An Improved Analytical Superscalar Microprocessor Memory Model

Xi E. Chen Tor M. Aamodt
Department of Electrical and Computer Engineering
University of British Columbia, Vancouver, BC, CANADA
{xichen,aamodt}@ece.ubc.ca

Abstract

As the number of transistors that can be integrated onto a single chip continues to increase exponentially, a growing challenge is modeling performance with reasonable accuracy in the early stages of processor design. While methodologies for execution driven simulations are well understood, comparatively little is known about how to develop accurate analytical models. Processor architects in industry have occasionally employed ad hoc analytical modeling techniques in an attempt to rapidly focus the search for higher performance designs. Moreover, analytical models can provide insights that a detailed performance simulator may not. This paper proposes techniques to accurately model the performance impact of long latency data cache misses in a superscalar microprocessor. A pending data cache hit results from a memory reference to a cache block for which a request has already been initiated by another instruction but has not yet completed (i.e., the requested block is still on its way from memory). These pending cache hits have a non-negligible influence on accuracy of analytical models when analyzing memory intensive benchmarks. We propose a technique to quickly identify pending data cache hits and account for their effect on performance by analyzing memory reference patterns without performing detailed performance simulations. We also propose a novel profiling method to take account of the maximum number of outstanding cache misses supported by the memory system. Overall, these approaches improve performance prediction accuracy by a factor of 3.9 on average (error decreases from 39.7% to 10.3%) for a set of memory intensive benchmarks when the maximum number of outstanding misses supported is unlimited. Moreover, on average our model achieves 151 and 170 times speedup over detailed simulations with less than 10% error, when the maximum number of outstanding misses supported is sixteen and eight, respectively.

1. INTRODUCTION

The traditional way to evaluate a microprocessor design is to create a cycle-accurate simulator modeling the microprocessor and then to run numerous simulations to quantify its performance. Not only is the task of creating such a simulator a time-consuming process, running performance simulations itself can be slow. Both are significant components of overall design-to-market time. As microprocessor design cycles stretch, architects effectively start each new project earlier and consequently with less accurate information about the eventual process technology that will be used, leading to designs that may not achieve the full potential of a given process technology node.

An orthogonal approach to obtain performance estimates for a proposed design is analytical modeling [7, 28]. An analytical model employs mathematical formulas that approximate the performance of the microprocessor being designed based upon program characteristics and microarchitectural parameters. One of the potential advantages of analytical modeling is that it can require much less time than crafting a performance simulator. Thus, when an architect has analytical models available to evaluate a given design, the models can significantly shorten the design cycle. While performance simulator infrastructures exist that leverage re-use of modular building blocks [8, 26], and workload sampling [23, 27] can reduce simulation time, another key advantage of analytical modeling is its ability to provide insights that a cycle-accurate simulator may not.

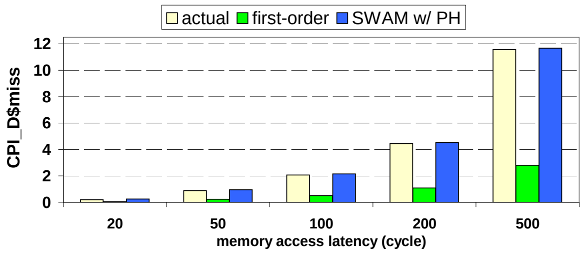


Figure 1: Comparison of actual CPI component due to long latency data cache misses versus modeled CPI component for *mcf* with different memory access latencies

Several analytical models have been proposed before [13, 17, 18, 19, 20, 21] and Karkhanis and Smith's first-order model [13] is the most recent one. Their first-order model separately estimates the *cycles per instruction* (CPI) component due to branch mispredictions, instruction and data cache misses, then adds each CPI component to an estimated CPI under ideal conditions (i.e., in the absence of the above-mentioned miss-events) to arrive at a final model for the performance of a superscalar processor. Although such a model ignores the interaction between different miss-events, previous work [12, 13] has shown that the model can be fairly accurate and is thus a good starting point. However, little prior work has focused on *analytically* modeling the performance impact of a limited (i.e., realistic) number of outstanding long latency data cache misses. In this paper we explore how to accurately and analytically model this important aspect of modern microprocessor designs.

One significant aspect of long latency memory systems is the large effect of pending data cache hits on overall performance. Here the term *pending data cache hit* refers to a memory reference to a cache block for which a request has already been initiated by another instruction but has not yet completed (i.e., the requested block is still on its way from memory). Figure 1 compares the actual CPI component due to long latency data cache misses to the modeled CPI component due to long latency data cache misses for *mcf* as memory latency increases. The first bar (actual) shows the result from a cycle-accurate simulator whose configuration is described in Section 4. The second bar (first-order) shows the result from a careful re-implementation of a previously described first-order model [13][1]. The third bar (SWAM w/ PH) illustrates the result from a new technique that we propose in this paper (see Section 3.4.1). In this paper, the modeled CPI due to long latency data cache misses ($CPI_{D\$miss}$) is always derived by dividing the modeled total extra cycles due to long latency data cache misses by the *total number of instructions committed.* From the data in Figure 1 we observe that the error becomes increasingly significant as memory latency grows. Therefore, to accurately model the performance of future superscalar microprocessors, it is necessary to appropriately model pending data cache hits.

This paper makes the following contributions:

- It shows that the impact of pending data cache hits on the performance of microprocessors with long memory latency is non-negligible for several memory intensive benchmarks. Therefore, their impact should be taken into account when analytically modeling future processor performance (Section 3.1).
- It proposes a way to identify potential pending data cache hits when profiling a dynamic instruction trace created by a cache simulator (not requiring any information from a performance simulator). It also shows how to analytically model the effects of those pending data cache hits (Section 3.1).
- It presents a novel technique to more accurately compensate for the potential overestimation of the modeled penalty per miss in earlier models, which incorporates an analysis of program characteristics. We find this approach significantly more accurate than the baseline approach we compare against (Section 3.2).
- It describes (in Section 3.3) a technique to analytically model the impact of a limited number of outstanding cache misses supported by a memory system on performance, thus increasing the realism (and hence applicability) of analytical models for processor designers. As the instruction window of future microprocessors becomes larger [6], a limited number of *Miss Status Holding Registers* (MSHRs) can have a dramatic impact on the performance of the whole system [25]. To the best of our knowledge, this is the first work describing how to analytically quantify the impact of support for a limited number of outstanding cache misses. The arithmetic mean of the absolute value of error[2] of a model that does *not* take account of a limited number of MSHRs is 32.6% and 32.4%, when the maximum number of outstanding misses supported is sixteen and eight, respectively; the arithmetic mean of the absolute value of the error of our model is 9.3% and 9.2%, respectively. (The geometric mean is 6.5% and 6.7%. The harmonic mean is 4.6% and 5.2%.)
- It also proposes two novel profiling techniques to better analyze data cache misses that are overlapped (Section 3.4).

Furthermore, we found these benefits accrue with no discernable increase in benchmark profile or analysis time. On average our analytical model is 156 and 170 times faster than detailed simulation, when the maximum number of outstanding misses supported is sixteen and eight, respectively.

The rest of this paper is organized as follows. Section 2 briefly reviews details of the first-order model required to understand our contributions. Section 3 describes how to accurately model the effects of pending data cache hits and a limited number of outstanding data cache misses supported. Section 4 describes the experimental methodology and Section 5 presents and analyzes our results. Section 6 reviews related work and Section 7 concludes the paper.

2. BACKGROUND: FIRST-ORDER MODEL

Before explaining the details of our techniques introduced in Section 3, it is necessary to be familiar with the basics of the first-order model of superscalar microprocessors. Karkhanis and Smith's first-order model [13] leverages the observation that the overall performance of a superscalar microprocessor can be estimated reasonably well by subtracting the performance losses due to different types of miss-events from the processor's sustained performance under the absence of miss events. The miss events considered include long latency data cache misses (e.g., L2 cache misses for a memory system with two-level cache hierarchy), instruction cache misses, and branch mispredictions.

[1]Our analysis indicates that pending data cache hits are very important to model accuracy. However, we note that the CPI errors we report for our ***baseline*** modeling technique (described in Section 2, and labeled "plain" in Section 5) are in some cases large relative to those reported in [13]. Our baseline modeling technique is our best attempt to implement the analytical model described in [13] based upon the details described in that paper and the follow-on work [12, 14]. We believe the discrepancy is partly due to our use of a smaller L2 cache size of 128 KB versus 512 KB used in [13] (we model smaller caches to stress the effects of long latency cache misses). We observed the difference for *mcf* drops from 79% to 66% when we model a similar sized cache. We speculate there are some features of their analytical model that were not fully documented in [12, 13, 14] (reportedly, pending hits were modeled, but the details were omitted due to space limitations [16]).

[2]In this paper, we use arithmetic mean of the absolute value of error to validate the accuracy of an analytical model, which we argue is the correct measure since it always reports the largest error numbers and is thus conservative in not overstating the case for improved accuracy. Note that we are interested in averaging the error of the CPI prediction on different benchmarks, not the average CPI predicted for an entire benchmark suite which often allows errors on individual benchmarks to "cancel out" in a way that suggests the modeling technique is more accurate than it really is. We also report the geometric mean and harmonic mean of the absolute error to allay any concerns these numbers might lead to different conclusions. In all case the improvements resulting from applying our new modeling techniques are robust enough that the selection of averaging technique does not (in our opinion) impact our conclusions.

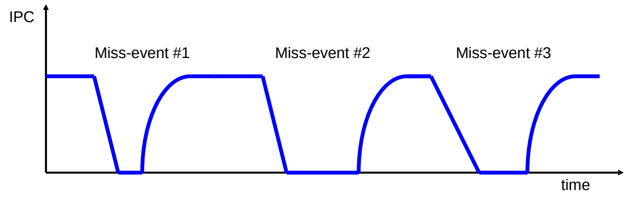


Figure 2: Useful instructions issued per cycle (IPC) over time used in the first-order model [13]

Figure 2 illustrates this approach. When there are no miss-events, the performance of the superscalar microprocessor is approximated by a stable IPC[3], expressed through a constant useful instructions issued per cycle (IPC) over time. When a miss-event occurs, the performance of the processor falls and the IPC gradually decreases to zero. After the miss-event is resolved (i.e., a mispredicted branch is resolved or a load missing in the data cache gets the data from memory), the decreased IPC ramps up to the stable value under ideal conditions. A careful analysis of this behavior leads to the first-order model [13].

While Figure 2 shows that a miss-event occurs only after the previous miss-events have been resolved, in a real processor it is possible for different types of miss-events to overlap. For example, a load instruction can miss in the data cache a few cycles after a branch is mispredicted, or a load instruction can miss in the data cache and an instruction fetch could miss in the instruction cache around the same time. However, it has been observed (and we confirmed) that overlapping between *different types* of miss-events is rare enough that ignoring it results in negligible error in typical applications [9, 10, 12, 13].

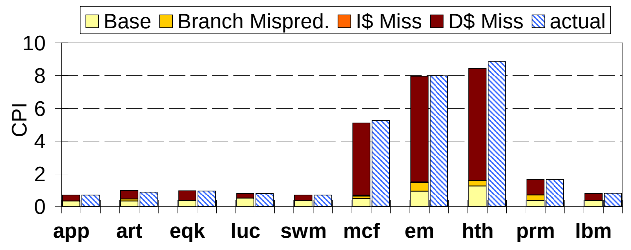


Figure 3: Comparison of the actual CPI to the CPI modeled by accumulating CPI components due to each miss-event in addition to the ideal CPI

Figure 3 compares the *cycles per instruction* (CPI) obtained from our performance simulator to the modeled CPI formed by adding the isolated CPI due to each miss-event to the base CPI (i.e., ideal CPI assuming perfect branch prediction and no penalty cycles for cache misses). The modeled CPI can be estimated as the sum of the base CPI and the CPI due to each miss-event [13]. Equation 1, below, expresses the modeled CPI as the sum of four CPI components. In this equation, CPI_{Base} represents the ideal CPI and CPI_{Bmsp} is the CPI due to branch mispredictions. $CPI_{D\$miss}$ and $CPI_{I\$miss}$ represents the CPI due to long latency data cache misses and instruction cache misses, respectively.

$$CPI_{Modeled} = CPI_{Base} + CPI_{Bmsp} + CPI_{D\$miss} + CPI_{I\$miss} \tag{1}$$

[3]While IPC often fluctuates even in the absence of miss events, the stable IPC value can be viewed as a long term average IPC in the absence of miss events.

As noted by Karkhanis and Smith, the interactions between microarchitectural events of the same type cannot simply be ignored. Although (demand) instruction cache misses will never overlap with each other, the overlapping of data cache misses must be appropriately modeled to achieve reasonable accuracy.

It has been observed that the number of instructions that can be issued per cycle is roughly proportional to the square root of the instructions in the window [17, 22], providing a simple way to approximate the ideal CPI. It has also been observed that for most benchmarks the ideal CPI can be approximated by the reciprocal of issue width when the instruction window is made large enough [13] (we employ this approximation in novel ways in Section 3.2).

Performance loss due to branch mispredictions is estimated by analyzing the transient IPC curve created based upon the IW characteristics of a program, where the IW characteristic is a function determining the average number of instructions that can issue per cycle (denoted as I) when given the number of instructions in the instruction window (denoted as W) [9, 11, 12, 13]. Since instruction cache misses never overlap with each other, the performance loss due to an instruction cache miss is simply modeled as the memory latency if the instruction fetch misses in the L2 cache or the L2 cache latency if the instruction fetch hits in the L2 cache [12, 13].

Short data cache misses (i.e., L1 data cache misses in this paper) are not regarded as miss-events in prior first-order models [12, 13] and they are modeled as long-execution-latency instructions when modeling CPI_{Base}. The term $CPI_{D\$miss}$ represents the CPI component due only to long latency data cache misses. In the rest of this paper, we use the term "cache misses" to represent long latency data cache misses.

Our baseline technique for modeling data cache misses, based upon Karkhanis and Smith's first-order model [13], analyzes dynamic instruction traces created by a cache simulator. In each *profile step*, a ROB_{size} number of consecutive instructions in the trace are put into the *profiling window* and analyzed, where ROB_{size} is the size of the re-order buffer. In other words, an instruction trace is divided into many blocks, each consisting of ROB_{size} number of consecutive instructions, then each block of instructions is analyzed. In each profile step, if all of the loads missing in the data cache are data independent with each other, they are considered overlapped (i.e., the overlapped misses have the same performance impact as a single miss). When data dependencies exist between misses, the maximum number of misses in the same data dependency chain is recorded and the execution of all the other misses are hidden under the chain.

In the rest of this paper, $num\_serialized\_D\$miss$ will represent the sum of the maximum number of misses measured in any single data dependency chain in a block of instructions, accumulated over all blocks making up the entire instruction trace. When all instructions in the trace have been analyzed, the $CPI_{D\$miss}$ can be estimated as

$$CPI_{D\$miss} = \frac{num\_serialized\_D\$miss \times mem\_lat}{total\_num\_instructions} \tag{2}$$

where mem_lat stands for the main memory latency.

The $CPI_{D\$miss}$ modeled in Equation 2 often overestimates the actual $CPI_{D\$miss}$ and a simple solution proposed by Karkhanis and Smith [13] is to subtract a fixed number of cycles per serialized data cache miss based upon ROB size to compensate. The need for this compensation is that, when a load issues and accesses the cache, it can be the oldest instruction in the ROB, the youngest instruction in the ROB, or somewhere in between. If the instruction is the oldest or nearly the oldest, the performance loss (penalty of the instruction) is the main memory latency. On the other hand, for example, if the instruction is the youngest or nearly the youngest one in the ROB and the ROB is full, its penalty can be partially hidden by the cycles required to drain all instructions before it, and can be approximated as $mem\_lat - \frac{ROB_{size}}{issue\_width}$ [13]. It has been observed that loads missing in the cache are relatively old when they issue [13]; and thus, perhaps the simplest (though not most accurate) approach is to use no compensation at all [13]. The mid-point of the two extremes mentioned above can also be used (i.e., a load missing in the cache is assumed to be in the middle of ROB when it issues), and the numerator in Equation 2 becomes $num\_serialized\_D\$miss \times (mem\_lat - \frac{ROB_{size}}{2 \times issue\_width})$ [12].

3. MODELING LONG LATENCY MEMORY SYSTEMS

In this section, we describe how to accurately account for many aspects of long latency memory systems. First, we propose a way to accurately model pending data cache hits in Section 3.1 and explain the major sources of error that occur when not modeling them. Then we present an approach to adjust for the overestimation of the modeled penalty per miss in Section 3.2 that takes account of application behaviour. In Section 3.3 we describe how to accurately model the impact of a limited number of outstanding misses on performance thus extending the model's utility. Finally, we present two novel profiling techniques to further increase the model's accuracy in Section 3.4.

3.1 Modeling Pending Data Cache Hits

The method of modeling long latency data cache misses described in Section 2 is based upon profiling dynamic instruction traces that are generated by a cache simulator [13]. Since a cache simulator provides no information about timing, it simply classifies the load or store first bringing a block from memory into the cache as a miss and all subsequent instructions accessing the block before the block is evicted as hits.

However, for many instructions classified as a hit by a cache simulator, their actual latency is much longer than the cache hit latency. For example, if there are two close load instructions accessing data in the same block that is not currently in the cache, the first load will be classified as a miss by the cache simulator and the second load as a hit, even though the data would still be on its way from memory in a real processor implementation. Therefore, since the second load is classified as a hit in the dynamic instruction trace, it is ignored in the process of modeling $CPI_{D\$miss}$ using the approach as described in Section 2.

More importantly, our detailed investigations uncovered that a significant source of errors results when two or more load

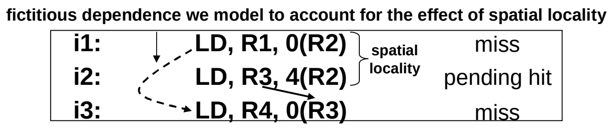


Figure 4: An example showing how two data independent misses (i1, i3) are connected by a pending hit (i2)

instructions that miss in the data cache are connected by a *third* pending data cache hit. We elaborate what "connected" means using the simple example in Figure 4. In this example, i1 and i3 are two loads classified as misses and they are data *independent* of each other, while i2 is a pending hit since it accesses the data in the same cache block as i1. The model described in Section 2 classifies i1 and i3 as overlapped and the performance penalty due to each miss using that approach is estimated as half of the memory access latency (total penalty is the same as if there is a single miss). However, this approximation is invalid since i3 is data dependent on the pending data cache hit i2; and i2 gets its data when i1 obtains its data from memory (i.e., i1 and i2 are waiting for the data in the same block). Therefore, in reality i3 can only start execution after i1 gets its data from memory (i.e., their execution should be serialized) although there is no true data dependence between i1 and i3 (or between i1 and i2). This scenario is very common since most programs contain significant spatial locality [1]. The appropriate way to model this situation is to consider i1 and i3 to be non-overlapped in our analytical model, similar to modeling a true data dependence between them. In general, two data independent loads (e.g., i1 and i3 in Figure 4) are "connected" by a pending data cache hit (e.g., i2 in Figure 4) when the pending hit accesses data in the same cache block as one load (e.g., i1 in Figure 4), and the other load (e.g., i3 in Figure 4) is data dependent on the pending hit. Although there are no data dependencies of any kind between the two loads (e.g., i1 and i3 in Figure 4), they have to execute serially due to microarchitectural effects.

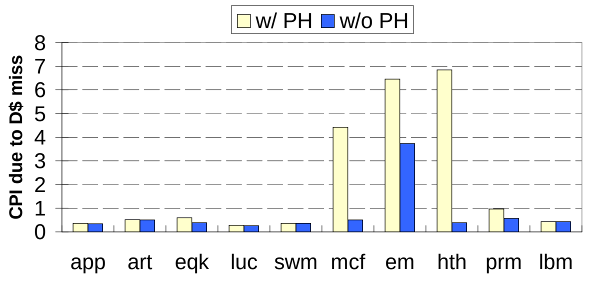


Figure 5: Impact of pending data cache hit latency on the CPI due to long latency cache misses (measured over all instructions committed)

Figure 5 shows the significant impact that pending data cache hits combined with spatial locality have on overall performance for processors with long memory latencies. The first bar (w/PH) illustrates measured $CPI_{D\$miss}$ for each benchmark on the detailed performance simulator described in Section 4, and the second bar (w/o PH) shows the measured $CPI_{D\$miss}$ when all the pending data cache hits are simulated as having a latency equal to the L1 data cache hit latency. From the figure we observe that the difference is significant for *eqk*, *mcf*, *em*, *hth*, and *prm*.

To model the effects of pending data cache hits *analytically*, we first need to identify them without a performance simulation. At first this may seem impossible as there is no information about timing from the cache simulator. We tackle this by assigning each instruction in the dynamic instruction trace a sequence number in program order and labeling each memory access instruction in the trace with the sequence number of the instruction that first brings the memory block into the cache. Then, when we profile the instruction trace, if a hit consumes data from a cache block that is first brought into the cache by an instruction that is still in the profiling window, it is regarded as a pending data cache hit.

For every pending hit identified using this approach (e.g., i2 in Figure 4), there is a unique instruction earlier in the profiling window that first brings in the cache block accessed by that pending hit (e.g., i1 in Figure 4). When we notice a data dependence between a later cache miss (e.g., i3 in Figure 4) and the pending hit (e.g., i2), we model a dependence between the early miss (e.g., i1) and the instruction that is data dependent on the pending hit (e.g., i3) since the two instructions (e.g., i1 and i3) have to execute serially due to the constraints of the microarchitecture.

Next we will discuss how to make our model more accurate by better estimating the performance loss (penalty) due to a load that misses in the cache rather than using fixed compensation based upon ROB size.

3.2 Accurate Miss Penalty Compensation

While the model described in Section 2 uses a fixed number of cycles to adjust the modeled $CPI_{D\$miss}$ (in Equation 2), we found that compensation with a fixed number of cycles (a constant ratio of the reorder buffer size) does not provide consistently accurate compensation for all the benchmarks that we studied, resulting in large modeling errors (see Figure 9 on page). To capture the distinct distribution of long latency data cache misses of each benchmark, we propose a novel compensation method. The new method is motivated by our observation that the number of cycles hidden for a load missing in the cache is roughly proportional to the distance between the load and the immediately preceding load that misses in the cache[4]. This happens because, we observe that, when a load instruction misses in the cache, most of the instructions between that load and its immediately preceding miss are independent of that load (i.e., these instructions are earlier than the load in program order). Therefore, we approximate the latency of the later load that can be hidden as the time used to drain those intermediate instructions from the instruction window, which we estimate as the distance between the two loads divided by the issue width. When we profile an instruction trace, the average distance between two consecutive loads missing in the cache is also collected and used to adjust the modeled $CPI_{D\$miss}$[5].

[4]We define the distance between two instructions to be the difference between their instruction sequence number.

[5]Note if the distance between two misses exceeds the instruction window size of the processor being modeled, the distance is counted as the window size since the latency of a miss can be hidden by at most $ROB_{size} - 1$ instructions.

Equation 3 shows how the $CPI_{D\$miss}$ is adjusted by subtracting a compensation term, $\frac{dist}{issue\_width} \times num\_D\$miss$, from the numerator in Equation 2. Here $dist$ is the average distance between two consecutive loads that miss in the cache and the term $\frac{dist}{issue\_width}$ represents the average number of cycles hidden for each cache miss, and the product of this term and the total number of loads missing in the cache ($num\_D\$miss$) becomes the total number of cycles used to compensate for the overestimation of the baseline profiling method.

$$CPI_{D\$miss} = \frac{num\_serialized\_D\$miss \times mem\_lat - comp}{total\_num\_instructions} \quad (3)$$

$$comp = (\frac{dist}{issue\_width} \times num\_D\$miss)$$

Next we will propose a technique to model the performance impact of a limited number of outstanding cache misses supported by a realistic memory system.

3.3 Modeling a Limited Number of MSHRs

The method of analytically modeling the CPI due to long latency data cache misses described in Section 2 assumes that at most ROB_{size} cache misses can be overlapped. However, this assumption is unreasonable for most modern processors where the maximum number of outstanding cache misses the system supports is limited by the number of *Miss Status Holding Registers* (MSHRs) [15] in the processor. In a real processor the issue stage has to stall when available MSHRs run out. To extend the existing model and increase its flexibility, we next show how to analytically model the impact of a limited number of MSHRs on overall performance.

Based upon the technique described in Section 2, the profiling window with the same size as the instruction window is always assumed to be full when modeling $CPI_{D\$miss}$. In order to model a limited number of outstanding cache misses, we need to refine this assumption. During a profile step, we stop putting instructions into the profiling window when the number of instructions that miss in the data cache and that have been analyzed is equal to N_{MSHR} (number of MSHRs), and then update $num\_serialized\_D\$miss$ only based upon those instructions that have been analyzed in this profile step[6].

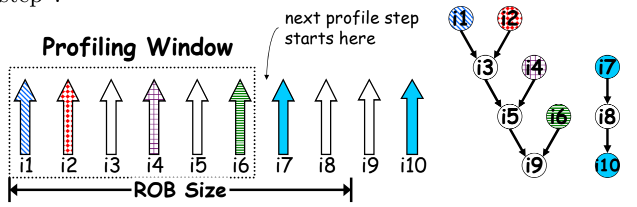


Figure 6: An example showing profiling with $ROB_{size} = 8$ and $N_{MSHR} = 4$. Each arrow corresponds to a dynamic instruction in the trace. Data cache misses are filled with patterns. Corresponding data dependency graph is shown to the right.

Figure 6 illustrates a simplified example demonstrating how the profiling technique works when the number of outstanding cache misses supported is limited to four. Once we

[6]In real execution, cache misses that are regarded as not present in the profiling window simultaneously due to lack of available MSHRs could actually be in the instruction window simultaneously. The profiling window only approximates the performance loss due to a limited number of MSHRs. We leverage this observation in Section 3.4.2.

encounter N_{MSHR} (four) cache misses in the instruction trace (i.e., $i1$, $i2$, $i4$, and $i6$), the profile step stops and *num_serialized_D\$miss* is updated (i.e., the profiling window is made shorter). In the example, the four misses are data independent of each other (and not connected with each other via a pending hit as described in Section 3.1), thus *num_serialized_D\$miss* will be incremented by one. Although $i7$ also misses in the cache, it is included in the next profile step since all four MSHRs have been used.

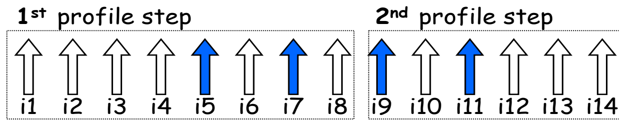
This enhancement leads us to the next two methods we propose to refine the selection of each profiling window's boundaries and thereby significantly increase the accuracy of our analytical model.

3.4 Profiling Window Selection

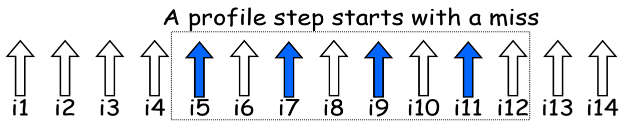
The profiling technique described in Section 2 (which we will refer to hereafter as *plain profiling*) partitions an instruction trace into blocks of instructions, the size of which is equal to the size of the instruction window. Then, each block of instructions is analyzed to model the performance loss due to data cache misses. In this section, we present two important refinements that we find better model the overlapping between cache misses. While applying the techniques described in previous sections (i.e., modeling pending hits, compensating based upon the average distance between two consecutive misses, and modeling a limited number of MSHRs), the two profiling techniques described below reduce average absolute error of plain profiling from 29.3% to 10.3%, from 26.0% to 9.3%, and from 23.9% to 9.2%, when the maximum number of MSHRs is unlimited, sixteen, and eight, respectively.

3.4.1 Start-with-a-miss (SWAM) Profiling

We observe that often the plain profiling described in Section 2 does not include all the cache misses that can be overlapped, due to the simple way it partitions an instruction trace. Figure 7(a) shows a simple example. In this example, we assume that all the cache misses (shaded arrows) are data independent of each other for simplicity. Using the profiling approach described in Section 2, a profile step starts at pre-determined instructions (for example, $i1$, $i9$, $i17$..., when ROB_{size} is eight). Therefore, although the latency of $i5$, $i7$, $i9$, and $i11$ can be overlapped, the plain profiling technique does not account for this.



(a) **Plain Profiling**


A profile step starts with a miss
i1 i2 i3 i4 i5 i6 i7 i8 i9 i10 i11 i12 i13 i14

(b) **SWAM Profiling**

Figure 7: An example comparing plain profiling and SWAM profiling with $ROB_{size} = 8$. Each arrow corresponds to a dynamic instruction. Data cache misses are shaded.

By making each profile step start with a cache miss, we find that the accuracy of the model improves significantly. Figure 7(b) illustrates the idea. Rather than starting a profile step with $i1$, we start a profile step with $i5$, and the profiling window will include $i5$ to $i12$. Then the next profile step will seek the first cache miss after $i12$ and start with it. We call this technique start-with-a-miss (SWAM) profiling and (in Section 5.1) we will show that SWAM profiling decreases the error of plain profiling from 29.3% to 10.3% when the number of MSHRs is not limited.

3.4.2 Improved SWAM for Modeling a Limited Number of MSHRs (SWAM-MLP)

The technique for modeling MSHRs proposed in Section 3.3 can be combined with SWAM to better model the performance when the number of outstanding cache misses supported by the memory system is limited. We refine this further to model a limited number of outstanding cache misses using SWAM by having each profile step start with a miss and finish either when the number of instructions that have been analyzed equals the size of the instruction window, or when the number of cache misses that have been analyzed equals the total number of MSHRs. However, choosing a profiling window independent of whether a cache miss is data dependent on other misses (or connected to other misses via pending hits as described in Section 3.1) leads to inaccuracy because data dependent cache misses cannot simultaneously occupy an MSHR entry.

To improve accuracy further, we stop a profile step when the number of cache misses that are data independent of misses that have been analyzed in the same profile step (rather than the number of cache misses being analyzed) equals the total number of MSHRs. In the rest of this paper we call this improved technique SWAM-MLP since it improves SWAM by better modeling memory level parallelism. SWAM-MLP improves model accuracy because, when a miss depends on an earlier miss in the same profiling window, the later miss cannot issue until the earlier one completes and the out-of-order execution allows another independent miss that is younger than both of the above misses to issue. Therefore, the number of instructions that miss in the data cache and that should be analyzed in a profile step should, in this case, be more than the total number of MSHRs. In Section 5.2, we show that SWAM-MLP further decreases modeling error of SWAM from 9.8% to 9.3% and from 12.8% to 9.2%, when the maximum number of MSHRs is sixteen and eight, respectively.

4. METHODOLOGY

To evaluate our analytical model, we have modified SimpleScalar [4] to simulate the CPI due to long latency data cache misses when accounting for a limited number of MSHRs. Table 1 describes the microarchitectural parameters used in this study. Note that the parameters for branch predictor and instruction cache are only used for collecting data shown in Figure 3. In the rest of this paper we focus on how to accurately predict $CPI_{D\$miss}$, which is the CPI due to long latency data cache misses when both branch predictor and instruction cache are ideal (this is the same methodology applied to model $CPI_{D\$miss}$ described in [13]).

To stress our model, we simulate a relatively small L2 cache

Table 1: Microarchitectural Parameters

Machine Width	4
ROB Size	256
LSQ Size	256
Branch Predictor	4KB gShare
L1 I-Cache	16KB, 32B/line, 4-way, 1-cycle latency
L1 D-Cache	16KB, 32B/line, 4-way, 2-cycle latency
L2 Cache	128KB, 64B/line, 8-way, 10-cycle latency
Memory Latency	200 cycles

Table 2: Benchmarks

Benchmark	Label	Miss rate	Suite
173.applu	app	31.1MPKI	SPEC 2000
179.art	art	117.1MPKI	SPEC 2000
183.equake	eqk	15.9MPKI	SPEC 2000
189.lucas	luc	13.1MPKI	SPEC 2000
171.swim	swm	23.5MPKI	SPEC 2000
181.mcf	mcf	90.1MPKI	SPEC 2000
em3d	em	74.7MPKI	Olden
health	hth	45.7MPKI	Olden
perimeter	prm	18.7MPKI	Olden
470.lbm	lbm	17.5MPKI	SPEC 2006

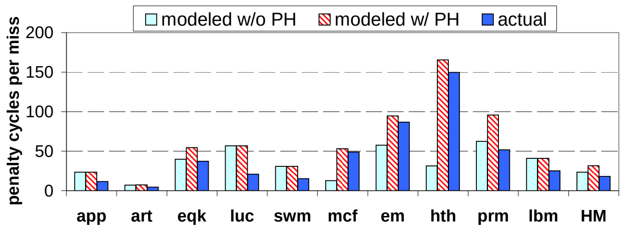
compared to contemporary microprocessors. We note that most of the benchmarks we used are relatively old and the size of the L2 cache we simulated is close in size to those employed in microprocessors shipping at the time when those benchmarks were released[7]. The benchmarks we choose are the ones from SPEC 2000 [24] and OLDEN [5] for which there are at least 10 long latency data cache misses every 1000 instructions simulated (10MPKI). Table 2 illustrates miss rates of these benchmarks and labels used to represent them in figures. For each benchmark, we select 100M representative instructions to simulate using the Sim-Point toolkit [23].

5. RESULTS

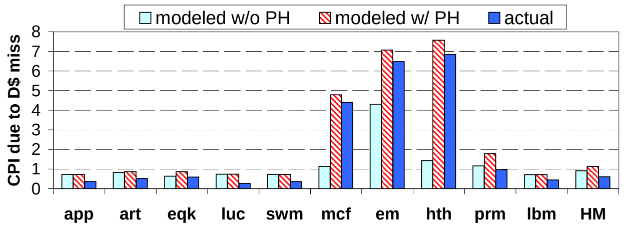
In this section, we first show (in Section 5.1) that SWAM profiling decreases modeling error from 39.7% to 10.3% while taking account of pending hits and using the compensation technique proposed in Section 3.2. Then in Section 5.2 we evaluate the techniques proposed in Section 3.3 and Section 3.4 to model the effects of a limited number of MSHRs and show that the error of our model is less than 10% when the maximum number of MSHRs varies from sixteen to eight. In Section 5.3 we illustrate that on average our analytical model is two orders of magnitude times faster than detailed simulations.

5.1 Modeling Pending Data Cache Hits

Figure 8(a) compares the modeled penalty cycles per miss to the actual penalty per miss[8] obtained from our performance simulator, and Figure 8(b) compares modeled $CPI_{D\$miss}$ (i.e., modeled total extra execution cycles due to long latency cache misses divided by the total number of instructions committed) to its simulated counterpart, in both cases assuming unlimited MSHRs. For each benchmark, the first bar (modeled w/o PH) is the result modeled by plain profiling technique without taking pending data cache hits into



(a) Penalty cycles per miss



(b) CPI component due to long latency data cache misses measured over *all* instructions

Figure 8: Penalty cycles per miss and CPI due to long latency cache misses for plain profiling (does not use compensation method described in Section 3.2, and assumes unlimited MSHRs)

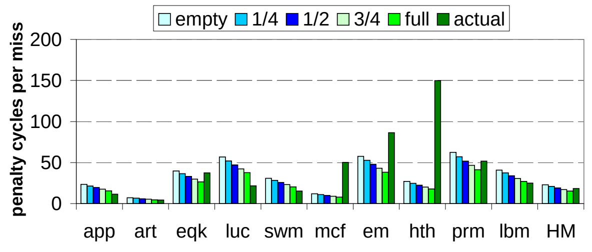
consideration and without applying any compensation method, and the second bar (modeled w/PH) is the result modeled by the same profiling technique incorporating the effects of pending data cache hits using the technique described in Section 3.1 without applying any compensation method. Note that we have shown the harmonic mean (HM) for all the benchmarks in Figure 8(a) and Figure 8(b). Due to the fact that positive errors (i.e., modeled results are larger than actual results) and negative errors (i.e., modeled results are less than actual results) cancel out, the harmonic mean of the modeled results without accounting for pending hits (modeled w/o PH) is close to simulated results. However, it is important to recognize that the accuracy of this technique on individual benchmark is quite poor. We find that the arithmetic mean of absolute error when not modeling pending hits (modeled w/o PH) is 71.2% and this error decreases to 66.5% while pending hits are modeled (modeled w/ PH). Note that the error of the model is quite large even with pending hits modeled since we do not yet include the compensation method described in Section 3.2. We will show below that modeling pending hits results in a much better accuracy than not modeling pending hits when the compensation method described in Section 3.2 is included.

The data in Figure 8(b) shows that for some benchmarks with heavy pointer chasing feature such as mcf, $em3$, and hth, ignoring the effects of pending data cache hits results in a dramatic underestimate for $CPI_{D\$miss}$. As discussed in Section 3.1, the reason for the underestimation is that some cache accesses classified as "hits" actually have long latencies because they must wait for a pending cache access.

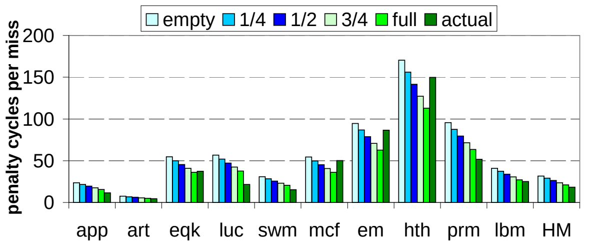
Section 2 describes prior proposals for compensating for the overestimation of modeled penalty cycles per serialized miss using a *fixed* number of cycles. Figure 9(a) and Figure 9(b) illustrate the modeled results after compensation with constant cycles. In these two figures, we show results using five different constant numbers of cycles to compensate for the overestimation. The first bar (empty) corresponds to

[7]We do not simulate SPEC 2006 since few SPEC 2006 applications compile with the current compilation infrastructure available for SimpleScalar.

[8]The actual penalty per miss is obtained by dividing the simulated total extra execution cycles due to long latency cache misses by the total number of misses. Due to the fact that the execution of some loads missing in the cache can be overlapped (memory level parallelism), the penalty per miss for each benchmark in Figure 8(a) is always less than the main memory latency (200 cycles in this study).



(a) Not modeling pending data cache hits

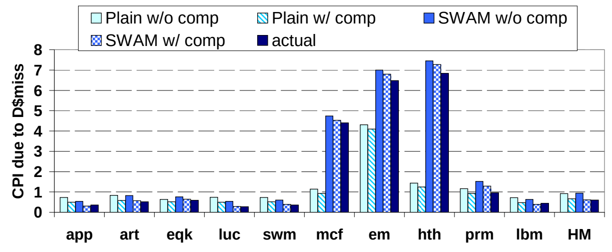


(b) Modeling pending data cache hits

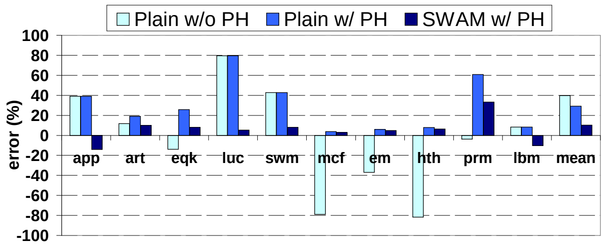
Figure 9: Penalty cycles per miss with fixed number of cycles compensation for plain profiling (Unlimited MSHRs)

the assumption that an instruction that misses in the cache is always the oldest one in the instruction window when it issues (accesses the first level cache). The second bar (1/4) corresponds to the assumption that there are always $\frac{1}{4}ROB_{size} = 64$ in-flight instructions older than a cache miss when it issues and it is similar to next two bars (1/2) and (3/4). The fifth bar (full) corresponds to the assumption that there are always $ROB_{size} - 1$ older instructions in the window when the instruction issues (i.e., the instruction is always the youngest one in the window when it issues). The last bar (actual) shows the simulated penalty cycles per cache miss. From this data we observe that there is no one fixed cycle compensation method performing consistently well for all the benchmarks we study. For example, in Figure 9(a) we observe that error is minimized using "full" for *app*, *art*, *luc*, *swm*, and *lbm*, but minimized using "empty" for *em*, *mcf*, and *hth*; however, *eqk* and *prm* requires something in-between. The harmonic means of each fixed cycle compensation method over all the benchmarks are also shown and we notice that due to the fact that positive and negative errors cancel out, the harmonic mean of some fixed cycle compensation methods is close to simulated results. However, it is important to recognize that their accuracy on individual benchmark is quite poor. By using the fixed cycle compensation method, we find that the smallest arithmetic mean of absolute error is 43.5% when not modeling pending hits and 26.9% when modeling pending hits, both from using "full" compensation.

To account for the distinct behavior of each benchmark, we use the average distance between two consecutive cache misses to compensate for the overestimation of the modeled extra cycles due to long latency data cache misses as proposed in Section 3.2. For each benchmark, this average distance is divided by the issue width of the machine. Then the result is multiplied by the total number of cache misses and the product is subtracted from the modeled extra cycles due to long latency cache misses before compensation, as described in Equation 3. Figure 10(a) compares the modeled CPI due to long latency data cache misses for both plain pro-



(a) CPI due to D$miss



(b) Modeling error

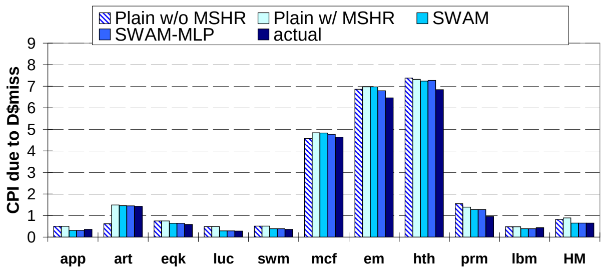
Figure 10: CPI due to D$miss and modeling error for different profiling techniques (Unlimited MSHRs)

filing technique described in Section 2 and the start-with-a-miss (SWAM) profiling technique described in Section 3.4.1 to the simulated results. The first bar (Plain w/o comp) and the third bar (SWAM w/o comp) for each benchmark correspond to the modeled results without any compensation; the second bar (Plain w/ comp) and the fourth bar (SWAM w/ comp) are modeled results after the compensation described in Section 3.2. As we expect, SWAM profiling is more accurate than plain profiling since it can capture more data cache misses that can be overlapped. Moreover, Figure 10(b) illustrates the error of each modeling technique, after compensation. We observe from Figure 10(b) that the arithmetic mean of the absolute error for SWAM profiling when pending data cache hits are modeled (SWAM w/PH) is about 3.9 times lower than plain profiling when pending hits are not modeled (Plain w/o PH) (i.e., the arithmetic mean of the absolute error decreases from 39.7% to 10.3%[9]).

5.2 Modeling Limited Number of MSHRs

All the results we have seen thus far are for modeling a processor with an unlimited number of Miss Status Holding Registers. This section shows modeled CPI component due to long latency data cache misses when the number of outstanding data cache misses supported by a memory system is finite. Figure 11(a) and Figure 11(b) compares the modeled CPI due to long latency data cache misses to the simulated results when the maximum number of MSHRs is sixteen and eight, respectively. (We show data for eight MSHRs since we note that Prescott has only eight MSHRs [3]). For each benchmark, the first bar (Plain w/o MSHR) represents the modeled $CPI_{D\$miss}$ from plain profiling (i.e., it is not aware of a limited number of MSHRs and always provides the same result for a benchmark) and the second bar (Plain w/ MSHR) shows the modeled $CPI_{D\$miss}$ from plain profiling with the technique of modeling a limited number of MSHRs (Section 3.3) included. The third and the fourth bar illustrates the modeled $CPI_{D\$miss}$ from SWAM (Sec-

[9]The geometric mean of the absolute error decreases from 26.4% to 8.2%, and the harmonic mean of the absolute error decreases from 15.3% to 6.9%.


(a) $N_{MSHR} = 16$

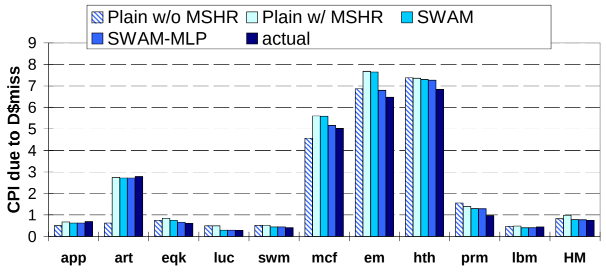
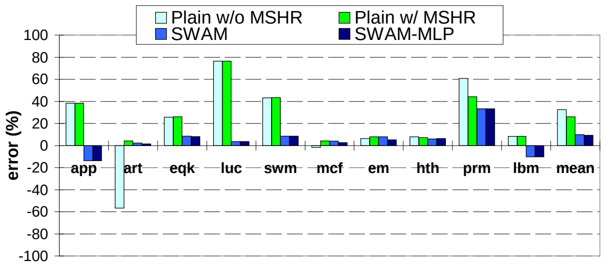

(b) $N_{MSHR} = 8$

Figure 11: CPI due to D$miss for $N_{MSHR} = 16$ and $N_{MSHR} = 8$.


(a) $N_{MSHR} = 16$

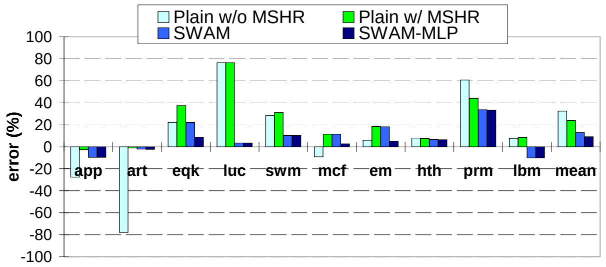

(b) $N_{MSHR} = 8$

Figure 12: Error of the modeled $CPI_{D\$miss}$ for $N_{MSHR} = 16$ and $N_{MSHR} = 8$.

tion 3.4.1) and SWAM-MLP (Section 3.4.2), respectively. For all these profiling techniques, pending hits are modeled using the method described in Section 3.1. Moreover, the modeling error for each technique is shown in Figure 12(a) and Figure 12(b), when the maximum number of MSHRs is sixteen and eight, respectively.

Similar to the situation when the number of MSHRs is unlimited, both plain profiling and SWAM profiling can be used to model a limited number of MSHRs and SWAM profiling performs consistently better than plain profiling. Moreover, with a limited number of MSHRs, the refined SWAM-MLP is better than SWAM. We notice that as the total number of MSHRs decreases, the advantage of SWAM-MLP over SWAM becomes significant, especially for *eqk*, *mcf*, and *em*[10], for which it is more likely to have data dependence among cache misses thus affecting the boundaries of the profiling window SWAM-MLP chooses. When the number of MSHRs is sixteen, SWAM decreases the arithmetic mean of the absolute error from 32.6% (Plain w/o MSHR) to 9.8%[11]. When the number of MSHRs is eight, SWAM decreases the arithmetic mean of the absolute error from 32.4% (Plain w/o MSHR) to 12.8%[12]. SWAM-MLP further decreases the error to 9.3% and 9.2%, when the number of MSHRs is sixteen and eight, respectively[13].

5.3 Speedup of the Analytical Model

One of the most important advantages of the analytical model we present in this paper versus detailed simulations is its fast speed of analysis. Figure 13 shows, for each benchmark, the speedup of our analytical model over the detailed simulator that is described in Section 4 with different maximum numbers of MSHRs[14]. From this figure we observe that on average the speedup of our analytical model over the detailed simulator increases while the maximum number of MSHRs decreases since the total number of cycles simulated for a fixed number of instructions increases. From Figure 13 we also notice that our analytical model is at least 91 times faster than the simulator (i.e., the first bar of swm), and on average the model is 150, 156, and 170 times faster than the detailed simulator when the number of MSHRs is unlimited (MSHR_INF), sixteen (MSHR_16), and eight (MSHR_8), respectively.

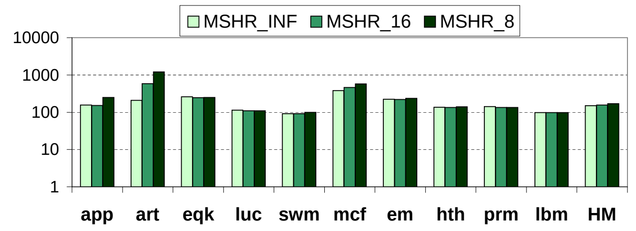


Figure 13: Speedup for different maximum numbers of MSHRs

6. RELATED WORK

There exist many analytical models proposed for superscalar microprocessors [17, 18, 19, 21]. A common limitation for early models is that they assume a perfect data cache and cannot be used to estimate the performance loss due to long latency data cache misses. As the gap between memory speed and microprocessor speed increases, long latency data cache misses must be properly modeled to provide useful insight for microprocessor designers.

Agarwal et al. [2] present an early analytical cache model es-

[10]These are the benchmarks which have large error when not modeling pending data cache hits as seen in Figure 8.

[11]It reduces the geometric mean from 19.4% to 7.4% and reduces the harmonic mean from 8.5% to 5.8%.

[12]It reduces the geometric mean from 21.5% to 9.7% and reduces the harmonic mean from 14.5% to 7.0%.

[13]The geometric mean of the absolute error is 6.5% and 6.7%, and the harmonic mean of the absolute error is 4.6% and 5.2%, when the number of MSHRs is sixteen and eight, respectively.

[14]We do not include the trace collection time when evaluating the speedup of our analytical model since a trace can potentially be reused multiple times to amortize the overhead.

timating cache miss rate when given cache parameters such as cache size, block size, associativity, etc., and their model also requires information from an address trace of the program being analyzed. However, in a superscalar, out-of-order execution microprocessor, the cache miss rate itself sometimes is not enough to predict the real performance of the program analyzed. Our analytical model can predict the actual performance of a program given a set of cache parameters and a relative simple analysis applied to an instruction trace.

Noonburg and Shen present a superscalar microprocessor model using a Markov chain model [20]. However, it does not model long memory latency. The first-order model proposed by Karkhanis and Smith [13] is the first to attempt to account for long latency data cache misses. Our analytical model improves the accuracy of our re-implementation of their techniques described in Section 2 (based upon details available in the literature) by modeling pending data cache hits, and extends it to estimate the impact of a limited number of outstanding cache misses on the overall performance.

7. CONCLUSIONS

In this paper we have proposed and evaluated several improvements to an existing analytical performance model for superscalar processors. We demonstrate that “first-order” analytical models are sensitive to the details of long latency cache miss events such as the modeling of pending data cache hits. We explain the importance of properly modeling pending data cache hits that “connect” otherwise data independent loads that miss in the cache for some memory intensive benchmarks, and propose a novel profiling technique to accurately model their effects on performance, not requiring a performance simulator. Moreover, we present a technique to quantify the impacts of a limited number of MSHRs in a microprocessor. We believe that as the instruction window of future microprocessors becomes larger, more memory level parallelism will be exposed and our model will help designers to better understand the influence of memory level parallelism on microprocessor performance. Overall, the techniques we introduce in this paper can reduce the modeling error by a factor of 3.9 for a set of memory intensive benchmarks. The error of our model, for the set of benchmarks we use, is 9.3% and 9.2%, when the maximum number of outstanding misses supported is sixteen and eight, respectively.

8. ACKNOWLEDGMENTS

We would like to thank Lieven Eeckhout, Wilson Fung, Ali Bakhoda, and the anonymous reviewers for their valuable comments on this work. This work was partly supported by the Natural Sciences and Engineering Research Council of Canada.

9. REFERENCES

[1] T. M. Aamodt and P. Chow. Optimization of Data Prefetch Helper Threads with Path-expression Based Statistical Modeling. In *ICS-21*, pages 210–221, 2007.

[2] A. Agarwal, M. Horowitz, and J. Hennessy. An Analytical Cache Model. *TOCS*, 7(2):184–215, 1989.

[3] D. Boggs, A. Baktha, J. Hawkins, D. T. Marr, J. A. Miller, P. Roussel, R. Singhal, B. Toll, and K. Venkatraman. The Microarchitecture of the Intel® Pentium® 4 Processor on 90nm Technology. *Intel® Technology Journal*, 8(1), 2004.

[4] D. Burger and T. M. Austin. The SimpleScalar Tool Set, Version 2.0. *SIGARCH Computer Architecture News*, 25(3):13–25, 1997.

[5] M. C. Carlisle. *Olden: Parallelizing Programs with Dynamic Data Structures on Distributed-Memory Machines.* PhD thesis, Princeton University, June 1996.

[6] A. Cristal, O. Santana, M. Valero, and J. F. Martínez. Toward Kilo-instruction Processors. *ACM TACO*, 1(4):389–417, 2004.

[7] L. Eeckhout, S. Nussbaum, J. E. Smith, and K. D. Bosschere. Statistical Simulation: Adding Efficiency to the Computer Designer’s Toolbox. *IEEE Micro*, 23(5):26–38, 2003.

[8] J. Emer, P. Ahuja, E. Borch, A. Klauser, C.-K. Luk, S. Manne, S. S. Mukherjee, H. Patil, S. Wallace, N. Binkert, R. Espasa, and T. Juan. Asim: A Performance Model Framework. *IEEE Computer*, 35(2):68–76, 2002.

[9] S. Eyerman. *Analytical Performance Analysis and Modeling of Superscalar and Multi-threaded Processors.* PhD thesis, Ghent University, 2008.

[10] S. Eyerman, L. Eeckhout, T. S. Karkhanis, and J. E. Smith. A Performance Counter Architecture for Computing Accurate CPI Components. In *ASPLOS-XII*, pages 175–184, 2006.

[11] S. Eyerman, J. E. Smith, and L. Eeckhout. Characterizing the Branch Misprediction Penalty. In *ISPASS*, pages 48–58, 2006.

[12] T. S. Karkhanis. *Automated Design of Application-specific Superscalar Processors.* PhD thesis, University of Wisconsin, 2006.

[13] T. S. Karkhanis and J. E. Smith. A First-order Superscalar Processor Model. In *ISCA-31*, pages 338–349, 2004.

[14] T. S. Karkhanis and J. E. Smith. Automated design of application specific superscalar processors: an analytical approach. In *ISCA-34*, pages 402–411, 2007.

[15] D. Kroft. Lockup-free Instruction Fetch/Prefetch Cache Organization. In *ISCA-8*, pages 81–87, 1981.

[16] Lieven Eeckhout. Personal Communication. May, 2008.

[17] P. Michaud, A. Seznec, and S. Jourdan. Exploring Instruction-Fetch Bandwidth Requirement in Wide-issue Superscalar Processors. In *PACT*, pages 2–10, 1999.

[18] P. Michaud, A. Seznec, and S. Jourdan. An Exploration of Instruction Fetch Requirement in Out-of-Order Superscalar Processors. *Int’l Journal of Parallel Programming*, 29(1):35–38, 2001.

[19] D. B. Noonburg and J. P. Shen. Theoretical Modeling of Superscalar Processor Performance. In *MICRO-27*, pages 52–62, 1994.

[20] D. B. Noonburg and J. P. Shen. A Framework for Statistical Modeling of Superscalar Processor Performance. In *HPCA-3*, pages 298–309, 1997.

[21] D. J. Ofelt. *Efficient Performance Prediction for Modern Microprocessors.* PhD thesis, Stanford University, 1999.

[22] E. M. Riseman and C. C. Foster. The Inhibition of Potential Parallelism by Conditional Jumps. *IEEE Transactions on Computers*, 21(12):1405–1411, 1972.

[23] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically Characterizing Large Scale Program Behavior. In *ASPLOS-X*, pages 45–57, 2002.

[24] Standard Performance Evaluation Corporation. SPEC CPU2000 benchmarks. http://www.spec.org.

[25] J. Tuck, L. Ceze, and J. Torrellas. Scalable Cache Miss Handling for High Memory-Level Parallelism. In *MICRO-39*, pages 409–422, 2006.

[26] M. Vachharajani, N. Vachharajani, D. A. Penry, J. A. Blome, and D. I. August. Microarchitectural Exploration with Liberty. In *MICRO-35*, pages 271–282, 2002.

[27] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe. SMARTS: Accelerating Microarchitecture Simulation via Rigorous Statistical Sampling. In *ISCA-30*, pages 84–97, 2003.

[28] J. J. Yi, L. Eeckhout, D. J. Lilja, B. Calder, L. K. John, and J. E. Smith. The Future of Simulation: A Field of Dreams. *Computer*, 39(11):22–29, 2006.