IMPROVING MEMORY HIERARCHY PERFORMANCE IN SYNCHRONOUS DESIGN

Saurabh Rawat[1], Dr Rakesh Kumar[2], Anushree Sah[3], Sumit Pundir[4], Bhaskar Nautiyal[5]

Department of Electronics
Graphic Era University, Dehradhun (India)

ABSTRACT

This paper surveys techniques used to improve memory hierarchy performance in the synchronous design. A number of commercial cache implementations are also described and discussed. The paper concludes by discussing candidate techniques that need to be studied in more detail for use in an asynchronous environment.

I INTRODUCTION

1 Measuring Performance

Memory hierarchy performance, represented by the average memory access time (T_{avg}) can be calculated as

$$T_{avg} = T_{hit} + (M \times T_{penalty})$$

Where T_{hit} is the cache access (or hit) time, M is the miss rate and $T_{penalty}$ is the cost incurred on a miss (the miss penalty).

Improving memory hierarchy performance is all about reducing the average memory access time (T_{avg})

This can be achieved by:

- Reducing the cache access time T_{hit} ,
- Reducing the cache miss rate M, thus change the ratio of low cost (T_{hit}) hits to expensive ($T_{penalty}$) misses ,
- Reducing the miss penalty $T_{penalty}$,
- Hiding read write latency, usually through increased concurrency ,
- Reducing memory traffic, causing less contention for main memory between the processor and other peripherals and giving the added bonus of reduced power consumption. First obvious technique for reducing processor memory traffic is caching itself.

II REDUCING CACHE HIT TIME

Hit time is critical to performance since it affects the majority of memory references and memory latency is often the limiting factor on system performance (and so clock frequency).An integrated cache on the same die as the

processor can support high bandwidth and low latency memory access by using a wide interface and eliminating the delay of pads and buses that arises with off chip access. Furthermore, on chip caching also decreases energy consumption in the memory system due to the reduction in off chip accesses. However, on chip area is often limited and so caches have to be small. The less control involved in a cache's implementation, the shorter the delay in the critical path through the hardware. Small and simple caches are ideal in this respect and a direct mapped cache is suggested to reduce hit time, since tag checking can be overlapped with the data access. However the major disadvantage of small and simple caches is that they are more likely to suffer from higher miss rates i.e. T_{hit} is reduced but M increases.

III REDUCING CACHE MISS RATE

To reduce the miss rate, some of the misses due to three C's –compulsory, capacity and conflict, must be eliminated. Compulsory misses are caused by loading data into an empty cache and are therefore unavoidable. It is possible to change mixture of miss types, for example: compulsory misses can be converted to conflict or capacity misses by changing cache size or other parameters. There are many parameters that can be adjusted to control the mixture of conflict, capacity and compulsory misses.

a. Larger Cache Size – to reduce the number of capacity misses is to increase cache size, but changing size also affects number of conflict misses. The total cache size can be increased by having more cache lines (of the same size), by enlarging the cache line whilst fixing the number of lines.

b. Longer Cache Line – Using short cache lines for a given cache size provide a lower miss penalty since less data is required to be fetched into the cache for each line fetch. Longer lines take better advantage of spatial locality, decreasing the number of compulsory misses since subsequent requests could become hits in the previously fetched, long cache line.

c. Higher Degree of Associativity – Direct mapped caches usually suffer from a large number of conflict misses. A direct mapped cache of size N has about the same miss rate as a 2- way set associative cache of size N/2, higher associativity with the same cache size can improve the cache hit rate.

d. Better Replacement Strategies – More effective replacement strategies allow associative caches to obtain further reductions in the number of conflict misses. The perfect replacement strategy is to reject the line that will be needed furthest forward in time.

e. Victim Cache – Jouppi proposed the victim cache, a small fully associative cache, as a technique to reduce the number of misses in a direct mapped main cache. The victim cache holds lines ejected from the main cache along with their corresponding addresses; these lines are buffered to be written into memory.

IV REDUCING CACHE MISS PENALTY

Cache misses usually occur rather infrequently compared to cache hits and furthermore, some cache misses can be eliminated by the techniques described earlier. However, the miss penalty related to main memory speed, which is likely to be slow compared to the speed of the cache or processor.

For example, with a miss rate of 10% in a 4-word line size cache with 10ns access time connected to a 50 ns access time main memory, the average memory hierarchy performance is T_{avg} + 10 ns + (0.1 X (50 ns x4)) = 30 ns according to the equation 3.1. T_{avg} is here dominated by the miss penalty as is usually, but not always, the case.

4.1 Giving Read Misses Priority Over Writes

The first miss penalty reduction technique is to give priority to read misses over write operations (also known as read-overtake-write). To implement this technique a write buffer is required to hold write data whilst read are allowed to proceed.

4.2 Line Fetch Mechanism

The conventional stall on miss line fetch scheme shown in figure XX can be improved upon in a number of ways as illustrated.

4.2.1 Early-restart

Early-restart allows the processor to obtain the required word as soon as the requested word is fetched. However, the processor still has to wait for an appreciable time for the required data – the worst-case is when the required word is the last word of the fetched line.

4.2.2 Requested-word-first

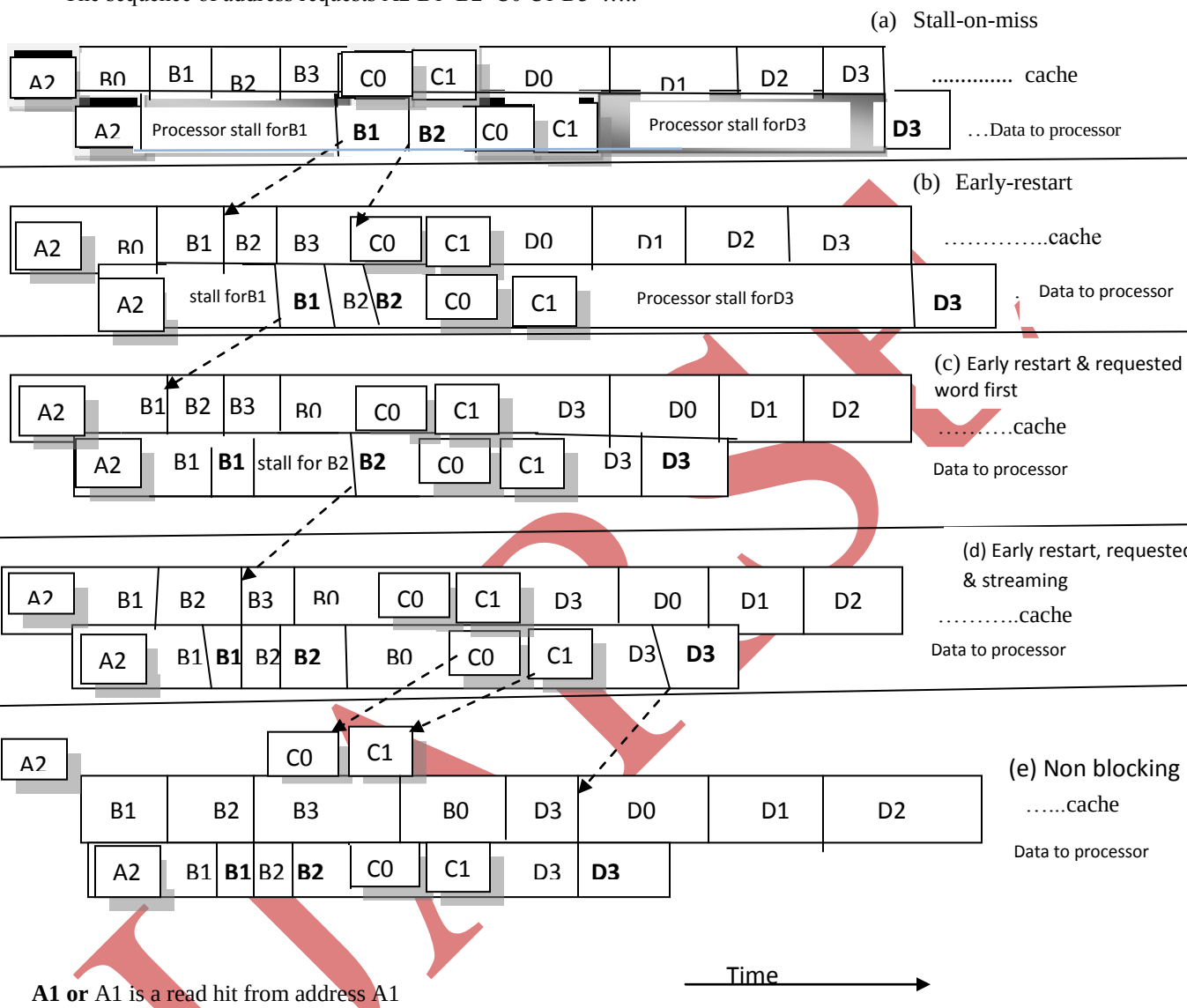
With the commonly used requested-word-first technique (also known as critical-word first or wrapped fetch), the required word is retrieved from main memory first followed by the other words in the line. Although employing the early-restart method with the requested-word-first technique shown in figure shortens the processor stall for the requested data, the processor still has to wait until the entire line fetch is completed before continuing with other read/write operations.

Figure 1 . Comparisons of line fetch schemes

Write requests may be special cases for some write hit policies. In a write through cache, writes are ‘fire and forget’ operations which are unaffected by line fetches. However, writes in a copy back cache and reads with either write hit policy may:

i. Cause a cache miss requiring a new lime fetch;
ii. Be to a word in a line that is still being fetched;
iii. Access a line that is already in the cache

The sequence of address requests A2 B1*B2*C0 C1 D3*…..



A1 or A1 is a read hit from address A1

A1 Or A1* is a read miss from address A1

V STREAMING

Streaming allows concurrency between the current line fetch process and processor accesses to other words of the same cache line being fetched. This behavior is useful in many cases such as long instruction fetch sequences. In case ii above, the processor can obtain the required data as soon as it arrives in the cache. However, subsequent requests may result in case iii. With this method, accesses to words other than those in the fetched line still have to wait for the line fetch process to complete before proceeding.

VI NON BLOCKING

If on a cache miss, the cache cannot continue to serve the processor until the required word was received from the lower level in the memory hierarchy, then this cache is a blocking cache. The blocking cache can be thought of as an in order cache design; data arrives at the processor in the same order that it was requested. Kroft's scheme, first known as lockup free and also known as non blocking combines the requested word first, early restart and streaming techniques to allow the processor to continue concurrently with a fetch line process proceeding in the background. The situation described in case iii above known as hit under miss can be exploited here where cache has the ability to work on other hit requests, waiting only for memory to supply further misses. A non blocking cache can be thought as an out of order cache design, by analogy with an out of order processor design where the processor does not have to execute instructions in the same order that they were fetched. When implementing streaming or non blocking in an asynchronous design environment, processor requests must be synchronized with the incoming fetch data to guarantee that required data present in the cache. This is simply done by having an extra valid bit for each data word.

VII HIDING LATENCY

Techniques for coping with memory latency are essential to achieve high processor utilization. Such techniques will become increasingly important in the future as the gap between processor and memory speeds continues to widen. Although latency hiding does not (directly) decrease the time taken for a hit or a miss, it potentially increases overall system throughput.

The latency of writes can be hidden by buffering write accesses with a writer buffer. This technique exploits the fact that a processor does not have to wait for a write to complete as long as it observes the effect of future written data. Therefore the processor can perform a write by simply issuing it to the write buffer, provided that future reads check the write buffer for matching addresses. The advantage of a write buffer is not only that the processor does not stall when executing a write, but also that multiple writes can be overlapped to exploit pipelining.

Buffering read accesses is more difficult because, unlike writes, the processor cannot proceed until the read access completes since it needs the data that is being read. With a non-blocking cache it is, however, possible to buffer and pipeline reads.

This section discusses two basic hardware techniques for tolerating memory latency prefetching and pipelining. These techniques complement the non-blocking cache by allowing concurrency, thus reducing average read latency.

7.1 Prefetching

Prefetching involves fetching data from the memory before it is actually needed by the processor. This technique hides the line fetch latency, reducing the miss penalty, if subsequent accesses can be serviced with this perfetched data. Prefetching can be done by using either software or hardware approaches. Software prefetching uses the

compiler to transform the code, usually by adding extra explicit fetch instructions to instruct the hardware which information is to be prefetched. This approach is not considered here.

Hardware-based prefetching, on the other hand, relies on either simple prefetch techniques that fetch a fixed pattern of data or more sophisticated techniques that approximate memory access patterns dynamically.

7.2 Pipelining

Pipelining is an implementation technique that exploits parallelism and is one of the most common techniques used to improve the performance of processors. It comes from the observation that instruction execution can be split into a number of independent stages chained into a pipeline, allowing a number of instructions to be operated upon concurrently, one in each different stage of execution. Pipelined processing is beneficial when all of the following are true:

Each task is relatively independent from the previous one.

Each task requires approximately the same sequence of stages.

The durations of time required by each of the different stages are approximately equal. (For asynchronous pipelining, the time per stage may not be constant but rather a function of both the stage and the data passing through it.)

VIII REDUCING MEMORY TRAFFIC

This section describes techniques for improving memory hierarchy performance by reducing the total memory traffic. All of the earlier techniques reduce latency related stalls but also increase the traffic between main memory and the processor. The main benefit of reducing the traffic is the power saved by not going off-chip to access external main memory, but lessening the traffic could also aid in reducing the miss penalty. Two common techniques write merging (X) and copy-back, are discussed in this section.

8.1 Write merging

Cache lines are usually larger than the size of any single piece of write data. Many modern write buffers have the ability to merge memory writes to save both write buffer space and memory traffic. This can be done by bringing together a new write operation with a previous write operation already resident in the write buffer. The new write is placed in the same write buffer entry as an existing write when the address of the new store falls inside the line address range of the existing entry. By this means two or more writes to the same location can be collapsed into one write or two or more writes to sequential locations in the same cache line can be merged into a single buffer entry and then written out using a high speed memory burst.

8.2 Copy-back write policy

The fundamental cache activities affecting write policies are reviewed in figure XX. At the beginning of each access is a comparison to determine whether the request is to a cacheable location. Uncacheable instruction or data accesses are passed on directly to the system bus and the operation (read-write) performed on the main memory.

The least complicated operation is a read hit in the cache when the data is simply read out straight from the cache and sent to the processor. However, in some (multi-level) cache systems an extra activity might be required to update the higher level in the cache system. A read miss is slightly more complicated: the line fetch process fetches the required data from the main memory (or a lower level in the memory hierarchy) along with data close to it. Whilst activities on a read access are fairly simple, there are several issues (policies and techniques) involved in a write, some of which need to be decided at a very early design stage.

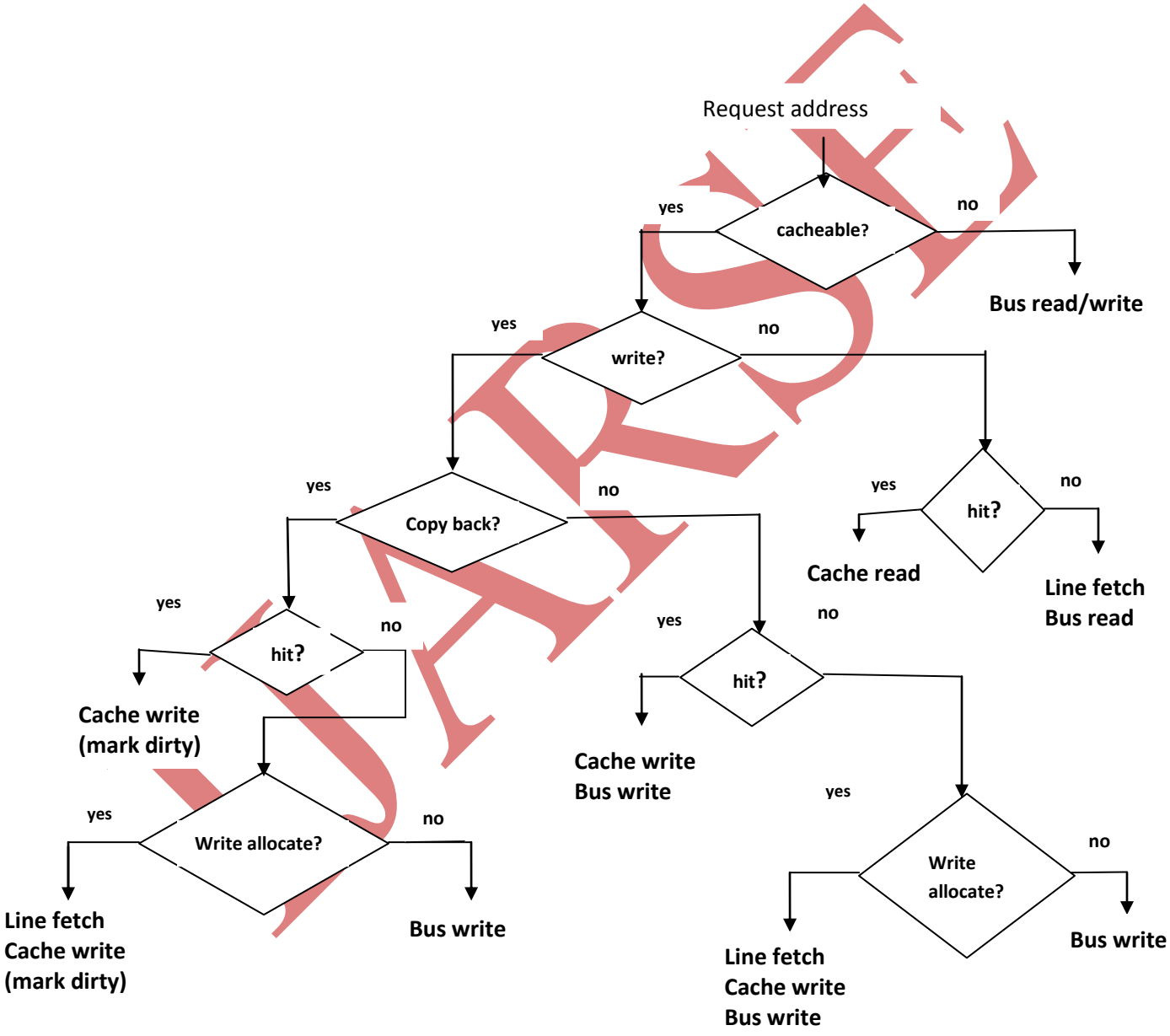


Figure 2 . Basic Bus operations

A short theoretical analysis shows how dramatic reductions are generated by a copy-back policy:

$$T_{avg} = T_{RH} + T_{RM} + T_{WH} + T_{WM}$$

Where $T_{RH}, T_{RM}, T_{WH} and T_{WM}$ are the contributions of cache read hit, read miss, write hit and write miss consecutively. The read hit contribution in both write-through and copy back caches are:

$$T_{RH} = R \times H \times T_{hit}$$

Where R represents the fraction of total read accesses over all accesses (for instruction and data) and H represents the hit rate.

Whilst a write hit in a copy- back cache only has to proceed in the cache, in a write-through cache a write operation must be performed in the main memory. The write hit contribution for write-through and copy-back caches can then be derived respectively as a follows:

$$T_{WH(write-through)} = W \times H \times T_{MEMwrite}$$
$$T_{WH(copy-back)} = W \times H \times T_{hit}$$

Where W represents the percentage of total write accesses, $T_{MEMwrite}$ is the time taken to update the main memory with the assumption that $T_{MEMwrite} \rangle\rangle T_{hit}$. Since a line fetch occurs on a read miss, the read miss contribution in a write-through cache is simply.

$$T_{RM(write-through)} = R \times M \times (T_{Rpenalty} + T_{hit})$$

Where $T_{Rpenalty}$ is the miss penalty for fetching a line? A line allocation in a copy-back cache involves a line eviction (needing to write dirty data back to the memory). Therefore, in the absence of a write buffer (either for decoupling the processor and the main memory or for decoupling copy-back allocation) the read miss contribution is given by:

$$T_{RM(copy-back)} = R \times M[(T_{Rpenalty} + T_{hit}) + (D \times T_{Wpenalty})]$$

Where D represents the percentage of dirty data amongst the evicted lines and $T_{Wpenalty}$ is a miss penalty for updating the main memory with a dirty evicted line and is directly related to $T_{MEMwrite}$

In a write-through cache (assuming a write-around policy) a write miss contribution is similar to that of the write hit in the same cache, however, in a copy-back cache (with a write-allocate policy), it is similar to that of the read miss in the same cache. Write miss contributions are thus:

$$T_{WM(write-through)} = W \times M \times T_{MEMwrite}$$
$$T_{WM(copy-back)} = W \times M[(T_{Rpenalty} + T_{hit}) + (D \times T_{Wpenalty})]$$

Simplifying the above formulae gives the memory hierarchy performance of write-through and copy-back caches respectively as:

$$T_{avg(write-through)} = R[T_{hit} + (M \times T_{Rpenalty})] + (W \times T_{MEMwrite})$$
$$T_{avg(write-through)} = T_{hit} + M[T_{Rpenalty} + (D \times T_{Wpenalty})]$$

For a read :write access ratio of 9:1 with a miss rate of 5% in a 4-word line size cache that has a 10 ns access time and is connected to a 50 ns access time main memory, and assuming that 10% of evicted lines are dirty in the copy-back cache with no write buffer, these give:

$$T_{avg(write-through)} = 0.9 \times [10ns + (0.05 \times (50ns \times 4))] + (0.1 \times 50ns) = 23ns$$

$$T_{avg(copy-back)} = 10ns + 0.05 \times [(50ns \times 4))] + (0.1 \times (50ns \times 4))] = 21ns$$

When a non-blocking scheme is applied, $T_{Rpenalty}$ can be reduced from 50ns X 4 + 200 ns to 50 ns, a miss penalty only for the required word thus reducing both $T_{avg(write-through)} and T_{avg(copy-back)}$ to 16.25 ns and 13.5 ns respectively. Furthermore, when a memory bursting mode (2-1-1-1) is applied, $T_{Wpenalty}$ can also be then reduced from 200ns to (50 ns + 25 ns +25 ns+25 ns)= 125 ns. Overall $T_{avg(copy-back)}$ is reduced to 13.1 ns with 20% improvement over the write-through cache.

IX OTHER NOTABLE TECHNIQUES

Two other techniques are commonly encountered in cache systems. Neither of these has a direct impact on a cache's general –purpose average performance but each offers specific benefits of common interest.

9.1 Sub-blocking

An architecturally different cache organization strategy to reduce cache power dissipation is to break a cache data array into multiple sub-blocks. Only the cache sub-block where the requested data may be located is addressed for each cache access. This technique saves power by making each access across a smaller cache. The proportion of power saved depends on the number of cache sub-blocks. Sub-banking as it is also known, is very attractive to computer architects designing energy-efficient microprocessors.

9.2 Cache lock-down

Since caches are transparent to user software, predicting the exact performance of a program in a system with a cache is difficult. This is an undesirable effect in many embedded systems which require real-time response. A technique commonly used in embedded systems to ensure deterministic behavior is to load critical code into the cache embedded systems to ensure deterministic behavior is to load critical code into the cache under supervisor software control and then, via special hardware support, prevent it from being evicted. This process is known as cache lock-down. Clearly, locking down most of the cache compromises its ability to acceler4ate the general performance of the machine, so it is important to have control of the lock-down mechanism at a fine granularity.

X CONCLUSION

The above cache system exemplifies the cache architectural techniques presented in synchronous implementations. This paper considers the possibility of using these techniques in the context of an asynchronous framework. The idea of multiple levels of cache is attractive in the context of asynchronous design where a wider variation in access time can be exploited in a manner that would prove expensive and difficult in a synchronous framework. This is because each unit in a synchronous system must complete its task in an integer number of clock cycles. The literature surveyed in this paper concentrated mostly on the architectural level hardware-based techniques that use alternative cache organizations for improving memory hierarchy performance. Improving one aspect of the cache performance usually comes at the expense of others.

Most of these techniques can be easily applied in an asynchronous system. However, non-blocking, read-overtaking-writes, forwarding and write-merging would present problems since they all require some degree of undesirable synchronization.

REFERENCES

[1] Jouppi, N. P. *Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers.* Proceedings of 17th Annual Int'l Symposium on Computer Architecture, 1990, pp. 364-373.

[2] Przybylski, M. Horowitz, and J. Hennessy.*Characteristics of performance-optimal multi-level cache hierarchies*. Proceedings of the 16thInternational Symposium on Computer Architecture, 1989, pp. 114 –121.

[3] Baer, Jean-Loup, and Wang, Wen-Hann. *On the inclusion properties for multi-level cache hierarchies*. 25 years of the international Symposia on Computer Architecture (selected papers), 1998,pp. 345 – 352

[4] Gee, et al.. *Cache Performance of the SPEC92Benchmark Suite.* IEEE Micro, Vol. 13, Number 4, August, 1993, pp. 17 – 27.http://www.cs.wisc.edu/~markhill/spec92miss.html

[5] Johnson, Teresa L., and Hwu, Wen-mei W. *Runtime Adaptive Cache Hierarchy management via Reference Analysis.* ISCA '97, Denver, CO, USA, pp. 315 – 326.

[6] Jouppi, Norman P, and Wilton, Steven J. E. *Tradeoffs in Two-Level On-Chip Caching.* Research Report 93/3, October 1993, Compaq Western Research laboratory.

[7] Steven Przybylski, Mark Horowitz, John L.Hennessy. *Performance Tradeoffs in Cache Design.* ISCA 1988: Honolulu, Hawaii, USA, pp. 290 – 298.

[8] Cheriton, D. R., Goosen, H. A. and Boyle, P. D.*Multi-level shared caching techniques for scalability in VMP-M/C.* Proceedings of the 16th annual International Symposium on Computer Architecture, 1989, pp. 16–24.

[9] Short, Robert L., Levy, Henry M. *A Simulation6Study of Two-Level Caches*. Proceedings of the 15th Annual Symposium on Computer Architecture. May 1988, pp. 81–88.

[10] Wan, Marlene, and George, Varghese. *Effect of Second Level Cache Parameterising Overall CachePerformance*.http://infopad.eecs.berkeley.edu/~varg/CS252_report /Final.html

[11] Hill. *A case for Direct Mapped Caches*. Computer,21:12. 1988, pp. 25-40.

[12] Smith, A. J. *Cache Memories*. Computing Surveys,14:3. September, 1982, pp. 473-530.