presented in the previous subsection, we know that the larger the $GCD(D, S)$ is, the more line conflicts there will be. Therefore, to reduce conflicts, we want to minimize $GCD(D, S)$. The GCD is minimum if the two numbers are relative prime. Our main objective here is to make D and S relative prime. Notice that D is program dependent and its value is unpredictable. To increase the chance that D and S are relative prime, we make S a prime or an odd number. This is the fundamental idea behind the one's complement cache. As a result, for any given set of memory accesses the one's complement cache maximizes the number of different congruence classes and minimize the number of data items that fall into a same congruence class.

2.2.1 Line Placement

The one's complement cache 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

[1] Actually, s-bit one's complement number has the range from $-2^{(s-1)}+1$ to $2^{s-1}-1$. But sign bit has no significance

numbers. 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. 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

$$A \ modulo \ (2^s - 1); \quad for \ some \ integer \ s, \qquad (4)$$

instead of A *modulo* 2^s.

2.2.2 Identifying a Line in the Cache

Each memory address, same as 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. Same as 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 the set-associative cache and hence takes the same amount of time as 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. 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 $2^s - 1$.

2.2.3 Cache Address Calculation

For generating a cache address, the tag field and the word (offset) field are the same as that for 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 $2^s - 1$, only one's complement arithmetic operations are needed. Suppose that the binary representation of a line address A contains i s-bit subfields: $A_1, \cdots A_i$. Then reduction of A modulo $2^s - 1$ can be done very easily by noting that $2^s = 1$, which is given by

$$A \ modulo \ (2^s - 1) = \sum_{k=1}^{k=i} A_k. \qquad (5)$$

Therefore, only 1's complement 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 to the normal address calculation as evidenced in the following discussion.

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 a number modulo 15. The main reason why we call one's complement is that all arithmetic operations performed on the index fields are done in one's complement arithmetic as explained shortly.

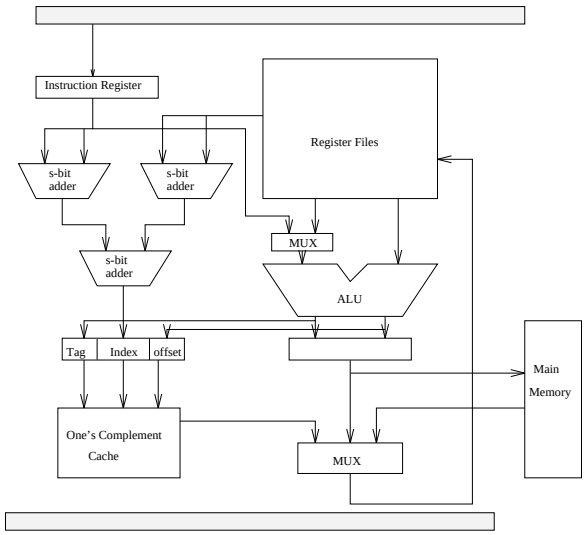

Figure 1: Block diagram of the one's complement cache design.

Figure 1 shows the block diagram of the address computation logic for a RISC processor. We take the MIPS R2000/R3000 [12] 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 an 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 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 to 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.

2.2.4 Virtual Memory System

In a processor that supports virtual memory system, 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. De-

pending 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 [13]. Some changes may be necessary in the cache design for physically addressed caches, which is one of our current research topics. In this situation, optimization of pipeline cache designs as suggested in [14] is one feasible way to go. In addition, our new design can also be applied to onchip TLB for fast address translations. It is well known that conflicts in TLB degrade system performance to a large extent. Avoiding conflicts in TLB is essential to a good processor design.

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 [15] cache simulator. 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.

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 both integer and floating point programs in the SPEC92. It has been shown in [16] that several integer and floating point programs exhibit poor cache performance. Since our main purpose here is to show that the new cache design 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 due to the time limits. 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. The selected programs are Gcc, Compress, Eqntott, Su2cor, Hydro2d, Swm256, Tomcatv and NASA7 which have high miss ratios as shown in [16].

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.

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 conventional cache and

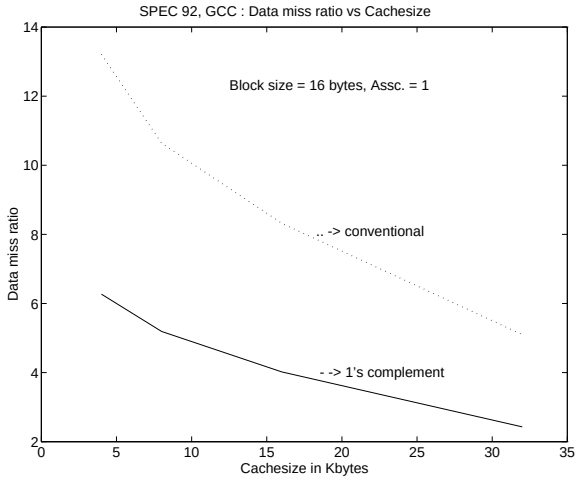

Figure 2: Data miss ratio of GCC vs cache size.

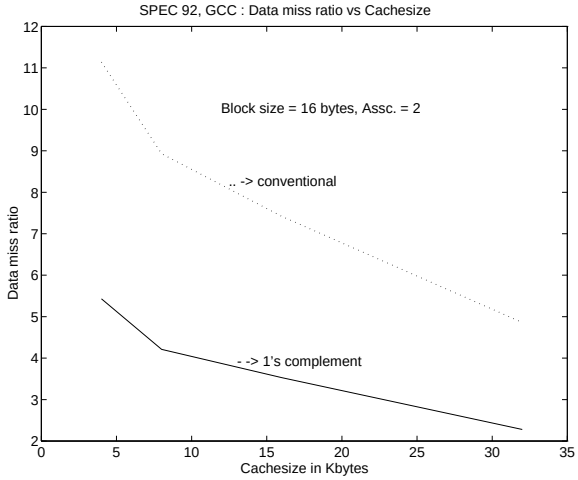

Figure 3: Data miss ratio of GCC vs cache size.

the one's complement cache. It is shown in [16] that different cache line sizes have a little effect on cache performance for this program. 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 less than half of the miss ratio of the conventional direct-mapped cache. In other words, the performance improvement of the one's complement cache over the conventional cache is more than 100%.

When we increase the degree of associativity for both caches, similar performance improvements are observed as shown in Figure 3. In this figure, the degree of associativity is increased to 2. The performance improvements are still over 100% for the cache size range considered.

Figure 4 shows the data miss ratios of program Compress which is a C program for data compression.

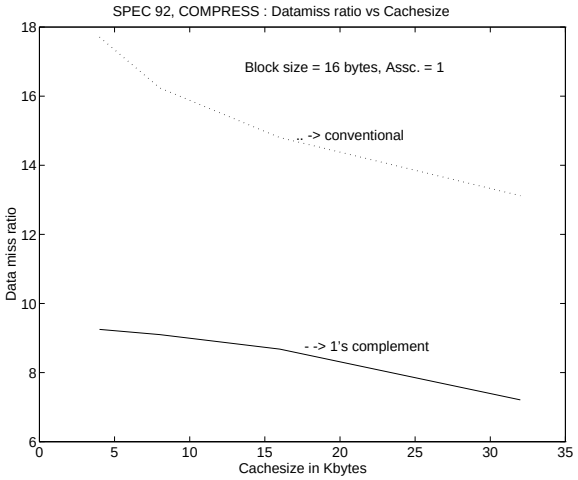

Figure 4: Miss ratio of COMPRESS vs cache size.

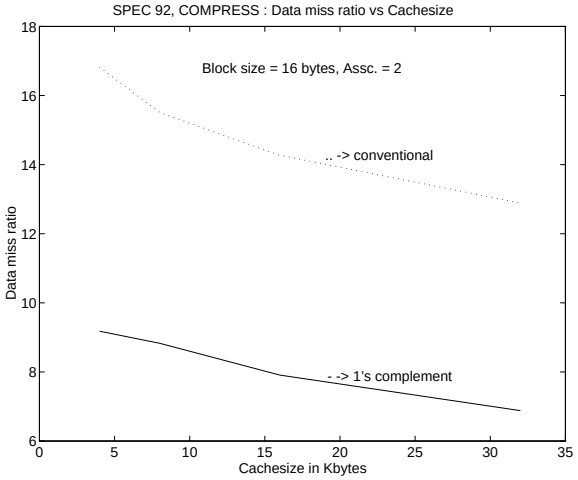

Figure 5: Miss ratio of COMPRESS vs cache size.

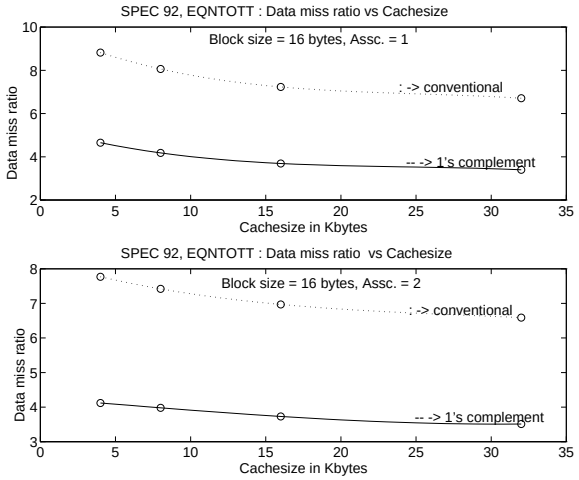

Figure 6: Miss ratio of EQNTOTT vs cache size.

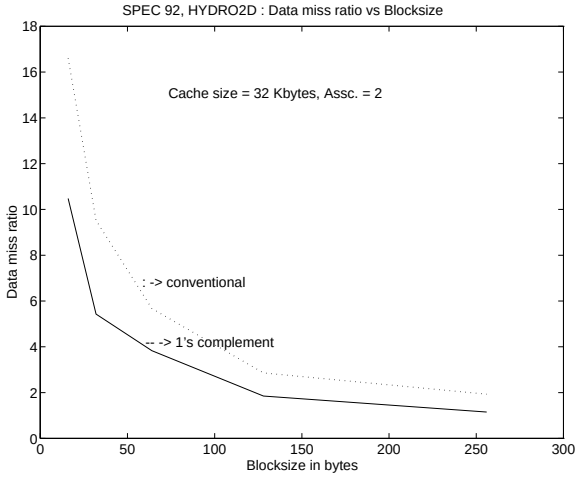

Figure 7: Miss ratio of HYDRO2D vs line size.

The miss ratios are plotted against cache sizes with line size fixed at 16 bytes and associativity 1. 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. Figure 6 plots the data miss ratio for two different degree of associativities of program Eqntott. It is shown in the figure that the one's complement cache significantly improve cache performance. The performance improvements varies from 84% to 97%.

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. [16] show that high degree of associativity does not significantly reduce 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 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 the conflicts are minimum. All above results and the results that follow show that the new cache design significantly reduces cache line conflicts from 2-way set-associative cache. Both our theoretical analysis presented in the previous sections and the experimental results presented here prove this fact.

Data miss ratios of HYDRO2D which is a floating point Fortran program are plotted in Figure 7. Previous

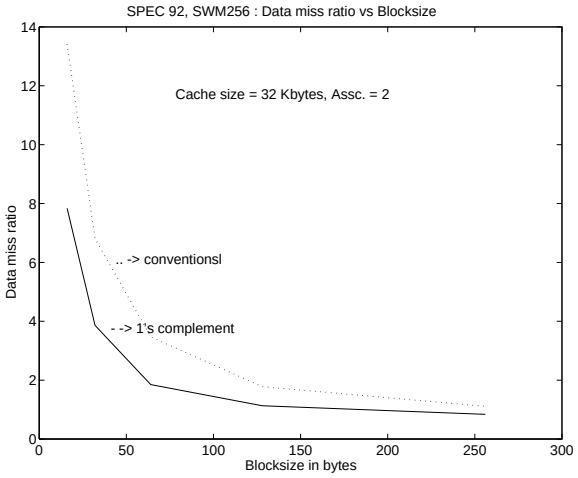

Figure 8: Miss ratio of SWM256 vs line size.

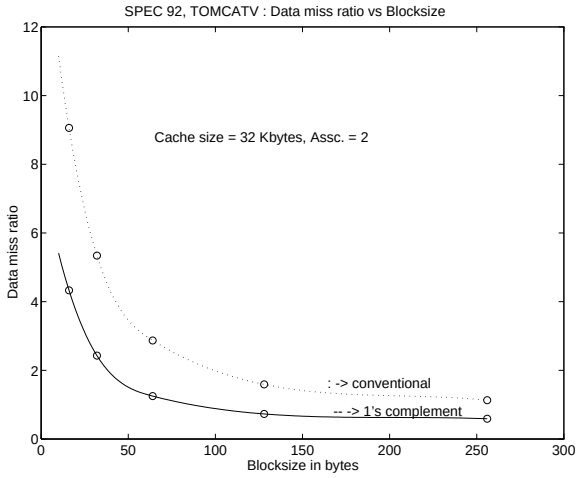

Figure 9: Miss ratio of TOMCATV vs line size.

simulation results reported in [16] indicate that cache performance of this program is more sensitive to cache line sizes than to 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 [16]. 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. [16] noted in their study that there are some anomalies in their results for the effect of associativity on miss ratio. That it, 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 ratio 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 a large GCD

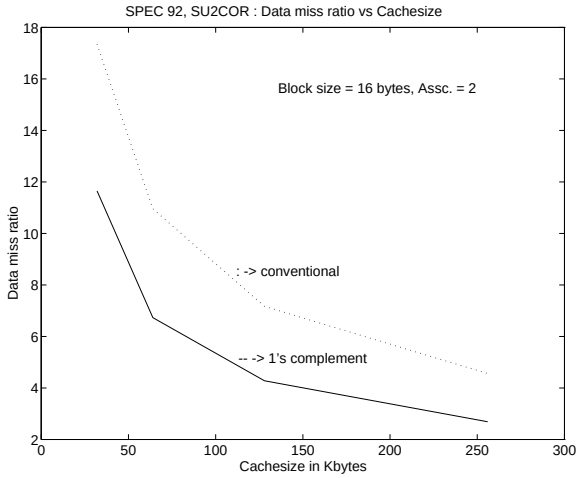

Figure 10: Miss ratio of SU2COR vs cache size.

of the set number and the address difference of consecutive data accesses as describe in Section 2 of this paper. However, the one's complement cache prevents this kind of line conflicts from occurring. The result is 90% to 130% performance improvements over 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 [16]. 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.

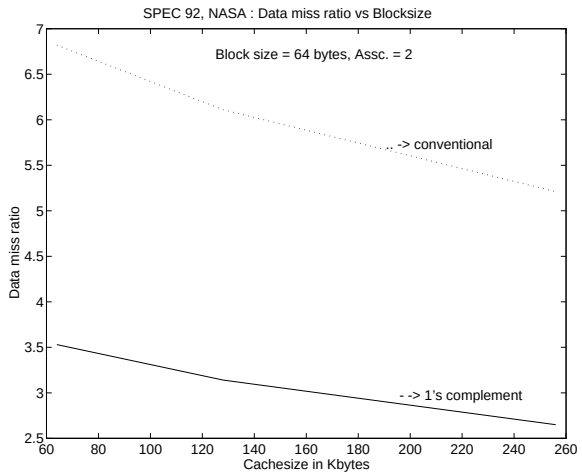

Figure 11: Data miss ratio of NASA7 vs cache size.

For program NASA7, previous research [16] 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.

5 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 the best cache design so far since it doubles the cache performance of 2-way set associative cache which has been used as a reference for evaluating conflict reductions. We have also shown in this paper that cache line conflicts of many application programs can not be reduced by increasing the degree of set-associativity. The one's complement cache, however, can solve this type of cache line conflicts. 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 cache are generally half as much as miss ratios of conventional set-associative caches.

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] J. Fang and M. Lu, "A solution of cache ping-pong problem in risc based parallel processing systems," in *Proc. of Int. Conf. on Parallel Processing*, pp. I–238–245, 1991.

[10] P. Budnik and D. J. Kuck, "Organization and use of parallel memories," *IEEE Trans. on Computers*, pp. 1566–1569, Dec. 1971.

[11] W. Oed and O. Lange, "On the effective bandwidth of interleaved memories in vector processor systems," *IEEE Trans. on Computers*, vol. C-34, pp. 949–957, Oct. 1985.

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

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

[14] K. Olukotun, T. Mudge, and R. Brown, "Performance optimization of pipelined primary caches," *19th Ann. Int'l Symp. on Computer Architectures*, pp. 181–190, May 1992. Queensland, Australia.

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

[16] 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.