Hybrid Analytical Modeling of Pending Cache Hits, Data Prefetching, and MSHRs

XI E. CHEN and TOR M. AAMODT, University of British Columbia

This article proposes techniques to predict the performance impact of pending cache hits, hardware prefetching, and miss status holding register resources on superscalar microprocessors using hybrid analytical models. The proposed models focus on timeliness of pending hits and prefetches and account for a limited number of MSHRs. They improve modeling accuracy of pending hits by $3.9\times$ and when modeling data prefetching, a limited number of MSHRs, or both, these techniques result in average errors of 9.5% to 17.8%. The impact of non-uniform DRAM memory latency is shown to be approximated well by using a moving average of memory access latency.

Categories and Subject Descriptors: C.4 [Computer Systems Organization]: Performance of Systems—*Modeling techniques*

General Terms: Performance

Additional Key Words and Phrases: Analytical modeling, pending hit, data prefetching, miss status holding register

ACM Reference Format:

Chen, X. E. and Aamodt, T. M. 2011. Hybrid analytical modeling of pending cache hits, data prefetching, and MSHRs. *ACM Trans. Architect. Code Optim.* 8, 3, Article 10 (October 2011), 28 pages.
DOI = 10.1145/2019608.2019609 <http://doi.acm.org/10.1145/2019608.2019609>

1. INTRODUCTION

To design a new microprocessor architects typically create or extensively modify a cycle-accurate simulator and run numerous simulations to quantify performance trade-offs. Both simulator creation or modification and data collection using the simulator can be significant components of overall design-to-market time. As microprocessor design cycles stretch with increasing transistor budgets, architects effectively start each new project 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 earlier version of the work in this article appeared in the *41st IEEE/ACM International Symposium on Microarchitecture (MICRO'08)* [Chen and Aamodt 2008a]. The new material in this article consists of: (1) an analysis of the trace of mcf showing the importance of accurately modeling pending cache hits in pointer chasing benchmarks; (2) a comparison of our novel compensation technique to the prior proposed fixed-cycle compensation techniques when pending hits are modeled and SWAM is applied; (3) a sensitivity analysis of our analytical models for various instruction window sizes and fixed memory access latencies; (4) an evaluation of the impact of nonuniform memory access latency due to DRAM timing and contention on the accuracy of our analytical models; (5) a detailed description and analysis of the reason that the overall average memory access latency is not enough to achieve reasonable accuracy for analytical models, as well as a proposal for using average latency in short time intervals when considering DRAM timing and contention. Author's address: T. M. Aamodt, Department of Electrical and Computer Engineering, University of British Columbia, 2356 Main Mall, Vancouver, BC, Canada V6T 1Z4; email: aamodt@ece.ubc.ca.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.

© 2011 ACM 1544-3566/2011/10-ART10 \$10.00

DOI 10.1145/2019608.2019609 <http://doi.acm.org/10.1145/2019608.2019609>

Fig. 1. Comparison of CPI component due to long data cache miss versus modeled result for *mcf* with different memory access latencies.

An orthogonal approach to obtaining performance estimates for a proposed design is analytical modeling [Yi et al. 2006]. An analytical model employs mathematical formulas that approximate the performance of the microprocessor being designed based on program characteristics and microarchitectural parameters. One of the potential advantages of analytical modeling is that it can require much less time than crafting and running a performance simulator. Thus, when an architect has analytical models available to evaluate a given design, the models can help shorten the design cycle. While cycle-accurate simulator infrastructures exist that leverage reuse of modular building blocks [Emer et al. 2002; Vachharajani et al. 2002], and workload sampling [Sherwood et al. 2002; Wunderlich et al. 2003] can reduce simulation time, another key advantage of analytical modeling is its ability to provide insights that a cycle-accurate simulator may not.

Several analytical models have been proposed before [Chow 1974; MacDonald and Sigworth 1975; Noonburg and Shen 1994, 1997; Jacob et al. 1996; Ofelt 1999; Michaud et al. 1999, 2001; Karkhanis and Smith 2004; Eyerman et al. 2009] and Karkhanis and Smith’s first-order model [Karkhanis and Smith 2004] is relatively accurate. Their first-order model separately estimates the *cycles per instruction* (CPI) component due to branch mispredictions, instruction cache misses, and data cache misses, then adds each CPI component to an estimated CPI under ideal conditions to arrive at a final model for the performance of a superscalar processor. However, little prior work has focused on analytically modeling the performance impact of data prefetching and the performance impact of hardware support for a *limited* number of overlapping long latency data cache misses due to finite MSHR resources. In this article we explore how to accurately model these important aspects of modern microprocessor designs.

One significant aspect of long latency data cache misses is the large effect of *pending data cache hits* (PH) on overall performance. In this work a pending data cache hit means a memory reference to a cache block for which a request has already been initiated by an earlier instruction but has not yet completed (i.e., the requested block is still on its way from memory). Accurately modeling pending data cache hits is important both for predicting performance of out-of-order execution and prefetching in the presence of long memory access latencies. When pending data cache hits (also referred to as *pending hits* later in this article) are treated as cache hits in the first-order model, the amount of memory level parallelism may be overestimated. This in turn may lead to an underestimate of the performance loss due to data cache misses.

For example, Figure 1 compares actual component of CPI due to long latency data cache misses to the same quantity predicted as described in the next section for *mcf*. The first bar (actual) shows the result from a cycle-accurate simulator whose configuration is described in Section 4. The second bar (baseline) shows the result from a careful

reimplementation of a previously proposed hybrid analytical model [Karkhanis and Smith 2004].¹ The third bar (SWAM w/PH) illustrates the result from a new technique that we propose in this article (see Section 3.5.1). In this figure (and throughout the rest of the paper), the CPI component resulting from long latency data cache misses ($CPI_{D\$miss}$) measures the total extra cycles due to long latency data cache misses divided by the total number of *instructions committed*. Figure 1 demonstrates that the underestimation is significant and the disparity grows with increasing memory latency.

A careful consideration of the impact of pending hits is central to our approach to modeling data prefetching. Taking account of pending hits improves our analysis of the number of misses that cannot be overlapped. We extend this analysis by proposing techniques to better select the cache misses that should be analyzed during each modeling step, further improving model accuracy. As the instruction window of future microprocessors becomes larger [Cristal et al. 2004], a limited number of *Miss Status Holding Registers* (MSHRs) can have a dramatic impact on the performance of the whole system [Tuck et al. 2006]. Modeling MSHRs requires additional consideration of which cache misses can be overlapped.

This article makes the following contributions.

- It explains why pending hits can have non-negligible impact for memory intensive applications and describes how to model their effect on performance in the context of a trace driven hybrid analytical model (Section 3.1).
- It presents a novel technique to more accurately compensate for the potential overestimation of $CPI_{D\$miss}$ in the context of the first-order superscalar model. This technique derives a compensation factor based upon individual program characteristics (Section 3.2).
- It proposes a technique to model $CPI_{D\$miss}$ when a data prefetching mechanism is applied in a microprocessor, without requiring a detailed simulator (Section 3.3).
- It describes a technique to analytically model the impact of a limited number of outstanding cache misses supported by a memory system (Section 3.4).
- It proposes two novel profiling techniques to better analyze overlapped data cache misses (Section 3.5).
- It also evaluates the impact of nonuniform memory access latency due to DRAM timing and contention on the accuracy of hybrid analytical models. This illustrates the need for further research on more accurate memory system analytical models (Section 5.8).

The proposed approach for modeling data prefetching is evaluated by using it to predict the performance impact of three different prefetching strategies and the average error of is shown to be 13.8% (versus 50.5% when pending hits are treated as normal hits). The proposed technique for modeling MSHRs reduces the arithmetic mean of the absolute error of our baseline from 33.6% to 9.5%. As with earlier hybrid modeling approaches [Karkhanis and Smith 2004; Karkhanis 2006; Karkhanis and Smith 2007], we find our model is two orders of magnitude faster than detailed simulations. Our improvements increase the realism (and hence applicability) of analytical models for microprocessor designers.

¹Our analysis shows that pending data cache hits can significantly impact model accuracy. However, we note that the prediction error of the CPI due to long latency data cache misses we report for our *baseline* modeling technique (described in Section 2) is in some cases large relative to those reported in Karkhanis and Smith [2004]. Our baseline modeling technique is our implementation of the first-order model described in Karkhanis and Smith [2004] based on the details described in that paper and the follow-on work [Karkhanis 2006; Karkhanis and Smith 2007]. We believe the discrepancy is partly due to our use of a smaller L2 cache size of 128 KB versus 512 KB used in [Karkhanis and Smith 2004], and partly due to their use of a technique similar to the one we describe in Section 3.5.1 [Karkhanis 2008].

Fig. 2. Useful instructions issued per cycle (IPC) over time used in the first-order model.

The rest of this article is organized as follows. Section 2 reviews the first-order model [Karkhanis 2006; Karkhanis and Smith 2004, 2007]. Section 3 describes how to accurately model the effects of pending data cache hits, data prefetching, and hardware that supports a limited number of outstanding data cache misses. 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 [Karkhanis and Smith 2004] 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.

Figure 2 illustrates this approach. When there are no miss-events, the performance of the superscalar microprocessor is approximated by a stable IPC, 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, the decreased IPC ramps up to the stable value under ideal conditions. A careful analysis of this behavior leads to the first-order model [Karkhanis and Smith 2004].

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. However, it has been observed that overlapping between *different types* of miss-events is rare enough that ignoring it results in negligible error in typical applications [Karkhanis and Smith 2004; Eyerman et al. 2006]. Figure 3 plots data from our simulation setup that verifies this observation for our benchmarks and realistic branch prediction and caches. The figure compares the *cycles per instruction* (CPI) obtained from our performance simulator to the CPI resulting from adding individual CPI components measured as follows. Each individual CPI component is measured as the difference in CPI when the impact on execution cycles is modeled versus when the structure is modeled as ideal during detailed simulation.

This article focuses on improving the accuracy of the modeled $CPI_{D\$miss}$ (i.e., CPI component due to long latency data cache misses) since it is the component with the largest error in prior first-order models [Karkhanis 2006; Karkhanis and Smith 2004]. Note that short latency data cache misses (i.e., L1 data cache misses that hit in the L2 cache in this paper) are not regarded as miss-events in prior first-order models [Karkhanis 2006; Karkhanis and Smith 2004] and they are modeled as long-execution-latency instructions when modeling the base CPI. In the rest of this paper,

Fig. 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.

we use the term “cache misses” to represent long latency data cache misses. As noted by Karkhanis and Smith [2004], the interactions between microarchitectural events of the same type cannot be ignored.

Our baseline technique for modeling data cache misses, based upon Karkhanis and Smith’s first-order model [Karkhanis and Smith 2004], analyzes dynamic instruction traces created by a cache simulator. To differentiate such models, which analyze instruction traces, from earlier analytical models [Chow 1974; MacDonald and Sigworth 1975; Agarwal et al. 1989; Noonburg and Shen 1994; Jacob et al. 1996; Noonburg and Shen 1997] that do not, we also refer to them as *hybrid analytical models* in this paper. In each *profile step*, a ROB_{size} number of consecutive instructions in the trace are put into the *profiling window* (or block) and analyzed, where ROB_{size} is the size of the reorder buffer. If all of the loads missing in the data cache in a profile step are data independent of 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 modeled to be hidden under this dependency chain. The reason for limiting the profile window to the size of the reorder buffer is that long latency memory accesses can only overlap if they are in the reorder buffer simultaneously. Increasing the profile window size further might result in an overestimate of memory level parallelism.

In the rest of this article, $num_serialized_D\$miss$ represents 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

$$CPI_{D\$miss} = \frac{num_serialized_D\$miss \times mem_lat}{total_num_instructions}, \quad (1)$$

where mem_lat stands for the main memory latency and $total_num_instructions$ is the total number of instructions committed (of any type).

The $CPI_{D\$miss}$ modeled in Equation (1) often overestimates the actual $CPI_{D\$miss}$ since out-of-order execution enables overlap of computation with long latency misses. A simple solution proposed by Karkhanis and Smith [2004] is to subtract a fixed number of cycles per serialized data cache miss based upon ROB size to compensate. The intuition 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

Fig. 4. An example showing how two data independent misses (i1, i3) are connected by a pending hit (i2), upon which i3 is data dependent.

hand, 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 $mem_lat - \frac{ROB_size}{issue_width}$ [Karkhanis and Smith 2004].

It has been observed that loads missing in the cache are usually relatively old when they issue [Karkhanis and Smith 2004]; and thus, perhaps the simplest (though not most accurate) approach is to use no compensation at all [Karkhanis and Smith 2004]. 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 (1) becomes $num_serialized_D\$miss \times (mem_lat - \frac{ROB_size}{2 \times issue_width})$ [Karkhanis 2006].

3. MODELING LONG LATENCY MEMORY SYSTEMS

In this section, we describe how we model pending cache hits, data prefetching, and a limited number of MSHRs.

3.1. Modeling Pending Data Cache Hits

The method of modeling long latency data cache misses described in Section 2 profiles dynamic instruction traces generated by a cache simulator [Karkhanis and Smith 2004]. Since a cache simulator provides no timing information, it classifies the load or store bringing a block into the cache as a miss and all subsequent instructions accessing the block before it is evicted as hits.

However, the actual latency of many instructions classified as a hit by a cache simulator 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 described in Section 2.

More importantly, a significant source of errors results when two or more data independent load 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 that miss 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 inaccurate 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 from the same block). Therefore, in the actual hardware, i3 can only start execution after i1 gets its data from memory although there is no true data dependence

Fig. 5. Impact of pending data cache hit latency on the CPI due to long latency cache misses (measured over all instructions committed from a detailed simulator).

between i1 and either i2 or i3. This scenario is common since most programs contain significant spatial locality. The appropriate way to model this situation is to consider i1 and i3 to be serialized in our analytical model, even though they are data independent and access distinct cache blocks.

Figure 5 shows the 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 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 this figure, we observe that the difference is significant for *eqk*, *mcf*, *em*, *hth* [Zilles 2001], and *prm*.

To model the effects of pending data cache hits analytically, we first need to identify them without a detailed simulator. At first, this may seem impossible since there is no timing information provided by 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 accesses data from a cache block that was first brought into the cache by an instruction still in the profiling window (equal to ROB_{size}), 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 brought 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 (i2), we model a dependence between the early miss (i1) and the instruction that is data dependent on the pending hit (i3) since the two instructions (i1 and i3) have to execute serially due to the constraints of the microarchitecture.

3.1.1. An Example from *mcf* Trace. To further illustrate the importance of accurately modeling pending cache hits in analytical models, Figure 6 shows a snapshot of data dependency chain from benchmark *mcf* in a profiling window. Each circle with an instruction sequence number in Figure 6 represents a dynamic instruction in the trace (note only relevant instructions are shown in the figure). Shaded circles stand for loads that miss in the data cache and circles filled with hatching represent pending cache hits.

In this figure i20 is a pending cache hit because it accesses data on the same block as i5 (i.e., block_A). There is a true data dependency between i20 and i33 (denoted by the solid arrow line from i20 to i33) since i33 needs data from i20 to calculate its effective address due to the pointer chasing behavior in *mcf*; therefore, although there is no true

Fig. 6. An example from the benchmark *mcf* showing the importance of correctly modeling pending cache hits (each circle represents a dynamic instruction).

data dependency between *i5* and *i33*, their executions must be serialized (denoted by the broken arrow line from *i5* to *i33*) since they are connected by the pending hit *i20*. In other words, *i5* and *i33* should be considered to be on the same data dependency chain when updating *num_serialized_D\$miss* after this profile step. In this common example in *mcf*, a similar pattern repeats seven times in a 256-entry profiling window (note that Figure 6 only shows three repetitions) and *num_serialized_D\$miss* should be incremented by eight since executions of the eight loads that miss in the cache (e.g., *i5*, *i33*, *i57*, and *i85* in Figure 6) are serialized.

On the other hand, if the pending cache hits (e.g., *i20*) are not modeled (e.g., by including the broken arrow lines in Figure 6) then executions of all the loads that miss in the data cache are modeled as being overlapped when assuming no limit to the maximum number of outstanding cache misses.² Then, *num_serialized_D\$miss* will only be incremented by one after the profile step, significantly underestimating the impact on CPI resulting from long latency data cache misses.

3.2. Accurate Exposed Miss Penalty Compensation

While the model described in Section 2 uses a fixed number of cycles to adjust the modeled $CPI_{D\$miss}$, 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 of the benchmarks that we studied, resulting in large modeling errors (see Figure 12). 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 missed in the cache (we define the distance between two instructions to be

²The case when the number of outstanding cache misses is limited is discussed in detail in Section 3.4.

the difference between their instruction sequence number). This is because when a load instruction misses in the cache, most of the instructions between that load and the immediately preceding long latency miss are independent of that load. Therefore, we approximate the latency of the later load that can be overlapped with useful computation 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 $CPI_{D\$miss}$. If the distance between two misses exceeds the window size, it is truncated since the miss latency can be overlapped by at most $ROB_{size} - 1$ instructions.

Equation (2), below, 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 (1).

$$CPI_{D\$miss} = \frac{num_serialized_D\$miss \times mem_lat - comp}{total_num_instructions}$$

$$comp = \left(\frac{dist}{issue_width} \times num_D\$miss \right). \quad (2)$$

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. 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.

3.3. Modeling Data Prefetching

Data prefetching is a technique to bring data from memory into the cache before it is required so as to hide (or partially hide) long memory access latency. Many hardware data prefetching strategies have been proposed before [Gindele 1977; Smith 1982; Baer and Chen 1991; Jouppi 1990]. In this section, we demonstrate how to extend our model described in Section 3.1 to estimate the $CPI_{D\$miss}$ when a data prefetching technique is employed without running detailed simulations.

To model the $CPI_{D\$miss}$ when a particular prefetching method is applied, the cache simulator is extended to implement that prefetching method while generating the dynamic instruction trace. While this does require some coding, we found that the resulting analytical model obtains very accurate results and is two orders of magnitude faster than detailed simulations. As described in Section 3.1, when our cache simulator generates an instruction trace, each memory access instruction in the trace is labeled with the sequence number of the instruction that first brought the data into the cache. If the data required by a load was brought into the cache by a prefetch, then the load is labeled with the sequence number of the previous instruction that triggered the prefetch.

In Section 3.1, when no prefetching mechanism is applied, an instruction trace generated by the cache simulator is analyzed by dividing it into ROB_{size} blocks of instructions called a profile window and each profile window is analyzed in a profile step. In each profile step, the maximum number of loads that are in a dependence chain and miss in the cache is recorded. However, when an effective prefetching method is implemented, many loads that would have missed in the cache become hits. To be more specific, many of them become pending hits, given that some of the prefetches cannot fully hide the memory access latency. We found that to accurately model prefetch performance, it is necessary to accurately approximate the timeliness of the prefetches and consequently the latencies of these pending hits.

Fig. 7. Algorithm for analyzing a pending hit in an instruction trace when a prefetching mechanism is applied.

Figure 7 illustrates how we analyze a pending hit in an instruction trace when a particular prefetching mechanism is applied. We extend the definition of a pending hit in Section 3.1 to include demand hit accesses to a block which was brought into the cache by a prefetch that was in turn triggered by an instruction earlier in the same profile window. All pending hits, whether caused by demand miss or prefetch, are analyzed using the algorithm in Figure 7. For each pending hit (crntInst in Figure 7), we find the instruction (prevInst in Figure 7) that brought crntInst's required data into the cache. We approximate crntInst's latency based upon the observation that typically the further prevInst is from crntInst, the more latency of crntInst can be hidden. The *hidden latency* of crntInst is estimated as the number of instructions between crntInst and prevInst divided by the issue width of the microprocessor being modeled.

Note that we employ the approximation of ideal CPI equal to $1/\text{issueWidth}$ in this calculation. Then, crntInst's latency is estimated as the difference between the memory access latency and the hidden latency, or zero if the memory latency is completely hidden. This latency is in cycles, and we normalize it by dividing it by the main memory latency since the accumulated *num.serialized.D\$miss* after each profile step is represented in units of main memory latency.

The part of the code marked B in Figure 7 models a significant phenomenon (late or tardy prefetches) that we observed in our study of various hardware prefetching mechanisms. Since the instruction trace being analyzed is generated by a cache simulator that is not aware of the out-of-order execution of the superscalar microprocessor being modeled, a pending hit due to prefetching indicated by the cache simulator is often actually a miss during out-of-order execution.

Figure 8 shows a simplified example illustrating how this may happen. In this example, there are eight instructions and they are labeled from i1 to i8 in program order. Figure 8 shows the data dependency graph constructed during profiling according to an instruction trace generated by a cache simulator assuming the pseudocode marked B in Figure 7 is not included. In Figure 8, i1 and i5 are loads missing in the data cache (represented by the shaded circles) and i6 triggers a prefetch that brings the data accessed by a load i8 into the cache when i6 issues (represented by the broken line arrow

Fig. 8. A tardy prefetch example motivating Figure 7 part B. Here tardy means the prefetch is triggered after the demand access.

Fig. 9. A timely prefetch example to illustrate application of Figure 7 part C. Here timely means the prefetch is initiated before the demand access.

labeled “prefetch” from i6 to i8). For each instruction, the longest normalized length of the data dependency chain up to and including that instruction is shown (in units of main memory latency). For example, “i3.length=1” above i3 in Figure 8 means that it takes one memory access latency from when i1 (the first instruction in the profile window) issues until i3 finishes execution since i3 is data dependent on i1, which missed in the cache. Since i8 is a pending hit (represented by the circle filled with hatching) and the associated prefetch is started when i6 issues, i8.length is calculated, without B, as the sum of i6.length and i8.lat, where i8.lat is estimated in part A in Figure 7 as $\frac{memLat - \frac{i8.iseq - i6.iseq}{issueWidth}}{memLat}$. In this example, i8.lat is almost equal to 1.0 since i8 is very close to i6.

Although the data accessed by i8 is regarded as being prefetched by the algorithm in Figure 7 without B, i8 is actually (as determined by detailed simulation) a miss rather than a pending hit due to out-of-order execution. In Figure 8, we observe that i6.length is bigger than i7.length. Therefore, before i6 (e.g., a load instruction) issues (and hence triggers a hardware generated prefetch), i8 has already issued and missed in the data cache. Thus, the prefetch provides no benefit. The code marked B in Figure 7 accurately takes account of this significant effect of out-of-order scheduling by checking if `crntInst (i8)` issues before the prefetch is triggered. We observed that removing part B in Figure 7 increases the average error for the three prefetching techniques that we model from 13.8% to 21.4% while adding part B slows our model by less than 2%.

An example in Figure 9 shows how the part of code marked C in Figure 7 models the case when a useful prefetch occurs in out-of-order execution (i.e., a prefetch which lowers CPI). In Figure 9, only nine relevant instructions are shown out of the 256 instructions included in a profile window (assuming ROB_{size} is 256). Among these nine instructions, i1 and i4 are loads that miss in the data cache and both i3 and i85 trigger prefetches, making i83 and i245, respectively, pending hits.

The number of cycles hidden in the prefetch triggered by i3 is estimated as $\frac{i83.iseq - i3.iseq}{issueWidth} = \frac{83 - 3}{4} = 20$ (when issue width is four), and then the remaining latency after normalization is calculated as $\frac{memLat - 20}{memLat} = 0.9$ (we assume throughout our

Fig. 10. 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.

examples that memory access latency is 200 cycles). However, since i83 is data dependent on i4 and $i4.length=2$, when i83 issues, its prefetched data has already arrived at the data cache and its real latency becomes zero (this case corresponds to the “else part” inside of part C in Figure 7).

The number of cycles hidden by the prefetch for i245 is estimated (from part A in Figure 7) as $\frac{i245.iseq - i85.iseq}{issueWidth} = \frac{245 - 85}{4} = 40$ with remaining normalized latency of $\frac{memLat - 40}{memLat} = 0.8$. Since the instruction triggering the prefetch (i85) and the instruction that i245 directly depends on (i86) finish execution around the same time (i.e., $i85.length = i86.length$), $i245.length$ becomes 2.8 and $i245.lat$ becomes 0.8 (this case corresponds to the “if part” inside of part C in Figure 7).

We note that the importance of modeling pending hits may decrease as the fraction of prefetch requests that are timely increases.

3.4. 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 since the maximum number of outstanding cache misses the system supports is limited by the number of *Miss Status Holding Registers* (MSHRs) [Kroft 1981; Farkas et al. 1995; Belayneh and Kaeli 1996] in the processor. In a real processor, the issue of memory operations to the memory system has to stall when available MSHRs run out.

Based on 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 first stop putting instructions into the profiling window when the number of instructions that miss in the data cache and 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 to that point.³

Figure 10 illustrates how the profiling technique works when the number of outstanding cache misses supported is limited to four. Once we 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 $num_serialized_D\$miss$

³In 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. Reducing the profiling window size only approximates the performance loss due to a limited number of MSHRs. We leverage this observation in Section 3.5.2.

Fig. 11. An example comparing plain profiling and SWAM profiling with $ROB_{size} = 8$. Each arrow is a dynamic instruction. Data cache misses are shaded.

is incremented by only one. Although $i7$ also misses in the cache, it is included in the next profile window since all four MSHRs have been used.

3.5. Profiling Window Selection

In this section, we present two important refinements to the profiling technique described in Section 2 (which we will refer to hereafter as *plain profiling*) to better model the overlapping between cache misses.

3.5.1. Start-With-A-Miss (SWAM) Profiling. We observe that often the plain profiling technique described in Section 2 does not account for all of the cache misses that can be overlapped, due to the simple way in which it partitions an instruction trace. Figure 11(a) shows an 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 based upon multiples of ROB_{size} (for example, $i1, i9, i17 \dots$, when ROB_{size} is eight and the first instruction is $i1$). Therefore, although the latency of $i5, i7, i9$, and $i11$ can be overlapped, the plain profiling technique does not account for this.

By making each profile window start with a cache miss, we find that the accuracy of the model improves significantly. Figure 11(b) illustrates this idea. Rather than starting a profile window with $i1$, we start a profile window with $i5$, so that the profiling window will include $i5$ to $i12$. Then, the next profile window will seek and start with the first cache miss after $i12$. We call this technique start-with-a-miss (SWAM) profiling and in Section 5.1 we will show that it decreases the error of plain profiling from 29.3% to 10.3% with unlimited MSHRs.

We explored a sliding window approximation (start each profile window on a successive instruction of any type), but found it did not improve accuracy while being slower. SWAM improves modeling accuracy because it more accurately reflects what the contents of the instruction window of a processor would be (a long latency miss would block at the head of the ROB).

3.5.2. Improved SWAM for Modeling a Limited Number of MSHRs (SWAM-MLP). The technique for modeling MSHRs proposed in Section 3.4 can be combined with SWAM to better model the performance when the number of outstanding cache misses supported by the memory system is limited. Perhaps the most straightforward approach for doing this is to have each profile window start with a miss and finish either when the number

Table I. Microarchitectural Parameters

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

of instructions that have been analyzed equals the size of the instruction window (ROB_{size}) or when the number of cache misses that have already 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 end a profile window when the number of cache misses that are data independent of prior misses in the profile window equals the total number of MSHRs. In the rest of this paper we call this technique SWAM-MLP since it improves SWAM by better modeling memory level parallelism. When a miss depends on an earlier miss in the same profiling window, the later miss cannot issue until the earlier one completes and SWAM-MLP improves model accuracy because it takes into account that out-of-order execution can allow 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 window should, in this case, be more than the total number of MSHRs.

A potential limitation of SWAM-MLP is modeling banked caches with per bank MSHR structures [Tuck et al. 2006]. Such banking introduces the possibility that isolated accesses within the profile window will be unable to be overlapped. We leave the study of extensions of SWAM-MLP that would address banked MSHRs to future work.

4. METHODOLOGY

To evaluate our analytical model, we have modified SimpleScalar [Burger and Austin 1997] to simulate the performance loss due to long latency data cache misses when accounting for a limited number of MSHRs. We compare against a cycle accurate simulator rather than real hardware to validate our models since a simulator provides insights that would be challenging to obtain without changes to currently deployed superscalar performance counter hardware [Eyerhan et al. 2006]. We believe the most important factor is comparing two or more competing (hybrid) analytical models against a single detailed simulator provided the latter captures the behavior one wishes to model analytically.

Table I describes the microarchitectural parameters used in this study. Note that we are focusing on predicting only the CPI component for data cache misses using our model. Hence, our comparisons in Section 5 is to a detailed cycle accurate simulator in which instruction cache misses have the same latency as hits and all branches are predicted perfectly. In the rest of this paper, we focus on how to accurately predict $CPI_{D\$miss}$, which is the performance loss 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 Karkhanis and Smith [2004]).

To evaluate the technique proposed in Section 3.3 for estimating the $CPI_{D\$miss}$ when a prefetching mechanism is applied, we have applied our modeling techniques to predict the performance benefit of three different prefetching mechanisms: prefetch-on-miss [Smith 1982], tagged prefetch [Gindele 1977], and stride prefetch [Baer and Chen

Table II. 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

1991]. When prefetch-on-miss [Smith 1982] is applied, an access to a cache block that results in a cache miss will initiate a prefetch for the next sequential block in memory given that the block is not in the cache. The tagged prefetch mechanism [Gindele 1977] adds a tag bit to each cache block to indicate whether the block was demand-fetched or prefetched. When a prefetched block is referenced, the next sequential block is prefetched if it is not in the cache. The stride prefetch technique [Baer and Chen 1991] uses a *reference prediction table* (RPT) to detect address referencing patterns. Each entry in the RPT is assigned a state and a state machine is applied to control the state of each entry. Whether a prefetch is initialized or not depends on the current state of the entry [Baer and Chen 1991]. In this study, we modeled a 128-entry, 4-way RPT that is indexed by the microprocessor's program counter (PC).

To stress our model, we simulate a relatively small L2 cache compared to contemporary microprocessors. We note that the size of the L2 cache that we simulated is close in size to those employed in microprocessors shipped at the time when those benchmarks we use were released. The benchmarks chosen are ones from SPEC 2000 [Standard Performance Evaluation Corporation] and OLDEN [Carlisle 1996] that have at least 10 long latency data cache misses for every 1000 instructions simulated (10MPKI). Table II illustrates the miss rates of these benchmarks and the labels used to represent them in figures. Moreover, for each benchmark, we select 100M representative instructions to simulate using the Sim-Point toolkit [Sherwood et al. 2002].

In this article, we use arithmetic mean of the absolute 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 that these numbers might lead to different conclusions. In all cases the improvements resulting from applying our new modeling techniques are robust enough that the selection of averaging technique does not impact our conclusions.

5. RESULTS

This section summarizes our experimental results.

5.1. Modeling Pending Data Cache Hits

Section 2 describes prior proposals for compensating for the overestimation of modeled penalty cycles per serialized miss using a fixed number of cycles. Figure 12(a)

Fig. 12. Penalty cycles per miss with fixed number of cycles compensation for plain profiling (Unlimited MSHRs).

and Figure 12(b) illustrate the modeled results after compensation with constant cycles both without, and with the pending hit compensation technique described in Section 3.1, respectively. In these two figures, we show results using five different constant compensation factors. The first bar (oldest) corresponds to 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 the next two bars, (1/2) and (3/4). The fifth bar (youngest) 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 cycle accurate simulation.

From this data, we observe that there is no one fixed cycle compensation method that performed consistently the best for all of the benchmarks we studied. For example, in Figure 12(a) we observe that error is minimized using “youngest” for *app*, *art*, *luc*, *swm*, and *lbm*, but minimized using “oldest” for *em*, *mcf*, and *hth*, while, *eqk* and *prm* requires something in-between. The harmonic mean for each fixed cycle compensation method is also shown and we notice that, due to the fact that positive and negative errors cancel out, the harmonic means of some fixed cycle compensation methods appear close to the detailed simulation results. However, it is important to recognize that their accuracy on individual benchmarks 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, resulting when employing “youngest” compensation.

Fig. 13. CPI due to D\$miss and modeling error for different profiling techniques (Unlimited MSHRs).

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 described in Section 3.2. Figure 13(a) compares the $CPI_{D\$miss}$ for both the plain profiling technique described in Section 2 and the start-with-a-miss (SWAM) profiling technique described in Section 3.5.1 (with pending hits modeled) to the results from detailed simulation. The first bar (Plain w/o comp) and the third bar (SWAM w/o comp) correspond to the modeled results without any compensation; the second bar (Plain w/comp) and the fourth bar (SWAM w/comp) are the modeled results with the compensation technique described in Section 3.2. The comparison of our novel compensation technique and the prior static compensation proposals is shown in Section 5.2.

Figure 13(a) and Figure 13(b) show that for benchmarks with heavy pointer chasing 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 this is that many data independent misses are connected by pending cache hits, which must be appropriately modeled. Moreover, as we expect, SWAM profiling is more accurate than plain profiling since it can capture more overlapping data cache misses. Figure 13(b) illustrates the error of each modeling technique after compensation. From Figure 13(b) we observe that the arithmetic mean of the absolute error (mean) decreases from 39.7% to 29.3% when modeling pending cache hits. Overall, we find SWAM combined with modeling of pending hits (SWAM w/PH) has about 3.9 times lower error than plain profiling without modeling pending hits (Plain w/o PH).⁴ Note that accuracy improves not just for “microbenchmarks” [Zilles 2001]: In Figure 13(b), comparing “Plain w/o

⁴The arithmetic mean of the absolute error decreases from 39.7% to 10.3%. Geometric mean of the absolute error decreases from 26.4% to 8.2%, and harmonic mean of the absolute error decreases from 15.3% to 6.9%.

Fig. 14. Modeling error of various compensation techniques when pending hits are modeled and SWAM is applied (Unlimited MSHRs).

PH” to “SWAM w/ PH,” we find that, on average, the arithmetic mean of the absolute error decreases from 31.6% to 9.1% for the five SPEC 2000 benchmarks *excluding mcf*.

5.2. Novel CPI Adjustment Technique

Figure 14 compares the modeling error of our novel compensation technique as described in Section 3.2 (labeled as “new”) to the prior proposed compensation techniques using a fixed number of cycles described at the beginning of this section (labeled as “oldest”, “1/4”, “1/2”, “3/4”, and “youngest”), when pending hits are modeled and SWAM is applied. From Figure 14 we observe that, when applying fixed cycle compensation techniques, for applications such as *mcf*, *em*, and *hth*, 1/4 (i.e., assuming that an instruction that misses in the data cache is always younger than $\frac{1}{4}ROB_{size}$ in-flight instructions in the instruction window) minimizes the error; for applications such as *app*, *art*, *luc*, *swm*, *prm*, and *lbm*, youngest (i.e., assuming that an instruction that misses in the data cache is always the youngest one in the instruction window) minimizes the error; for *eqk*, the lowest error is achieved at “3/4” (i.e., somewhere in-between). On average, the optimal fixed cycle compensation technique is youngest. However, rather than curve-fitting the actual result, our new compensation technique takes into account of the individual program characteristics and improves the accuracy of the best fixed cycle compensation technique (youngest) by 33.9%, reducing the arithmetic mean of absolute error from 15.5% to 10.3%.

5.3. Modeling Different Prefetching Techniques

In this section, we evaluate $CPI_{D\$miss}$ when modeling the three prefetching techniques mentioned in Section 4 with unlimited MSHRs. Figure 15(a) compares the actual $CPI_{D\$miss}$ to the modeled one for the three prefetching methods. For each prefetching method, both the prediction when each pending hit is analyzed according to the algorithm described in Figure 7 (labeled “w/PH”) and the prediction when pending hits are treated as normal hits (labeled with “w/o PH”) are shown. We use SWAM in both cases. When employing the algorithm in Figure 7, we apply SWAM as follows: When we analyze the trace we let each profile window start with a miss or a hit due to a prefetch. The latter refers to a demand request whose data was brought into the data cache by a previous prefetch (we start with it since its latency may not be fully hidden and thus it may stall commit). Figure 15(b) shows the error of the model for each benchmark.

Fig. 15. CPI due to D\$miss and modeling error while prefetch-on-miss (POM), tagged prefetch (Tag), or stride prefetch (Stride) technique is applied.

From Figure 15(b) we observe that if pending hits are not appropriately modeled (i.e., a pending hit is simply treated as a hit and not analyzed based upon the algorithm in Figure 7), the modeled $CPI_{D\$miss}$ always underestimates the actual $CPI_{D\$miss}$. The reason is that with a prefetching technique applied, a large fraction of the misses occurring when there is no prefetching become pending hits since prefetches generated by that prefetching technique cannot fully hide the memory access latency of those misses. By using the method of analyzing pending hits that we propose in Section 3.3 to model prefetching, the arithmetic mean of the absolute error decreases from 22.2% to 10.7% for prefetch-on-miss, from 56.4% to 9.4% for tagged prefetch, and from 72.9% to 21.3% for stride prefetch (i.e., the arithmetic mean of the absolute error decreases from 50.5% to 13.8% overall for the three data prefetching techniques modeled).

5.4. Modeling Limited Number of MSHRs

All of the results that we have seen thus far are for modeling a processor with an unlimited number of MSHRs. This section compares modeled $CPI_{D\$miss}$ when the number of available MSHRs is limited. Figures 16(a), 17(a), and 18(a) compare the modeled $CPI_{D\$miss}$ to the simulated results when the maximum number of MSHRs in a processor is sixteen, eight, and four, respectively. We show data for eight MSHRs and four MSHRs since we note that Prescott has only eight MSHRs [Boggs et al. 2004] and Williamette has only four MSHRs [Hinton et al. 2001]. For each benchmark, the first bar (Plain w/o MSHR) shows the modeled $CPI_{D\$miss}$ from plain profiling (i.e., it is not aware that there are a limited number of MSHRs and always provides the same result for a given 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 3.4) included. The third and the fourth bar illustrates the modeled

Fig. 16. CPI due to D\$miss and modeling error for $N_{MSHR} = 16$. Fig. 17. CPI due to D\$miss and modeling error for $N_{MSHR} = 8$.

$CPI_{D\$miss}$ from SWAM (Section 3.5.1) and SWAM-MLP (Section 3.5.2), respectively. For these four profiling techniques, pending hits are modeled using the method described in Section 3.1. The modeling error based on the data in Figure 16(a)–18(a) is illustrated in Figure 16(b)–18(b).

SWAM-MLP is consistently better than SWAM. We observe that as the total number of MSHRs decreases, the advantage of SWAM-MLP over SWAM becomes significant, especially for *eqk*, *mcf*, *em*, and *hth*, for which it is more likely to have data dependence among cache misses thus affecting the size of the profiling window that SWAM-MLP chooses. SWAM decreases the arithmetic mean of the absolute error from 32.6% (Plain w/o MSHR) to 9.8%, from 32.4% to 12.8%, and from 35.8% to 23.2%, when the number of MSHRs is sixteen, eight, and four, respectively.⁵ SWAM-MLP further decreases the error to 9.3%, 9.2%, and 9.9%⁶ (i.e., SWAM-MLP decreases the error of plain profiling (Plain w/o MSHR) from 33.6% to 9.5% when the number of MSHRs is limited). For the SPEC 2000 benchmarks excluding *mcf*, average error reduces from 48.1% to 7.0% comparing Plain w/o MSHR to SWAM-MLP.

5.5. Putting It All Together

We also evaluated the combination of the techniques for modeling prefetching (Section 3.3) and SWAM-MLP to model the performance of the three prefetching methods with limited MSHRs. On average, the error of modeling prefetching is 15.2%,

⁵Geometric mean is reduced from 19.4% to 7.4%, from 21.5% to 9.7%, and from 21.8% to 10.9%; harmonic mean is reduced from 8.5% to 5.8%, from 14.5% to 7.0%, and from 10.2% to 5.1%.

⁶Geometric mean of the absolute error is 6.5%, 6.7%, 5.2%, and harmonic mean of the absolute error is 4.6%, 5.2%, 3.3%, when the number of MSHRs is sixteen, eight, and four, respectively.

Fig. 18. CPI due to D\$miss and modeling error for $N_{MSHR} = 4$.

17.7%, and 20.5%, when the number of MSHRs is sixteen, eight, and four, respectively (average of 17.8% across all three prefetch methods).

5.6. Speedup of the Hybrid Analytical Model

One of the most important advantages of the hybrid analytical model we present in this paper versus detailed simulations is its fast speed of analysis. On average our model is 150, 156, 170, and 229 times faster than the detailed simulator when the number of MSHRs is unlimited, sixteen, eight, and four, respectively, with a minimum speedup of $91\times$. Moreover, for estimating the performance impact of prefetching, on average our model is 184, 185, 215, and 327 times faster than the detailed simulator when the number of MSHRs is unlimited, sixteen, eight, and four, respectively, with a minimum speedup of $87\times$. These speedups were measured on a 2.33 GHz Intel Xeon E5345 processor.

5.7. Sensitivity Study

In this section we perform several sensitivity studies to test the accuracy of our analytical models for various main memory access latency and instruction window size.

Figure 19(a) to 19(d) compare the predicted $CPI_{D\$miss}$ to the result obtained from the simulator described in Section 4 for various main memory latency (200, 500, and 800 cycles), when the number of MSHRs is unlimited, sixteen, eight, and four, respectively. We keep other parameters the same as Table I. For each figure, there are three data points for each application, corresponding to three different memory latencies. From these figures we observe that our models accurately track the impact of varying main memory latency on the CPI due to long data cache misses. Overall, the arithmetic mean of absolute error calculated over all data points is 9.39% and the correlation coefficient between the predicted and simulated results is 0.9983. The error is relatively constant as memory latency varies (errors of 10.9%, 9.0% and 8.3% for memory latencies of 200, 500 and 800 cycles, respectively).

Figure 20(a) to 20(d) compare the predicted $CPI_{D\$miss}$ to the result obtained from the simulator described in Section 4 for various instruction window size (64, 128, and 256), when the number of MSHRs is unlimited, sixteen, eight, and four, respectively. We keep other parameters the same as Table I. For each figure, there are three data points for each application, corresponding to three different instruction window sizes. From these figures we observe that our models accurately track the impact of varying instruction window size on the CPI due to long data cache misses. Overall, the arithmetic mean of absolute error calculated over all data points is 9.26% and the correlation coefficient between the predicted and simulated results is 0.9951. The error is relatively constant as instruction window size varies (errors of 8.1%, 8.7% and 10.9% for window sizes of 64, 128 and 256, respectively).

Fig. 19. Comparison of the predicted $CPI_{D\$miss}$ to the measured $CPI_{D\$miss}$ from detailed simulation for various main memory access latency (200, 500, and 800 cycles).

Fig. 20. Comparison of the predicted $CPI_{D\$miss}$ to the measured $CPI_{D\$miss}$ from detailed simulation for various instruction window size (64, 128, and 256).

Table III. DRAM Timing Parameters

Parameter	# cycles	Parameter	# cycles
t_{CCD}	4	t_{WL}	2
t_{RRD}	2	t_{WTR}	2
t_{RCD}	3	t_{RP}	3
t_{RAS}	8	t_{RC}	11
t_{CL}	3		

5.8. Impact of DRAM Timing on the Accuracy of Analytical Models

All the models we have discussed thus far assume a uniform fixed memory access latency (as the detailed simulator described in Section 4 does). In other words, these models ignore the nonuniformity of memory access latency due to DRAM timing and contention. In this section, we show the impact of DRAM timing and contention on the accuracy of our analytical models and discuss how our models can be applied when the assumption of uniform fixed memory access latency is removed. We find that simply using the average memory access latency across an entire application execution results in very large errors but that using the average latency over shorter intervals can significantly reduce this error.

To study the impact of the nonuniform memory access latency caused by DRAM timing and contention, we simulate a DRAM system based upon DDR2-400 [Micron Technology, Inc]. Table III shows the detailed DRAM timing parameters we simulate. For our study, we simulate eight banks for a DRAM chip and assume that the frequency of the microprocessor that we model is five times the DRAM frequency. We use first-come first-served (FCFS) as our DRAM scheduling policy.

Figure 21(a) compares the simulated $CPI_{D\$miss}$ (labeled with “actual”), when DRAM timing is modeled, to the $CPI_{D\$miss}$ predicted by our analytical model (labeled with “SWAM_avg_all_inst”) that accounts for pending hits and uses the compensation technique described in Section 3.2 along with the SWAM technique described in Section 3.5.1. Here we use the average memory access latency collected over all load instructions from the detailed simulator for the $mem.lat$ in Equation (2). Note we have assumed that the average memory access latency is available. Figure 21(b) shows the error of the technique for individual benchmark and the arithmetic mean of the absolute error over all benchmarks. From the figure we notice that the accuracy of our analytical models decreases significantly, compared to the case when the memory access latency is assumed to be fixed (see Figure 13). The arithmetic mean of the absolute error over all benchmarks is 117.1%. We address the reason for the significant error and propose solution in the following.

The significant error from SWAM_avg_all_inst in Figure 21(b) indicates that the average memory access latency over all instructions itself is not sufficient to capture all the information that we need to accurately predict the CPI due to data cache misses when DRAM timing and contention is taken into account. Figure 22 illustrates, for each benchmark, the average memory access latency of load instructions calculated every 1024 instructions.⁷ Moreover, Figure 22 shows, by horizontal lines, the average memory access latency calculated over all instructions that is used to predict the result from SWAM_avg_all_inst as shown in Figure 21(a). We notice that the average memory access latency calculated over all instructions is not a good metric capturing the nonuniformity of memory access latency over time. As Figure 22(f) illustrates, although the average memory access latency for *mcf* is quite high, the fraction of instructions experiencing

⁷Note we only show the first 10000 groups of 1024 instructions per benchmark for clarity of figures.

Fig. 21. Comparison of the simulated CPI due to D\$miss accounting for DRAM timing to the predicted result using the average memory access latency (Unlimited MSHRs).

a very high latency is relatively low. On average, during 9373 groups out of the 10000 groups of 1024 instructions shown in Figure 22(f), the average memory access latency is significantly lower than the global average (i.e., the horizontal line), resulting in a $7.7\times$ overestimate for the actual $CPI_{D\$miss}$ when using the average latency calculated over all instructions, as shown in Figure 21(a).

To better account for the impact of the nonuniform memory latency on the performance, we use the average memory access latency collected every 1024 instructions, rather than the average over all instructions, for the mem_lat in Equation (2), while corresponding predicted $CPI_{D\$miss}$ and modeling error is shown in Figure 21(a) and Figure 21(b) (labeled with “SWAM_avg_1024_inst”), respectively. Here we assume that an analytical model is available to predict the average memory access latency during a certain number of instructions given an instruction trace. The detail of such a model is beyond the scope of this work and is left as our future work. We notice that, by using the fine-grained average for a group of instructions to better take into account the nonuniform behavior of memory access latency, SWAM_avg_1024_inst improves the accuracy of SWAM_avg_all_inst by a factor of 5.3, reducing the average error from 117% to 22%. This compares favorably to the 10.3% error

Fig. 22. Average memory access latency of loads calculated every 1024 instructions. Horizontal lines show the average memory access latency calculated over all instructions.

obtained in Section 5.1 when comparing to a detailed simulator using a fixed memory latency.

Recently, many proposals have been put forward for more aggressive memory controllers [Rixner et al. 2000; Nesbit et al. 2006; Mutlu and Moscibroda 2008; Kim et al. 2010] including some that intelligently prioritize demand versus prefetch memory requests [Lee et al. 2008]. Such memory controllers can create wider distributions in memory access latency while increasing effective memory bandwidth. Given the increase in modeling error we observe when using a relatively simple FCFS memory controller we believe developing accurate analytical models for DRAM memory access is an important area for future work.

6. RELATED WORK

There exist many analytical models proposed for superscalar microprocessors [Noonburg and Shen 1994, Ofelt 1999, Michaud et al. 1999, 2001]. A common limitation of early models is that they assume a perfect data cache. As the gap between memory and microprocessor speed increases, data cache misses must be properly modeled to achieve reasonable accuracy.

Agarwal et al. [1989] present an analytical cache model estimating cache miss rate given cache parameters. However, in a superscalar, out-of-order execution microprocessor, the cache miss rate itself is not enough to predict the real performance of the program analyzed. Jacob et al. [1996] propose an analytical memory model that applies a variational calculus approach to determine a memory hierarchy optimized for the average access time given a fixed hardware budget with an (implicit) assumption that memory level parallelism does not occur. Noonburg and Shen [1997] present a superscalar microprocessor Markov chain model that does not model long memory latency. The first-order model proposed by Karkhanis and Smith [2004] 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 technique described in Section 2 (based upon details available in the literature) by modeling pending data cache hits, and extends it to estimate the performance impact of data prefetching techniques and a limited number of outstanding cache misses.

More recently, Eyerman et al. [2009] proposed a modification to the first-order model simplifying the analysis by examining the impact of miss events at the dispatch rather than issue stage. This formulation is useful in examining scaling relationships between different pipeline parameters and can be used to show that optimal performance is achieved at points where the pipeline depth times the square root of the pipeline width is a constant. The focus on pending hits, prefetching and miss status holding registers in this article is orthogonal. The use of SWAM profiling may capture benefits similar to profiling individual miss intervals.

Some earlier work has investigated the performance impact increasing the number of MSHRs using detailed simulations [Farkas et al. 1995; Belayneh and Kaeli 1996]. The hybrid analytical model that we propose can be used to estimate the performance impact of a limited number of MSHRs without requiring a detailed simulator.

Concurrent with our work, Eyerman [2008] proposed an approach similar to SWAM described in Section 3.5.1, except that the profile window slides to begin with each successive long latency miss. Reportedly, a pending hit compensation mechanism was used [Esckhout 2008].

7. CONCLUSIONS

In this article, several improvements to an existing analytical performance model for superscalar processors are proposed and evaluated. A proposed technique for modeling pending data cache hits reduces error from 39.7% to 29.3% while a technique for selecting the beginning of profile windows reduces error further to 10.7%. Accounting for the impact of program characteristics on the overlap of computation and memory access error reduces error by 33.9% compared to fixed compensation techniques. The basic model can be extended to estimate the performance of a microprocessor when an arbitrary data prefetching method is applied. Moreover, a technique to quantify the impacts of a limited number of MSHRs is proposed. The average error is 13.8% when modeling several prefetching strategies and 9.5% when modeling a limited number of supported outstanding cache misses. This paper also explores the sensitivity of the modeling technique. As memory access latency and instruction window size varies the model achieves correlation coefficient of 0.9983 and 0.9951, respectively. Finally,

the large impact of realistic memory access latencies (accounting for DRAM timing and contention) on the data memory access component of CPI is quantified and it is shown that a windowed average of memory latency can significantly reduce error. The later results suggest that future work should explore mechanisms to analytical model the impact of memory controllers on memory access latency.

ACKNOWLEDGMENTS

We would like to thank Tejas Karkhanis, Lieven Eeckhout, Ali Bakhoda, Wilson Fung, Johnny Kuan, Andrew Turner, Henry Wong, George Yuan, and the anonymous reviewers for their valuable comments on earlier versions of this article. This work was partly supported by the Natural Sciences and Engineering Research Council of Canada.

REFERENCES

- AGARWAL, A., HOROWITZ, M., AND HENNESSY, J. 1989. An analytical cache model. *Trans. Comput. Syst.* 7, 2, 184–215.
- BAER, J.-L. AND CHEN, T.-F. 1991. An effective on-chip preloading scheme to reduce data access penalty. In *Proceedings of the Conference on Supercomputing*. 176–186.
- BELAYNEH, S. AND KAEI, D. R. 1996. A discussion on non-blocking/lockup-free caches. *SIGARCH Comput. Archit. News* 24, 3, 18–25.
- BOGGS, D., BAKTHA, A., HAWKINS, J., MARR, D. T., MILLER, J. A., ROUSSEL, P., SINGHAL, R., TOLL, B., AND VENKATRAMAN, K. 2004. The microarchitecture of the Intel Pentium 4 processor on 90nm technology. *Intel Technol. J.* 8, 1.
- BURGER, D. AND AUSTIN, T. M. 1997. TheSimpleScalar tool set, Version 2.0. *SIGARCH Comput. Archit. News* 25, 3, 13–25.
- CARLISLE, M. C. Q996. Olden: Parallelizing .programs with dynamic data structures on distributed-memory machines. Ph.D. thesis, Princeton University.
- CHEN, X. E. AND AAMODT, T. M. 2008a. Hybrid analytical modeling of pending cache hits, data prefetching, and MSHRs. In *Proceedings of the 41st Annual ACM/IEEE International Symposium on Microarchitecture*. 59–70.
- CHEN, X. E. AND AAMODT, T. M. 2008b. An improved analytical superscalar microprocessor memory model. In *Proceedings of the 4th Workshop on Modeling, Benchmarking and Simulation* 7–16.
- CHOW, C. K. 1974. On optimization of memory hierarchies. *IBM J. Resear. Dev.* 2, 194–203.
- CRISTAL, A., SANTANA, O., VALERO, M., AND MARTINEZ, J. F. 2004. Toward kilo-instruction processors. *ACM Trans. Architect. Code Optim.* 1, 4, 389–417.
- ECKHOUT, L. 2008. Personal communication.
- EMER, J., AHUJA, P., BORCH, E., KLAUSER, A., LUK, C. K., MANNE, S., MUKHERJEE, S. S., PATIL, H., WALLACE, S., BINKERT, N., ESPASA, R., AND JUAN, T. 2002. Asim: A performance model framework. *IEEE Comput.* 35, 2, 68–76.
- EYERMAN, S. 2008. Analytical performance analysis and modeling of superscalar and multithreaded processors. Ph.D. thesis, Ghent.
- EYERMAN, S., ECKHOUT, L., KARKHANIS, T. S., AND SMITH, J. E. 2006. A performance counter architecture for computing accurate CPI Components. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*. 175–184.
- FARKAS, K. I., JOUPPI, N. P., AND CHOW, P. 1995. How useful are non-blocking loads, stream buffers and speculative execution in multiple issue processors? In *Proceedings of the 1st International Symposium on High-Performance Computer Architecture*. 78.
- GINDELE, J. D. 1977. Buffer block prefetching method. *IBM Techn. Disclo. Bull.* 20, 2, 696–697.
- HINTON, G., SAGER, D., UPTON, M., BOGGS, D., CARMEAN, D., KYKER, A., AND ROUSSEL, P. 2001. The microarchitecture of the Pentium 4 processor. *Intel Techn. J.* 5, 1.
- JACOB, B. L., CHEN, P. M., SILVERMAN, S. R., AND MUDGE, T. N. 1996. An analytical model for designing memory hierarchies. *IEEE Trans. Comput.* 45, 10, 1180–1194.
- JOUPPI, N. P. 1990. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*. 364–373.
- KARKHANIS, T. S. 2006. Automated design of application-specific superscalar processors. Ph.D. thesis, University of Wisconsin.

- KARKHANIS, T. S. AND SMITH, J. E. 2004. A first-order superscalar processor model. In *Proceedings of the 31st Annual International Symposium on Computer Architecture*. 338–349.
- KARKHANIS, T. S. AND SMITH, J. E. 2007. Automated design of application specific superscalar processors: an analytical approach. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*. 402–411.
- KARKHANIS, T. S. 2008. Personal Communication.
- KROFT, D. 1981. Lockup-free instruction fetch/prefetch cache organization. In *Proceedings of the 8th Annual International Symposium on Computer Architecture*. 81–87.
- MACDONALD, J. E. AND SIGWORTH, K. L. 1975. Storage hierarchy optimization procedure. *IBM J. Resear. Dev.* 19, 2, 133–140.
- MICHAUD, P., SEZNEC, A., AND JOURDAN, S. 1999. Exploring instruction-fetch bandwidth requirement in wide-issue superscalar processors. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*. 2–10.
- MICHAUD, P., SEZNEC, A., AND JOURDAN, S. 2001. An exploration of instruction fetch requirement in out-of-order superscalar processors. *Int. J. Paral. Program.* 29, 1, 35–38.
- MICRON TECHNOLOGY, INC. 2Gb DDR2 SDRAM Component : MT47H128MI6HG-25. <http://download.micron.com/pdf/datasheets/dram/ddr2/2gbddr2.pdf>.
- NOONBURG, D. B. AND SHEN, J. P. 1994. Theoretical modeling of superscalar processor performance. In *Proceedings of the 27th Annual ACM/IEEE International Symposium on Microarchitecture*. 52–62.
- NOONBURG, D. B. AND SHEN, J. P. 1997. A framework for statistical modeling of superscalar processor performance. In *Proceedings of the 3rd International Symposium on High-Performance Computer Architecture*. 298–309.
- OFELT, D. J. 1999. Efficient performance prediction for modern microprocessors. Ph.D. thesis, Stanford University.
- SHERWOOD, T., PERELMAN, E., HAMERLY, G., AND CALDER, B. 2002. Automatically characterizing large scale program behavior. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*. 45–57.
- SMITH, A. J. 1982. Cache memories. *ACM Comput. Surv.* 14, 3, 473–530.
- STANDARD PERFORMANCE EVALUATION CORPORATION. SPEC CPU2000 benchmarks. <http://www.spec.org>.
- TUCK, J., CEZE, L., AND TORRELLAS, J. 2006. Scalable cache miss handling for high memory-level parallelism. In *Proceedings of the 39th Annual ACM/IEEE International Symposium on Microarchitecture*. 409–422.
- VACHHARAJANI, M., VACHHARAJANI, N., PENRY, D. A., BLOME, J. A., AND AUGUST, D. I. 2002. Microarchitectural exploration with liberty. In *Proceedings of the 35th Annual ACM/IEEE International Symposium on Microarchitecture*. 271–282.
- WUNDERLICH, R. E., WENISCH, T. F., FALSAFI, B., AND HOE, J. C. 2003. SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*. 84–97.
- YI, J. J., EECKHOUT, L., LILJA, D. J., CALDER, B., JOHN, L. K., AND SMITH, J. E. 2006. The future of simulation: A field of dreams. *Computer* 39, 11, 22–29.
- ZILLES, C. B. 2001. Benchmark health considered harmful. *SIGARCH Comput. Archit. News* 29, 3, 4–5.

Received September 2010; revised April 2011; accepted May 2011