Performance Potentials of Compiler-directed Data Speculation

Youfeng Wu, Li-Ling Chen, Roy Ju, Jesse Fang

Programming Systems Research Lab
Intel Labs
2200 Mission College Blvd.
Santa Clara, CA 95052-8119
{youfeng.wu, li-ling.chen, roy.ju, jesse.fang}@intel.com

Abstract

Compiler-directed data speculation has been implemented on Itanium systems to allow for a compiler to move a load across a store even when the two operations are potentially aliased. This not only breaks data dependency to reduce critical path length, but also allows a load to be scheduled far apart from its uses to hide cache miss latencies. However, the effectiveness of data speculation is affected by the sophistication of alias analysis technique as well as the aggressiveness of the instruction scheduler. In general, the more sophisticated is the alias analysis technique, the less performance gain is from data speculation, and the more aggressive is the instruction scheduler, the more opportunity is for data speculation. In this paper we evaluate in various scenarios the performance potentials of data speculation for SPEC2000C benchmarks. For each scenario, we determine the performance contributions of data speculation due to both critical path reduction and cache miss latency reduction. We also show interesting statistics about the effects of scheduling constraints, the percentage of critical dependencies, the impacts of cache miss latencies, and the distances between the load locations before and after data speculation.

***Keywords**: data speculation, instruction scheduling, alias analysis, performance evaluation, and EPIC architecture*

1. Introduction

For wide, in-order-issue processors like Itanium [1], a memory load operation often appears on a critical execution path, and needs to be scheduled earlier to improve performance. When a load may cause a cache miss, it is even more important to move it early to hide its latency. However, a load may be dependent upon aliased stores that precede the load in a program. This makes it difficult to schedule the load before the stores.

Compiler-directed data speculation has been proposed as a runtime disambiguation mechanism, which allows for a compiler to move a load across a store even when the two operations may be potentially aliased [11], [1], [7]. When a load is speculatively executed before the potentially aliased stores as an "advanced load" (ld.a), an Advanced Load Address Table (ALAT) records the load address and bookkeeping information. When a store operation is subsequently performed, the store address is compared with each valid address recorded in ALAT. If it is the same as the address in an ALAT entry, the entry is invalidated. At the original location of the load, an "advanced load check" operation (ld.c) is executed to check against possible invalidation. It examines ALAT to see if the entry for the speculatively executed load is still valid. If the entry is valid, the ld.c becomes a no-op. Otherwise, ld.c reloads the value. Figure 1 shows an example of data speculation.

store reg *p reg1 = ld *q	reg1 = **ld.a** *q store reg *p reg1 = **ld.c** *q

Figure 1. Data speculation of a load

store reg *p reg1 = ld *q reg2=use reg1	reg1 = **ld.a** *q reg2 = use reg1 store reg *p reg1 = **chk.a** *q, recovery_label recovery_label: reg1 = ld *q; reg2 = use reg1

Figure 2. Data speculation of a load and its uses

A more aggressive data speculation allows for additionally moving the operations using the result of an ld.a before stores. At the original location of the load, a

0-7803-7756-7/03/$17.00 ©2003 IEEE

"check advanced load " operation (chk.a) is placed to check against possible invalidation. If the ld.a is invalidated, chk.a transfers control to a recovery code to redo the load and all of its dependent operations that have been speculatively executed. Figure 2 shows an example of speculating a load and its dependent operations.

Data speculation can also be combined with control speculation [14] to move a load above branches as well as stores. Figure 3 shows an example in which a load is moved above a branch and a store as a "control speculative advanced load" (ld.sa).

store reg *p
if(cond)
  reg1 = ld *q
	reg1 = **ld.sa** *q
	store reg *p
	if(cond)
	  reg1 = **ld.c** *q

Figure 3. Data speculation of a load above a branch

Data speculation targets the memory dependencies that cannot be determined as either always conflict or never conflict at compile-time. If a load always conflicts with a store, there is no need for data speculation since the load should not be moved above the store. If a load never conflicts with a store, the compiler can move the load passing the store without using data speculation. In the paper, a memory dependency from a load to a store that cannot be determined definitely at compile-time will be called a "maybe" dependency.

The state of the art compilers can resolve many of the maybe dependencies by static alias analysis [8]. This tends to diminish the usefulness of data speculation. In situations where static alias analysis is not feasible, however, many of the memory dependencies that never conflict will be declared as maybe dependency, and thus data speculation can be applied more widely.

Data speculation has been implemented on Itanium systems to allow for a compiler to move a load across a store even when the two operations are potentially aliased. This not only breaks data dependency to reduce critical path length, but also allows a load to be scheduled far apart from its uses to hide cache miss latencies. However, the effectiveness of data speculation is affected by the sophistication of alias analysis technique as well as the aggressiveness of the instruction scheduler. In general, the more sophisticated is the alias analysis technique, the less performance gain is from data speculation, and the more aggressive is the instruction scheduler, the more opportunity is for data speculation. In this paper we evaluate in eight scenarios the performance potentials for SPEC2000C benchmarks running on the Itanium systems. Our results indicate that, without memory disambiguation and with an ideal scheduler, the potential gain from data speculation is about 8.8% (5.4% from critical path reduction and 3.4% from hiding cache miss latency). With the state of the art memory disambiguation and an ideal scheduler, the potential gain from data speculation is around 4.4% (3.2% from critical path reduction and 1.2% from hiding miss latency). With a real scheduler and assuming all memory pairs are aliased, the potential gain from data speculation is around 3.2% (1.8% from critical path reduction and 1.4% from hiding miss latency). With a sophisticated alias analysis and a realistic scheduler, the potential gain from data speculation is around 1.9% (1.4% from critical path reduction and 0.5% from hiding miss latency).

We also show the following interesting statistics about effects of scheduling constraints, the percentage of critical dependencies, the impacts of cache miss latencies, and the data speculation distances. For example, our data indicate that,

- Only about 2% to 2.5% of dependency edges are maybe dependency critical to performance. Those are the dependency edges that data speculation can be useful.
- Majority of the loads can be moved by data speculation no more than 20 cycles early. This limits the effectiveness of data speculation to hide long latency events (e.g., L3 cache misses).

The rest of the paper is organized as follows. Section 2 discusses the related works. Section 3 outlines the methodology for our experiment. Section 4 presents the experimental results. Section 5 concludes the paper and provides future directions.

2. Related works

A software-only approach to dynamic memory disambiguation is the run-time disambiguation proposed by Nicolau [15]. By inserting explicit address comparison and conditional branch instructions, this approach allows general code movement across ambiguous stores. Huang et al [9] extended the idea to use predicated instructions to help parallelize the comparison code based on a hardware approach. The Memory Conflict Buffer (MCB) scheme, first presented by Chen [4], extends the idea of run-time disambiguation by introducing architectural support to eliminate the need for explicit address comparison instructions. The MCB scheme adds two new instructions: 1) *preload*, which performs a normal load, but signals the hardware if a possible dependency violation exists for this load, and 2) *check*, which directs the hardware to determine if a violation has occurred and if so branch to conflict correction code. The effectiveness of the MCB approach is evaluated by the IMPACT group [7]. The performance gains were reported for subsets of Spec92C and Spec92F benchmarks and a few Unix programs. For the subset of Spec92C benchmarks, the potential gain from data speculation by ignoring all maybe aliases over the baseline without static alias analysis is on the average

37%. The potential gain from data speculation by ignoring all maybe aliases over the baseline with a static alias analysis is on the average 11%. The actual gain obtained from a simulator on an 8-issue and a 4-issue processor is about 4% and 3%, respectively.

Based on the MCB approach, several research and commercial architectures have adopted the idea of hardware and software collaborative run-time disambiguation: HP PD Architecture [12], IBM DAISY Architecture [6], and Itanium Architecture [10], [16]. Data speculation was studied in conjunction with control speculation and predication on the IMPACT EPIC Architecture [1]. Ju et al [11] experimented with data speculation for the Spec95C benchmarks. They simulated the six-issue Itanium system and reported a gain of about 2.5%.

Since the early studies, several things have changed, including the compilation technology, benchmarks, hardware implementation, as well as new usages of data speculations. Our work in this paper establishes the performance potentials of data speculation with the latest compilation technology, on the SPEC2000C benchmarks, and for various scenarios.

3. Methodology

Data speculation is primarily used to improve performance with the following benefits:

- Reduce schedule cycle counts by moving load away from critical path.
- Hide data cache miss latencies by advancing loads earlier from their uses.

To evaluate the two benefits, we first compute the total dependency heights (TDH) of a program. The TDH of a program is the frequency-weighted sum of dependency heights of all program execution paths. It represents the schedule cycle counts of the program with an ideal scheduler (e.g., a scheduler with aggressive code replication and control speculation) and unlimited machine resource. A TDH is computed in the following steps.

- Collect path profile. We experiment with both the global path [4], in which a path may span multiple loop levels, and intra-loop path, in which a path contains only blocks from the same loop level.
- Build dependency graph for each path. Each edge is annotated with the Itanium instruction latency.
- Compute dependency height. The dependency height for a path is the height of the dependency graph for the path.
- Sum the dependency height of all paths in all functions as follows, where the set of functions in a program is denoted by *{functions}*; the set of paths for a function f is denoted by *paths(f)*, the frequency for a path p is denoted by *freq(p)*, and the dependency height for the path is calculated as *height(p)*.

$$\text{TDH} = \sum_{\forall f \in \{functions\}} \sum_{\forall p \in path(f)} freq(p) * height(p)$$

Data speculation may remove some of the "maybe" dependencies to shorten the schedule length on the critical paths and thus can reduce TDH. We evaluate the benefit of data speculation on the schedule cycle count reduction by comparing the total dependency heights (TDH) of a program with removing maybe dependency edges and without removing maybe dependency edges. We will refer to the gain on schedule cycle count reduction due to data speculation the "*schedule gain*".

Data speculation may remove a "maybe" dependency so a load may be scheduled early to reduce cache miss penalty. To evaluate the benefit of data speculation on hiding data cache miss latency, we collect data cache miss rates for all loads and use the weighted latencies for the loads during TDH computations. For a load with L1, L2, and L3 miss rates of r1, r2, and r3, respectively, the weighted latency is calculated as

```
weighted_latency = L1_HIT_LATENCY*(1-r1)
      + L1_MISS_LATENCY *r1
      + L2_MISS_LATENCY *r2
      + L3_MISS_LATENCY *r3.
```

We will refer to the performance gain from reducing schedule cycle count and hiding cache miss latency the "*schedule&prefetch* gain".

We also measure the schedule gain and schedule&prefetch gain in a real scheduler. The scheduler is implemented in the Itanium production compiler and can schedule code in large DAG regions [3] with detailed modeling of the Itanium microarchitecture [5]. Since Itanium is a statically scheduled machine, we compare the schedule cycle counts with and without data speculation to measure the "schedule gain". Similarly, we use the weighted latency of the loads and measure the schedule&prefetch gain.

The schedule cycle counts measured by TDH or the real scheduler correspond to the program time spent in CPU execution, and the schedule&prefetch cycles correspond to the program time spent in CPU execution and data cache service. In order to obtain the overall performance gain from data speculation, we need to know many machine cycles are spent in the CPU or the memory subsystem. Figure 4 shows the cycle distribution for SPEC2000C running on an 800 MHz Itanium processor, with a 16K 4-way set associative L1 data cache, a 96K 6-way set associative unified L2 cache, 4M 4-way set associative unified L3 cache, and 2 GB of main memory.

An L1 hit takes 2 cycles and an L1 miss takes 8 cycles. An L2 miss takes 25 cycles, and an L3 miss takes 110 cycles. On the average, the CPU execution cycles account for about 28% of total cycles, and the data cache miss stalls account for about another 29% of total cycles on average. The rest of the cycles are spent in such micro-architecture stalls as, branch mis-prediction, DTLB miss, instruction cache miss, register stack engine activity, etc.

To derive the overall performance gain of data speculation on reducing schedule cycle count (overall schedule gain), we weigh the schedule gain computed from TDH or the real scheduler by the percentage of CPU execution cycles. For example, assume the schedule gain due to data speculation for a benchmark is 10% and the CPU execution cycles account for about 30% of total cycles of the benchmark. The "overall schedule gain" would be 10% * 30% = 3%. Similarly, to derive the overall performance gain of data speculation on reducing schedule cycle count and hiding cache misses (overall schedule&prefetch gain), we weigh the schedule&prefetch gain by the combined percentage of CPU execution cycles and data cache miss cycles. For example, assume the schedule&prefetch gain due to data speculation for a benchmark is 8% and the CPU execution cycles and data cache miss stalls together account for about 60% of total cycles of the benchmark. The "overall schedule&prefetch gain" would be 8% * 60% = 4.8%.

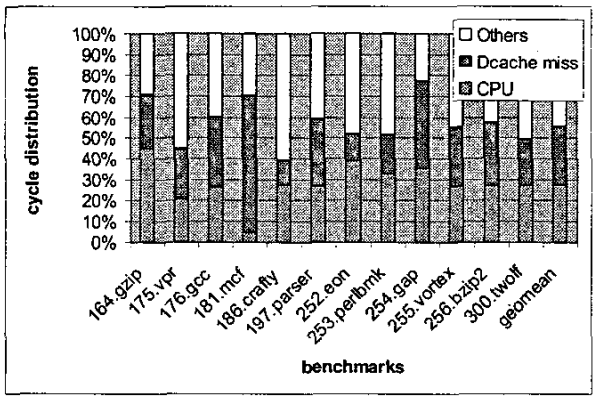

Figure 4. Total cycle distribution on Itanium

Each of the above performance gains can be calculated with the following two modes of memory disambiguations:

- Without static alias analysis and assuming every pair of memory operations is aliased (no-disam). In this setup, the dependency graph constructed to compute TDH or in the real scheduler includes a dependency edge between every pair of memory operations (one of them is a store).
- With static alias analysis (static-disam): In this setup, the dependency graph includes a dependency edge between a pair of memory operations (one of them is a store) only if the static alias analysis says that they are aliased.

Therefore we have the following three parameters giving a total 8 combinations of performance gains due to data speculation.

- Data speculation benefits: {reducing schedule cycle counts, prefetching for cache misses}
- Scheduling methods: {TDH, real scheduler}
- Disambiguation modes: {no-disam, static-disam}

We need to keep in mind, however, that the gains computed with our methodology are based on the following simplifying assumptions.

- Only the effect of data speculation in a DAG instruction scheduler is evaluated. The potential benefit of data speculation on global optimizations and iterative scheduling (e.g., software pipelining) are not included.
- All maybe dependencies are assumed to have low conflict probabilities and the recovery overhead is ignored.
- Assume fixed cache miss latency for each static load. In a real execution, the load miss latency may vary depending on the machine status. For example, an L1 miss on Itanium may take 6 to 8 cycles and we simply use a fixed number of cycles (e.g., 8) in our experiment.

4. Experimental Results

Our experiment is conducted using a research compiler for Itanium processor family (IPF) running with the SPEC2000C benchmarks (252.Eon benchmark is not included due to an experimental issue). The research compiler is based on a production compiler [13] with added components for flexibility in researches. The binaries used in the experiment are compiled with best compilation option for performance (*QO3 option)*. The generated codes are highly optimized with all of the inter-procedural, intra-procedural, architectural specific, and profile-guided optimizations, assisted with aggressive inter-procedural alias analysis [8], which is disabled in the no-disam mode, and whole program knowledge. In particular, many scope enhancement transformations, such as function inlining and loop unrolling, are applied before the TDH computations and the instruction scheduling.

In this section, we report results on the performance potential in terms of schedule gain, schedule&prefetch gain, overall schedule gain, and the overall schedule&prefetch gain, with two disambiguation configurations: no-disam and static-disam, and two scheduling configurations: TDH and a real scheduler. We first show the eight combinations of performance gains

without weighting them by the percentages of total cycles. We then show the overall performance gains weighted by the percentages of total cycles. We conclude this section with the statistics on the effects of scheduling constraints, the percentage of critical dependencies, the effects of scheduling constraints, and the data speculation distances.

4.1. Schedule Gains

Figure 5 shows the schedule gains for the SPEC2000C benchmarks. With the ideal scheduler and unlimited machine resources (TDH), data speculation improves the performance much more significantly than with a realistic scheduler on a real machine (19% v.s. 7% with no-disam and 11% v.s. 4.2% with static disam). An exception to this trend is the 300.twolf benchmark, for which data speculation provides more performance gain with a realistic scheduler than with the ideal scheduler (10% v.s. 9%). This seems due to the heuristics nature of the realistic scheduler. The realistic scheduler uses heuristics to approximate the NP hard scheduling problem. The base performance (without data speculation) with the realistic scheduler happens to be low and thus yields high performance gain for data speculation.

Furthermore, the sophisticated alias analysis reduces the benefit of data speculation significantly (19% to 11% with TDH and 7% to 4.2% with the realistic scheduler). There is also an exception to this trend. With the realistic scheduler, 253.perlbmk shows a 2% performance gain with static-disam and shows no performance gain with no-disam. We believe this is again due to the heuristics nature of the realistic scheduler.

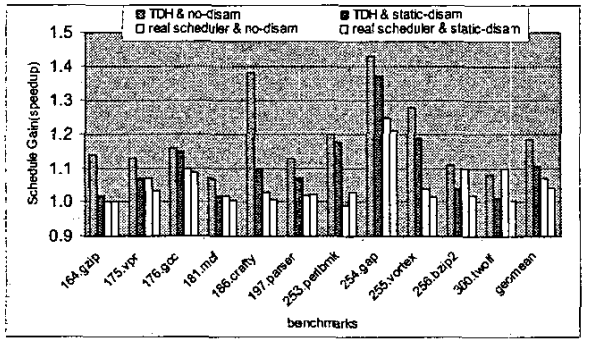

Figure 5. Schedule Gains

4.2. Schedule&prefetch Gains

The schedule&prefetch gain is shown in Figure 6. On the average, the schedule&prefetch gain with TDH is about 15% with no-disam and 7% with static-disam. With a real scheduler, the schedule&prefetch gain is 5.3% with no-disam and 3.3% with static-disam.

Notice that schedule&prefetch gain may be lower than schedule gain. For example, the schedule&prefetch gain with no-disam is 15%, compared to the 19% schedule gain with no-disam. This is because schedule&prefetch gain captures the average effect of data speculation on both reducing schedule cycle counts and hiding cache miss latency, while schedule gain captures only the effect of data speculation on reducing schedule cycle counts. Since the effect of data speculation on hiding miss latency is smaller than the effect on reducing schedule cycle counts, the average of the two is smaller than the higher one. In the next two subsections, we will show that the *overall* schedule&prefetch gain is higher than the *overall* schedule gain.

Table 1 summarizes the schedule gains and schedule&prefetch gains.

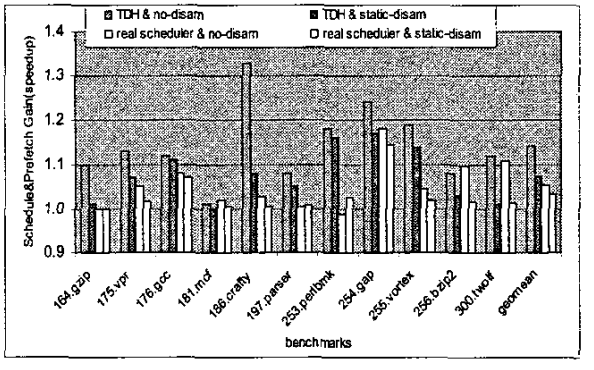

Figure 6. Schedule&prefetch Gains

Types of gains \ Disambiguation setups	no disam	static disam
schedule gain with TDH	19.0%	11.0%
schedule&prefetch gain with TDH	15.0%	7.0%
schedule gain with a real scheduler	7.1%	4.2%
schedule&prefetch gain with a real scheduler	5.3%	3.3%

Table 1. Summary of Schedule and Schedule&Prefetch Gains.

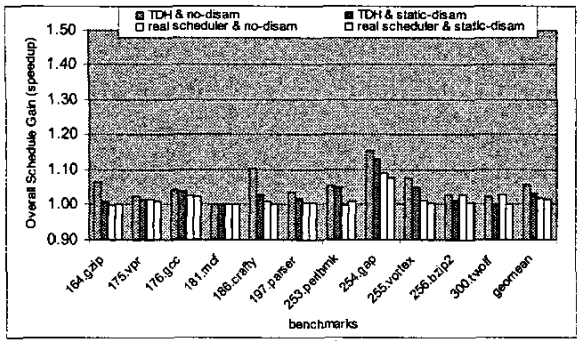

Figure 7. Overall Schedule Gains.

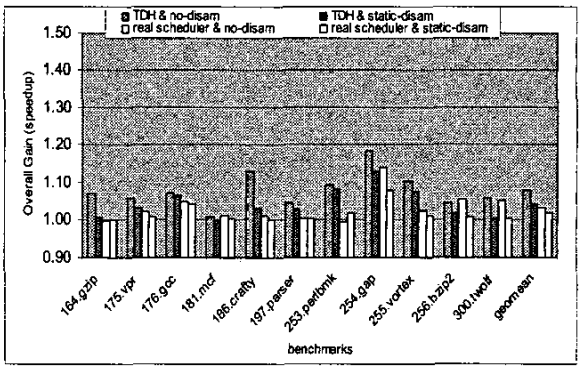

Figure 8. Overall Schedule&prefetch Gains

4.3. Overall Schedule Gain

The overall schedule gain is shown in Figure 7. The overall schedule gain with TDH is about 5.4% with no-disam and 3.2% with static-disam. With a real scheduler, the overall schedule gain is 1.8% with no-disam and 1.4% with static-disam.

4.4. Overall Schedule&prefetch Gain

Figure 8 shows that the overall schedule&prefetch gain with TDH is about 8.8% with no-disam and 4.4% with static-disam. With a real scheduler, the overall schedule&prefetch gain is 3.2% with no-disam and 1.9% with static-disam.

Table 2 summarizes the overall gains and overall schedule&prefetch gains.

Types of gains \ Disambiguation setups	no disam	static disam
overall schedule&prefetch gain with TDH	8.8%	4.4%
overall schedule gain with TDH	5.4%	3.2%
overall schedule&prefetch gain with a real scheduler	3.2%	1.9%
overall schedule gain with a real scheduler	1.8%	1.4%

Table 2. Summary of Overall Schedule and Schedule&Prefetch Gains.

4.5. Effects of Scheduling Constraints

In the above, the TDH computation includes the following two scheduling constraints.

- Non-speculative instruction dependency: Instructions such as store and check (e.g. chk.a) cannot be speculated above branches on Itanium. A control-dependency is added from each non-speculative instruction to the preceding branch that is not post-dominated by the non-speculative instruction.
- Intra-loop path: each path collected from path profiling must contain blocks from the same loop level.

In this experiment we selectively relax the two constraints and compare the effects on the TDH schedule gains with static-disam. Figure 9 shows the schedule gains with and without the constraints. The gain drops from 14% to about 11% when both constraints are added. This suggests a general trend that the schedule gain usually decreases when more constraints are added.

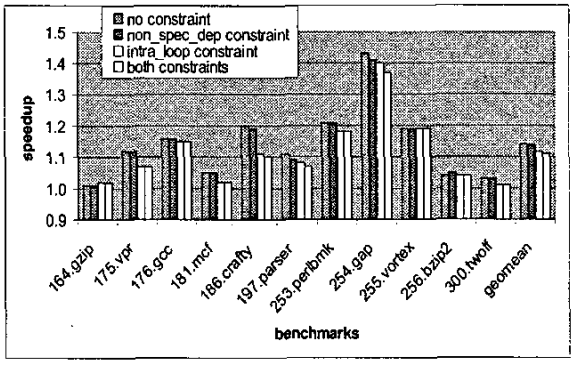

Figure 9. Effects of Scheduling Constraints on Schedule Gains.

4.6. Critical Maybe Aliases

This experiment determines the percentage of maybe dependencies that are critical.

A "maybe" flow dependency is critical if it satisfies the following two conditions:

- Slack condition: it is on the critical path (without enough scheduling slack to hide its latency) of the dependency graph.
- Improvement condition: when the maybe dependency is removed, the dependency height of the graph will reduce. To check for the Improvement condition, we re-compute the height of the dependency graph after removing the edges that satisfy the slack condition. If the new height is smaller than the previous height, the removed edges satisfy the Improvement condition, therefore they are critical maybe edges.

Clearly, the order of identifying the critical edges may result in different set of critical edges. Our heuristics first collects the list of maybe dependency edges in program order, and then repeatedly traverses the list to remove identified critical edges. The number of critical edges identified this way should be an upper bound on the minimal number of critical edges.

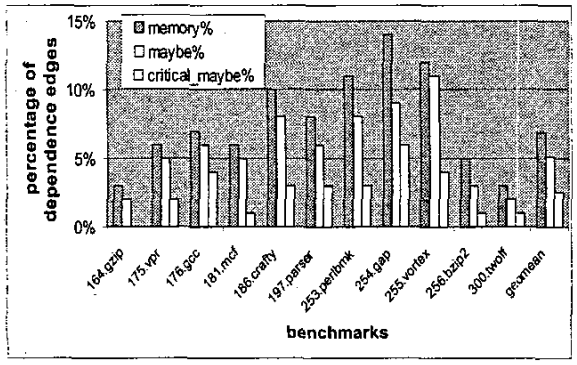

Figure 10. Percentage of Critical Maybe Dependencies

The result of critical edge analysis is shown in Figure 10. In this experiment, TDH is used and the dependency graph has neither scheduling constraints nor cache miss latencies added. On the average, about 6.9% of all dependency edges are memory dependency (memory%); about 5.2% of all dependency edges are "maybe" dependency (maybe%); about 2.5% of all dependency edges are critical "maybe" dependency (critical maybe%). Overall, only a small percentage of dependency edges are critical maybe edges that can be improved by data speculation. This result may suggest why the potential gain of data speculation with SPEC2000C benchmarks is not as high as we would expect.

When scheduling constraints and data cache miss latencies are included in the dependency graph, the percentage of maybe edges that is critical drops slightly (see Figure 11). On the average only about 2% of dependency edges are critical maybe edges.

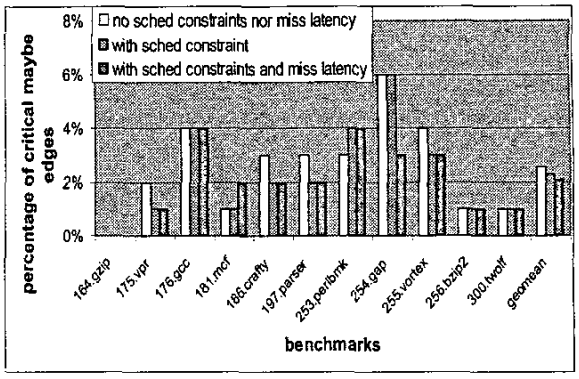

Figure 11. Percentage of critical maybe dependencies

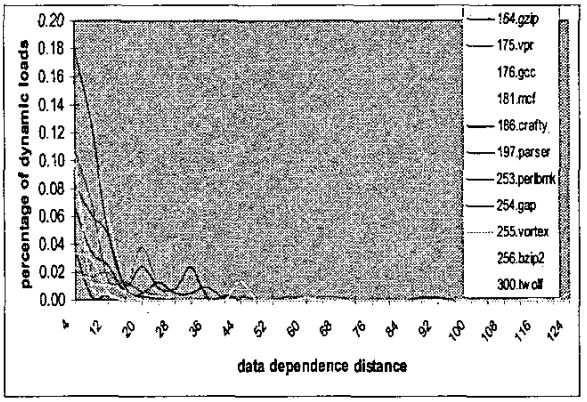

Figure 12. Data speculation distances without constraints and miss latency

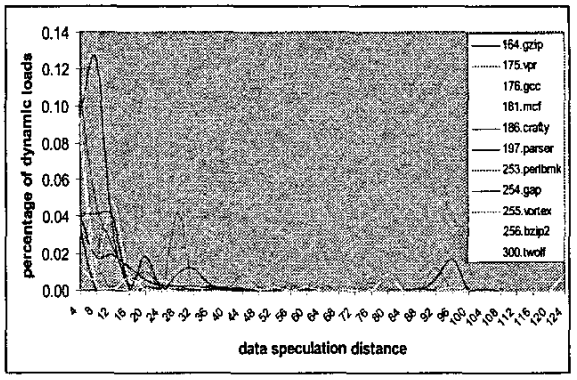

Figure 13. Data speculation distances with constraints and miss latency

4.7. Data Speculation Distance

The data speculation distance for a load is the maximum distance from the original load location and the

final location when a "maybe" dependency edge is ignored by data speculation. The distance is measured as the difference in earliest start cycles of the load before and after the "maybe" dependencies are ignored. Figure 12 shows the data speculation distances for the TDH computation without scheduling constrains or cache miss latencies being added. In comparison, Figure 13 shows the data speculation distances for the TDH computation with scheduling constrains and cache miss latencies being added. In both cases, over 80% of loads have distance of zero cycles, and the majority of the distances are less than 20 cycles. One noticeable exception is the 254.gap benchmark with scheduling constraints and cache miss latency: about 2.3% of dynamic loads with distance greater than 92 cycles. In general, data speculation could be used to hide L1 (<=8 cycles) and L2 (<= 25 cycles) miss latencies, but not enough to hide L3 miss latency (>= 100 cycles). This may explain why the effect of data speculation on hiding cache miss latency is less significant than on reducing critical path length.

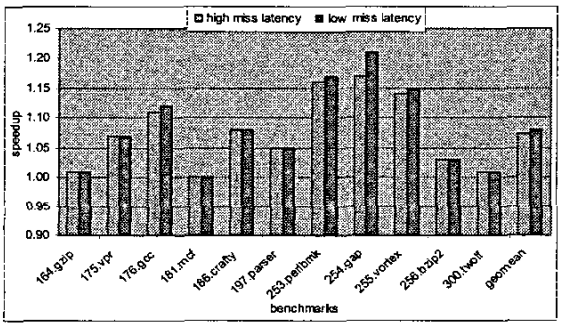

Figure 14. Effects of Miss Latencies

4.8. Effects of Miss Latencies

Intuitively, longer cache miss latencies result in longer dependency chains in the dependency graph and thus provide more opportunities for data speculation to reduce critical path length. In this experiment we examine the effect of cache miss latencies on the schedule&prefetch gains with static-disam. We consider the following two latency configurations:

- Low miss latencies: An L1 miss takes 6 cycles. An L2 miss takes 20 cycles, and an L3 miss takes 100 cycles.
- High miss latencies: An L1 miss takes 8 cycles. An L2 miss takes 25 cycles, and an L3 miss takes 110 cycles. This is the configuration that we use in the rest of experiments.

Figure 14 shows the schedule&prefetch gain with the high and low average miss latencies. With low miss latency, the schedule&prefetch gain actually increases slightly. This result seemingly suggests that data speculations are applied more to the loads with low miss latencies than to the loads with high miss latencies.

5. Summary and Future Work

In this paper we evaluate in eight scenarios the performance potentials for SPEC2000C benchmarks. Our results indicate that, without memory disambiguation and with an ideal scheduler, the potential gain from data speculation is about 8.8% (5.4% from critical path reduction and 3.4% from hiding cache miss latency). For certain compilation systems, such as dynamic binary translators, which cannot afford sophisticated memory disambiguation, data speculation may be very useful to enhance performance. With the state of the art memory disambiguation and an ideal scheduler, the potential gain from data speculation is around 4.4% (3.2% from critical path reduction and 1.2% from hiding miss latency). With a real scheduler and assuming all memory pairs are aliased, the potential gain from data speculation is around 3.2% (1.8% from critical path reduction and 1.4% from hiding miss latency). With a sophisticated alias analysis and a realistic scheduler, the potential gain from data speculation is around 1.9% (1.4% from critical path reduction and 0.5% from hiding miss latency). Although the average potential from data speculation is not high, certain benchmarks, such as 254.gap and 176.gcc could potentially get a significant performance boost from data speculation.

We also show the following interesting statistics about effects of scheduling constraints, the percentage of critical dependencies, the effects of scheduling constraints, and the data speculation distances. For example, we show the following results.

- Only about 2% to 2.5% of dependency edges are maybe dependency and critical to performance. Those are the dependency edges that data speculation can be useful.
- Majority of the loads can be moved by data speculation no more than 20 cycles early. This limits the effectiveness of data speculation to hide long latency event (e.g., L3 cache misses).

In the future, we would like to investigate the difference between the potential gain and the measured gain on real machines. We would like to attribute the performance discrepancy to specific sources, such as, ALAT implementation, recovery overhead, register pressure, instruction scheduling heuristics, etc. Figure 15 shows the performance gains of data speculation (load only, without speculating uses) over without data speculation on an Itanium machine [10]. The *O3* binaries are compiled with aggressive inter-procedural pointer analysis and whole program knowledge [8]. The O2 binaries, on the other hand, are not compiled with

inter-procedural pointer analysis. For QO3, data speculation on the average shows no noticeable performance gain, even though our limit study shows that there is a 1.9% performance potential (see the cell "overall schedule&prefetch gain with a real schedule and static-disam" in Table 2). For O2, the average performance gain from data speculation is about 2.1%. This result demonstrates the usefulness of data speculation in a compilation environment with a less sophisticated alias analysis.

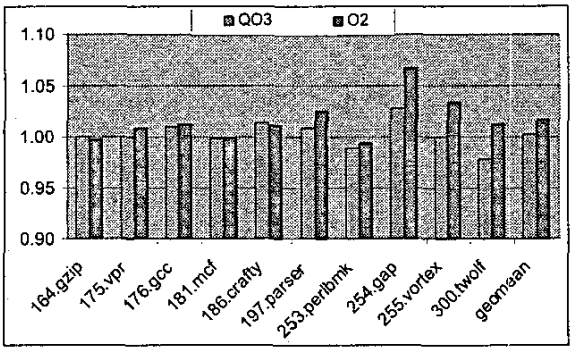

Figure 15. Performance gains on an Itanium machine

We would also like to identify the compiler limitations that are responsible for the performance gap between the ideal scheduler and a real scheduler. This understanding may suggest improvement to the instruction scheduler to bring more performance out of data speculation on a real machine.

Finally, we would like to investigate the conflict probabilities of the critical maybe dependency edges. So far in this study, we have treated all the dependencies are unlikely to occur. However, some of the maybe dependencies may conflict with stores frequently at runtime, and thus data speculation along those dependency edges may trigger the recovery code, which would severely degrade performance. We should profile the critical maybe edges to study their conflict rates, and only remove critical maybe edges with low conflict probability. Since the amount of critical maybe dependencies is small, this profile can be obtained inexpensively.

Acknowledgement

We would like to thank Sun Chan, Dan Lavery, and Chu-cheow Lim for their support during this study and their comments on an early draft of the paper. We appreciate the comments from the anonymous reviewers that helped improve the quality of the paper.

References

[1] August, David, Daniel A. Connors, Scott A. Mahlke, John W. Sias, Kevin M. Crozier, Ben-Chung Cheng, Patrick R. Eaton, Qudus B. Olaniran, and Wen-mei W. Hwu, "Integrated Predicated and Speculative Execution in the IMPACT EPIC Architecture." Proceedings of the 25th International Symposium on Computer Architecture, July, 1998

[2] Ball, T and J. Larus, "Efficient Path Profiling," MICRO-29, December 2-4, 1996.

[3] Bharadwaj, J.; Menezes, K.; McKinsey, C., "Wavefront scheduling: path based data representation and scheduling of subgraphs," Proceedings 32nd Annual International Symposium on Microarchitecture, 1999, Page(s): 262 –271

[4] Chen, W. Y., *Data Preload for Superscalar and VLIW Processors*, Ph.D. Thesis, University of Illinois, Urbana, IL, 1993.

[5] Crawford, J., "Introducing the Itanium Processors." IEEE Micro, Volume: 20 Issue: 5, Sept.-Oct. 2000, Page(s): 9 –11

[6] Ebcioglu, K. and E. Altman, "DAISY: Dynamic Compilation for 100% Architectural Compatibility", IBM Research Report RC 20538, Aug 1996.

[7] Gallagher, David M., Chen, William Y., Hwu, Wen-mei W., "Dynamic Memory Disambiguation Using the Memory Conflict Buffer," ACM SIGPLAN notices, NOV 01 1994 v 29 n 11, Page: 183

[8] Ghiya, Rakesh, Daniel Lavery and David Sehr, "On the importance of points-to analysis and other memory disambiguation methods for C programs," Proceedings of the ACM SIGPLAN'01 conference on Programming language design and implementation, 2001, pp. 47 – 58.

[9] Huang, A. S., G. Slavenburg, and J. P. Shen, "Speculative Disambiguation: A Compilation Technique for Dynamic Memory Disambiguation, "In *Proc.of the 21st Annual Int'l Symp. on Computer Architecture*, pp. 200-210, April 1994.

[10] Intel Corp, "Itanium™ Application Developers Architecture Guide," May 1999, and Intel® Itanium™ Processor Hardware Developer's Manual, 2000.

[11] Ju, R. D-C, K. Nomura, U. Mahadevan, and L.-C. Wu, "A Unified Compiler Framework for Control and Data Speculation," Proc. of 2000 Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT), pp. 157 - 168, Oct. 2000.

[12] Kathail, V., M. Schlansker, B. Rau, *HPL PlayDoh Architecture Specification: Version 1.0*, Hewlett-Packard Laboratories Technical Report, HPL-93-80, Feb. 1994.

[13] Krishnaiyer, R., D. Kulkarni, D. Lavery, W. Li, C. Lim, J. Ng, and D. Sehr, "An Advanced Optimizer for the IA64 Architecture, IEEE Micro, Vol 20, No 6, Nov 2000, 60-68

[14] Mahlke, S. A., W. Y. Chen, R. A. Bringmann, R. E. Hank, W. W. Hwu, B. R. Rau, and M. S. Schlansker, "Sentinel Scheduling: A Model for Compiler-Controlled Speculative Execution," Transactions on Computer Systems, Vol. 11, No. 4, Nov. 1993

[15] Nicolau, A., "Run-time Disambiguation: Coping with Statically Unpredictable Dependencies," *IEEE Trans. on Computers*, Vol. 38, No. 5, pp. 663-678, May 1989.

[16] Schlansker, M.S, Rau, B.R. "EPIC: Explicitly Parallel Instruction Computing," Computer, Volume: 33 Issue: 2, Feb. 2000, pp 37-45

[17] Smotherman, M, S. Krishnamurthy, P.S. Aravind, and D. Hunnicut, "Efficient DAG Construction and Heuristics Calculation for Instruction Scheduling, Micro-25, 1991, pp93-102.