A Study of the Effects of Compiler-Controlled Speculation on Instruction and Data Caches

Roger A. Bringmann * Scott A. Mahlke † Wen-mei W. Hwu

Center for Reliable and High-Performance Computing
University of Illinois
Urbana, IL 61801

Abstract

Compiler-controlled speculation has been shown to be effective in increasing instruction level parallelism (ILP) found in non-numeric programs. However, it is not clear the extent to which speculatively scheduled code may affect the instruction and data caches. In particular, the amount of time spent resolving cache misses may be significant enough to prevent the more aggressive speculation models from attaining their best potential performance results. The objective of this paper is to quantify these effects using aggressive speculation models.

Index terms - instruction cache effects, data cache effects, compiler-controlled speculation, safe speculation, VLIW, superscalar

1 Introduction

Instruction scheduling is the process used by the compiler to re-order instructions in an effort to minimize program execution time. Since instruction scheduling is NP-Hard, heuristics are used to approximate the best schedule. One common approach to scheduling is to perform list scheduling using greedy heuristics to approximate a *globally optimal* schedule [1]. Regardless of the scheduling heuristics, instructions are ordered based upon some priority mechanism. At each cycle, the instructions with the highest priority that have resolved all dependences and meet the issue requirements of the processor are scheduled.

The implementation of a scheduler is straightforward if list scheduling is applied only within basic blocks. Unfortunately, there is insufficient instruction level parallelism (ILP) available within basic blocks of non-numeric benchmarks to fully utilize the functional units of wide issue superscalar and VLIW architectures [2, 3, 4]. Therefore global scheduling techniques such as *trace scheduling* [5] and *superblock scheduling* [6] have been proposed to permit greater scheduling and optimization freedom beyond basic block boundaries. Using these techniques, the program is divided into a set of traces or superblocks that represent frequently executed paths. These traces or superblocks contain multiple basic blocks and as a result can contain multiple conditional branches. When building a dependence graph for a trace or superblock, control dependence arcs are added from conditional branches to subsequent instructions. In order to gain additional scheduling freedom beyond the natural basic block boundaries found within these traces or superblocks, the compiler must remove some of these control dependence arcs. This permits speculation of instructions past conditional branches, thus the term compiler-controlled speculation.

When an instruction is speculated above a branch, it is executed regardless of the direction taken by the branch. As such, the speculated instruction could introduce instruction cache (Icache) and data cache (Dcache) effects that may not have been present in the original unscheduled program. If these effects are significant, much of the performance potential of aggressive speculation may be lost. This makes it critical that the processor and computer system designers understand the requirements of the speculation models being used by the compiler if they are to balance the potential performance of the compiled code with the cache implementation.

The next section briefly describes the static speculation models used in the experiments. Section 3 discusses the expected cache effects. Section 4 presents experimental results showing how these speculation models affect various configurations of instruction and data caches. Finally, concluding remarks are given in Section 5.

2 Scheduling Models

In order to gain greater scheduling freedom, instructions must be allowed to speculate above conditional branches found within a trace or superblock. In some cases, speculation of these instructions can introduce a *scheduling error* that can cause unexpected program termination. An example of such a scheduling error would be scheduling a divide before a branch that was implicitly preventing a divide-by-zero. The decision on what to do if the highest priority instruction could potentially introduce a scheduling error is

*Roger Bringmann is now with QMS, Inc.
†Scott Mahlke is now with HP Labs

1. **Avoid Errors** - do not permit an instruction to speculate if it could cause a scheduling error [2].
2. **Ignore Errors** - assumes that the likelihood of a scheduling error is small and will therefore speculate a non-excepting form of the instruction. As a result, any scheduling errors are hidden. This model requires non-excepting forms of each potentially excepting instruction that is speculated [2].
3. **Resolve Errors** - speculates instructions that could cause a scheduling error but assumes that the processor has some mechanism to resolve the error. Three examples of speculation models that fall into this category are boosting [7], sentinel scheduling [8] and write-back suppression [9].

Figure 1: Classifications of compiler-controlled speculation models.

based upon the speculation model and the processor support. As shown in Figure 1, the existing speculation models can be categorized into three classes based upon these decisions.

Each of the speculation models used for experimentation contain certain characteristics that permit different degrees of scheduling freedom. As such, they are expected to introduce varying Icache and Dcache effects. In order to evaluate the speculation models fairly, all benchmarks are aggressively optimized with superblock techniques [6]. Each of the speculation models are used to schedule the optimized code. This paper does not cover any of the *resolve error* speculation models since their recovery processes are not directly comparable to the other models.

2.1 No Speculation Model

This model provides a baseline for the typical Icache and Dcache effects that occur without code speculation. As a result, this model provides the best scheduling results that are attainable with no additional Icache and Dcache effects introduced from compile-time scheduling.

2.2 Restricted Speculation Model

Restricted speculation (formally called restricted code percolation [2]) assumes that correct program execution is always required as defined by the *avoid errors* class. Using this model, the compiler can only speculate instructions that will never cause an exception. The conservative definition of potentially excepting instructions used by this model assumes that if there is any way that an instruction could cause an exception, it will be classified as potentially excepting and may not be speculated. As such, this definition ignores the cases where the context in which the instruction is used can sometimes indicate if the instruction may or may not cause an exception. This conservative model prevents speculation of any memory instructions, integer divide and remainder, and all floating point instructions. This model functions as a low-end for the speculation models. The advantage of this model is that it does not introduce any Dcache effects as a result of speculating memory instructions. The only Dcache effects introduced above those introduced by the no speculation model are a direct result of the increased register pressure created by the more aggressively speculated code.

2.3 Safe Speculation Model

Safe speculation expands the scheduling freedom of restricted speculation by using program analysis to identify potentially excepting instructions that can never cause scheduling errors or will introduce no new scheduling errors. An example of such a potentially excepting instruction would be a load from an array. If the potential values that the array may access can be proven to be within the declared array bounds, then the load is a safe load. This model also falls under the *avoid errors* classification. The advantage of this speculation model is that it requires no special hardware support in the processor as required for the *resolve errors* class and the *ignore errors* class and does not have the inherent risks associated with the *ignore errors* class. The safe speculation model results reported in this paper are based on inter-procedural and intra-procedural analysis algorithms reported in [10].

2.4 General Speculation Model

General speculation (formally called general code percolation) falls under the *ignore errors* classification. It requires a non-excepting form of every potentially excepting instruction that is desirable to speculate [2]. Thus, if a potentially excepting instruction is to be speculated, it will be replaced by its non-excepting form. Potentially excepting instructions are typically load instructions, integer divide and remainder, and all floating point instructions. The results on general speculation reported in this paper are based upon a full model where every potential exception causing instruction has a non-excepting counter part in the instruction set. The following architectures are examples that have implemented subsets of this model in an effort to increase instruction level parallelism.

1. Multiflow - non-excepting floating point instructions [11]
2. Cydra 5 - non-excepting floating point instructions, ability to disable exceptions for memory and arithmetic operations [12].
3. HP Precision Architecture - non-excepting floating point instructions, non-excepting dereferenced null pointer [13].

3 Expected Cache Effects

This section will provide a qualitative analysis of the expected instruction and data cache effects resulting from

Opcode	Description
add	32-bit integer add
beq	conditional branch on equal
bgt	conditional branch on greather than
bne	conditional branch on not equal
jump	unconditional branch
ld_c2	load signed 16-bit value
ld_i	load signed 32-bit value
ld_uc	load unsigned 8-bit value
ld_uc2	load unsigned 16-bit value
lsl	32-bit logical shift left
mov	move 32-bit value
st_c	store 8-bit value
sub	32-bit integer subtract
xor	32-bit exclusive or

Table 1: Instruction opcodes and descriptions.

Cycle	Instruction - (instruction id, opcode)				
1	45, ld_uc	48, mov	49, add	50, add	
2					
3	46, st_c	47, lsl	51, bgt	52, mov	
4	53, bgt	54, add	57, mov		
5	55, add	58, add			
6	56, ld_i				
7					
8	59, beq	60, mov			
9	61, add				
10	62, bne	66, ld_uc	63, lsl		
11	64, add				
12	68, lsl	67, st_c	65, mov	69, bgt	70, mov
13	71, bgt	72, add	75, mov		
14	73, add	76, add			
15	74, ld_i				
16					
17	77, beq	78, mov			
18	79, add				
19	80, bne	81, lsl	83, add	84, add	85, add
20	82, add	86, jump			

Figure 2: The most important loop in cccp scheduled using no speculation model.

compiler-controlled speculation. To accomplish this goal, scheduled code examples from two benchmarks are presented. These code examples were chosen because they show extreme cache effects due to speculation. The instruction opcodes and their descriptions for the examples are given in Table 1. The examples were scheduled with the no speculation and general speculation models using an eight-issue superscalar processor that has uniform functional units and instruction latencies of the HP-PA 7100 (see Table 5). The Icache and Dcache block sizes were 64 bytes each.

Icache Effects

Speculating instructions above branches moves them from less frequently executed paths to more frequently executed paths. As such, the instruction working set is increased which should result in more Icache requests and subsequently more Icache misses. The first example benchmark, cccp, is used to show the expected Icache effects. To accomplish this, the most frequently executed loop within cccp (found in the *rescan* function) was used. Based upon profile information, the IMPACT superscalar optimizer decided to unroll this loop three times. Tables 2 and 3 respectively show the scheduled code for the no speculation and general speculation models. As these tables show, none of the branches from Table 2 have been delayed in Table 3. In addition, the schedule was reduced from 20 cycles for the no speculation model to 8 cycles for the general speculation model. It should also be noted that scheduling with the no speculation model provide insufficient freedom to schedule more than 5 instructions in any cycle for the 8-issue processor.

While none of the branches in the general speculation schedule were issued later than in the no speculation schedule, the location of the branches within the Icache blocks did change as shown by Tables 4 and 5. The most notable difference is that branch instruction 59 is located in block 2 of the no speculation Icache layout while it is located in block 3 of the general speculation Icache layout. As a result, there is one additional Icache block before the branch. If branch 59 is infrequently taken, this may not increase Icache misses since both no speculation and general speculation loops are contained within only 4 Icache blocks. However, as Table 2 shows this branch is taken 6192 times. This means that there is an additional Icache block in the working set of the taken path of branch 59 in the general speculation schedule than in the no speculation schedule. The increased working set of this taken branch increases the chance of mapping conflicts with other important Icache blocks. As such, the advantages of the more aggressive schedule have resulted in greater risk of Icache misses.

Dcache Effects

Speculating load instructions above branches moves them from less frequently executed paths to more frequently executed paths. This will not only have effects on the Icache, but will also increase the frequency that the load requests are made. As such, the data working set is increased which should result in more Dcache requests and subsequently more Dcache misses. The second benchmark, compress, is used to show the expected Dcache effects. To accomplish this, the most frequently executed loop within compress (found in the *compress* function) was used. Based upon profile information, the IMPACT superscalar optimizer decided to unroll this loop three times. Tables 6 and 7 respectively show the scheduled code for the no speculation and general speculation models. As the tables show, the no speculation model used 37 cycles while the general speculation model required only 18 cycles. It should also be noted that scheduling with the no speculation model provide insufficient freedom to schedule more than 6 instructions in any cycle for the 8-issue processor.

Table 3 shows the increased execution frequency of the six speculated loads from the general speculation schedule of this loop. By speculating a load above a particular branch, the memory reference patterns of the control flow paths reached from that branch have been altered. Depending upon the cache configuration, this could introduce more Dcache conflicts. For example, by speculating load instruction 163 above branch 159 in Table 7, the memory reference pattern of the paths reached by the taken path of this branch have been altered. Based upon the increased execution frequency of load number 163, and the resultant change in memory reference patterns, Dcache miss rates caused by this load could increase. Due to speculation of other loads and the change in their memory reference patterns, the to-

Cycle	Instruction - (instruction id, opcode[* = speculative])							
1	45, ld_uc	54, add*	57, mov*	60, mov*	52, mov*	72, add*	48, mov	49, add
2	55, add*	58, add*	61, add*	50, add	73, add*	63, lsl*	75, mov*	70, mov*
3	47, lsl	46, st_c	66, ld_uc*	51, bgt	53, bgt	64, add*	76, add*	78, mov*
4	56, ld_i	79, add*	81, lsl*					
5	68, lsl*							
6	59, beq	62, bne	67, st_c	74, ld_i*	65, mov	69, bgt	71, bgt	
7								
8	77, beq	80, bne	82, add	83, add	84, add	85, add	86, jump	

Figure 3: The most important loop in cccp scheduled using general speculation model.

cccp loop		compress loop	
Branch Instruction ID	Times Taken	Branch Instruction ID	Times Taken
51	0	159	46234
53	20	164	11
59	6192	166	0
62	1753	179	39594
69	0	184	8
71	5	186	0
77	2272	199	25981
80	130	204	1
86	9182	210	26824
		214	0
Total	19554	Total	138653

Table 2: Branch taken frequencies. (Total corresponds to the total entrance frequency of the loop.)

Icache Block	Instruction - (instruction id, opcode)							
1								
								45, ld_c
2	48, mov	49, add	50, add	46, st_c	47, lsl	51, bgt	52, mov	53, bgt
	54, add	57, mov	55, add	58, add	56, ld_i	59, beq	60, mov	61, add
3	62, bne	66, ld_c	63, lsl	64, add	68, lsl	67, st_c	65, mov	69, bgt
	70, mov	71, bgt	72, add	75, mov	73, add	76, add	74, ld_i	77, beq
4	78, mov	79, add	80, bne	81, lsl	83, add	84, add	85, add	82, add
	86, jump							

Figure 4: Icache layout for cccp loop after no speculation model (16 instruction block).

Icache Block	Instruction - (instruction id, opcode[* = speculative])							
1								45, ld_uc
	54, add*	57, mov*	60, mov*	52, mov*	72, add*	48, mov	49, add	55, add*
2	58, add*	61, add*	50, add	73, add*	63, lsl*	75, mov*	70, mov*	47, lsl
	46, st_c	66, ld_uc*	51, bgt	53, bgt	64, add*	76, add*	78, mov*	56, ld_i
3	79, add*	81, lsl*	68, lsl*	59, beq	62, bne	67, st_c	74, ld_i*	65, mov
	69, bgt	71, bgt	77, beq	80, bne	82, add	83, add	84, add	85, add
4	86, jump							

Figure 5: Icache layout for cccp loop after general speculation model (16 instruction block).

Cycle	Instruction - (instruction id, opcode)					
1	147, sub	151, sub	156, add			
2	148, mov	157, add				
3	149, lsl	152, mov				
4	153, lsl	150, add				
5	154, xor					
6	155, lsl					
7	158, ld_j					
8						
9	159, bne	161, add	160, lsl			
10	162, add					
11	163, ld_uc2	164, bgt	165, ld_c2			
12						
13	166, beq	167, sub	171, sub	176, add		
14	168, mov	177, add				
15	169, lsl	172, mov				
16	173, lsl	170, add				
17	174, xor					
18	175, lsl					
19	178, ld_j					
20						
21	179, bne	181, add	180, lsl			
22	182, add					
23	183, ld_c2	184, bgt	185, ld_c2			
24						
25	186, beq	187, sub	191, sub	196, add		
26	188, mov	197, add				
27	189, lsl	192, mov				
28	193, lsl	190, add				
29	194, xor					
30	195, lsl					
31	198, ld_j					
32						
33	199, bne	201, add	200, lsl			
34	202, add					
35	203, ld_c2	204, bgt	205, ld_uc	209, add	208, add	207, add
36						
37	206, mov	210, bne	211, mov	212, add	213, mov	214, jump

Figure 6: The most important loop in compress scheduled using no speculation model.

Load Instruction ID	Branches Speculated Above	Increase in Execution Frequency
163	159	46234
165	159, 164	46245
183	179	39594
185	159, 164, 166, 179, 184	85847
203	199	25981
205	159, 164, 166, 179, 184, 186, 199, 204	111829

Table 3: Increase in execution frequency of speculated loads in the compress loop after scheduling with the general speculation model.

tal increase in Dcache misses for instruction 163 could be greater than the increase in its execution frequency.

4 Experimental Evaluation

This section will quantify the effects that increasing levels of scheduling freedom can have on instruction and data caches. The speculation models used in the experiments from least aggressive to most aggressive are no speculation, restricted speculation, safe speculation and general speculation. Section 4.1 discusses the experimental approach used to generate the cache effects. Section 4.2 discusses the experimental results.

Benchmark	Benchmark Description
008.espresso	truth table minimization
022.li	lisp interpreter
023.eqntott	boolean equation minimization
026.compress	compress files
072.sc	spreadsheet
cccp	GNU C preprocessor
cmp	compare files
eqn	format math formulas for troff
grep	string search
lex	lexical analyzer generator
qsort	quick sort
tbl	format tables for troff
wc	word count
yacc	parser generator

Table 4: Benchmarks.

4.1 Methodology

Compiler support for each of the speculation models has been implemented in the IMPACT-I C compiler. The IMPACT-I compiler is a prototype optimizing compiler designed to generate efficient code for VLIW and superscalar processors [2]. The benchmarks used in this study are the 14 non-numeric programs shown in Table 4. The benchmarks consist of 5 non-numeric programs from the SPECint92 suite and 9 other commonly used non-numeric programs. Each of the benchmarks were aggressively optimized with superblock techniques [6] and scheduled using the four speculation models varying the processor issue width from 1 to 8 instructions per cycle.

The processor model used in this study is an in-order

Cycle	Instruction - (instruction id, opcode[* = speculative])							
1	147, sub	151, sub	161, add*	181, add*	165, ld_uc*	167, sub*	171, sub*	156, add
2	148, mov	162, add*	182, add*	157, add	196, add*	176, add*	185, ld_uc*	187, sub*
3	149, lsl	152, mov	197, add*	177, add*	201, add*	191, sub*	205, ld_uc*	
4	153, lsl	168, mov*	150, add	202, add*				
5	154, xor	169, lsl*	172, mov*					
6	160, lsl*	155, lsl	173, lsl*	188, mov*				
7	163, ld_uc2*	158, ld_i	189, lsl*	192, mov*				
8	193, lsl*							
9	174, xor*	159, bne	170, add*	164, bgt	166, beq	206, mov*		
10	175, lsl	180, lsl*						
11	183, ld_uc2*	178, ld_i						
12								
13	194, xor*	179, bne	190, add*	184, bgt	186, beq			
14	195, lsl	200, lsl*						
15	198, ld_i	203, ld_uc2*						
16								
17	199, bne	204, bgt	207, add	209, add	208, add	210, bne		
18	211, mov	212, add	213, mov	214, jump				

Figure 7: The most important loop in compress scheduled using general speculation model.

Function	Latency	Function	Latency
Int ALU	1	FP ALU	2
memory load	2	FP multiply	2
memory store	1	FP divide(single-precision)	8
branch	1 / 1 slot	FP divide(double-precision)	15

Table 5: Instruction latencies.

Cache Sizes:	4K - 256K, perfect
Cache Associativity:	direct-mapped, two-set associative
Cache Block Size:	64 bytes with 12 cycle miss latency
Dcache Type:	blocking cache
Dcache Write Policy:	write-through, no write-allocate

Table 6: Cache configurations used in experiments.

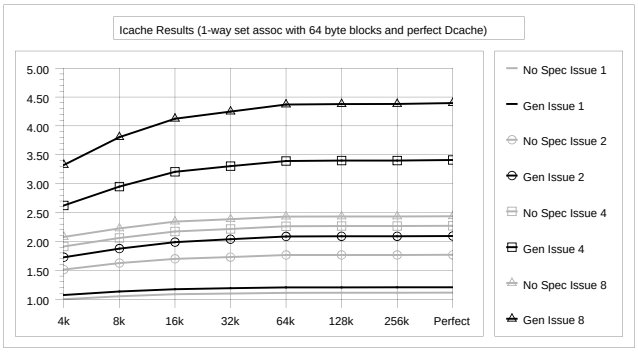

Figure 8: Icache effects for no speculation and general speculation models.

issue superscalar processor with register interlocking. The processor is assumed to have uniform functional units, 1 branch delay slot, and the instruction set of the HP PA-RISC processor. The instruction latencies assumed are those of the HP PA-RISC 7100 (see Table 5). For each machine configuration, the program execution times are derived from execution driven simulations of the benchmarks in Table 4. During the simulations, the issue widths were varied from 1 to 8 based upon the processor model that the code was scheduled for. Dynamic branch prediction was assumed using a 1024 entry direct mapped BTB with a 2 bit counter and a 2 cycle misprediction penalty. A perfect Dcache was used when measuring the Icache effects and a perfect Icache was used when measuring the Dcache effects. The cache configurations used for the experiments are given in Table 6.

4.2 Results

The shear volumes of data produced from the simulations made it impossible to present the individual benchmark results in this paper. In an effort to be more concise, the results presented in the subsequent figures are generated by computing the arithmetic mean of speedups for each speculation model, cache size and issue rate. Speedup was computed by dividing the execution time of the respective benchmark using the no speculation model at issue 1 with a 4K direct mapped Icache and Dcache by the execution time of the same benchmark using the specified speculation model at the specified cache size and issue rate.

Icache Performance Results

Figure 8 shows the performance results for direct mapped caches for the extreme speculation models - no speculation and general speculation. The first thing to observe from this figure is that the curves for the no speculation model show very little change regardless of the issue rate. In particular, there was an increase of only .35 IPC (16.9%) at issue 8 from a 4K to a 64K Icache. In contrast, the curves for the general speculation model showed a noticeable increase from the lower issue rates to the higher issue rates. In particular, their is an increase of .36 IPC (20.9%) at issue 2, .77 IPC (29.3%) at issue 4, and 1.05 IPC (31.5%) at issue 8. Thus, the benefits from larger cache sizes are more pronounced as the issue rate increases. Finally, the performance for all speculation models stabilized with a 64K Icache.

Figure 9 shows the performance results for 2-way set associative caches for no speculation and general speculation models. By comparing this figure to Figure 8, it is clear that there is little advantage in higher associativities with Icaches

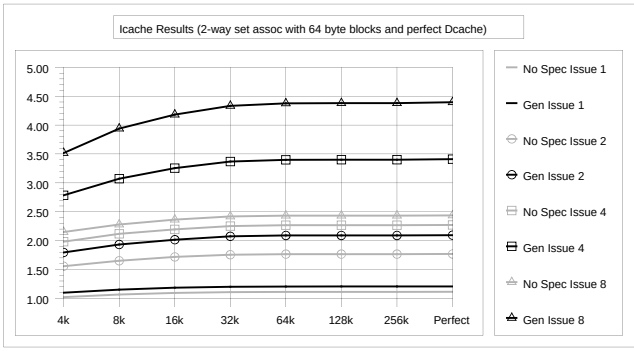

Figure 9: Icache effects for no speculation and general speculation models.

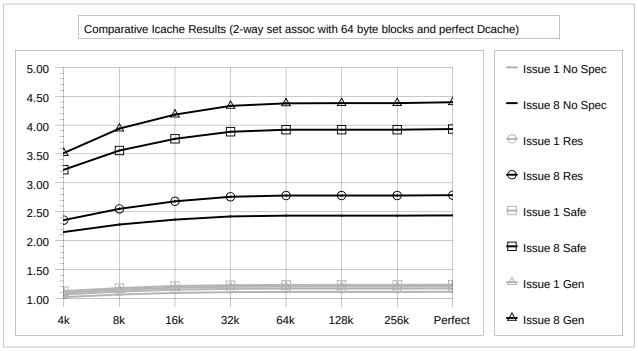

Figure 10: Icache effects for all speculation models at issue 1 and issue 8.

Benchmark	Read Requests	Read Misses	Miss Rate
008.espresso	412641852	1501759	0.36
022.li	35649513	943031	2.64
023.eqntott	1027576863	761471	0.07
026.compress	78221684	1716563	2.19
072.sc	72122569	1132815	1.57
cccp	3094004	37823	1.22
cmp	2198695	35	0.01
eqn	32813682	1508122	4.60
grep	1580207	2078	0.13
lex	46035584	208868	0.45
qsort	70546739	4041	0.01
tbl	2603306	45092	1.73
wc	1630199	35	0.01
yacc	43309632	350120	0.83
Average	130716038	586561	1.13

Table 7: Icache Access and Miss Rates at Issue 1 (direct mapped cache).

larger than 8K regardless of the issue rate or speculation model. Even at the lowest cache sizes, general speculation was only able to show a 6 percent speedup at 8-issue using 2-way set associative Icaches over direct mapped Icaches.

Figure 10 shows the comparative Icache results for all of the scheduling models at issue-1 and issue-8. As the figure shows, there is no significant performance advantage in using any of the aggressive speculation models for a single issue processor. Since only one instruction can be issued per cycle, the only potential slots that can be filled in the schedules of the integer benchmarks are branch and load delay slots. Therefore, there is very little opportunity to improve the performance of the benchmarks through more aggressive speculation. As a result of little speculation, only minor Icache effects are observed.

In contrast to the single issue performance, there is a clear advantage in using more aggressive speculation models at 8-issue. The no speculation model shows a 13.1 percent improvement between 4K and 64K Icaches. The restricted speculation model shows an 18.0 percent improvement, the safe speculation model shows 21.4 percent improvement and the general speculation model shows a 24.5 percent improvement over the same cache configurations. Thus, while the cache size was only a minor impediment to performance with lower issue rates, it is clearly a larger impediment to performance with higher issue rates for more aggressive speculation models. However, this set of benchmarks were not able to benefit from Icaches larger than 64K.

One additional point should be noted from the 8-issue results shown in Figures 10. The most aggressive speculation model's performance ranged from only 8.9 to 11.8 percent higher than safe speculation. Thus, safe speculation has great potential since it requires no special processor support that could potentially lead to slower clock rates. Also, it introduces none of the risks that result from ignoring scheduling errors like general speculation.

Analysis of Icache Results

To more fully understand the performance results, the Icache behavior is broken down in Tables 7 and 8. Table 7 contains the absolute number of read requests and read misses as well as the miss rate for each of the benchmarks in the base case. The numbers from Table 8 represent the read requests and read misses as a percentage of the totals presented in the final row of Table 7. As Table 8 shows, the more aggressive speculation models tend to reduce the number of Icache read requests. This can be justified by understanding how the simulator's fetch model works. The fetch model fills buffers equivalent to twice the issue rate of the processor in an effort to provide the processor with the issue-width number of instructions at each cycle. Thus, each cycle, the fetch unit fetches a block of instructions to fill the fetch buffer. Any instructions that cannot be placed into the fetch buffer will be discarded and potentially fetched again the next cycle. Since the more aggressive speculation models have more independent instructions each cycle to choose from, the compiler is better able to group independent instructions together and reduce interlock. As such, more instructions can be issued each cycle, which reduces the need to re-fetch the same cache block repeatedly.

As Table 8 shows, even though the number of read requests decreased, the absolute miss rates increased for both 4K and 64K from the least aggressive speculation models to the most aggressive speculation models. In particular, there was a 1 percent increase in the miss rate from no speculation to general speculation. There was practically no change in the Icache miss rates with 64K Icaches since the Icache was sufficiently large to hold the working set for all speculation

Speculation Model	4K Caches Read Req	4K Caches Read Misses	4K Caches Miss Rate	64K Caches Read Req	64K Caches Read Misses	64K Caches Miss Rate
None	0.426	0.779	0.82	0.456	0.172	0.17
Restricted	0.379	0.848	1.00	0.401	0.176	0.20
Safe	0.286	0.841	1.32	0.300	0.173	0.26
General	0.255	1.068	1.88	0.265	0.176	0.30

Table 8: Average Icache Access and Miss Rates at Issue 8 (2-way set associative cache).

Icache Block	No Speculation Misses	General Percolation Misses	General Percolation - No Speculation	Percent Change
1	2003	1980	-23	-0.01
2	3408	2486	-922	-27.05
3	2323	6952	4629	199.27
4	3227	1334	-1893	-58.66
Total	10961	12752	1791	16.34

Table 9: Icache Misses for the no speculation and general speculation models of the cccp loop example at Issue 8 (2-way set associative, 4K Icache).

models. While the miss rates for general speculation at 8-issue with a 64K cache is only 1.5 percent lower then the miss rate with a 4K cache, the performance was 24.5 percent higher. Thus, even a small increase in the miss rate can significantly impact the performance for the more aggressive speculation models. The impact on performance would be even more pronounced if the cache miss latency was greater than the simulated 12 cycles.

The cccp loop example shown in Tables 4 and 5 can be used to illustrate the reasons for the increase in the miss rate with the 4K Icache. Table 9 shows the Icache misses caused by the first instructions in each Icache blocks. The misses caused by the instruction at the start of the loop are represented with Icache block 1. There was only a negligible difference in the miss rates for the two speculation models in this block. Icache blocks 2 and 4 decreased their cache misses from the no speculation model to the general speculation model. Icache block 3 showed a significant increase in Icaches misses. Most of these misses can be attributed to migration of the misses from Icache blocks 2 and 4 to Icache block 3 due to the small 4K Icache. However, even after considering the migration of misses, there was an overall increase in misses for the loop by 16.34 percent which is attributable to the additional Icache block before the frequently taken branch number 59 in the Icache layout for the general speculation model.

Dcache Performance Results

Figure 11 shows the performance results for direct mapped Dcaches for the extreme speculation models. The first thing to observe from this figure is that the curves for the no speculation model show a much smaller increase in performance than general speculation at the same issue rates. In particular, there was an increase of only .51 IPC (27.5%) at issue 8 from a 4K to a 64K Dcache while the general speculation model showed an increase of 1.32 IPC (45.4 %). In contrast to the Icache results, the performance for general speculation model still demonstrates a noticeable

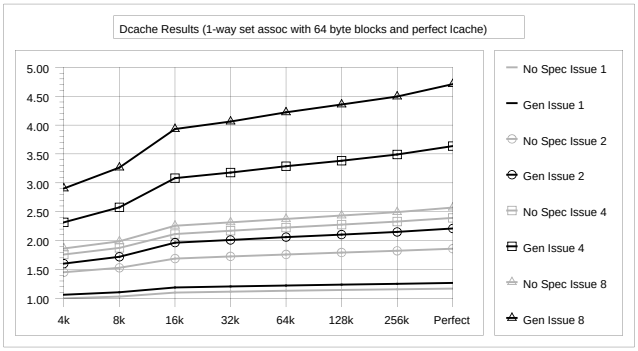

Figure 11: Dcache effects for no speculation and general speculation models.

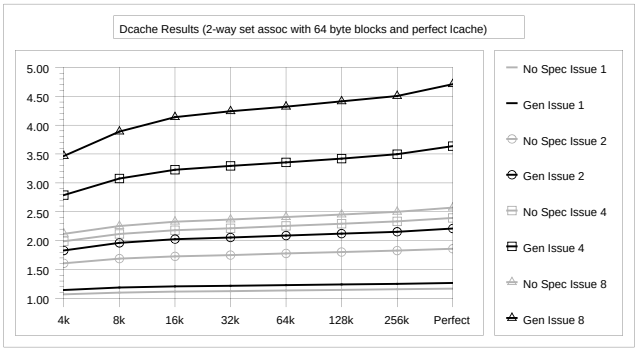

Figure 12: Dcache effects for no speculation and general speculation models.

improvement with Dcache sizes larger than 64K.

Figure 12 shows the performance results for 2-way set associative Dcaches for no speculation and general speculation models. By comparing this figure to Figure 11, it is clear that higher associativity significantly benefits the smaller Dcaches. In particular, general speculation showed a 19 percent improvement in performance at 8-issue for a 2-way set associative 4K Dcache over a direct mapped 4K Dcache. The no speculation model showed a 14 percent improvement in performance at the same cache configurations. Both speculation models showed some performance improvement with higher associativity when using Dcaches as large as 128K. Thus, higher associativity can be better used to offset the limitations of smaller Dcaches than the smaller Icaches.

Figure 13 shows the comparative Dcache results for all of the scheduling models at issue 1 and issue 8. As the figure shows, there is no significant performance advantage in using any of the aggressive speculation models for a single issue processor. However, at issue 8, there is a clear advantage in using the more aggressive speculation models. An increase in the Dcache size from 4K to 64K using the no speculation model resulted in a performance improvement of 13.8 percent while the restricted speculation model showed an increase of 16.3 percent. Safe speculation increased per-

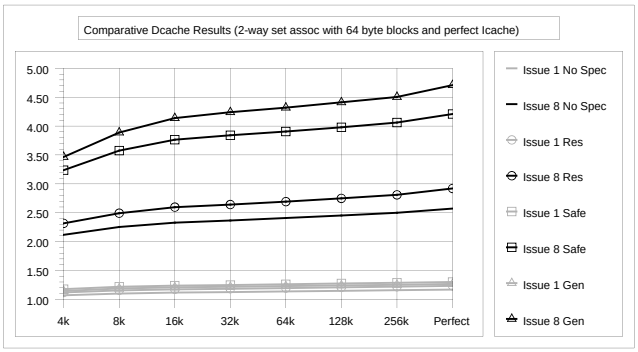

Figure 13: Dcache effects for all speculation models at issue 1 and issue 8.

Benchmark	Read Requests	Read Misses	Miss Rate
008.espresso	81609167	8517176	10.44
022.li	8059907	732281	9.09
023.eqntott	193278137	14396275	7.45
026.compress	9626725	3520353	36.57
072.sc	14788900	2094508	14.16
cccp	422760	21996	5.20
cmp	436248	163749	37.54
eqn	4343317	769148	17.48
grep	239327	2885	1.21
lex	7862729	314967	4.01
qsort	12206746	734301	6.02
tbl	548733	33554	6.11
wc	141527	2635	1.86
yacc	7488032	677821	9.05
Average	24382304	2284404	11.87

Table 10: Dcache Access and Miss Rates at Issue 1 (direct mapped cache).

formance by 20.6 percent and general speculation increased performance by 24.6 percent over the same region. While there was no performance advantage from increasing the Icache beyond 64K, this was not the case with the Dcache. The no speculation model improved its performance to 21.5 percent higher than 4K with perfect Dcaches. Restricted speculation improved to 26.2 percent higher than 4K. Safe speculation improved to 30.1 percent higher and general speculation improved to 35.9 percent higher. Thus, small Dcaches have been shown to be a significant impediment to the potential performance of more aggressive speculation models at higher issue rates.

Analysis of Dcache Results

To more fully understand the performance results, the Dcache behavior is broken down in Tables 10 and 11. Table 10 contains the absolute number of read requests and read misses as well as the miss rate for each of the benchmarks in the base case. The numbers from Table 11 represent the read requests and read misses as a percentage of the totals presented in the final row of Table 10. Table 11 shows that the Dcache accesses increase with the more aggressive speculation models. This is caused by an increase in the working set size resulting from speculation of additional load instructions.

The decrease in the miss rate from the less aggressive to the more aggressive speculation models is miss-leading since

	4K Caches			64K Caches		
Speculation Model	Read Req	Read Misses	Miss Rate	Read Req	Read Misses	Miss Rate
None	1.008	0.694	6.45	1.009	0.171	1.59
Restricted	1.017	0.712	6.56	1.019	0.181	1.66
Safe	1.143	0.725	5.94	1.145	0.173	1.41
General	1.313	0.776	5.54	1.315	0.193	1.38

Table 11: Average Dcache Accesses and Miss Rates at Issue 8 (2-way set associative cache).

Load Instr	No Spec. Misses	General Spec. Misses	General Spec. - No Spec.	Percent Change
158	124951	133596	8645	6.92
163	82058	132654	50596	61.65
165	1641	2444	803	48.93
178	84617	92107	7490	8.85
183	47746	91250	43504	91.11
185	890	1685	795	89.33
198	47566	50379	2813	5.91
203	23380	49579	26199	112.06
205	501	2346	1845	368.26

Table 12: Dcache Misses for the no speculation and general speculation models of the compress loop example at Issue 8 (2-way set associative, 4K Dcache).

the read requests have significantly increased. The Dcache misses actually increase from the less aggressive to the more aggressive speculation models. In particular, there was an 11.8 percent increase in the Dcache misses from the no speculation model to the general speculation model with a 4K Dcache and a 12.9 percent increase with a 64K Dcache. The 4.16 percent lower miss rate for general speculation with a 64K Dcache versus a 4K Dcache corresponds to a performance increase of 24.6 percent. Thus, the Dcache size can significantly affect the potential performance of aggressive speculation models.

The compress loop example shown in Tables 6 and 7 can be used to illustrate the reasons for the increases in Dcache misses. Table 12 shows the Dcache misses generated by the load instructions in the no speculation and general speculation codes based upon a 4K Dcache. It can be seen from this data that there were moderate to significant increases in Dcache misses from the no speculation case to the general speculation case. By comparing the increased Dcache miss rates for load instructions 163, 183 and 203 with their respective increases in execution frequency given in Table 3, it is apparent that the increase in miss rates for these loads was not constrained by the their increase in execution frequency. Other speculative loads actually caused further Dcache misses for these loads. In addition, the non-speculated load instructions 158, 178 and 198 also showed an increase in Dcache misses that is attributable to other speculated loads.

5 Conclusions

This paper has presented experimental results for four compiler-controlled speculation models over a variety of issue rates and cache configurations. The results indicate that the more aggressive speculation models create larger

instruction and data working sets. As such, processor designers need to ensure that cache configurations can tolerate the increased working set if they expect to attain the *best* performance from aggressive speculation models. These experiments have shown that increasing the Icache and Dcache from 4K to 64K resulted in a performance increase of approximately 26 percent for the general speculation model at issue 8. Additionally, the results indicate that 2-way set associativity beneficially reduces misses for Dcaches up to 128K. In contrast, 2-way set associativity was only beneficial for Icaches up to 8K.

While small Icaches and Dcaches can significantly limit the potential performance of more aggressive speculation models, there is still an advantage in using the more aggressive speculation models at higher issue rates even if the cache configuration is held constant. Even though some of the potential advantages of the more aggressive speculation models are negated by the higher miss rates, it was not sufficient to offset the performance advantages. In particular, general speculation at issue 8 was 63.6 percent faster than no speculation with the same 4K cache configuration and issue rate. Safe speculation was 50.2 percent faster and restricted was 9.6 percent faster. When using a 64K cache, general speculation was 80 percent faster than no speculation. Safe speculation was 61.1 percent faster and restricted speculation was 14.3 percent faster. The improvements in performance were almost identical for the experiments that used a perfect Icache and varied the Dcache as those that used a perfect Dcache and varied the Icache. Thus, aggressive speculation effects the Icache and the Dcache in a similar fashion.

Acknowledgements

The authors would like to thank all members of the IMPACT research group for their comments and suggestions. This research has been supported by the National Science Foundation (NSF) under grant MIP-9308013, Joint Services Engineering Programs (JSEP) under Contract N00014-90-J-1270, Intel Corporation, the AMD 29K Advanced Processor Development Division, Hewlett-Packard, SUN Microsystems, NCR and the National Aeronautics and Space Administration (NASA) under Contract NASA NAG 1-613 in cooperation with the Illinois Computer laboratory for Aerospace Systems and Software (ICLASS).

References

[1] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*. New York, NY: McGraw-Hill, 1991.

[2] P. P. Chang, S. A. Mahlke, W. Y. Chen, N. J. Warter, and W. W. Hwu, "IMPACT: An architectural framework for multiple-instruction-issue processors," in *Proceedings of the 18th International Symposium on Computer Architecture*, pp. 266–275, May 1991.

[3] E. M. Riseman and C. C. Foster, "The inhibition of potential parallelism by conditional jumps," *IEEE Transactions on Computers*, vol. c-21, pp. 1405–1411, December 1972.

[4] M. D. Smith, M. Johnson, and M. A. Horowitz, "Limits on multiple instruction issue," in *Proceedings of the 3rd International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 290–302, April 1989.

[5] J. A. Fisher, "Trace scheduling: A technique for global microcode compaction," *IEEE Transactions on Computers*, vol. c-30, pp. 478–490, July 1981.

[6] W. W. Hwu, S. A. Mahlke, W. Y. Chen, P. P. Chang, N. J. Warter, R. A. Bringmann, R. G. Ouellette, R. E. Hank, T. Kiyohara, G. E. Haab, J. G. Holm, and D. M. Lavery, "The Superblock: An effective technique for VLIW and superscalar compilation," *Journal of Supercomputing*, vol. 7, pp. 229–248, January 1993.

[7] M. D. Smith, M. A. Horowitz, and M. S. Lam, "Efficient superscalar performance through boosting," in *Proceedings of the Fifth International Conference on Architecture Support for Programming Languages and Operating Systems (ASPLOS-V)*, pp. 248–259, October 1992.

[8] S. A. Mahlke, 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, November 1993.

[9] R. A. Bringmann, S. A. Mahlke, R. E. Hank, J. C. Gyllenhaal, and W. W. Hwu, "Speculative execution exception recovery using write-back suppression," in *Proceedings of 26th Annual International Symposium on Microarchitecture*, December 1993.

[10] R. A. Bringmann, "Determining instructions that are safe to speculate at compile-time," tech. rep., Center for Reliable and High-Performance Computing, University of Illinois, Urbana, IL, May 1994.

[11] R. P. Colwell, R. P. Nix, J. J. O'Donnell, D. B. Papworth, and P. K. Rodman, "A VLIW architecture for a trace scheduling compiler," in *Proceedings of the 2nd International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 180–192, April 1987.

[12] J. C. Dehnert and R. A. Towle, "Compiling for the cydra 5," *Journal of Supercomputing*, vol. 7, pp. 181–227, January 1993.

[13] H. Packard, *PA-RISC 1.1 Architecture and Instruction Set Reference Manual*. Cupertino, CA, 1990.