A Shared-Way Set Associative On-Chip Cache

José Luis Hamkalo, Andrés Djordjalian, Bruno Cernuschi-Frías [1]

[1] Facultad de Ingeniería

Universidad de Buenos Aires and CONICET

Paseo Colón 850, (1063) Buenos Aires, ARGENTINA

Email: jhamkal@fi.uba.ar, Phone: +(5411)-4343-0891 EXT 278, FAX: +(5411)- 4331-1852

Abstract—

A new cache memory organization called "Shared-Way Set Associative" (SWSA) is described in this paper. It consists of a modified two-way set associative scheme in which one way is larger than the other. We show how better use of memory is obtained, without the costs that higher-associativities have. An expression for calculating the non-integer degree of associativity of SWSA caches is given. Several replacement policies are discussed. Miss rate statistics for the SPEC95 and additional benchmarks are presented for first and second level SWSA caches, together with a detailed analysis of conflicts using the D3C classification of misses. For large caches the miss rates of SWSA caches are similar to those 33% larger two-way set associative caches. The issue of hardware implementation is addressed, and we explain why SWSA caches may have advantages, specially with configurations with very unbalanced ways which have miss rates that are very similar to those of slightly smaller two-way caches. We conclude that shared-way set associativity shows benefits compared to two-way set associativity, and may also be favorably compared with direct-mapping and even to higher associativities depending on other architectural and technological issues.

Keywords— **Cache memory, associativity, replacement policy.**

I. INTRODUCTION

CPU caches reduce the average memory access time of computer systems by exploiting the temporal and spatial locality present in most of real reference streams [13]. The property that is most relevant for this work is temporal locality, which means that data that was recently used is very likely to be used again

in the near future. One way to make use of temporal locality is to keep in the cache the references that were most-recently used (MRU). Though this criterion is known to be inferior to the optimum unrealizable criterion [2], it exploits temporal locality very well. A fully-associative placement policy is the only one that guarantees that the least-recently used (LRU) block in the cache can be victimized when room for a new block is needed during a cache miss, thus it is the one that complies best with the aforementioned criterion. As we go from full-associativity through set-associativities of decreasing degree towards direct-mapping we are moving further away from a criterion that exploits temporal locality very well, and this explains why miss ratios increase correspondingly [14]. On the other hand, direct-mapping offers the best access times [8]. Also [8] reports increasing access times with increasing associativities. Access time is an important issue in the choice of the degree of associativity. For example, on-chip integration of the second level cache reduces the complexity and access time penalty of higher associativities. Also there is more need for the better miss rates offered by higher associativities because sizes must be reduced to fit in the die. So, it is valuable to increase the associativity if second level caches are integrated on the CPU die. It is reasonable to think that patterns like these may evolve with the advance of technology. For example, higher transistor counts per chip will allow the use of larger second-level caches, though the reduction of local miss rates produced by higher associativities diminish as size increases [4]. This may suggest the use of lower associativities increasing then the importance of access time versus local miss rate. Another possibility is the use of a third level of on-die cache, which would change the priorities of the second level. Flynn shows reasons to believe that a third level on-die will show benefits with feature sizes of 0.1 μm which will be the industry standard in a few years [3].

New architectures have been proposed to increase the cache performance. Skewed caches [17], improves hit statistics of set-associative designs with minimum penalty in access time. A multi-bank n way set-associative cache memory is built with n distinct memory banks. A block of data with base address A

may be physically mapped on physical block $f(A)$ in any of the distinct banks. This vision of a set-associative cache fits with its physical implementation: n banks of static RAM memories. The skewed caches propose a very slight modification in this design: different mapping functions are used for the distinct cache banks i.e., a block of data with base address A may be mapped on physical line $f_0(A)$ in cache bank 0 or in $f_1(A)$ in cache bank 1, etc. So data may cause conflict for a cache block on bank 0, but not on bank 1 on a skewed-associative cache, improving the hit ratio. The hardware modification incurs a over cost for implement the mapping functions and the necessary gates are placed in the critical path lengthening the hit time. Column-associativity [1] and group-associativity [14] proposes modifications to direct-mapped architectures to improve miss rates while maintaining advantageous access times. For example column-associative caches reduce the conflicts that arise in direct-mapped caches by allowing conflicting addresses to dynamically choose alternate hashing functions, so that some of the conflicting data can reside in the cache. The critical hit access path is unchanged but hits may need more than one cycle because the alternate hashing functions to search the cache are used once at a time. Also this scheme add a rehash bit to each cache set which indicates whether that set stores data that is referenced by an alternate hashing function and hardware to swap cache block content is needed. Stream buffers and victim caches [9] are additions to direct-mapped caches that decrement their miss penalties. In the victim cache the miss rate is the same of the direct-mapped cache but there are two kinds of misses, the normal ones and faster ones served by a small fully-associative cache placed between the first and the second level caches. Another proposed scheme is the difference-bit cache [10]. The difference-bit cache is a two-way set-associative cache with an access time that can be smaller than that of a conventional cache (with the actual value depending on the technology). This is achieved by noticing that the two tags for a set have to differ at least by one bit and by using this bit to select the way. In contrast with other approaches that predict the way and have two types of hits (primary of one cycle and secondary of two or more cycles), all

hits of the difference-bit cache are of one cycle. The difference-bit cache has a miss rate that is equal to a two-way set-associative cache. On the other hand, this proposal is only well suited for the two-way case. Also this organization requires a cache with virtual addresses and tags, since the bits needed would not be available in time if the addresses or tags have to be translated. Peir et al. present a summary of some of these architectures in [15].

The scheme described in this paper is another alternative. It consists of a modified two-way scheme that makes better use of memory without the additional costs that higher-associativities present. We will show how it may lead to improvements depending on the implementation.

The paper is organized as follows. Section II explains the motivation for the new scheme and why we suggest that it may lead to a more efficient use of memory than a 2-way set associative organization with a comparable complexity. The architecture is described next, together with a detailed description of some replacement policies. Miss rate statistics are presented and analyzed in Section III. Section IV discusses issues related to access times and the implementation of the proposed scheme. In section V extensions of the proposed scheme to higher associativities are given and finally in Section VI the conclusions are given.

II. Shared-Way Set Associative (SWSA) Caches

A. Motivation

Set-associative and direct-mapped caches usually retain only a subset of the N most recently used (MRU) memory blocks, where N is the number of blocks in the cache [14]. Peir et al use the term "holes" for the memory blocks that are kept by the cache but do not belong to this MRU subset and claim that frames containing holes are underutilized [14]. They provide experimental data for the average percentage of holes in caches of different associativities during the execution of TPC-C. On data caches the percentage of holes is roughly 40%, 30% and 22% for direct-mapping, 2-way and 4-way associativities respectively.

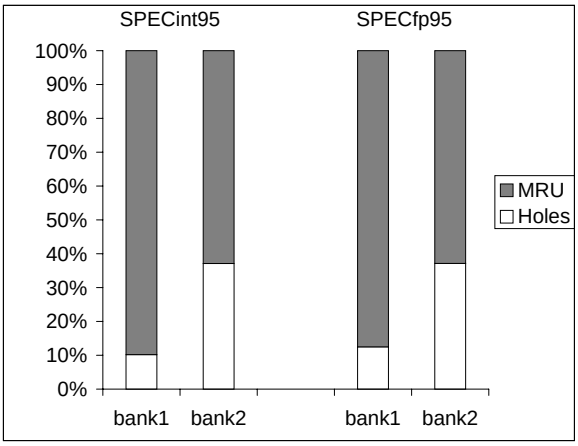


Fig. 1. Distribution of holes

Instruction caches hold fewer holes: 36%, 25% and 17%. A fully-associative LRU-replacement cache would keep no holes. These numbers illustrate what was discussed in the introduction.

For the purpose of making our point let's suppose that, in a two-way set associative cache, the MRU block of each set is kept in bank 1 (the first way) and the LRU in bank 2 (the second way). A replacement policy that does this is called "swap" [9]. We have measured the percentage of holes in each of these banks for a 16Kbytes data cache running the SPEC95 benchmarks. The results are shown in Fig. 1. The number of holes for each bank is very unbalanced as bank 2 holds a considerably larger percentage. Results for other cache sizes are similar to those shown in Fig. 1. Instruction caches under the SPECfp95 benchmarks also show a considerable unbalance. For the SPECint95 benchmarks the number of holes in bank2 is near 50%. Bank 2 is therefore less exploited than bank 1. This conclusion motivated us to evaluate the performance of a scheme similar to a two-way set associative but with bank 2 smaller than bank 1. This leads to a new cache memory organization for which new placement and replacement policies are required. We call this new cache organization "Shared-Way Set Associative" (SWSA) [7].

B. Definition of Shared Way Set Associative (SWSA) Caches

It is possible to think a two-way set associative cache of size C as the result of to cut a direct mapped cache in two equal halves of size $C/2$. The sets are formed by creating a one to one association between the lines of the two halves (see Fig. 2a). Bit-mapped indexing is usually used to access the sets. For the different case in which the direct mapped cache is cut in two parts of different sizes C_1 and C_2, an association between the lines of the two ways as shown in Fig. 2b can be used. For simplicity we assume one memory bank for each way. More than one line in the first bank is associated to only one line in the second bank. Thus different sets share one line in the second bank or second way. The "degree of sharing" is defined as the number of blocks in the first bank that share the same block in the second bank. The degree of sharing can be calculated as the ratio between the number of blocks in the first and second banks. This organization is called "Shared Way Set Associative (SWSA) Cache."

For SWSA caches the cache block selection is done by using an independent bit- mapped indexing for each bank (we use banks with sizes which are powers of two). An index 1 is used to access the bank 1. The number of bits for index 1, I_1, is given by $C_1/L = 2^{I_1}$, where L is the line size. Similarly to access bank 2 another index is used, index 2, with a number of bits given by $C_2/L = 2^{I_2}$. In figure 3 a block diagram for a SWSA cache hit logic is given. The example in figure 3 is for a case where bank 1 is twice larger than bank 2. So index 1 need one bit more than index 2 for access the bank 1 lines (i.e. the tags used for bank 1 have one bit less than those used for bank 2). In figure 3, i bits of index plus b bits of byte select are enough for the access of a data word in bank 2. For access bank 1, $i + 1$ bits of index are needed. Similarly, t bits are required for the tags used in bank 1 and $t + 1$ for the tags used in bank 2. A tag match (hit) in bank 1 or bank 2 will assert the MatchOut bit and the proper data word will be chosen with the multiplexer and placed in DataOut.

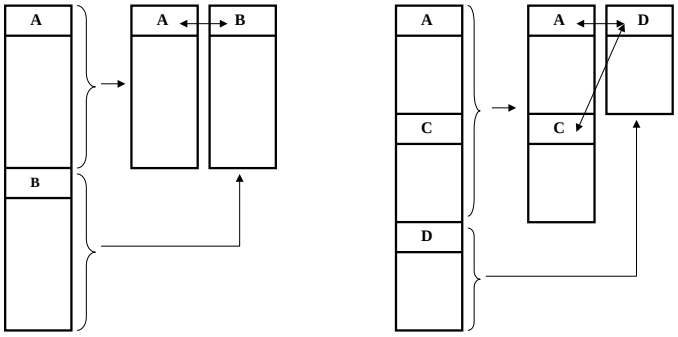


Fig. 2. a)Reassembled direct-mapped cache as a two way set associative cache. b) The proposed scheme

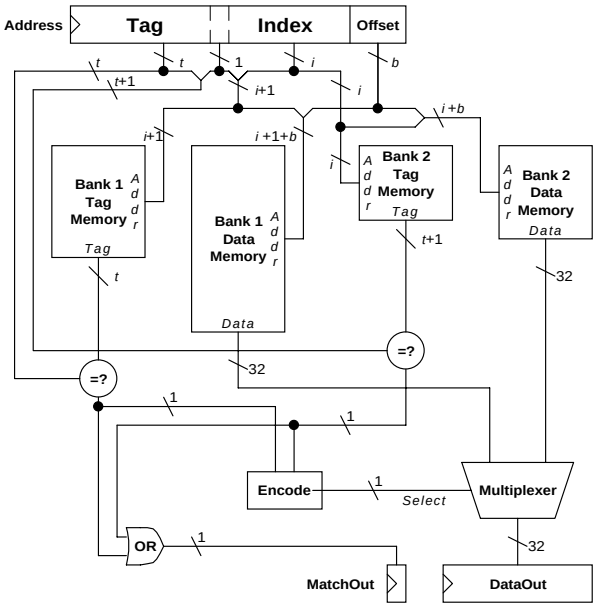


Fig. 3. Block diagram for a SWSA cache hit logic

C. *Degree of Associativity of SWSA Caches*

The degree of associativity n for a standard n-way set associative cache is obtained as the quotient between the total number of blocks (lines) in the cache B and the number of sets in the cache S:

$$n = \frac{B}{S} \tag{1}$$

For a cache with a total cache size C and block size L, is $B = \frac{C}{L}$ and considered in (1) is

$$n = \frac{C}{LS} \tag{2}$$

We also will use (2) to obtain an operational definition to calculate the degree of associativity of a SWSA cache. The total cache size in a SWSA cache is $C = C_1 + C_2$, where C_1 and C_2 are the sizes of banks 1 and 2 respectively. In a SWSA cache organization, like in a standard set associative cache, the number of sets in the cache coincide with the number of blocks in the first bank and then is $S = C_1/L$. Then, from (2) the associativity n for a SWSA cache is:

$$n = \frac{C}{LS} = \frac{(C_1 + C_2)}{\frac{LC_1}{L}} = \frac{(C_1 + C_2)}{C_1} = 1 + \frac{C_2}{C_1} \tag{3}$$

Using (3), the degree of associativity for a SWSA cache can be easily calculated. For example if bank 1 has a size of 16Kbytes and bank 2 has a size 8kbytes then the SWSA cache has a total size of 24Kbytes and from (3) its degree of associativity is $n = 1.5$. Standard two-way set associative and direct-mapped caches are special cases of SWSA caches, and equation (3) reflect this observation. In a two-way set associative cache is $C_1 = C_2 = \frac{C}{2}$ (the ways in the second bank are not shared) and from (3) is $n = 2$. For the case of a direct mapped cache bank 2 does not exist and thus is $C_2 = 0$, and from (3) it results $n = 1$.

D. Replacement Policies

With shared-way set associativity the structure of the sets is redefined so new replacement policies are needed. This section describes three possibilities. For the first one, the criterion of holding the MRU blocks that can be held is followed exactly. The other two policies are simplifications of the previous one. The simulations presented in Section III were performed with these two simpler policies.

D.1 A LRU Replacement Policy with Reallocation

When a miss happens, there are only three blocks that may be victimized if we are to comply with the shared-way set associativity:

1. The block that was indexed in the first bank (block 1).
2. The block that was indexed in the second bank (block 2).
3. The block that occupies the place that would be occupied by the block that was indexed in the second bank if it was moved to the first (block 3).

For example, if blocks A and D were checked in Fig. 2b, and block D mapped to the place occupied by block C, then the candidates for replacement would be A, D and C correspondingly. Note that the first and third block could be the same, resulting in only two candidates. This would be the case for the last example if block D mapped to the place occupied by block A. Our first replacement policy chooses the LRU of these three candidates. To make this possible, blocks need to be moved from one bank to the other, as well as storage and maintenance of information is needed to find out which block was LRU.

A policy that satisfies these requirements, which we name “LRU replacement policy with reallocation,” is:

Hit: The requested item is in block 1 or block 2; only the temporal information is updated.

Miss:

a) If the candidates for replacement are only block 1 and block 2 then the LRU of these two blocks is replaced.

b) If not, then:

b1) If block 3 is the LRU then it is replaced with block 2 (reallocation) and the new incoming block is placed in bank 2.

b2) If block 3 is not the LRU then replacement is done as in case a).

Temporal information is updated either for cases a) or b).

D.2 The Swap Replacement Policy

There are two requirements that complicate the previous replacement policy: movement of blocks within banks and storage and maintenance of temporal information. We can partially drop the latter with an expectable small degradation of performance if we use a swap replacement policy:

Hit: If the hit is in the first bank the contents of the cache do not change. If it is in the second bank then block 1 and 2 are swapped, so as to maintain the LRU block within this pair always in the second bank.

Miss: Block 2 is victimized. Block 1 is moved to the second bank and the new block is placed in the first bank. Again, the LRU of the two blocks is placed in the second bank.

This policy has the useful property of inclusion. All the references that hit on a direct-mapped cache (DM) of size $C_{\text{DM}} = C_1$, also hit on a SWSA cache of size $C_{\text{SWSA}} = C_1 + C_2$ if it uses the swap replacement policy, because this policy guarantees that at any given time the DM cache and the C_1 bank in the SWSA cache have exactly the same contents, and thus, the second bank C_2 in the SWSA cache gives the possibility of additional hits for the SWSA cache. A similar analysis can be made when two SWSA caches of different associativities (sizes) are compared. Consider two SWSA caches, SWSA and SWSA',

of size $C_{\text{SWSA}} = C_1 + C_2$ and $C'_{\text{SWSA}} = C_1 + C'_2$ respectively, with $C'_2 > C_2$. As with the previous example, the C_1 banks for the two caches have exactly the same contents. The references that cannot be contained in the C_1 banks are directed to the C_2 and C'_2 banks. From the C_2 and C'_2 banks point of view, they are direct mapped caches of sizes C_2 and C'_2 that are receiving the same pattern of memory requests. Because direct mapped caches have the property of inclusion, all the hits in the C_2 bank are included in the C'_2 bank ($C'_2 > C_2$). So all the references that hit in the SWSA cache also hit in the SWSA' cache. SWSA caches with successively greater C_2 banks (spanning from a DM cache of size C_1 with $C_2 = 0$, to a two-way set associative cache of size $2C_1$ with $C_2 = C_1$) always have lower or equal miss rates as a consequence of the property of inclusion.

D.3 A LRU Replacement Policy

A different simplification of the LRU with reallocation policy is made by dropping the first requirement of movement of blocks within banks. By doing so the second requirement is simplified because no comparisons are needed between blocks in the first bank. One LRU bit for each block in the first bank is enough to compare those blocks with the corresponding block in the second. This leads to an easy to implement policy that we call "LRU replacement policy":

Hit: If the requested item is in block 1 the corresponding LRU bit is set to one. If it is in block 2 all the LRU bits associated to it are set to zero.

Miss: If the corresponding LRU bit is set to zero then block 1 is replaced and its LRU bit is set to one. If it was set to one then block 2 is replaced and all the LRU bits associated to it are set to zero.

A final remark is that the three policies that were considered converge to the standard LRU replacement policy for the particular case of two-way set associativity.

III. HIT/MISS PERFORMANCE EVALUATION

A. Methodology

For the evaluation of SWSA L1 and L2 caches, the SPECint95 and SPECfp95 benchmarks were used with trace driven simulations. All the eight integer benchmarks and ten floating point benchmarks were simulated. With the integer benchmarks a mean of 700 million instructions by benchmark were simulated, noting that the integer traces were collected after the execution of the first 500 million load/store instructions. With the floating point benchmarks a mean of 1200 million instruction by benchmark were simulated, this time starting from the beginning of the benchmarks. Traces were collected with the ATOM tool [19]. A line size of 32 bytes and the swap replacement policy were used. Later in this section the LRU replacement policy is also considered.

Mean miss rates obtained for instruction and data SWSA L1 caches (SW) are shown in Fig. 4. Each graph has three small SW curves; these curves are formed with the points corresponding to associativities equal to $1\frac{1}{32}$, $1\frac{1}{16}$, $1\frac{1}{8}$, $1\frac{1}{4}$ and $1\frac{1}{2}$ in this order. For example, the first point with size greater than a power of two has an associativity of $1\frac{1}{32}$, the next one of $1\frac{1}{16}$, and so on. Curves for direct-mapped (DM), two-way set associative (2W), four-way set associative (4W) and fully-associative (FA) LRU caches are also plotted for comparisons purposes.

B. Analysis of the Results

The addition of a small second bank of size C_2 to a direct-mapped cache of size C_1 results in a significant reduction of miss rates as it may be seen in Fig. 4 by comparing the points with associativities $1\frac{1}{32}$ to those of DM caches. As the size of the second bank C_2 increases the miss rates diminish with a very sharp tendency. It can be observed that the curves for SWSA caches intersect the 2W tendency curves near the points corresponding to associativities of $1\frac{1}{16}$ and $1\frac{1}{8}$, indicating that SWSA caches with those asso-

ciativities roughly attain the level of performance of two-way set associative caches of comparable size. The associativities of those caches are very low. Note that the degree of sharing is 16 and 8 respectively. Points with associativities of $1\frac{1}{4}$ and $1\frac{1}{2}$ are generally under the two-way tendency curve. It is interesting to observe that, for large sizes, caches with an associativity $1\frac{1}{2}$ have miss rates that are very close to those of two-way set associative caches of significantly greater size. For example in Fig. 4, caches with sizes of 48K ($n = 1.5$) and 64k ($n = 2$) have very similar miss rates although the two-way cache is 33% larger.

For standard LRU set associative caches the property of inclusion arise when a given cache memory increase his size (usually duplicating the number of sets) and then all the references that hit in the original cache hits in the bigger cache. This observation applies to SWSA cache too, and then bigger SWSA caches of a given associability have lower or equal miss rate statistics. SWSA caches offer the possibility of increase the total cache size by increase the associativity. For example for a given SWSA cache of size $C = C_1 + C_2$, a bigger cache if obtained if only C_2 is increased to C_2' and C_1 remains constant. This increased SWSA cache has a greater degree of associativity than the original given SWSA cache according to expression 3. Focusing in these cases, i.e. a SWSA cache with a bank 1 of constant size C_1 and a bank 2 of variable size from $C_2 = 0$ (DM cache) up to $C_2 = C_1$ (two-way set associative cache), SWSA caches with the swap replacement policy have the property of inclusion. So independently of the accesses pattern SWSA caches of increasing associativities has better or equal hit rate statistics.

C. Analysis Using the D3C Model

In this section the performance of SWSA caches is analyzed using the D3C model [5]. With this model a non-compulsory miss is categorized as a conflict miss if the LRU distance of the reference that produced it is less than or equal to the number of blocks in the cache. If the LRU distance is greater than the number of blocks in the cache then the miss is a capacity miss. Conflict misses for SWSA (SW) caches

are plotted in Fig. 5. Direct-mapped (DM) and two-way set associative (2W) caches are also considered for comparison purposes. Compulsory misses are the same for all configurations (and close to zero) so they are not plotted; neither are capacity misses as they are nearly coincident.

From Fig. 5 the following observations can be done. SWSA caches with high degrees of sharing, such as 32 or 16, have more conflicts than two-way set associative caches of comparable size for instructions under both integer and floating point benchmarks, and for data under integer benchmarks, as it can be seen by comparing the first two points of each SW curve with the closest points of the 2W curves. However, with degrees of sharing of 4 or 2 they perform better than two-way. This behavior is more noticeable with instruction caches. For small data caches under the floating point benchmarks, shared-way set associativity performs better than two-way set associativity, and their performance is roughly equal for large caches. An interesting observation is that for instruction and data caches a 1.5 SWSA cache often has the same conflict miss rate than the next two-way set associative cache, which is 33% larger. This means that enlarging the second bank of a 1.5-way SWSA cache to convert it in a two-way set-associative cache of greater size usually does not result in better solved conflicts, so this increment in size only produces the benefit of a reduction of the capacity misses. Thus when there are no more capacity misses, or when they remain constant as the cache increases (this may happen because nearly all these misses have LRU distances that are much greater than the cache sizes that are being considered), 1.5 SWSA caches have the same miss ratio as two-way set associative caches that are 33% larger.

D. Miss Ratios with the LRU Replacement Policy.

SWSA caches using a LRU replacement policy as described in Section II were also simulated. The miss rates obtained for instruction and data caches under SPEC95 benchmarks were slightly better or equal than those for the swap replacement policy, so these results are not plotted and it may be assumed that for

L1 SWSA caches the swap and LRU replacement policies produce almost the same miss rate statistics. These experimental results make SWSA caches with the LRU replacement policy very interesting due to the combination of very good miss rate statistics and ease of implementation.

E. Analysis using the simple loop model

In our methodology to evaluate SWSA caches, we constrain the indexing functions used to accesses the cache banks to bit mapped functions. This bit mapped indexing provides the easiest implementation and the minimum access times to the banks. Bit mapped indexing leads to power of two memory bank sizes (cache blocks sizes are assumed to be power of two). Because SWSA cache use banks of different size, it's not possible to obtain a SWSA cache of power of two total cache size. Then, with bit mapped indexing its not possible to compare direct mapped or set associative caches with SWSA caches of the same size. The use of non power of two memory banks (i.e. no bit mapped indexing) leads to greater hardware complexity, longer cache access times and potentially better miss rates for typical workloads. Better miss rates are possible because the no bit mapped indexing function may cause an effect similar to the produced by the skewed functions used in skewed cache memories.

In order to provide results for a simulation that use DM, 2W and SWSA caches of the same size, we used a theoretical memory access model as the workload: the Simple Loop Model. The simple loop model has been proposed and analyzed in [18] and further analyzed in [6]. This model reference a contiguous zone of memory, with a stride between the references of exactly one cache block, except for the last reference that jumps backwards in memory to close the loop. The number of iterations of the loop tend to infinite. The simple loop model reference memory very orderly, so the cache sets are filled very orderly too. this fill pattern is not altered by the choice of a non power of two number of sets (as opposite to a real workload where the cache memory could be beneficed from a hashing of memory blocks over the cache sets). In

the next example we compare DM, 1.5 SWSA and 2W LRU caches of 12Kbytes, 24Kbytes and 48Kbytes, using a loop of 30Kbytes. The results are given in table I.

TABLE I

MISSES FOR DM, SWSA AND 2W CACHES UNDER THE SIMPLE LOOP MODEL.

	DM	SWSA	2W
$C = 12Kbytes$	100%	100%	100%
$C = 24Kbytes$	40%	53%	60%
$C = 48Kbytes$	0%	0%	0%

From table I, for 12 Kbytes caches the 30 Kbytes loop is big enough to produce trashing in all the sets, so the miss rate is 100% for the three organizations. For 48 Kbytes caches, the caches are bigger than the loop. After the first turn of the loop, the three organizations can retain completely the loop and then the miss is tend to 0%. The 24 Kbytes caches are an intermediate case where the behavior difference between the three organization arise. The lowest miss rate is obtained for the DM cache ($miss\ rate_{\ DM} = 40\%$), next de SWSA cache ($miss\ rate_{\ SWSA} = 53\%$) and finally the worst miss rate is for the 2W cache ($miss\ rate_{\ 2W} = 60\%$). For standard set associative caches and the simple loop model it is known that miss rates are worst as the cache is more associative [18], [6]. The results given in table I agree with the last and also shows that the 1.5 24kbytes SWSA cache, gives a miss rate worst than a DM cache and better than a 2W cache. This results agree with the definition for the associativity for SWSA caches given in this work in the sense that SWSA caches are more associative than DM caches and less than 2W caches. Although the simple loop model is a theoretical model, match the memory access patterns of some real programs that use loops to access instructions or big data structures and then is useful to support generality for our proposed organization.

F. More simulation results

In order to support generality of our proposed organization, we provide results for other benchmarks with different characteristics from the used above. We used the set of Java benchmarks from the University of Wisconsin [12]. All the set of nine benchmarks were simulated using the traces available in [12]. Mean miss rates were obtained for LRU SWSA L1 data caches. Direct mapped, two way and four way set associative caches were simulated for comparison purposes. The results are shown in figure 6. Figure 6 shows results of similar characteristics than the obtained for the SPEC95 benchmarks, i.e. the miss rates for SWSA caches usually are under the interpolation line for 2W miss rate curves. Then, the analysis made in section B holds for the Java benchmarks and confirm the way SWSA caches perform as compared to direct mapped and standard set associative caches.

G. Miss Ratios of Second-Level SWSA Caches

SWSA L2 caches were also evaluated under the SPECint95 and SPECfp95 benchmarks in the way indicated in Section III. Two 16Kbytes 2-way set associative caches with LRU replacement policy and 32 byte lines were used as instruction and data L1 caches. The L2 caches under study used the swap replacement policy. Mean miss rates were obtained for the integer and floating point benchmarks are shown in Fig. 7. It can be observed form fig. 7 that the 768 kbytes SWSA L2 cache performs as a one megabyte cache for the integer benchmarks and only slightly worse for the floating point benchmarks. The 640 kbytes cache is also very interesting as it shows miss rates very close to the one megabyte cache, which is 56% larger.

IV. Implementation Issues

Shared-way set associativity makes better use of memory than two-way associativity. This leads to interesting hit/miss statistics as it was shown in previous sections, but associativities higher than two provide even better results. Shared-way set associativity may offer a speed benefit compared to conventional architectures only if its access times are better than those of associativities higher than two. To analyze this issue it is relevant to know why higher associativities increase access times. Running an extended version of CACTI [20][16], with different configurations of associativities between two and eight, showed that more than 90% of the increase in access time is due to the delay of the data output multiplexor, which raises because it has a number of inputs that is proportional to the associativity. The slightly larger tags also augment the delay of the tag array wordlines and comparators but this difference is almost insignificant. Shared-way set associativity does not add inputs to the multiplexor, so this suggests that the access time penalty of associativities higher than two is not present. Thus, shared-way set associativity could result in better overall speeds when compared to traditional architectures. SWSA caches are better appreciated if we consider that some architectures may work with no more than two ways. An example is the bit-difference circuitry referenced earlier [10]. Kulkarni et al. showed that it reduces data access time for an associativity of two but not for greater associativities [11]. Hence, if a design uses a difference-bit scheme, shared-way set associative architectures are interesting alternatives.

The access of data for each way is generally done in parallel, and selecting which will drive the output, based on the comparison of tags, is usually the critical timing path. In SWSA caches this selection can be done using only the tag comparison for the second bank, by choosing the data from this bank if there was a match, or from the first bank if there was not. Smaller memories are faster, so removing the larger tag memory from the critical path could improve the access time. This result is even more important if we consider that CPUs not necessarily use the data output and the hit signal simultaneously. If data was

available before the hit signal it could be convenient to let the CPU begin execution with that data and later roll back if there was not a hit, because it would have remained in an idle state anyway. Some of the alternatives to conventional organizations that were mentioned in the introduction rely on this kind of predictive execution. SWSA caches with a high degree of sharing would be advantageous if this scheme is adopted. Also, if an address hits in the faster tag way, it is convenient to terminate the lookup in the slower way, with the consequent saving in power consumption.

Another way to exploit the unbalance of the sizes of the tag banks with second-level caches is to access the tags of the second way in parallel with the access of the first level cache. So, if the second level cache must be accessed, only one data bank would have to be searched, leading to significant power savings. This would most surely require multiporting the tag bank of the second way and having much more accesses on it, but shared-way set associativity would alleviate this problem, because the second way would be considerably smaller.

V. Extension of SWSA caches to higher associativities.

In this section a SWSA cache that uses more than two ways is described. Figure 8 shows a scheme for a SWSA cache that uses four ways, corresponding one memory bank for each way. Each bank is allowed to be of size less or equal than the previous one, starting from the left with a bank of a given size. The banks must be restricted to a power of two size, so that bit mapped indexing may be used. Given a memory address issued by the processor, the selected set is conformed using a bit map independently for each bank. Let C_j be the size of bank j and L the block size, an index is used to access each bank, the number of bits for index j, I_j, is given by $\frac{C_j}{L} = 2^{I_j}$. The example in fig. 8 shows the case where bank 1 is twice bank 2 and bank 2 is twice bank 3 and 4. So if a given number of bits I_1 is necessary for index 1, I_2 needs one bit less than index 1 to access the bank 2 lines (i.e. the tags used for bank 2 have one bit more than the tags

used for bank 1). Similarly, I_3 and I_4 need two bit less than index 1 to access the bank 3 and bank 4 blocks (i.e. the tags used for banks 3 and 4 have two bits more than the tags used for bank 1).

When more than two ways are used in a SWSA cache, it is difficult to characterize the cache from a single degree of associativity and the total cache size. A n-way SWSA cache is characterized with a tuple of numbers and a given total cache size or bank 1 cache size as follow: each number in the tuple gives the size ratio between the corresponding cache memory bank and the first bank. For example a SWSA cache with four banks of sizes 8Kbytes, 4Kbytes, 2Kbytes, and 2Kbytes respectively is characterized by the 4-tuple $1\frac{1}{2}\frac{1}{4}\frac{1}{4}$. As another example, for a standard 4 way set associative cache the corresponding tuple is 1111. The description given in this section shows how a SWSA cache can be extended using more than two ways. A detailed analysis must be done in order to evaluate the performance of these extended schemes and will be addressed in future work.

VI. Conclusions

Shared-Way Set Associativity is an organization that makes a more efficient use of memory than conventional two-way set associativity without the additional costs that higher associativities have. It provides new alternatives for cache organization that may offer benefits depending, within other factors, on the state of technology. Shared-way set associativity differs from other approaches in that it enables the use of bit-mapped indexing on caches of total sizes that are not powers of two without having to add ways. This gives a degree of freedom to obtain area-efficiency.

The main characteristic of SWSA caches is that one way is larger than the other. Each line of the smaller way is thus shared between several of the larger. There are then alternatives, each corresponding to a different degree of sharing, that span from two-way (degree of sharing equal to one) to direct-mapped (no second way). The degree of associativity of SWSA caches calculated with the given formula reflects

this fact as it ranges from 2 to 1 correspondingly.

We performed simulations with traces extracted from SPEC95 benchmarks using two replacement policies (swap and LRU). One of them (LRU), is very easy to implement. The results for configurations with associativities from 1.125 to 1.5 were generally better than the interpolations of the results for two-way. It is interesting to note that for large caches the miss rates of 1.5-way were similar to those of 33% larger two-way caches. In order to support generality we also give additional simulation results using Java traces and one theoretical memory access model. The issue of access time was addressed, and we discussed that SWSA caches may have advantages, specially with configurations with a high degree of sharing. These have miss rates that are very similar to those of slightly smaller two-way caches, so benefits in access time would result in better overall speeds. Simulations for second level caches were also done and the conclusions derived from the results are the same as those for first level caches.

SWSA cache architectures do not require any elaborate new hardware. Nevertheless, SWSA caches use the same hardware scheme used for standard set associative caches, but implementing ways of different size. Rather than produce overheads, the introduced asymmetry leads to better cache access time and lower power consumption.

Shared-way set associativity shows benefits when compared to two-way set associativity, and also when compared to direct-mapping depending on other architectural issues. This suggests their use as first-level caches. The evolution of technology, specially per chip higher transistor counts and integration of third-level caches, may lead to a tendency towards lower associativities for second level caches, eventually making shared-way set associative caches an attractive possibility.

ACKNOWLEDGMENTS

We gratefully acknowledge Professor Daniel Etiemble from LRI, Université de Paris-Sud, and Professor Alberto Dams from the Universidad de Buenos Aires. This work was partially financed by the Universidad de Buenos Aires, grants No. TI-09 and I-025, and the Consejo Nacional de Investigaciones Científicas y Técnicas , CONICET, grant PIP-4030.

REFERENCES

[1] A. Agarwal, S. D. Pudar, "Column-Associative Caches: a Technique for Reducing the Miss Rate of Direct-Mapped Caches," *Proc. of the 20th ISCA,* pp. 179-180, 1993.

[2] L. A. Belady, "A Study of Replacement Algorithms for a Virtual-Storage Computer," *IBM Systems Journal*, vol. 5, no. 2, 1966.

[3] M. J. Flynn, "What's Ahead in Computer Design?" *Euromicro'97 Proc.*, Budapest, pp. 4-9, Sep. 1997.

[4] J. D. Gee, M. D. Hill, D. N. Pnevmatikatos and A. J. Smith, "Cache Performance of the SPEC92 Benchmark Suite," *IEEE Micro,* vol. 13 no. 4, pp. 17-27, Aug. 1993.

[5] J. L. Hamkalo, B. Cernuschi-Frías, "A Taxonomy for Cache Memory Misses," *Proc. of the 11th Symposium on Computer Architecture and High Performance Computing*, Natal, Brazil, pp. 67-73, 1999.

[6] J. L. Hamakalo, B. Cernuschi-Frías, "Theoretical Analysis of Cache Statistics for the Simple Loop Model," *International Journal of Computers & Applications*, v.21, n.1, p.13-18, 1999.

[7] J. L. Hamkalo, A. Djordjalian, B. Cernuschi-Frías, "A Shared-Way Set Associative Architecture for On-Chip Caches," *Proc. of the ISCA 16th International Conference on Computers and Their Applications*, Seattle, Washington USA, pp. 125-128, March 28-30, 2001.

[8] M. D. Hill, "A Case for Direct-Mapped Caches," *IEEE Computer*, vol. 21 no. 12, pp. 25-40, Dec. 1988.

[9] P. N. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers," *Proc. of the 17th ISCA*, pp. 364-373, 1990

[10] T. Juan, T. Lang and J. J. Navarro, "The Difference-bit Cache," *Proc. of the 23th ISCA*, pp. 114-120, 1996.

[11] A. Kulkarni, N. Chander, S. Pillai and L. John, “Modeling and Analysis of The Difference-Bit Cache,” *Proc. of the Great Lakes Symposium on VLSI’98,* Lafayette, Louisiana, Feb. 1998

[12] M. M. Martin and M. Plakal, “On-line Java Data Traces”, http://www.cs.wisc.edu/ plakal/javatraces/overview.html.

[13] D. A. Patterson, J. L. Hennessy, *Computer Architecture, A Quantitative Approach.* San Mateo, California: Morgan Kaufmann Publishers, 1995.

[14] J. K. Peir, Y. Lee and W. W. Hsu, “Capturing Dynamic Memory Reference Behavior with Adaptive Cache Topology,” *Proc. of the ASPLOS VIII*, San Jose, California, pp. 240-250, 1998.

[15] J. K. Peir, W. W. Hsu and A. J. Smith, “Functional Implementation Techniques for CPU Cache Memories,” *IEEE Trans. on Computers*, vol. 48 no. 2, pp. 100-110, Feb. 1999.

[16] G. Reinman and N. P. Jouppi, “Extensions to CACTI,” http://research.compaq.com/wrl/people/jouppi/CACTI2.html.

[17] A. Seznec, “A Case for a Two-Way Skewed-Associative Cache,” *Proc. of the 20th ISCA*, pp. 169-178, 1993.

[18] J. E. Smith, and J. R. Goodman, “Instruction Cache Replacement Policies and Organizations,” *IEEE Trans. on Computer, C-34, (3)*, 1985, 234-241.

[19] A. Srivastava, A. Eustace, “ATOM: A System for Building Customized Program Analysis Tools,” *Proc. of the 1994 ACM Conf. on Prog. Lang. Design and Implementation (PLDI)*, pp. 196-205, 1994.

[20] S. J. E. Wilton and N. P. Jouppi, “An Enhanced Access and Cycle Time Model for On-Chip Caches,” Research Report 93/5, DEC Western Research Laboratory, July 1994.

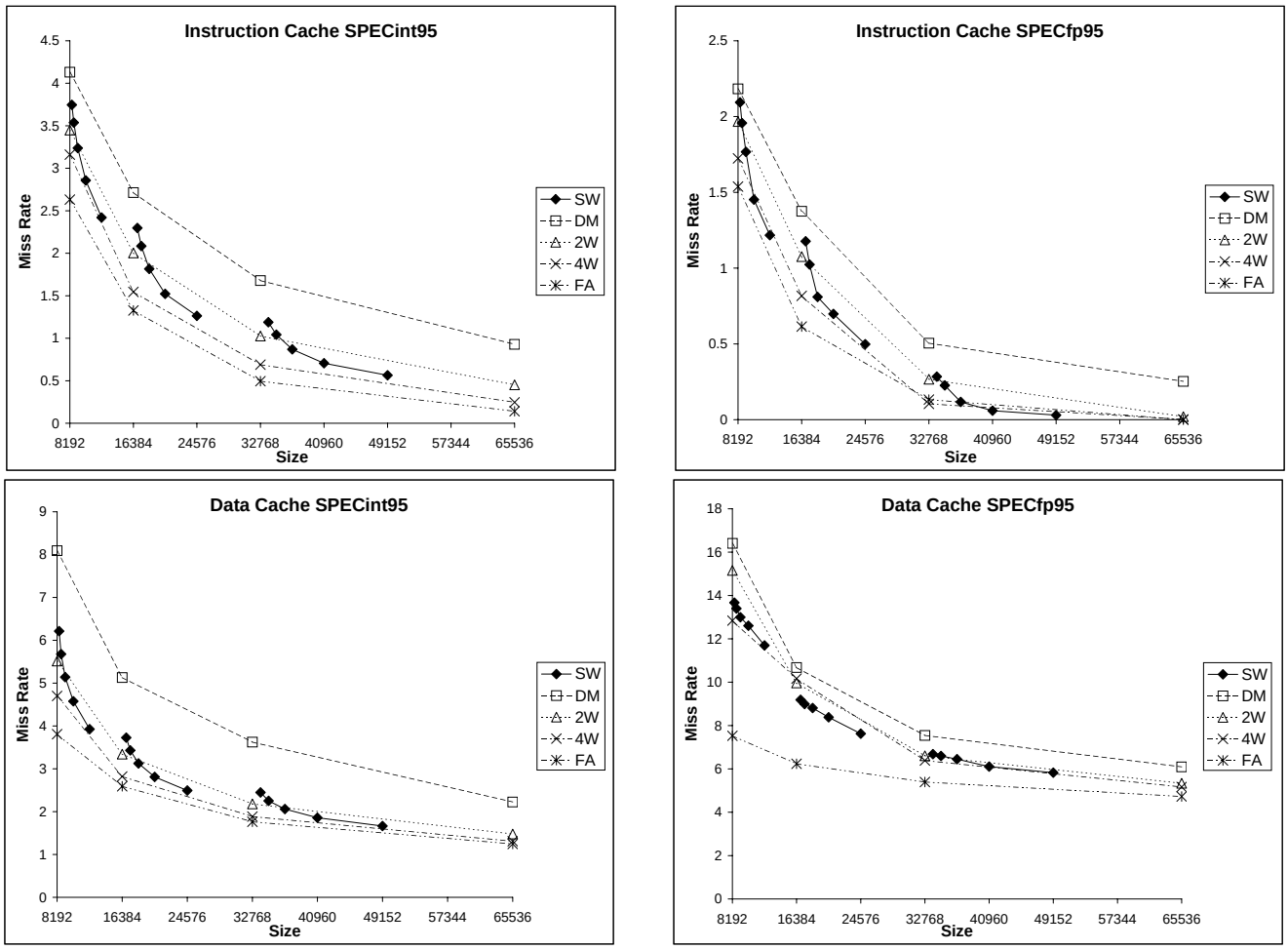


Fig. 4: Mean miss rate of SWSA (SW) L1 instruction and data caches on SPEC95 benchmarks. Curves for direct mapped (DM), two-way set associative (2W), four-way set associative (4W), and fully-associative (FA) caches are plotted for comparison purposes. Each SW curve has five points corresponding to associativities $1\frac{1}{32}$, $1\frac{1}{16}$, $1\frac{1}{8}$, $1\frac{1}{4}$ and $1\frac{1}{2}$.

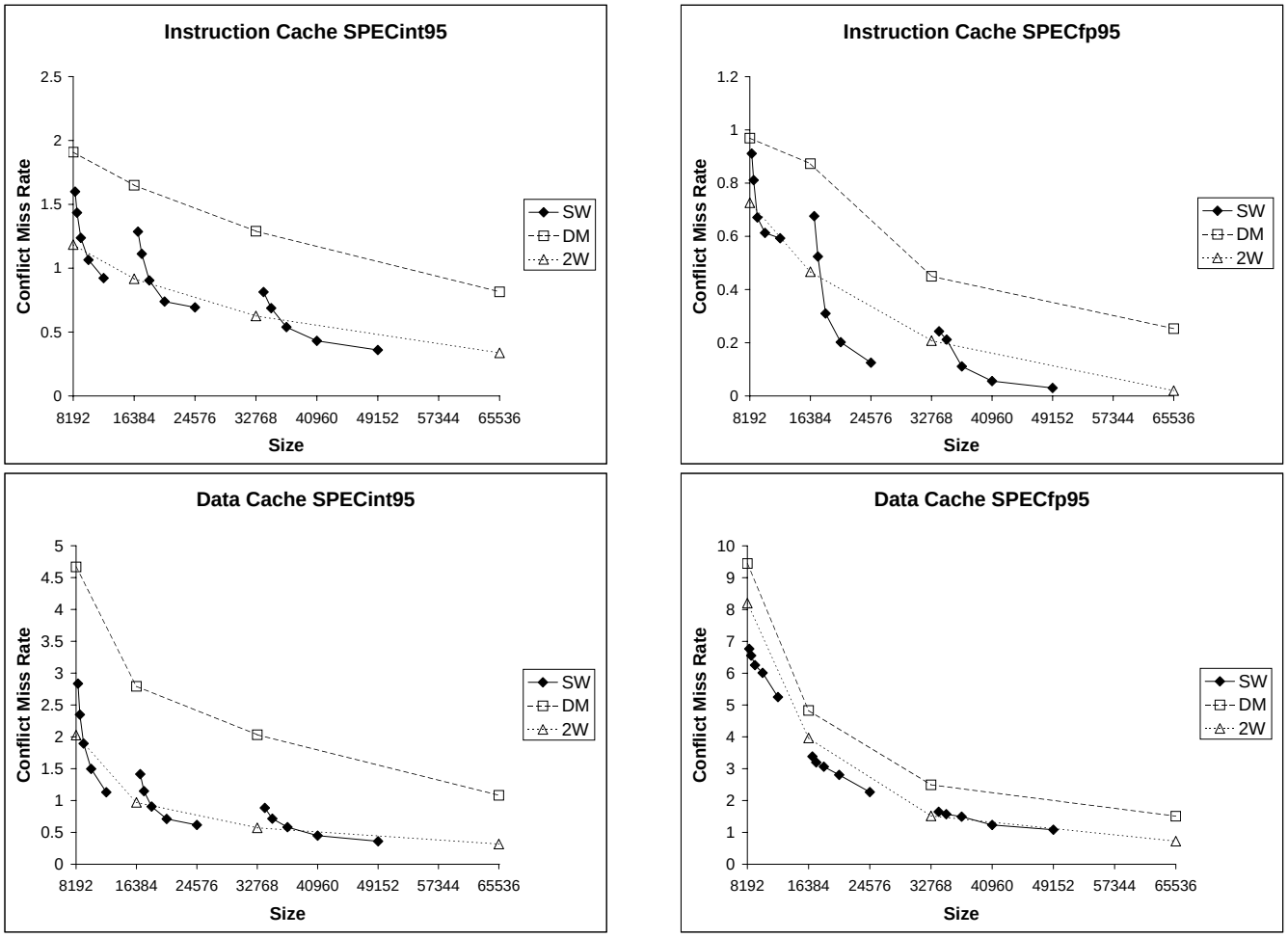


Fig. 5: Mean conflict miss rates for the SPEC95 benchmarks for Shared Way Set Associative (SW) L1 instruction and data caches. Direct mapped (DM) and two-way set associative (2W) cache curves are plotted for comparison purposes. Each contiguous curve for the SW curve has five points corresponding to associativities $1\frac{1}{32}$, $1\frac{1}{16}$, $1\frac{1}{8}$, $1\frac{1}{4}$ and $1\frac{1}{2}$.

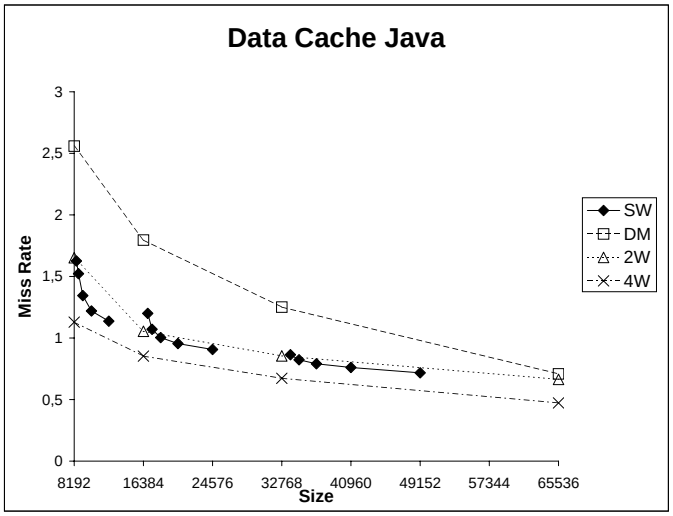
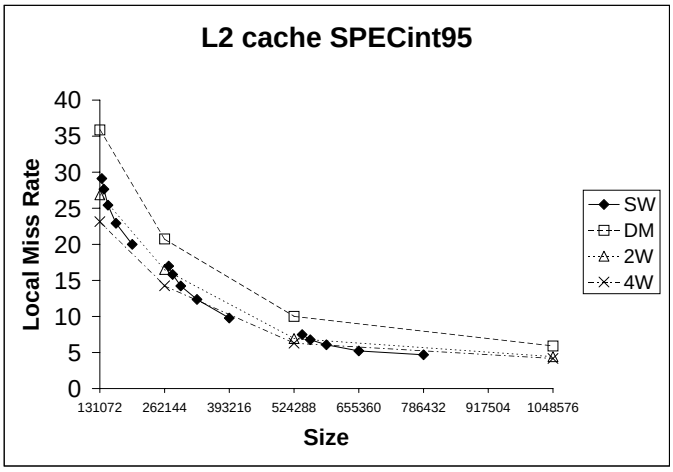


Fig. 6: Mean miss rate for the Wisconsin Java benchmarks for SWSA (SW) L1 data caches. Curves for direct mapped (DM), two-way set associative (2W) and four-way set associative (4W) caches are given for comparison purposes.



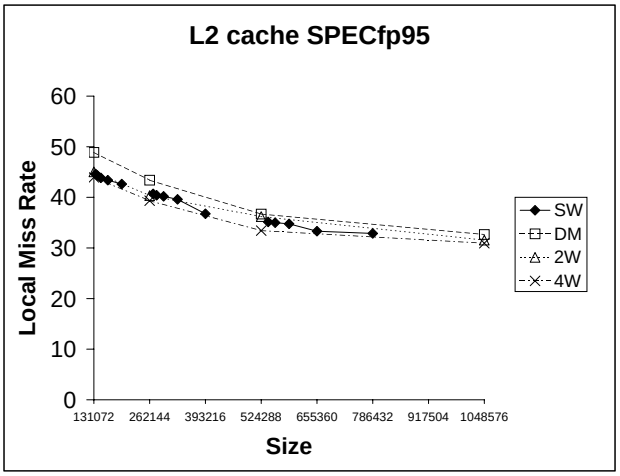


Fig. 7: Local mean miss rate for the SPEC95 benchmarks for SWSA L2 caches. Curves for direct mapped (DM), two-way (2W) and four-way (4W) set associative caches are plotted for comparison purposes.

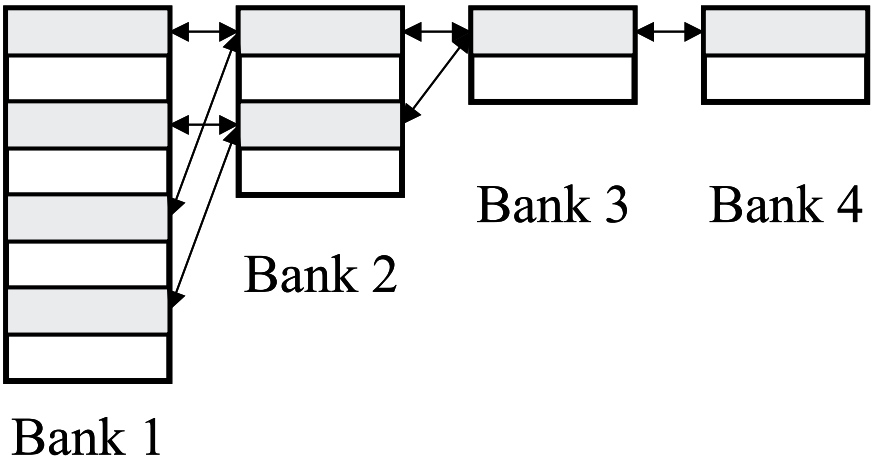


Fig. 8. A four bank SWSA cache example