A Discussion on Non-Blocking/Lockup-Free Caches

Samson Belayneh and David R. Kaeli
Northeastern University
Department of Electrical and Computer Engineering
Boston, MA 02115

Abstract

Cache memories are commonly used to bridge the gap between processor and memory speed. Caches provide fast access to a subset of memory. When a memory request does not find an address in the cache, a cache miss is incurred. In most commercial processors today, whenever a data cache read miss occurs, the processor will stall until the outstanding miss is serviced. This can severely degrade the overall system performance.

To remedy this situation, non-blocking (lockup-free) caches can be employed. A non-blocking cache allows the processor to continue to perform useful work even in the presence of cache misses.

This paper summarizes past work on lockup free caches, describing the four main design choices that have been proposed. A summary of the performance of these past studies is presented, followed by a discussion on potential speedup that the processor could obtain when using lockup free caches.

1. Introduction

As the gap between processor and memory speeds continues to grow, new techniques are needed to limit the cost of cache misses. One technique that has recently been discussed in the context of scalable shared memory multiprocessors is using a lockup free cache. The concept behind this type of cache is to be able to process cache accesses in the presence of cache misses. Then the only time the processor would have to stall on a data cache read is when the operand that is being read is needed in a subsequent instruction or when the miss bypassing hardware is not able to handle any more misses (a lockup free cache generally has a finite number of misses that can be handled). As long as the processor does not encounter data dependencies, and as long as there is sufficient hardware to handle the maximum number of concurrent misses outstanding, the processor should not have to stall during memory loads.

In this paper we will focus on lockup free data caches. Most past discussions on lockup free caches have been targeted to data caches, and specifically to handling read misses. It is not clear how a lockup free cache could benefit the instruction stream.

In section 2, we will provide some background on lockup-free caches. In section 3, we will describe four lockup-free cache organizations. In section 4, we discuss the performance advantages of using a lockup-free cache, and in section 5, we summarize our discussion on non-blocking caches.

2. Background

The first work known to the authors on lockup free caches was presented by Kroft [1], where he described special registers called Miss Status/Information Holding Registers (MSHR's) which hold information about outstanding misses. These special registers, along with the necessary control logic, contain

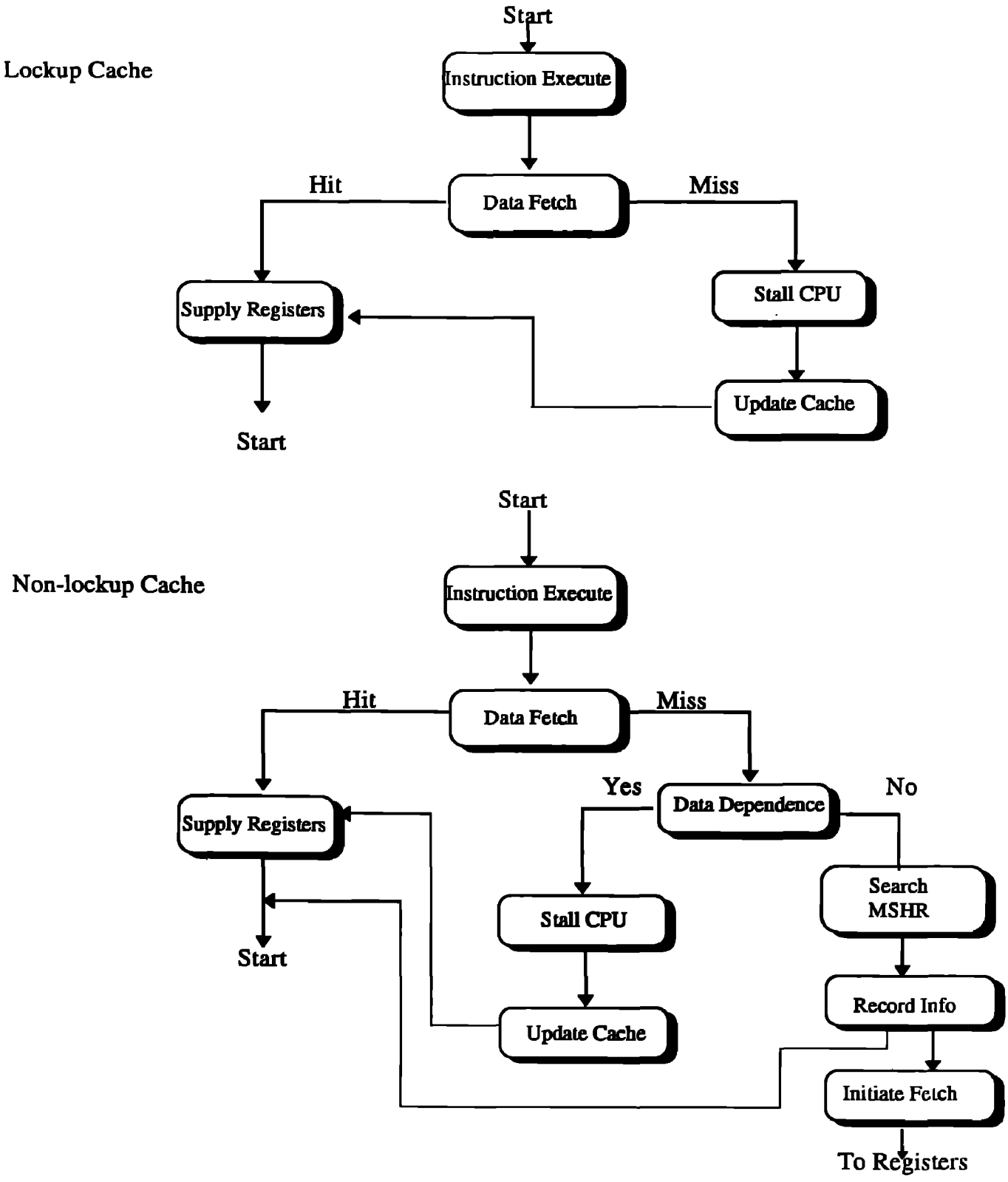

Figure 1. Flow Diagrams of a Lockup and Non-lockup Cache - Processor Organization.

enough information to enable the processor to overlap the processing of a cache read miss with the processing of subsequent instructions. Figure 1 provides flowcharts that attempt to illustrate some of the fundamental differences between a non-lockup free cache and a lockup free cache. Kroft showed that the number of useful MSHR's decreases dramatically with an increasing number of MSHR's (see Figure 2 below). This is because the amount of concurrent computation that can be performed, before encountering an instruction which is dependent on the data for which a miss is being serviced, is limited. We discuss this point further in section 4.

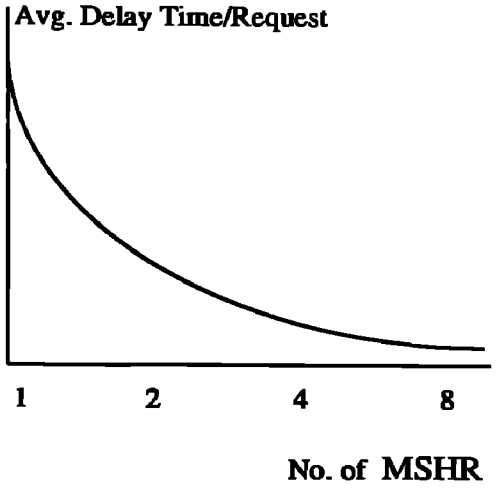

Figure 2. Performance vs. Number of MSHR.

Sohi and Franklin have studied the effectiveness of increasing the number of MSHR's using an analytical model. In their analysis they concluded that the optimal number of MSHR's is four. This result is somewhat workload dependent, but agrees with simulations that we have performed using the ATOM simulator [2] running the SPEC92 benchmark suite.

In addition to MSHR's, a lockup free cache must include some pipeline scoreboarding to provide the pipeline with information regarding outstanding misses. Using such a mechanism, the processor can track the outstanding memory references that need to be satisfied.

While non-blocking caches can be used for uniprocessor caches, their real benefit can be reaped for shared-memory multiprocessor machines [3]. Non-blocking caches are especially useful for multiprocessors since, generally, the latency on a miss is larger in multiprocessor systems. However, the memory and interconnect systems can often support multiple outstanding accesses. This should help alleviate some of the higher bandwidth requirements needed for a non-blocking cache design.

3. Organization of Non-Blocking Caches

There are several schemes that can be used to organize a non-blocking cache. Farkas and Jouppi [4] describe four organizations that have different cost/performance ratios.

In the simplest organization, an MSHR entry is used to hold such information as the block valid bit, block request address, word valid bit, word destination field and the format field (see Figure 3). Each MSHR has its own comparator so that all MSHR entries can be searched associatively on a miss.

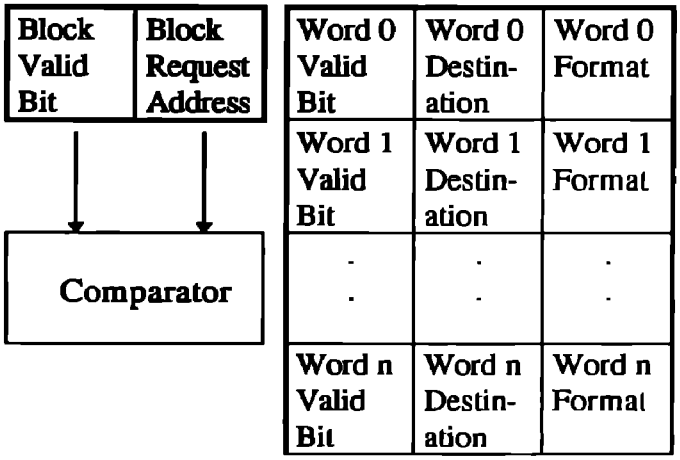

Figure 3. A Simple MSHR organization (as described in [4]).

In this organization, one MSHR entry can handle only one miss to a particular block (a single *hit-under-miss*). Any subsequent miss to the same word in a block causes the processor to stall.

A more sophisticated organization described in the literature is an explicitly addressed MSHR, where multiple misses to the same cache block can be tolerated without stalling the processor. This organization, while structurally similar to the simple organization, is potentially better since multiple misses to the same cache block can be handled concurrently.

Another MSHR organization proposed is an in-cache MSHR [5]. In-cache MSHR's try to reduce the extra hardware used for searching a non-blocking organization. The cache block that causes the miss is an effective MSHR, keeping all information on the miss in the cache block. The disadvantage of this organization is that in direct-mapped caches (which are commonly used due to their simplicity), there can only be one in-flight miss that can be supported at a time. Also, there needs to be a transit bit in the cache that indicates that the block is being used for holding MSHR information and is not a valid cache entry. The down side of this is, in a large cache, using extra cache bits for managing cache misses may require more resources than a specific set of MSHR's dedicated solely for this purpose.

Inverted MSHR's are the most aggressive and non-restrictive approach proposed in [4]. In this type of organization, there are as many MSHR entries as there are destinations (e.g. general purpose registers, both integer and floating-point).

Figure 4 shows the layout for an Inverted MSHR. In the inverted MSHR organization, when a miss occurs all valid information is recorded for each destination in the inverted MSHR and a cache block fill is initiated. On subsequent misses, all MSHR entries are searched to check if a miss is outstanding. If a cache block fill is already in process for this block, only the inverted MSHR entry corresponding to the destination of the requested address is recorded.

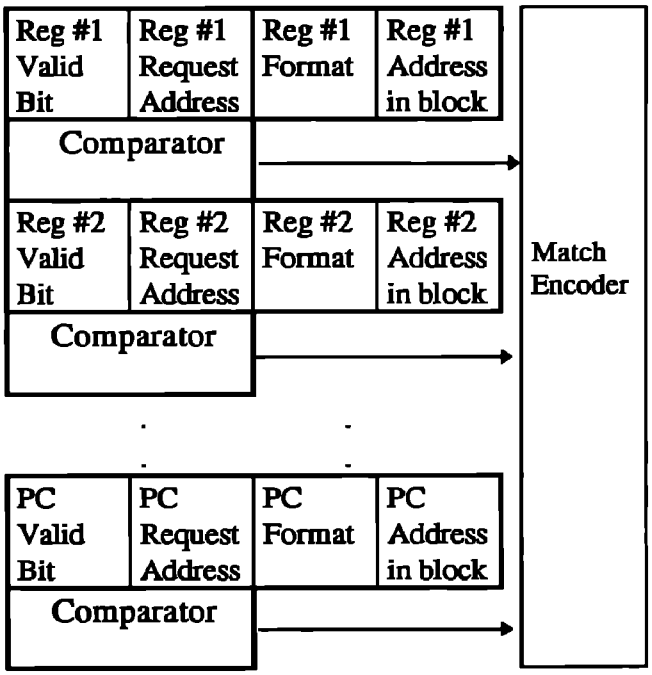

Figure 4. An Inverted MSHR Organization (as described in [4]).

When a block of data is returned from lower level memory (L2 cache or main memory), all entries in the MSHR must be probed to identify those destinations that are waiting for data. For those entries that match, each waiting destination will be filled with the data that was returned using the information contained in the MSHR fields.

4. Performance of Non-Blocking Loads and the Effect of Compiler Optimization

As opposed to prefetching and other pre-miss optimization techniques, a non-blocking load

scheme is a post-miss optimization technique. As a result, non-blocking loads do not reduce the cache miss rate. They can, however, reduce the *miss penalty* significantly depending on the organization of the non-blocking scheme used and whether there is an optimizing compiler employed that can schedule instructions away from data dependencies. In this section, we will first summarize the performance of various non-blocking (MSHR) organizations and the impact of miss penalty on the performance of non-blocking loads. Then we will consider the effect of code optimization in the presence of a non-blocking cache.

In order to study the performance improvement obtained by non-blocking loads, all store operations are assumed to be non-blocking (an infinite write buffer is assumed). Furthermore, the instruction cache is assumed to always produce a hit so that all performance improvements can be attributed solely to the non-blocking loads.

To measure the performance of non-blocking loads in isolation, the term miss CPI (MCPI) is used. All stalls in the model are due only to stalls caused by the load instructions. Hence,

MCPI = (Total Data Access Penalty/Total Number of Load Instructions)

The misses used in the above calculation are due to either the existence of a true-data dependency or outstanding misses that could not be supported because of the limits of the non-blocking hardware.

The following graph is a summary of some of the simulation results obtained in [4]. The simulation was done using the VLIW Multiflow Compiler that schedules instructions based on the load latency. The miss penalty is fixed at 16 cycles and an 8K byte direct-mapped cache with a block size of 32 bytes is used. The graph shows that for the floating-point programs (*doduc, hydro2d and tomcatv*) the performance of the more complicated non-blocking caches is much better than the simple hit under miss scheme. However, the integer benchmarks do not show such improvement for the more complex schemes. For these programs, the simpler implementation will be sufficient in hiding the majority of the latency. This indicates that the integer programs are dominated by true-data dependency stalls rather than stalls caused by structural hazards. Since stalls caused by true-data dependencies can not be hidden by the non-blocking scheme, the importance of the non-blocking caches is not as significant as in the floating-point programs.

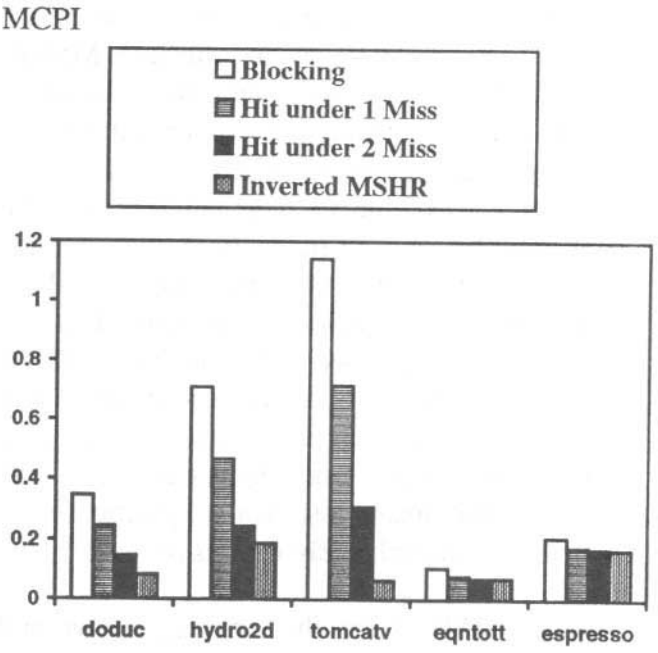

Figure 5. MCPI for Selected SPEC Benchmarks (results summarized from [4]).

Many techniques have been proposed to reduce data dependencies including multithreading, out-of-order issue and out-of-order completion. In the absence of data dependencies, performance of the non-blocking caches will be bound by the MSHR resources available.

We would like to understand some bounds on the optimal performance attainable using lockup-free caches. To do this, we model a cache using the following assumptions: a) WAR and RAW hazards can be eliminated using latency hiding techniques, and b) only WAW hazards that are the result of two loads to the same register destination will stall the processor (when the first load miss is still outstanding). Modeling these assumptions we see that past some threshold number, increasing the number of MSHR's does not improve performance. This optimal number of MSHR's is determined by the amount of structural-hazard stalls in the program.

The simulation results presented next were produced using ATOM on a DEC Alpha 21064. Instrumentation and analysis programs are custom built to study the performance of non-blocking caches. The cache model used is an 8K direct-mapped cache with a cache block size of 32 bytes.

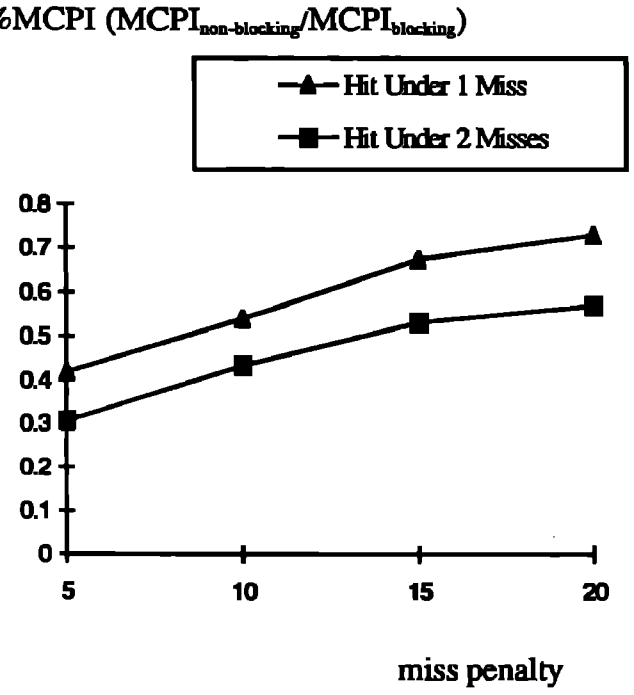

Figure 6. Average %MCPI vs. Miss Penalty.

Increasing the miss penalty will increase the probability that more instructions will be executed during a fetch operation. This will likely increase the number of in-flight misses. Furthermore, longer miss penalties aggravate the data dependency between instructions. Figure 6 shows the performance of a non-blocking cache that supports a hit-under-one-miss and a hit-under-two-misses scheme compared to that of a blocking cache. We compare these two designs, varying the miss penalty. We see that the usefulness of the non-blocking caches diminishes slightly with increasing miss penalty.

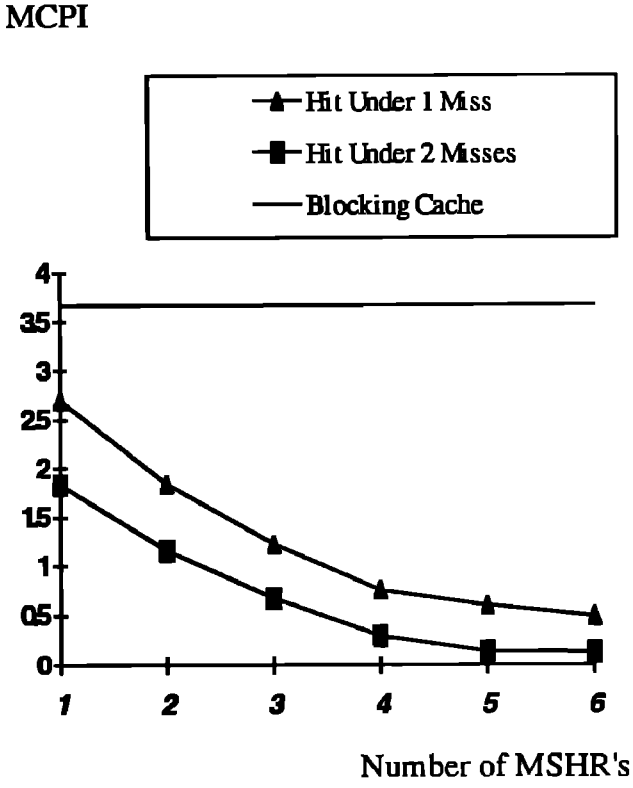

Figure 7. MCPI for *eqntott* vs. the number of MSHR's.

Figure 7 shows the performance improvement obtained for *eqntott* as the number of MSHR's is increased from 1 to 6 for a fixed cache miss penalty of 15 cycles. As we can see here, the number of MSHR's is an important parameter

that should be determined accurately to obtain the best cost/performance for non-blocking caches. In our simulation for *eqntott,* data dependent cache misses (outstanding misses to the same destination register) occurred for 29.4% of all load misses. This percentage does not decrease by increasing the number of MSHR's.

All improvements observed by increasing the number of MSHR's are due to reduction of stalls caused by structural-hazards. However, after 6 MSHR's the improvement is quite small as is evident from the graph. Obviously, the MCPI of the blocking cache remains constant and is plotted only for comparison.

4.1 Compiler Assistance

Instruction scheduling can assist non-blocking schemes significantly. The aim is to create as much distance as possible between a load instruction and the first use of the data. Algorithms have been developed to schedule instructions so that the maximum benefit can be obtained [6]. These algorithms schedule instructions within a basic block, but need to consider the effect of the reschedule on memory bandwidth (i.e. loads should not be heavily clustered).

Register renaming is also effective in removing dependencies caused by write-after-read (WAR) and write-after-write (WAW). This allows more freedom in moving around instructions for a better schedule. Since generally, the distance between set and use in an unoptimized code is small, instruction scheduling algorithms can be effective in increasing this distance, and hence further improve the performance of non-blocking techniques [6].

Figure 8 shows the performance of using these optimizations with a non-blocking cache. The MCPI is normalized to the MCPI of the original code. As can be seen, instruction scheduling and register renaming can further improve performance in most cases.

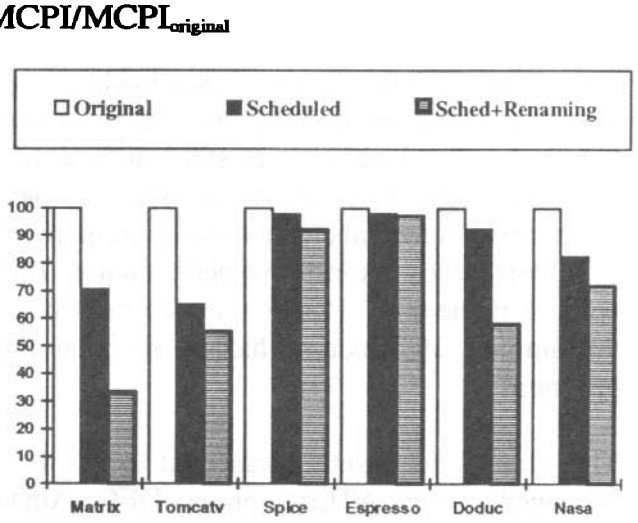

Figure 8. Effect of Instruction Scheduling [6].

5. Conclusion

Non-blocking caches hide the latency associated with cache misses by allowing the processor to execute instructions until data dependency or structural hazards make it necessary to stall. As a result, misses can be outstanding. Extra hardware resources are required to handle outstanding misses. MSHR's are used so that a data request can be performed concurrently with the execution of other instructions. We have looked at a number of ways that have been proposed to organize non-blocking caches.

As the number of MSHR's is increased the complexity and cost of the non-blocking scheme increases quite rapidly hence there is a tradeoff that has to be made in choosing a particular organization. The simple hit-under-miss organization is shown to be the most cost effective for most of the integer benchmarks while the more complex organizations would be more likely to improve the MCPI of the

numeric benchmarks.

We also considered the performance of a non-blocking cache in the absence of data dependencies. Our results indicate as miss penalty increases, loads to the same register destination will tend to dominate. Increasing the number of MSHR's will give diminishing returns.

Compiler optimization using instruction scheduling algorithms and register renaming can further improve the performance of non-blocking caches. Non-blocking caches when used with compilers that reschedule code have shown to provide the best performance.

Non-blocking caches have been shown to be effective in improving the performance of a system. However, the cost of implementing non-blocking caches can impose limitations on their application. As processors become more powerful, it is likely that non-blocking caches will be commonly used. This should motivate future research directed towards improving the design implementation of lockup-free caches.

References

1. David Kroft. Lockup-Free Instruction Fetch/ Prefetch Cache Organization. Proceedings of the 8th Int. Symp. on Computer Architecture, May 1981, pp. 81-87.

2. Amitabh Srivastava and Alan Eustace. ATOM: A system for Building Customized Program Analysis Tools. Proceedings of the ACM SIGPLAN '94 Conference on Programming Languages, March 1994.

3. D. Lenoski, J. Laudon, K. Gharachorloo A. Gupta and J. L. Hennesy. The Stanford DASH Multiprocessor. Proceedings of the 17th Int. Symp. On Computer Architecture, June 1990, pp. 148-159.

4. Keith I. Farkas and Norman P. Jouppi. Complexity/Performance Tradeoffs with Non-Blocking Loads. Proceedings of the 21st Intl. Symp. on Computer Architecture, pp. 211-222, 1994.

5. Manoj Franklin and Gurindar Sohi. Non-Blocking Caches for High Performance Processors. Unpublished, 1991.

6. Tien Fu Chen and Jean-Loup Baer. Reducing memory Latency via Non-blocking and Prefetching Caches. Fifth ASPLOS Conference, October 1992, pp. 51-61.

7. Keith I. Farkas, Norman P. Jouppi and Paul Chow. How Useful Are Non- blocking Loads, Stream Buffers, and Speculative Execution in Multiple Issue Processors? Western Research Laboratory, December 1994.

8. John H. Edmondson and et al. Internal Organization of the Alpha 21164, a 300-MHz 64-bit Quad-issue CMOS RISC Microprocessor. Digital Equipment Corp. 1995.