Tradeoffs in Processor/Memory Interfaces for Superscalar Processors

Thomas M. Conte
Department of Electrical and Computer Engineering
University of South Carolina
Columbia, South Carolina 29208
conte@ece.scarolina.edu

1 Introduction

The current scheme of dealing with data cache misses is not well-suited for superscalar processors. In this scheme, the processor is blocked by holding its clock low until the missing cache block can be fetched from memory and inserted into the cache. From the processor's viewpoint, the miss did not occur. From the user's viewpoint, the execution time was lengthened in direct proportion to the number of cache misses. This scheme has the potential of reducing the parallelism of superscalar processors with high issue rate to sub-unity values.

The above scheme can be termed *blocking.* Two less-restrictive schemes are possible for interfacing processors to the first level of the data memory system:

Limited blocking: In this scheme, when a miss occurs, any subsequent access to the cache is frozen, as in the *blocking* scheme. The processor clock is not halted. Rather, the processor is allowed to continue to execute other instructions that do not access memory. If sufficient independent instructions are ready to execute, this scheme can effectively hide the performance impact of the cache miss. Limited blocking was suggested in [1].

Non-blocking: In this scheme, the cache is modified to update its contents on a cache miss while it continues to service requests for data that is present in the cache. Such a design is termed a *non-blocking cache of degree n* if it can service n outstanding misses while continuing to service requests from the processor. When greater than n misses are outstanding, this scheme reverts to the *limited blocking* scheme. Non-blocking caches were introduced by Kroft in [2].

There have been several studies of the *non-blocking* scheme besides the original work by Kroft. Sohi and Franklin made several observations concerning the design of non-blocking caches in a two-level hierarchy [3]. They also asserted that the *limited blocking* scheme was identical in performance to *blocking.* Johnson performed comparisons between the same two schemes and concluded the opposite, that *non-blocking* and *limited-blocking* had similar performance [1]. What complicates these studies is the miss rate of the data cache itself. It is difficult to use a set of benchmarks with varying cache needs with a fixed-sized cache and achieve the same miss rate for each benchmark. Yet a fair study should fix the miss rate, not the cache size, and evaluate the three schemes on a level playing field. Otherwise, the prospect of making a false conclusion as a result of a poorly designed cache is too great.

An additional problem with evaluating processor/memory interface schemes comes from the other side of the interface, the processor. It is difficult to remove the artifacts of the design of the processor from the performance of the interface schemes. A crippled processor that does not efficiently exploit instruction-level parallelism will not tax any of the schemes sufficiently. Any conclusions of the relative merits of each scheme would be tainted by this poorly designed processor.

This paper addresses the relative merits of the three processor/memory interface schemes by constructing a fair comparison between the schemes. Six members of the SPEC89 benchmark set are used [4]. For each benchmark (where possible) a cache size is selected that achieves a fixed miss ratio. Miss ratios of 5% and 10% are used for the study. This effectively removes the fixed cache size problem and replaces it with *fixed cache performance.*

The processor is a superscalar, full-Tomasulo out-of-order execution engine that issues multiple instructions per cycle [5]. Issue rates of two, four, and eight instructions per cycle are considered. The function units are given an aggressive set of latencies, but the number of function units are left unbounded. This decision removes the side-effect of at least one aspect of processor design: function unit selection. The resultant processor exploits the highest-achievable degree of parallelism, emulating a high-quality processor design. The reservation stations are generic. They can be used by any function unit. This forms a *scheduling window* of reservation stations. Results are presented below for a window size of 32 entries and for an unlimited window size.

The penalty for a cache miss, T_{MISS}, the number of cycles it takes for a cache miss to be repaired, can skew results if it is not varied. This study considers two values of T_{MISS}, 10 cycles and 20 cycles. It is found that this parameter has a large effect on the performance of the two schemes.

The following section briefly introduces the trace collection and simulation methods. The results are presented in the third section of the paper and conclusions are drawn concerning the relationship between the three schemes for processor/memory interfacing.

2 Simulation Methods

The benchmarks used in this paper are six of the 10 members of the SPEC89 benchmark suite [4]. These members are: *doduc, eqntott, espresso, gcc, matrix300* and *xlisp*. The traces for the results presented below were collected using the Spike tracing tool fitted into the GNU C compiler (version 1.40). Profiling has shown that a large portion of the execution of several of the SPEC benchmarks occurs inside Unix library code. These libraries were also instrumented and included as parts of the benchmarks. The instruction set assumed for this work is derived from the intermediate code of the GNU C compiler.

The primary metric of processor performance used in this paper is *parallelism*, or *instructions per cycle (IPC)*. This is the expected number of instructions in the execution stage of the processor for a cycle of a benchmark's execution.

A benchmark that runs for 90 seconds on a 20 million instructions/second machine executes 1.8 billion instructions. The simulation time required for such a task is too great to be tractable. Improving the performance of the simulation allows larger workloads to be used. One simulation approach that has appeared in studies of instruction-level parallelism tradeoffs is to use the first four to 10 million instructions [1]. Such an approach has the problem that the first few million instructions might only capture the initialization phase of the benchmark and not the portion of the benchmark that performs actual work.

Instead of the above approach, this paper advocates a statistical sampling approach [5]. The results presented herein are based on taking 40 samples of size 10,000 instructions from the trace. The gap between samples is typically 10 million instructions, although this gap is varied per benchmark to guarantee that samples are taken from the majority of the execution of each benchmark. *It is not the purpose of this paper to validate the sampling approach.* This sampling regimen has been shown to be accurate in capturing the behavior of the full trace, yielding *relative* errors between 2% to 13% for the instructions per cycle metric of a processor with a relatively high issue rate of eight instructions/cycle and no cache miss penalty (see [5]). A multiprogramming environment is assumed for the simulations to make the results more realistic. Multiprogramming assumptions fit naturally with sampling. For these simulations, the multiprogramming quantum is set equal to the sample size.

Table 1: Cache sizes required to achieve $\hat{\rho} = 0.05, 0.10$.

	To achieve:	
Benchmark	$\hat{\rho} = 0.05$	$\hat{\rho} = 0.10$
doduc	32KB	8KB
eqntott	8KB	2KB
espresso	32KB	4KB
gcc	64KB	8KB
matrix300	*	128KB
xlisp	8KB	4KB

(* data cache design criteria could not be met)

Cache designs are taken from a large design space of potential caches and selected to be the smallest cache having a miss ratio, ρ, such that $\rho \leq \hat{\rho}$. The values of 0.05 and 0.10 for $\hat{\rho}$ are used. The caches are direct mapped and the block size is 32 bytes. The write policies are write back/write allocate. The caches are designed to perform in a multiprogramming environment. Note that this does not impact the achieved miss ratio for the caches, the caches are selected taking the multiprogramming load into account. The specific cache sizes for each target miss ratio are listed in Table 1. Some benchmarks require much higher cache sizes for the miss ratio criterion of $\hat{\rho} = 0.05$. One benchmark (matrix300) accesses a sufficient number of unique memory locations such that $\hat{\rho} = 0.05$ cannot be achieved at any cost (it is only used for simulations with $\hat{\rho} = 0.10$ below). This illustrates the value of fixing the cache *performance* rather than cache size in order to compare processor/memory interface schemes on a level playing field.

3 Empirical Evaluation

This section uses members of the SPEC89 benchmark set as inputs to a combined processor/data cache simulation. The IPC results for a *perfect* cache— a cache that never generates a miss– will be used for comparison.

Table 2: Geometric mean of IPC (and percentage of *perfect* IPC) for the three schemes.

	issue rate 2		4		8	
Scheme	$\hat{\rho} =$ 0.05	$\hat{\rho} =$ 0.10	$\hat{\rho} =$ 0.05	$\hat{\rho} =$ 0.10	$\hat{\rho} =$ 0.05	$\hat{\rho} =$ 0.10
blocking, $T_{MISS} = 10$						
IPC	1.18	1.09	1.55	1.39	1.74	1.54
%	77%	71%	70%	63%	65%	58%
blocking, $T_{MISS} = 20$						
IPC	0.97	0.83	1.19	1.01	1.30	1.09
%	65%	54%	54%	45%	49%	41%
limited blocking, $T_{MISS} = 10$						
IPC	1.36	1.28	1.73	1.57	1.90	1.70
%	89%	84%	78%	71%	71%	64%
limited blocking, $T_{MISS} = 20$						
IPC	1.12	0.99	1.30	1.11	1.39	1.17
%	73%	65%	59%	50%	52%	44%
non-blocking, $T_{MISS} = 10$						
IPC	1.44	1.42	2.00	1.93	2.27	2.17
%	94%	93%	90%	87%	85%	81%
non-blocking, $T_{MISS} = 20$						
IPC	1.36	1.49	1.79	1.68	1.97	1.83
%	89%	97%	81%	76%	74%	69%

Table 2 presents the geometric mean of the IPC and the corresponding percentage of perfect cache performance for the three shemes with $T_{MISS} = 10$ and 20 cycles.

The performance for $\hat{\rho} = 0.05$ and $\hat{\rho} = 0.10$ data cache prototypes demonstrate the problems with the *blocking* schemes. From the table, the best mean performance is for $T_{MISS} = 10$, issue rate two, $\hat{\rho} = 0.05$, the least-demanding situation. Even in this case, the *blocking* scheme's performance is 77% of the *perfect* cache performance. The performance degrades considerably at higher issue rates. Clearly the *blocking* scheme has unacceptable performance for moderate to high issue rates and high miss penalty (T_{MISS}). The impact of this observation is that although the *blocking* scheme is the least-complicated of the three schemes, its performance is only marginally acceptable for an issue rate of two instructions per cycle. At higher issue rates, *blocking* removes all the advantage of designing a superscalar processor microarchitecture.

The *limited-blocking* scheme has acceptable performance for issue rates of two and four instructions per cycle when $T_{MISS} = 10$. In general, the performance of *limited-blocking* is sufficient to warrant its use in conservative situations where these parameters are in effect.

The complexity of the *non-blocking* hardware is justified by the increased performance that the *non-blocking* scheme provides. The table demonstrate that the *non-blocking* scheme can achieve approximately 97%–69% of the *perfect* cache miss performance with unlimited processor resources. This scheme hid miss penalty well. *Non-blocking* cache designs are needed when either the miss penalty is large or the issue rate is on the order of four to eight instructions per cycle.

The geometric mean of IPC across all benchmarks for the three schemes and the *perfect* case (no cache miss penalty) are shown in Figures 1 ($T_{MISS} = 10$) and 2 ($T_{MISS} = 20$) for the $\hat{\rho} = 0.10$ data cache prototypes. These two graphs show the relative ordering

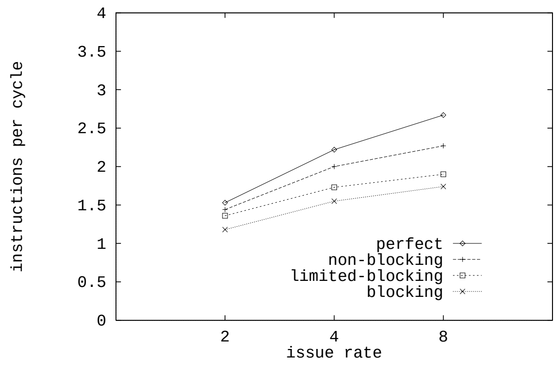

Figure 1: The geometric mean of IPC and IPC for *perfect* case for issue rates two, four and eight instructions per cycle, $T_{MISS} = 10$.

between the schemes from low to high performance as: *blocking*, *limited-blocking*, and *non-blocking*. Figure 2 also demonstrates that *limited-blocking* is only marginally better than *blocking* when the cache miss penalty is high at 20 cycles.

Some of the performance can be lost due to a limited scheduling window. To check whether scheduling window size had undue effect on the differences between the three schemes, the entire set of experiments were re-run with unlimited window size. The geometric mean IPC for unlimited window size for the $\hat{\rho} = 0.10$ data cache prototypes are presented in Figures 3 and 4. These figures demonstrate that lim-

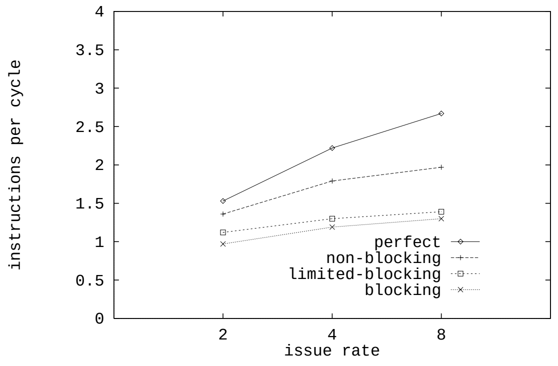

Figure 2: The geometric mean of IPC and IPC for *perfect* case for issue rates two, four and eight instructions per cycle, $T_{MISS} = 20$.

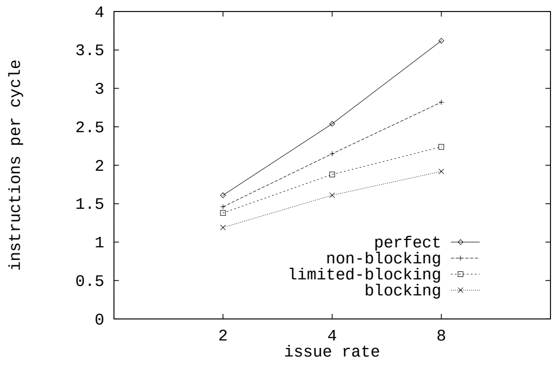

Figure 3: The geometric mean of IPC and IPC for *perfect* case for issue rates two, four and eight instructions per cycle, $T_{MISS} = 10$ with unlimited window size.

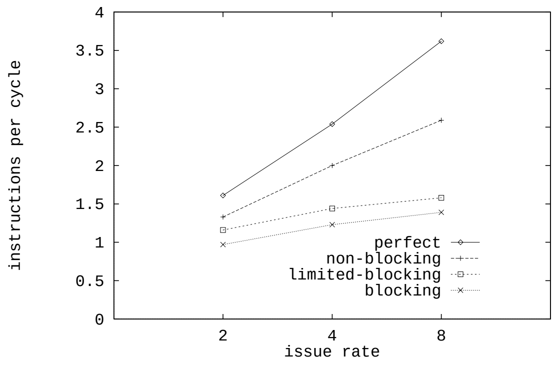

Figure 4: The geometric mean of IPC and IPC for *perfect* case for issue rates two, four and eight instructions per cycle, $T_{MISS} = 20$ with unlimited window size.

ited window size attributed to the low performance of the three schemes for issue rates of eight instructions per cycle. In particular, the *non-blocking* scheme performs acceptably even at high miss penalties of $T_{MISS} = 20$. The unlimited window size did not help the *limited-blocking* scheme for $T_{MISS} = 20$, it maintains a marginal performance increase over *blocking* in these simulations. Large scheduling windows are difficult to design, however, suggesting that this potential performance will not be tapped for some time to come. (Complete results for unlimited window size can be found in [5]).

4 Conclusion

This paper simulated a broad range of possible processor/memory interfaces for superscalar processors. In the evaluation, all effort was made to compare the different interface schemes on a level playing field with equivalent cache performance and sufficient processor resources. The results are uniquely unbiased about the relative merits of the three schemes. In general, cache designs must be fixed and processor resources must be restricted. If the resultant cache and processor are well designed (i.e., not the system bottleneck), the results of this paper can be used to make intelligent decisions about how to interface the cache design to the processor design.

References

[1] W. M. Johnson, *Super-scalar processor design.* PhD thesis, Department of Electrical Engineering, Stanford University, Stanford, California, June 1989.

[2] D. Kroft, "Lockup-free instruction fetch/prefetch cache organization," in *Proc. 8th Ann. Int'l. Symp. Computer Architecture*, pp. 81–87, May 1981.

[3] G. S. Sohi and M. Franklin, "High-bandwidth data memory systems for superscalar processors," in *Proc. 4th Int'l. Conf. on Architectural Support for Prog. Lang. and Operating Systems.*, (Santa Clara, CA), pp. 53–62, Apr. 1991.

[4] "Spec newsletter," Feb. 1989. SPEC, Fremont, CA.

[5] T. M. Conte, *Systematic computer architecture prototyping.* PhD thesis, Department of Electrical and Computer Engineering, University of Illinois, Urbana, Illinois, 1992.