Just Say No: Benefits of Early Cache Miss Determination

Gokhan Memik[†], Glenn Reinman[‡], William H. Mangione-Smith[†]

[†]Department of Electrical Engineering,
University of California, Los Angeles
{memik, billms}@ee.ucla.edu

[‡]Computer Science Department,
University of California, Los Angeles
reinman@cs.ucla.edu

ABSTRACT

As the performance gap between the processor cores and the memory subsystem increases, designers are forced to develop new latency hiding techniques. Arguably, the most common technique is to utilize multi-level caches. Each new generation of processors is equipped with higher levels of memory hierarchy with increasing sizes at each level. In this paper, we propose 5 different techniques that will reduce the data access times and power consumption in processors with multi-level caches. Using the information about the blocks placed into and replaced from the caches, the techniques quickly determine whether an access at any cache level will be a miss. The accesses that are identified to miss are aborted. The structures used to recognize misses are much smaller than the cache structures. Consequently the data access times and power consumption is reduced. Using SimpleScalar simulator, we study the performance of these techniques for a processor with 5 cache levels. The best technique is able to abort 53.1% of the misses on average in SPEC2000 applications. Using these techniques, the execution time of the applications are reduced by up to 12.4% (5.4% on average), and the power consumption of the caches is reduced by as much as 11.6% (3.8% on average).

1. INTRODUCTION

It is known that the performance of processor cores increase at a faster pace than the access times to memory resulting in an increasing performance problem. As this gap grows, designers are forced to develop techniques to close this gap. The most common technique is to use multiple-levels of memory hierarchy. Two level on-chip caches have been one of the common properties of high-end microprocessors in the recent years, and high-end processors have started employing three levels of on-chip caches (e.g. three levels of on-chip caches on McKinley [6]). If the trends continue, the next generation processors will use 4, 5, or even more levels of on-chip caches. In this paper, we propose techniques that will improve the cache access times and reduce the power consumption of processors with multi-level (more than 2) caches.

Two important factors limit the size of the level 1 caches and hence force designers use multiple level caches. First, the access times of caches increase with their size. Second, aggressive processors perform multiple accesses to the caches in a single cycle. Therefore, designers cannot build arbitrary sizes of caches. On the other hand, in general the cache footprints of the applications are increasing, therefore there is an increasing need for accessing large amounts of data at faster speeds. Hence, cache miss ratios are likely to increase or at least stay the same in next generation processors.

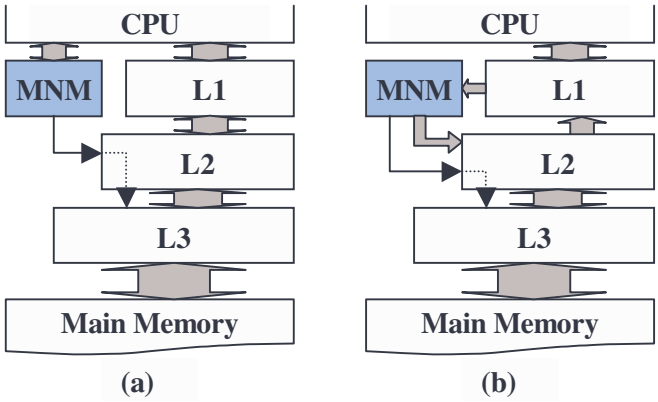

Figure 1. Different Mostly No Machine (MNM) positions in a processor with 3 cache levels. In (a) the MNM and L1 cache is accessed in parallel, in (b) MNM is accessed only after the L1 miss.

We propose techniques to quickly identify a set of misses. Consider a 5 level memory hierarchy. If we can determine that the first three levels of cache accesses are going to miss and the request will be supplied from the fourth level of hierarchy, we can bypass the accesses to the first three levels. Thereby, the delay of the data access will be reduced and the power consumed by the misses can be prevented. The techniques identify misses for both instruction and data cache accesses. The first proposed technique uses the information about the replaced blocks from the caches and determines the exact location of the accessed block. Other techniques store partial information about which addresses are kept in a certain cache level. Using this information a decision is made whether the access may hit or will definitely miss. These techniques are built into a **Mostly No Machine** (MNM) which is accessed either in parallel with the level 1 cache(s) or is accessed only after a level 1 cache miss. Figure 1 depicts the two positions. In Figure 1 (a), the MNM is accessed in parallel with the level 1 instruction and data caches (the instruction and data caches are not drawn separately in Figure 1 for brevity). At this position, the MNM identifies some (but not all) cases where the L2 cache will miss. In these cases, the access to L2 cache can be bypassed, directly accessing level 3 cache. Similarly, level 3 cache accesses can be bypassed directly sending requests to the main memory. Once a miss is determined, the decision is tagged along with the data request and propagates with the access through the cache levels. This tag forces caches to bypass if they are identified to miss by the MNM. Since the MNM structures are much smaller than the corresponding cache structures, the MNM reduces the delay and power consumption of the cache system.

Power consumption is one of the important issues in the modern processors. As the semiconductor feature size is reduced, more logic units can be packed into a given area. This reduces the delay of logic units, but increases the overall power consumption of the processor. Therefore, with each new generation of processors, the total power consumption as well as the power per area tends to increase, putting the overall execution of the processor in danger. Caches consume a significant portion of the overall power in the processors. By quickly identifying the misses, we show that power reduction can be achieved. Specifically, in this paper we

- present experimental results showing the fraction of the power consumed by the cache misses in a processor with multi-level caches,
- present novel techniques that use partial information about the blocks in a cache to identify whether the cache access will be a hit or a miss,
- use miss bypassing to improve cache access times, and
- show through simulation that by using the techniques power reductions and performance improvements are achieved.

We next present motivational data. In Section 2, we give an overview of the MNM and discuss the complexity of employing an MNM. Section 3 presents the different MNM techniques we have considered. In Section 4, we present the experimental results. Section 5 discusses the related work. In Section 6 we conclude the paper with a summary.

1.1 Effects of Cache Misses

The MNM is useful when there is considerable number of cache misses and these misses can be identified using small structures utilized by the MNM. Hence, we have first performed simulations to see the fraction of delay and power consumption caused by cache misses. We simulate SPEC2000 applications using processors with different levels of caches. The processors used in the simulations for 2 and 3 level caches are 4-way processors. The results for 5 and 7 level caches are obtained using an 8-way processor with resources (RUU size, LSQ size, etc.) twice of the processor for 2 and 3 level cache simulations. The time effects of misses are measured in data access time. We define the *data access time* to be the average time spent between when the CPU makes a request to the L1 cache and the time the data is supplied to the CPU. If all requests are satisfied by the level 1 cache, the data access time is equal to the level 1 cache latency. In other cases, it is the sum of number of requests satisfied by each memory level including the main memory weighted with the latency of accessing the data in that memory level. Section 2 presents a formula to calculate the data access time using the hit rates at each cache level. The misses increase the data access times, because if the data will be supplied by the nth level cache, all the cache levels before n will be accessed causing unnecessary delay. If these misses are known in advance and not performed, the data access time will be reduced.

Figure 2 presents the time fraction of the data access times caused by cache misses. Each bar in the figure represents the fraction of time spent for cache misses in the respective configuration. For example, a 30% fraction for a processor with 5 levels of cache means that on an average access, 30% of the time is spent accessing caches that miss and 70% of the time is actually spent in the correct cache level to access the data. We see that as the number of levels is increased, the fraction of misses increases. In the simulated processor with 5 levels, the misses cause 25.5% of the data access times. In Section 4, we will show that such a fraction can have significant effect on the performance.

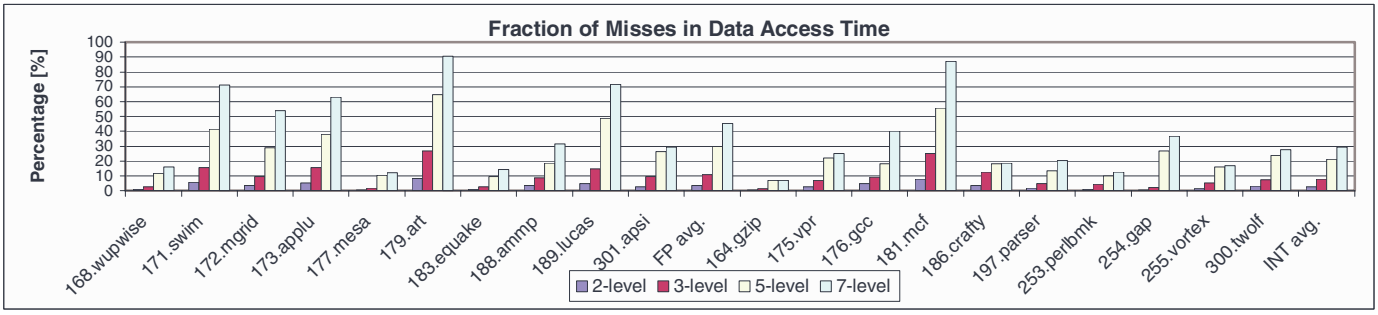

Figure 2.Fraction of time spent for cache misses in the overall data access times.

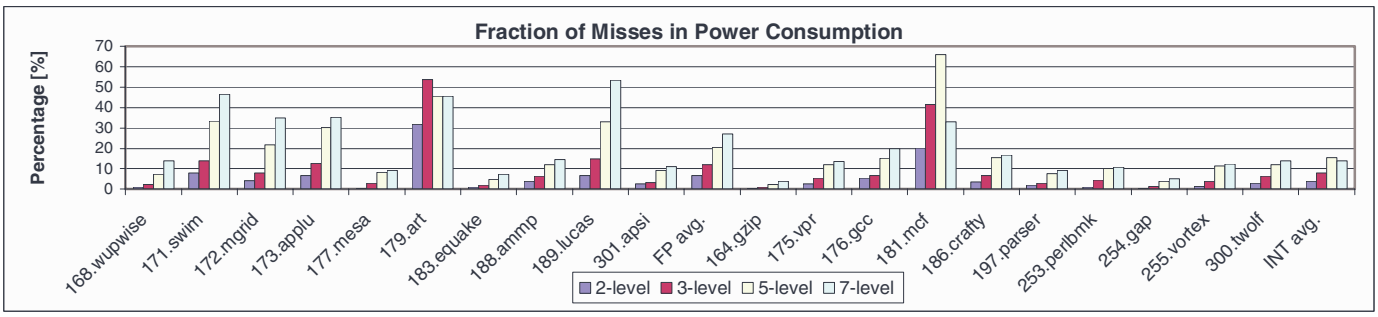

Figure 3.Fraction of cache power consumption for cache misses in the overall cache power consumption.

Figure 3 presents the fraction of power consumption of the caches caused by cache misses. Similar to data access times, as the number of levels is increased, the fraction generally increases. The fraction reduces for 181.mcf and 179.art for higher levels of caches, because these applications have relatively high miss ratios for the lower level caches. The miss ratios are larger for low-level caches. Therefore, the fraction of misses in data access times increase with higher levels of caches, but the fraction in power consumption decreases, because power consuming larger caches have small miss rates. Nevertheless, we see that on average approximately 18% of the overall cache power is consumed for the misses in the processor with 5 level caches. These results show that the effects of cache misses will become an important factor in both data access times and power consumption, motivating the exploration of techniques to minimize the negative effects of cache misses.

2. MOSTLY NO MACHINE OVERVIEW

In this section, we discuss how a processor should be modified to utilize the MNM techniques and also give an overview of how the MNM works. The MNM propagates the decision with the help of additional tags. When the MNM identifies a miss, *miss signals* for each cache level is generated and is tagged to the control signals sent to the higher levels of caches. There is a miss bit for each level of cache (except the level 1). The i^{th} miss bit dictates whether the access should be performed at level i, or whether the address should be bypassed to the next cache level without performing any access to the cache structure. Another possible approach might be to implement control buses from the MNM to each cache level. This will reduce the data access times even further, but might increase the processor complexity.

There are several locations where the MNM can be realized. Figure 1 depicts two possible locations of an MNM in a processor. In Figure 1 (a) (**parallel MNM**), the L1 cache(s) and the MNM are accessed in parallel. In this configuration, the number of ports in the MNM is equal to the sum of ports in the level 1 data and instruction caches. When the address is given to the MNM, it identifies which cache has a block that contains the specific address. When the L1 cache miss is detected, we know which caches to access, because in all techniques, the MNM delay is smaller than the L1 delay. Hence, the miss signals are generated prior to the detection of L1 cache miss, and it is tagged to the control signals after the L1 miss detection. In Figure 1 (b) (**serial MNM**), the MNM is accessed only after level 1 cache misses. The advantage of this configuration is the reduced power consumption by the MNM and the reduced complexity of the MNM block due to fewer ports required. If the level 1 hit ratios are high, the parallel MNM will not generate any useful information for most of the accesses. The serial MNM, on the other hand, has the disadvantage of higher cache access times. For the serial MNM, the access times to any cache level except the level 1 caches are increased by the delay of the MNM.

Another possible configuration is to employ a distributed MNM that identifies the misses before each cache level. The misses are recognized using information about the blocks in each cache level. This information can be distributed to each cache. Such a configuration will have better power consumption, but will increase the access times.

To incorporate an MNM into a processor, the cache structures should be modified. Each cache structure should be extended with logic to detect the miss signal and bypass the access if necessary. In the parallel MNM, the bus to the L1 core should also be accessed by the MNM (this can be achieved by snooping). In the serial MNM, the request generated by the L1 is sent to the MNM, which forwards the request to the L2. This modification is necessary to guarantee the synchronization between the access and the miss signal. The data supplied from the L2 cache, on the other hand, will be directly sent back to the L1. So, there is need for separate address/data busses between the L1 and L2 cache. If the number of cache levels is increased, there might be separate instruction and data caches behind the first level (e.g. separate level 2 instruction and data cache). In such a cache system, the MNM is still placed between the level 1 and level 2 caches. For such a system, the MNM will have two inputs (from level 1 data and instruction caches) and two outputs (to level 2 instruction and data caches). The final modification needed by the MNM is that each cache has to send the information about the blocks that are replaced from the cache. This information is needed for the bookkeeping required at the MNM. Since the request for a block goes through the MNM, the block address that will be placed into any cache can be easily calculated. But, as we will show in the next section, the MNM also needs information when a block is replaced from the cache. This can be achieved through separate bus signals sent from each cache to the MNM indicating the block address replaced or these addresses can be attached to the blocks sent to the level 1 caches.

As explained in the previous sections, the MNM improves the data access times. In the following, we develop an analytical model to approximate the benefits of any MNM technique. Equation 1 presents the formula to calculate the average data access time in a processor with multiple cache levels.

$$\Sigma_{i=1 \text{ to memory\_levels}} \left(\Pi_{n=1 \text{ to } i-1} \text{miss\_rate}_n\right) * \left(\text{cache\_hit\_time}_i * (1 - \text{miss\_rate}_i) + \text{cache\_miss\_time}_i * \text{miss\_rate}_i\right)$$

Equation 1. Data access times without MNM

The cache_hit_time corresponds to time to access data at a cache and cache_miss_time is the time to detect a miss in a cache. The data access times in case when an MNM is utilized can be found using Equation 2.

$$\Sigma_{i=1 \text{ to memory\_levels}} \left(\Pi_{n=0 \text{ to } i-1} \text{miss\_rate}_n\right) * \left(\text{cache\_hit\_time}_i * (1 - \text{miss\_rate}_i) + \text{cache\_miss\_time}_i * \text{MNM\_aborted}_i * \text{miss\_rate}_i\right)$$

Equation 2. Data access times with MNM

3. MNM TECHNIQUES

This section presents the proposed techniques for identifying cache misses. The first technique gathers information about

the blocks that are replaced from the caches. The remaining techniques store information about the blocks stored in the cache. All the techniques except the first technique, has separate structures storing information about the different caches. When a decision has to be made, the collective information is processed by accessing all the structures in parallel and interpreting the data stored in these structures.

When a block is placed into a cache, all the bytes in that block are placed into the cache. Therefore, to recognize misses, there is no need to store information about the exact bytes that are stored in a cache. Instead in all the techniques, the block addresses are stored. The block address is composed of the tag and the index sections of an address. Figure 4 shows the block address portion of an address. For example, if the cache has 128-byte block size, each address is shifted 7 bits right before it is entered to the MNM. Therefore, in the remainder of this paper, an address entered to the MNM corresponds to a block address placed into or replaced from the cache.

The techniques do not assume the inclusion property of caches. In other words, we assume that if cache level i contains a block b, block b is not necessarily contained in cache level i+1. In addition, the MNM checks for the misses on cache level i+1, even if it cannot identify a miss in cache level i. Since the miss signals are propagated with the access, level i+1 can still bypass the access even if the access is performed in cache level i. For example, if the MNM identifies the miss in L3 cache but could not identify at L2 cache, first the L2 cache will be accessed. If it misses, the L3 cache will be bypassed because a miss was identified for the cache. *In all the techniques, a miss for a cache level is indicated by a high output of the MNM structures.*

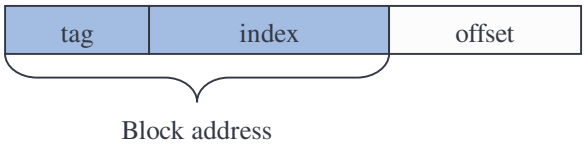

Figure 4.Portion of the address used by the MNM.

3.1 Replacements MNM (RMNM)

The Replacements MNM stores the addresses that are replaced from the caches. If a block is replaced from a cache, it means that the address is not present in the specific cache anymore. Therefore, access to the address will miss. Note that cold misses cannot be captured with this technique.

The information about the replaced blocks is stored in an RMNM cache. The RMNM cache is addressed by the addresses of the current reference or the address of the block being replaced. They are shifted according to the block size of the level 2 cache(s). Therefore, if a block is replaced from a cache with a larger block size, multiple accesses to the RMNM cache are necessary. Specifically, there will be (block $size_{LARGE}$ / block $size_{L2}$) accesses performed to the RMNM cache indicating that accesses to any of the addresses will miss. Instead of storing the replaced blocks from each cache in a separate RMNM cache, we have chosen to have a single RMNM cache that stores information about each cache level. The RMNM cache has a block size of (n-i) bits, where n corresponds to number of separate caches and i correspond to the number of level 1 caches. Each bit in the block corresponds to a cache and there is no information stored about the first level caches. When i^{th} bit for an address is set, that means the block is replaced from the cache and hence a miss will occur for the access. Table 1 shows a scenario for a 2-level cache structure, where a sequence of addresses is accessed by the core. Since there are only two levels of caches, each RMNM block contains a single bit indicating the hit/miss for the second level cache. Table 1 shows the block addresses placed into and replaced from the caches and the RMNM cache. When the block with address `0x2FC0` is replaced from the L2 cache, it is placed into the RMNM cache and the subsequent access, which will miss, is captured by the RMNM.

Table 1. An example scenario showing how the RMNM works. The abbreviations are: *pl.* stands for placed into the cache, *repl.* stands for replaced from the cache.

	Event		
Access	L1	L2	RMNM cache
0x2FF4	pl. 0x2FF0	pl. 0x2FC0	
0x20F4	repl.0x2FF0 pl. 0x20F0	repl.0x2FC0 pl. 0x20C0	place 0x2FC0 with data 1
0x2FF4	repl.0x20F0 pl. 0x2FF0	repl.0x20C0 pl. 0x2FC0	The miss at level 2 cache is identified.

3.2 Sum MNM (SMNM)

The sum technique stores information regarding the relative locations of the bit values on the addresses that are high. Specifically, for each access a hash value is generated. If the hash value of the access does not match any of the hash values of the existing cache blocks, a miss is captured. The specific hash function is given in Figure 5. The result of the function is `sum` value that is modified in the for loop. Figure 6 shows the design of the circuit implementing the sum mechanism. The figure is drawn with the `SUM_WIDTH` value equal to 3. The D flip-flops at the bottom of the Figure 6 store the hash values for the addresses in the cache. At the start of the execution, the bits are reset. Then, when a block is placed into the cache, the address is placed into the circuit as input and the resulting hash value is stored in the flip-flops. On an access, if the hash value of the requested address is equal to one of the hash values in the flip-flops, then the access is performed (output of the design will be zero). If it does not match any of the hash values, then, the corresponding cache access is bypassed. Note that the complexity of the logic units is $O(SUM\_WIDTH^4)$, whereas the number of flip-flops at the bottom is $O(SUM\_WIDTH^3)$. Specifically, we need

$$1 + sum\_width * (sum\_width + 1) * (2sum\_width + 1) / 6$$

Equation 3. Number of D Flip-flops for the sum circuit.

D flip-flops at the bottom of Figure 6, which is O(SUM_WIDTH^3). The amount of logic for each level can be estimated by Equation 3. Since there are sum_width levels, the total logic units will be bounded by O(SUM_WIDTH^4).

```
sum = 0;
for (i = 1; i < (SUM_WIDTH + 1); i++) {
      if (tag & 0x1)
            sum += i * i;
      tag = tag >> 1;
}
```

Figure 5.Sum hash function

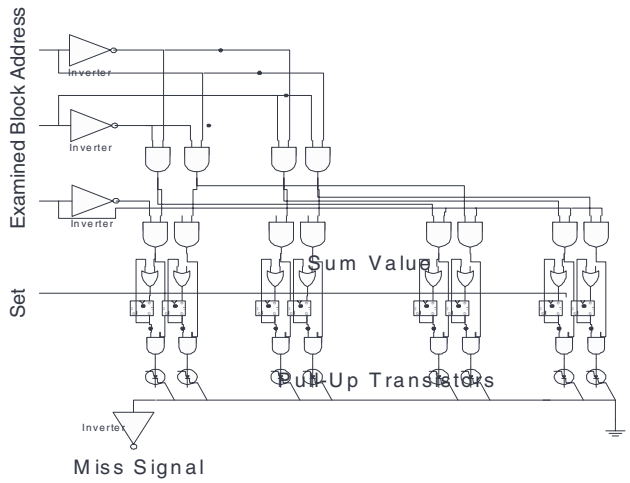

Figure 6.Circuit implementing the sum mechanism (*checker*)

We name the design in Figure 6, which examines a portion of the tag, a *checker* circuit. Some SMNM configurations use multiple checkers that examine different portions of the block address. The SMNM configuration for each cache level is denoted by sumwidthxreplication, where sumwidth equals to the sum_width at each checker and the replication is the number of parallel checkers implemented. For example, SMNM_12x3 means that there are 3 parallel checkers each with sum_width equal to 12. Regardless of the sum_width, if there are multiple checkers, the first one examines the least significant bits, the second examines the bits starting from the 7th leftmost bit and the third one examines the bits starting from the 13th. Figure 7 shows how multiple checkers work in parallel. The configuration drawn is SMNM_10x2. Each checker applies the algorithm presented above. Then, if any of the checkers capture a miss (return 0), the access is bypassed.

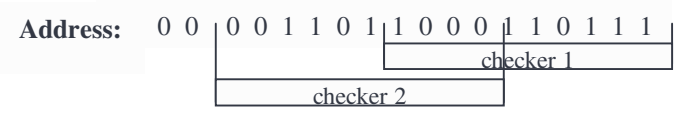

Figure 7.SMNM example.

3.3 Table MNM (TMNM)

The third technique proposed is called the Table MNM. TMNM stores the least significant N bits of the block addresses in the caches. If the least significant N bits of the access do not match any one of the stored values, then the access can be bypassed.

The values are stored in the TMNM table, an array of size 2^N bits. The least significant N bits of the address are used to address the TMNM table. The locations corresponding to the addresses stored in the caches are set to 0 and the remaining locations are set to 1. During an access, the least significant N bits of the block access address are used to address this table. The value stored at the corresponding location is used as the miss signal. An example TMNM for N equals to 6 is drawn in Figure 8. The cache in the example only has two blocks with the block addresses as shown in the figure. When the request comes to the MNM, the corresponding position is calculated and the bit position is read from the TMNM table. In Figure 8, this location is high, which means that the access is going to be miss and can be bypassed. There can be several block addresses that are mapped to the same bit position in the TMNM table. Therefore the values at the TMNM tables are counters instead of single bits. We use a counter of 3 bits in our simulations. When a new block is placed into the cache, the counter value at the corresponding position is incremented unless it is saturated. If a block is replaced from a cache, the corresponding counter value is decremented unless the counter is saturated. A saturated value occurs when there are 8 different blocks that are mapped to the same location. In this case, we cannot conclude whether there are 8 or more blocks mapped to the same location, therefore the counter becomes an indicator that any access mapped to this position "may be a hit". The counter values are reset when the caches are flushed.

Similar to the SMNM, we have performed simulations with multiple tables examining different portions of the address. Therefore each TMNM is defined as TMNM_nxreplication, where n corresponds to the number of bits checked by each table and replication is the number of tables examining different positions of the address.

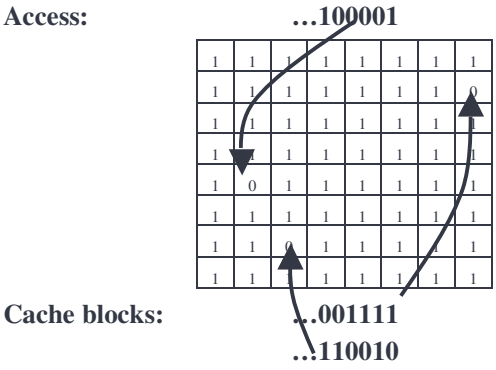

Figure 8.TMNM example.

3.4 Common-Address MNM (CMNM)

Common-Address MNM tries to capture the common values at the block addresses. When an application accesses a memory position x, it is likely that the memory positions close to x will be accessed in the following cycles. Common-Address value tries to capture this locality by examining the most significant bits of the address. This is achieved with the help of a virtual-tag finder and a table similar to the TMNM table. Virtual-tag finder stores the most significant n-bits stored in the cache. Then, for each address it generates a virtual-tag that is attached to the remaining m bits of the address and used to

access the table. Figure 9 depicts the events for finding the address used to access the CMNM table. Assuming a 32-bit address, the CMNM enters the most significant (32 – m) bits to the virtual-tag finder. Virtual-tag finder has k registers storing the encountered most significant portions of the cache blocks. The masked bits of the address are compared to the values. If the new address matches any of the existing values, the virtual-tag finder outputs the index of the matching register. This index is attached to the least-significant m bits of the examined address and used to access the CMNM table. Similar to the TMNM, this table indicates whether the address should return a miss or a "maybe". If the address of the block to be placed into the cache does not match any register values in the virtual-tag finder, mask value for the registers are shifted left until a match is found. Then the mask values are reset to their original position except the register that matched the value.

When an address is checked, there are two ways to identify a miss. First, the (32 – m) most significant bits of the address are entered to the virtual-tag finder. If the value does not match any of the register values in the virtual-tag finder, the access is marked as a miss. If a register matches the address, the index is attached to the least significant m bits of the address and the CMNM table is checked. If the corresponding table position has a 1, again a miss is indicated.

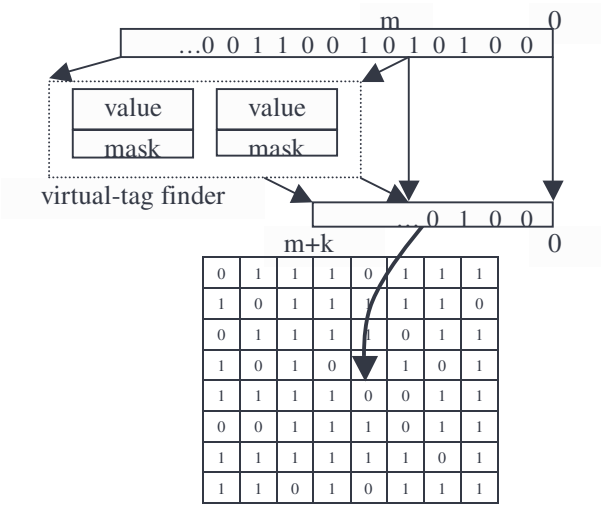

Figure 9.CMNM example.

The CMNM techniques are defined by the number of registers in the virtual-tag finder and the m value. For example CMNM_4_10 corresponds to a CMNM checker with the first 22 most significant bits used to access the virtual-tag finder (which has 4 registers) and the least significant 10 bits used to access the CMNM table. Similar to the TMNM table, each address of the CMNM table is a 3-bit counter.

3.5 Hybrid MNM (HMNM)

The previous MNM techniques explained in this section perform different transformations on the address and use small structures to identify some of the misses. A natural alternative is to combine these techniques to increase the overall accuracy of recognition of the misses. Such a combined MNM is defined as Hybrid MNM. We have performed simulations with a variety of combinations of the proposed techniques and measured the effect on the accuracy in Section 4.

3.6 Discussion

The MNM techniques discussed in this section never incorrectly indicate that bypassing should be used, but do not detect all opportunity for bypassing. In other words, if the MNM indicates a miss, then the block certainly does not exist in the cache, however if the output is a "maybe" then the access might still miss in the cache. The techniques are developed intentionally with this property. The miss signals should be reliable because the cost of indicating an access will miss when the data is actually in the cache is high (redundant access to a higher level of memory hierarchy should be performed), whereas the cost of a hit misindication is relatively less (a redundant tag comparison at the cache).

4. EXPERIMENTS

In this section, we measure the effectiveness of various MNM techniques. Section 4.1 discusses the simulation environment. In Section 4.2, we present simulation results measuring the success of the MNM techniques. Section 4.3 presents the improvement in the execution time of the simulated applications using the parallel MNM. Finally, Section 4.4 explains the power reduction effects of the serial MNM.

4.1 Methodology

We have performed several simulations to measure the effectiveness of the proposed techniques. The SimpleScalar 3.0 [12] simulator is used in all the experiments. The necessary modifications to the simulator have been implemented to measure the effects of the multiple level caches and the MNM techniques. We simulate 10 floating-point and 10 integer benchmarks from the SPEC2000 benchmarking suite [14]. The applications are compiled using DEC C V5.9-008 and Compaq C++ V6.2-024 on Digital UNIX V4.0. Sherwood et. al [11] show how to speed up the simulation of SPEC applications without affecting the simulation results. We have simulated 300 Million instructions from each application after fast-forwarding application-specific number of instruction determined by Sherwood et. al [11]. Important characteristics of the applications are explained in Table 2.

In all the experiments, we simulate an 8-way processor with 5 cache levels. The processor has separate level 1 and level 2 instruction and data caches. Therefore, there are a total of 7 different cache structures in the processor. The level 1 caches are 4 KB, direct-mapped caches with 32-byte block size and 2 cycle latency. The level 2 caches are 16 KB, 2-way associative caches with 32-byte linesize and 8 cycle latency. Unified level 3 cache is a 128 KB, 4-way associative cache with 64-byte blocks and 18 cycle latency. Unified level 4 cache is a 512 KB, 4-way associative cache with 128-byte block size and 34 cycle latency. Finally, unified level 5 cache is a 2 MB, 8-way associative cache with 128 byte blocks and 70 cycle latency. For all configurations, the MNM has a 2-cycle delay. The memory access delay is 320 cycles.

4.2 MNM Miss Coverage

The success of the MNM techniques is measured in *coverage*. Coverage is the fraction of the misses identified by the technique over all cache misses. For example, if the access will hit in level 4 cache and miss in all the previous cache levels, potentially 2 caches can be bypassed because we do not predict misses in the first level cache. If the MNM recognizes the miss in level 2, but not for level 3, then the coverage is 50%, because only half of the misses are identified. Note that coverage is not affected by the location of the MNM. The location affects the delay and power consumption.

Figure 10 presents the coverage for different RMNM configurations. Different configurations are labeled according to the RMNM cache used in the simulations. RMNM_n_m corresponds to an RMNM cache of associativity m and the number of blocks equal to n. The configuration is used for all the cache levels. We see that although the average coverage of RMNM is low, for some applications it can recognize majority of the misses. The reason lies in the nature of the applications. If the conflict and capacity misses constitute a large portion of the misses, RMNM may capture them. However, if there are too many replications that pollute the RMNM cache or the cold misses dominate the misses in a cache, RMNM has a low coverage. The largest configuration simulated is a 2 KB, 8-way associative cache, which identifies 24% of the misses on average.

The coverage results for SMNM configurations are presented in Figure 11. The labeling is explained in Section 3.2. For an SMNM configuration, all the structures are replicated for 4 cache levels. The SMNM coverage is small except for the application 301.apsi. A closer investigation of the results reveal that SMNM structures can identify the misses in the smaller caches better. The 301.apsi application has a high miss ratio for level 2 instruction cache and high hit ratios for the remaining caches. This increases the overall weight of the misses for the small caches. Figure 12 presents the coverage for different TMNM configurations. The labeling of different configurations are explained in Section 3.3. The 12x3 configuration has the best coverage, recognizing 25.6% of the misses on average. We see that the TMNM_10x3 configuration has a better coverage in all applications than the 11x2 coverage, which uses larger structures. This shows that using multiple tables in parallel can boost the performance significantly. Figure 13 plots the coverage for the CMNM configurations. Among the proposed techniques the CMNM has the best coverage. The CMNM_8_12 technique identifies 46.4% of the misses on average. Although this is a large structure, compared to the caches the delay and power consumption is very small.

Table 2. Important characteristics of the simulated SPEC2000 applications.

Application	# cycle [M]	# dl1 acc [M]	# il1 acc [M]	dl1 hit rate [%]	dl2 hit rate [%]	il1 hit rate [%]	il2 hit rate [%]	ul3 hit rate [%]	ul4 hit rate [%]	ul5 hit rate [%]
168.wupwise	290.4	94.4	350.0	94.2	60.9	98.0	100.0	57.3	56.9	12.7
171.swim	839.7	100.4	300.4	80.4	36.9	99.9	55.5	63.7	70.1	32.1
172.mgrid	480.6	109.8	305.9	68.3	72.2	97.4	99.8	69.6	61.8	48.1
173.applu	710.5	114.7	308.9	83.4	52.9	99.9	0.6	68.1	66.9	34.2
177.mesa	289.8	113.5	339.2	91.4	92.3	93.4	95.4	90.0	76.9	49.2
179.art	1249.4	112.1	327.8	57.6	21.3	99.9	95.4	27.5	43.8	88.5
183.equake	318.3	112.8	468.2	96.2	98.8	92.9	76.9	99.3	53.7	19.8
188.ammp	309.6	118.4	318.3	90.4	44.6	99.9	100.0	59.8	54.2	75.2
189.lucas	680.1	84.2	300.0	74.0	39.8	99.9	0.0	61.9	36.9	47.5
301.apsi	296.0	115.4	343.0	87.1	76.1	97.8	47.2	75.9	90.3	67.5
FP avg.	546.4	107.6	336.2	82.3	59.6	97.9	67.1	67.3	61.1	47.5
164.gzip	175.3	77.1	362.6	94.4	65.2	99.9	91.4	86.2	86.2	32.7
175.vpr	394.8	118.6	381.6	87.8	69.1	95.2	91.6	63.1	63.1	99.6
176.gcc	313.6	111.2	323.4	88.8	67.0	94.9	61.8	91.3	91.3	61.9
181.mcf	1767.4	185.6	625.0	65.4	35.7	99.9	97.4	46.3	46.3	45.1
186.crafty	529.5	111.5	371.4	84.5	82.6	90.8	60.7	96.8	96.8	43.6
197.parser	331.4	110.0	389.1	89.9	65.9	98.7	97.7	76.1	76.1	67.7
253.perlbmk	443.1	127.5	424.7	96.8	48.2	99.4	84.4	69.0	69.0	32.0
254.gap	183.4	67.6	331.7	91.9	53.1	98.8	78.8	70.8	70.8	37.5
255.vortex	407.7	127.0	342.9	92.7	84.0	91.2	60.8	96.0	96.0	50.1
300.twolf	388.3	98.1	397.0	87.2	58.5	96.8	62.2	72.5	72.5	99.5
INT avg.	493.4	113.4	394.9	87.9	62.9	96.6	78.7	76.8	76.8	57.0
Arith. Mean	519.9	110.5	365.5	85.1	61.2	97.3	72.9	72.1	69.0	52.2

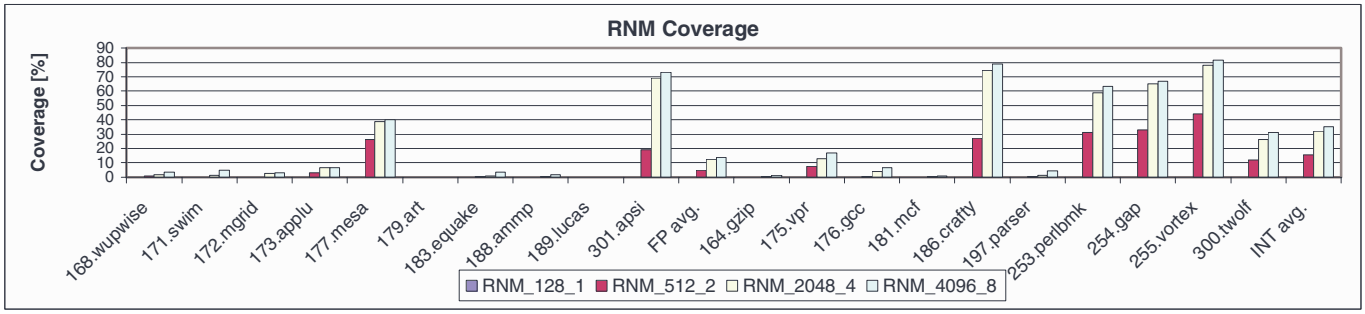

Figure 10. RMNM coverage.

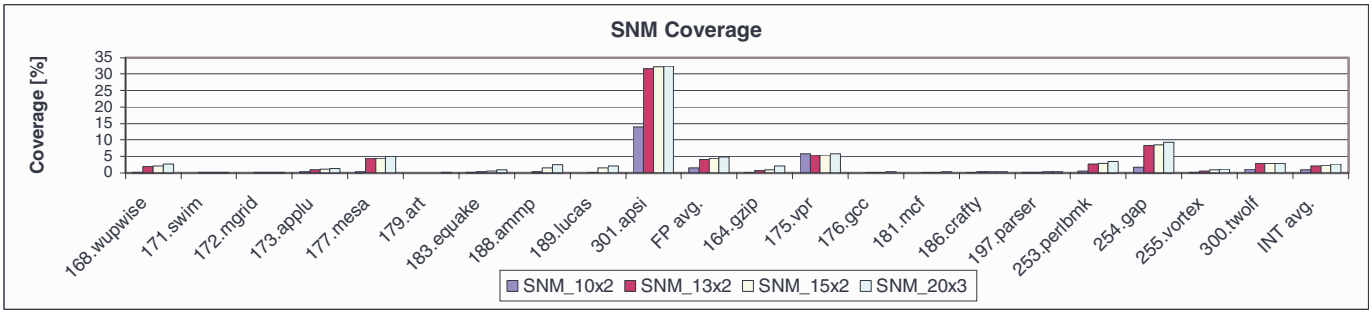

Figure 11. SMNM coverage.

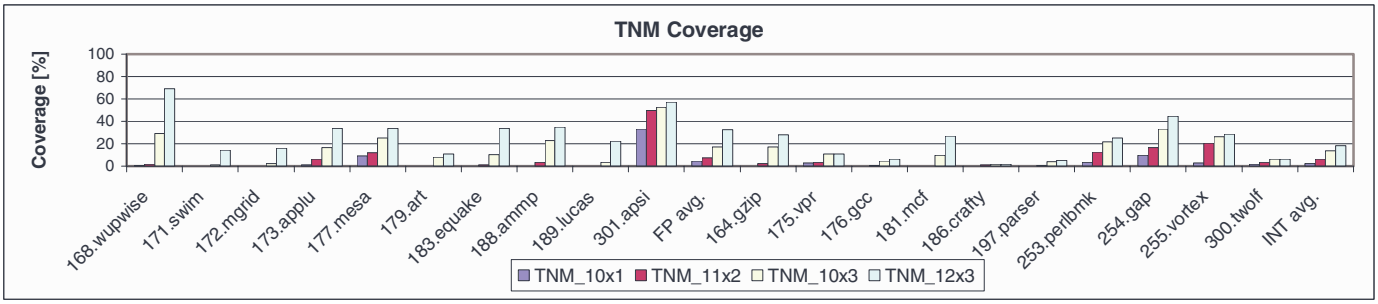

Figure 12. TMNM coverage.

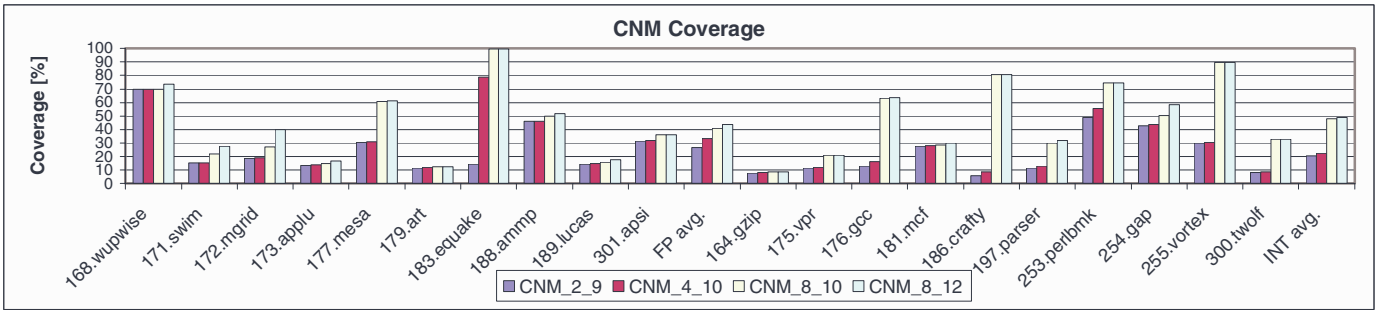

Figure 13. CMNM coverage.

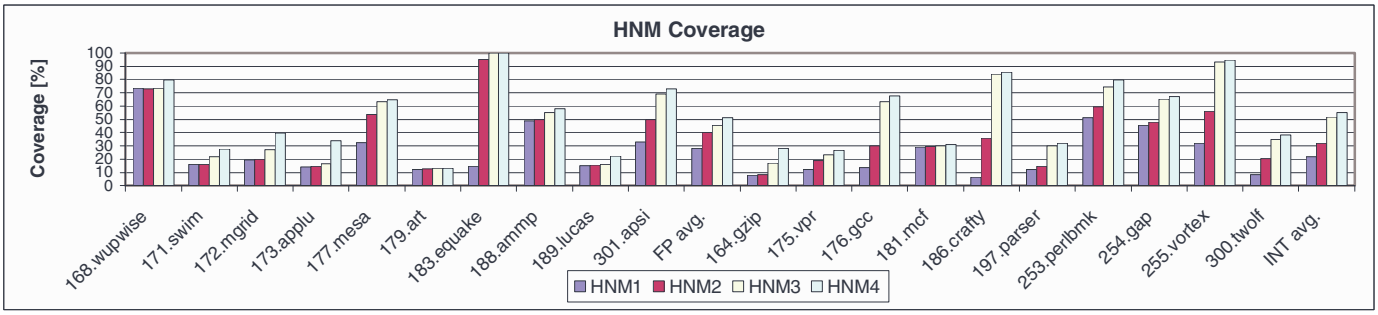

Figure 14. HMNM coverage.

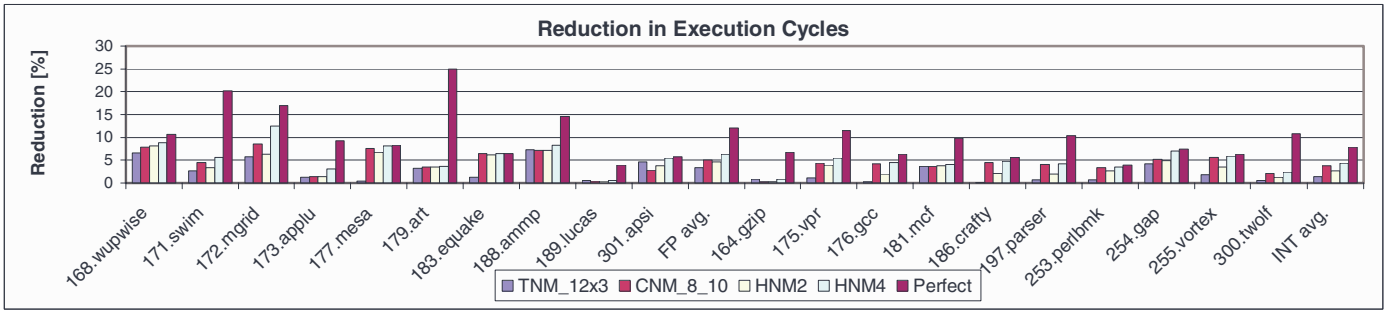

Figure 15. Reduction in execution cycles.

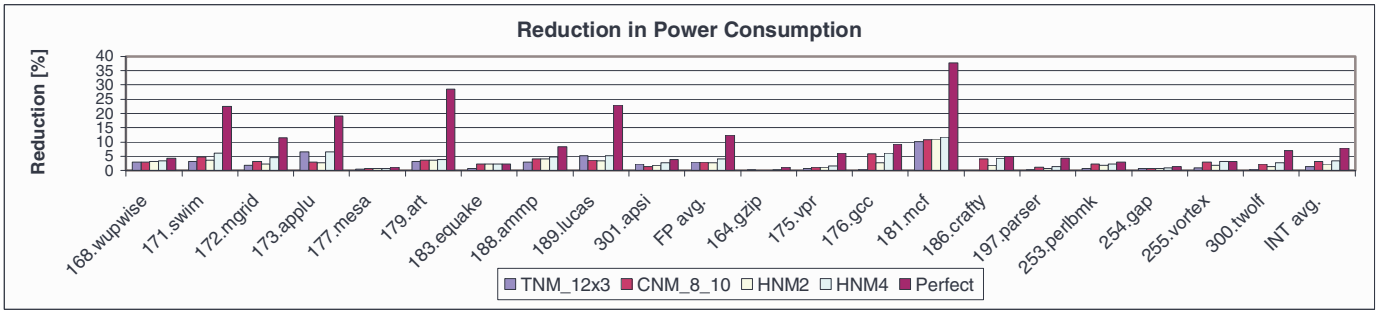

Figure 16. Reduction in the power consumption of the caches.

The results for the HMNM are presented in Figure 14. The techniques used in each HMNM for different configurations are listed in Table 3. The simulated configurations have an increasing level of complexity in terms of delay and area. Nevertheless, the delay for the most complex configuration HMNM4 is smaller than the delay of the 4 KB level 1 caches. The HMNM4 configuration has an average 53.1% coverage.

Table 3. HMNM configurations. In addition to the techniques listed, HMNM1 employs an RMNM_128_1, HMNM2 employs an RMNM_512_2, HMNM3 employs an RMNM_2048_4, and HMNM4 employs an RMNM_4096_8.

Cache	HMNM1	HMNM2	HMNM3	HMNM4
Levels 2 and 3	SMNM10x2 + TMNM_10x1	SMNM13x2 + TMNM_10x1	SMNM15x2 + TMNM_10x1	SMNM20x3 + TMNM_10x3
Levels 4 and 5	CMNM2_9 + TMNM_10x1	CMNM4_10 + TMNM_11x2	CMNM8_10 + TMNM_10x3	CMNM8_12 + TMNM_12x3

4.3 Execution Time Measurements

We have performed several simulations with the parallel MNM to see measure its effects on the execution time of the SPEC 2000 applications. The results are summarized in Figure 15. Figure 15 presents results for 4 different techniques (TMNM_12x3, CMNM_8_10, HMNM2, and HMNM4). To see the limits on the performance improvements of the MNM techniques, we also report simulation results of a perfect MNM. The perfect MNM always knows where the data is and hence bypasses all the caches that misses. The HMNM4 technique reduces the execution cycles by as much as 12.4%, and by 5.4% on average. The perfect MNM reduces the execution time by as much as 25.0% and 10.0% on average.

4.4 Power Reduction Measurements

This section presents the measurements for power consumption of the caches using the MNM techniques. For the experiments, the MNM structures are accessed after level 1 cache misses. We have found the power consumption of the simulated caches, as well as the MNM structures using the CACTI 3.1 [16]. For the SMNM structures, we have implemented the RTL descriptions of the checkers in VHDL and used Synopsys Design Compiler [15] to find the power and delay properties.

Figure 16 presents the power reduction in the cache system using the MNM techniques. The HMNM4 configuration is able to reduce the power consumption of the caches by as much as 11.6% (3.8% on average). The perfect MNM that identifies all the misses perfectly without consuming any power, reduces the power consumption by as much as 37.6%, and 10.2% on average.

4.5 Discussion

We have seen that the misses even for large caches can be accurately identified using small structures utilized in the MNM. We also showed that the information about misses might be used to increase the performance or reduce the power consumption of the caches in modern processor. However, the utilization of the information about cache misses is not limited to these two usages. For example, miss information can be used in instruction scheduling step. The scheduler can use the miss information to prevent scheduling of the memory instructions that will miss in the level 1 cache and other instructions dependent on these memory instructions. Another usage might be to reduce the power consumption of other caching structures such as the TLBs.

5. RELATED WORK

Related work falls into a group of studies conducted for reducing the negative effects of cache misses. Arguably the most important technique to reduce cache miss penalty is the non-blocking caches, also called the lock-up free caches [7].

Non-blocking caches do not block after a cache miss, being able to provide data to other requests. Sohi and Franklin [13] discuss a multi-port non-blocking L1 cache. Farkas and Jouppi [3] explore alternative implementations of the non-blocking caches. Farkas et. al [4] studies the usefulness of the non-blocking caches. Other important techniques for reducing cache miss penalty is giving priority to read misses over write misses, subblock placement, early restart, and critical word first on a miss, which gives priority to the accessed word over the other sections of the cache block [5]. Seznec et. al [10] studies caches in an out-of-order processor to find optimal linesize to reduce the cache miss penalty. Although these techniques share the same goal of reducing the cache miss penalty similar to the proposed techniques, the techniques employed have no resemblance to our proposed mechanisms. In all the above-mentioned techniques the miss is detected after the cache structures are accessed.

Way prediction [2] and selective direct-mapping [1] were proposed to improve set-associative cache access times. Powell et al. [9] use these techniques to reduce the energy consumption of set-associative caches. Our techniques identify whether the access will be a miss in the cache rather than predicting what associative way of the cache will be accessed.

In the context of multiple processor systems, Moshovos et al. [8] propose filtering techniques for snoop accesses in the SMP servers. Similar to our work, they identify the hits or misses in the level 2 cache. However, the predictions are made for snoop accesses originating from other processors. We identify misses in a single-core processor with multiple caches and the requests originate from the core instead of arriving to the processor from other cores.

6. CONCLUSION

In this paper, we have presented techniques to reduce the penalty of cache misses. Particularly, we have shown techniques identifying the misses in different cache levels. These techniques are implemented in a Mostly No Machine (MNM). When an access is identified to miss, the access is directly bypassed to the next cache level. Thereby the cache structures are not accessed, reducing the delay and the power consumption associated with the misses. We have first shown that as the number of cache levels employed in processors increase, the fraction of the time and power spent for cache misses also increase. Then, we have presented 5 different techniques of varying complexity to recognize some of the cache misses. We show that using the small structures MNM is able to identify and hence prevent 53.4% of the misses in a processor with 5 cache levels. The MNM can be aggressively accessed in parallel with the level 1 caches reducing the execution time of the applications. It can also be accessed after a level 1 cache miss, primarily to reduce the power consumption of the caches. Specifically, we have shown that the execution time of SPEC 2000 applications are reduced by 5.4% on average (ranging from 0.6% to 12.4%), whereas the power consumption is reduced by 3.8% on average (ranging from 0.4% to 11.6%) using an Hybrid MNM technique.

REFERENCES

1. Batson, B. and T.N. Vijaykumar. Reactive associative caches. In Proceedings of *2001 International Conference on Parallel Architectures and Compilation Techniques*, Sep. 2001.

2. Calder, B. and D. Grunwald. Predictive Sequential Associative Caches. In Proceedings of *Second IEEE Symposium on High-Performance Computer Architecture*, Feb. 1995.

3. Farkas, K. and N. D. Jouppi. Complexity/Performance Tradeoffs for Non-blocking Loads. In Proc. of *21st International Symposium on Computer Architecture*, April 1994.

4. Farkas, K., N. D. Jouppi, and P. Chow. How Useful are Non-Blocking Loads, Stream Buffers, and Speculative Execution in Multiple Issue Processors? In. Proc. of 1st *International Symposium on High Performance Computer Architecture*, Jan. 1995.

5. Hennessy, J. L. and D. A. Patterson, *Computer Architecture: A Quantitative Approach*. 1990, San Mateo, CA: Morgan Kaufmann.

6. Intel, Inc. Intel Itanium 2 Processor at 1.0 GHz and 900 MHz Datasheet. Intel Document Number 250945-001, July 2002.

7. D. Kroft. Lock-up Free Instruction Fetch/Prefetch Cache Organization. In Proc. of *8th International Symposium on Computer Architecture*, May 1981.

8. Moshovos, A., G. Memik, B. Falsafi, and A. Choudhary. JETTY: Snoop filtering for reduced power in SMP servers. In Proceedings of *International Symposium on High Performance Computer Architecture (HPCA-7)*, Jan 2001, Toulouse / France.

9. Powell, M.D., A. Agarwal, T.N. Vijaykumar, B. Falsafi, and K. Roy. Reducing set-associative cache energy via way-prediction and selective direct-mapping. In Proceedings of *34th International Symposium on Microarchitecture*, Dec. 2001, Austin / TX.

10. Seznec, A. and F. Lloansi. About Effective Cache Miss Penalty on Out-Of-Order Superscalar Processors. Technical report IRISA-970, November 1995.

11. Sherwood, T., E. Perelman and B. Calder. Basic Block Distribution Analysis to Find Periodic Behavior and Simulation Points in Applications, In Proc. of *Intl. Conference on Parallel Architectures and Compilation Techniques (PACT 2001)*, Sept 2001. Barcelona, Spain.

12. SimpleScalar LLC. SimpleScalar Home Page. http://www.simplescalar.com

13. Sohi, G. and M. Franklin. High Bandwidth Data Memory Systems for Superscalar Processors. In. Proc. of 4th *International Conference on Architectural Support for Programming Languages and Operating Systems*, April 1991.

14. Standard Performance Evaluation Corporation. SPEC CPU2000 V1.2. http://www.spec.org/osg/cpu2000

15. Synopsys Inc. Synopsys Design Compiler - Overview. http://www.synopsys.com/products/logic

16. Wilton, S. and N. Jouppi. An enhanced access and cycle time model for on-chip caches. July 1995.