Published in the Proceedings of the International Symposium of Microarchitecture (MICRO29), Dec. 1996

Design Decisions Influencing the UltraSPARC's Instruction Fetch Architecture

Robert Yung
Sun Microsystems Laboratories
Sun Microsystems Inc.
yung@sun.com

ABSTRACT

Designing a modern microprocessor is a complex task that demands careful balance between cycle time, cycles-per-instruction and area costs. In particular, the instruction fetch unit greatly affects the performance of a multi-issue processor. It must provide adequate bandwidth to sustain peak instruction issue rate and must predict future instruction sequences with high accuracy. In the UltraSPARC prefetch and dispatch unit design, we examined a technique that combined two prediction methods: predictive set-associative cache and in-cache prediction. This combination was compared with alternative designs such as direct-mapped and set-associative caches, and a branch history table and a branch target buffer. We chose the combined prediction technique for its fast cycle time, lower cycles-per-instruction, and lower area costs. This paper summarizes the trade-off decisions made in the design of the UltraSPARC instruction prefetch and dispatch unit.

1. Introduction

Designing an instruction prefetch and dispatch unit (PDU) has become a critical and complex task in a modern processor such as Sun Microsystem's UltraSPARC^{TM} [8]. It is critical because the throughput of a processor depends on the ability of the processor's PDU to fetch and issue needed instructions. It is complex for two reasons. As the speed gap between a processor and its memory system widens, there is greater reliance on high speed caches to sustain peak instruction dispatch bandwidth. A first level cache should utilize only a small fraction of the integrated circuit area to reduce cost and minimize access speed, but still provide most of the instructions needed by the PDU. The second complexity arises with increases in instruction issue width and pipeline depth. An efficient PDU should anticipate, with high accuracy, the outcome of issued yet pending branches, and prefetch and issue in the predicted execution path. If a prediction is later found to be incorrect, a PDU should quickly invalidate incorrectly issued instructions and restart fetching from the right instruction addresses.

The design of the UltraSPARC prefetch and dispatch unit centered around two questions:

- What is the optimal cache organization to sustain peak instruction dispatch rates?
- What branch prediction technique achieves high prediction accuracy with small misprediction penalties?

The answer to the first question is typically a choice of a direct-mapped or a set-associative cache; both have been thoroughly researched in academia and industry in the last thirty years with no definitive preference to either [1, 4, 5, 10, 12, 13]. Neither cache organization offered the best trade-offs for the UltraSPARC design. Instead, we proposed a new technique called the ***predictive set-associative cache (PSAC)*** [20], implemented in the UltraSPARC processor, in an attempt to combine the benefits of higher cache hit rates associated with set-associative caches and the shorter access time of direct-mapped caches.

The second question was how to handle branches. Two factors that make instruction fetching difficult are high dynamic branch density in programs and the large number of pending branches possible in a multi-issue processor. Percentages of dynamic branches in integer and commercial programs are usually over 20%, with more than half being taken branches [18]. High scalarity and deep pipelining increase the number of pending branches in a multi-issue processor. In search of a branch prediction solution for the UltraSPARC design, static prediction and software prediction were rejected because of their low prediction accuracy or difficulties in applying them to all programs. The combination of a branch history table (BHT) and a branch target buffer (BTB) offers good prediction accuracy [9]. Unfortunately, a BTB access could not be completed within the target cycle time without incurring a one cycle predicted taken branch penalty. Instead, a new branch prediction technique, ***in-cache prediction (ICP)*** [19], integrated into the instruction cache was proposed and implemented in the UltraSPARC. An ICP provides high branch prediction accuracy and no branch following latency; it can be implemented within the UltraSPARC target cycle time.

Solutions to these two questions were complicated by further tradeoffs such as area, power dissipation, design complexity and scalability for future enhancement. This paper offers some insight into the choices made in the UltraSPARC prefetch and dispatch unit design. Section 2 surveys instruction fetch unit organization in recent microprocessors. Section 3 describes the UltraSPARC instruction prefetch and dispatch unit and summarizes its design goals. Section 4 discusses the benchmarks and methodology used to drive the design process. Section 5 presents miss rates and cycle-per-instruction (CPI) costs for the PSAC and ICP schemes and some alternatives. These results include design alternatives not examined in the original UltraSPARC design. They show quantitatively the consequences of choices made in the design and point out possible future improvements. Section 6 examines area cost to implement the PSAC and ICP, and some alternatives. Section 7 concludes with a summary and suggests possible future work.

2. Related Work

With increases in transistor budget and performance goals, eight recent microprocessors, shown in Table 1, incorporate large and fast instruction caches and advanced branch prediction techniques to sustain high instruction dispatch and execution rates. Many microprocessors contain an on-chip, direct-mapped, single cycle access, level one instruction cache, as recommended in [10]. [5] suggested a hashing scheme to minimize conflict misses in a direct-mapped cache, but its complexity and cycle time impacts prohibit its use in a commercial design. [6] outlined a scheme similar to PSAC with its benefits but did not include data specifying its miss rates and miss causes.

All eight microprocessors implement branch prediction. The most popular scheme is a branch history table combined with either a branch target address buffer or dynamic target address calculation in parallel with instruction cache access. If a branch is predicted not taken in the BHT or is absent from the BTB, it is presumed to be not taken and its address is calculated. The combination of a BHT and a BTB provides high accuracy in branch direction and target address prediction but has two disadvantages. First, because the next fetch address selection depends on the outcome of the BTB search, it may add to the critical path of a design. Second, if the current cache line contains more than one branch, the branch history and branch target address of each branch must be accessed and prioritized to determine the next fetch address. This most likely adds to the critical path of a design. In this case, the next fetch may be assumed not taken and if wrong, the target address can be refetched in the following cycle. This approach reduces cycle time at the expense of a two cycle taken branch latency which leads to increased CPI for taken branches.

Except for the UltraSPARC and the AMD 5K86, all the BHTs and BTBs are implemented with RAM structures separate from the instruction cache. Incorporating the ***next fetch address (NFA)*** into a cache was described in [6] and [11]. Both schemes store only target addresses in the NFA, while sequential addresses of predict-not-taken branches are calculated. Our study in [22] shows that there is little difference in prediction accuracy between storing only target addresses in the NFA and storing either a branch target or a sequential address in the NFA. Storing only target addresses in the NFA has both the BTB disadvantages, while the alternate does not.

	Sun UltraSPARC	**DEC 21164**	**HAL R1**	**HP PA-8000**
Clock Frequency	167 / 250 Mhz	300 / 366 Mhz	154 Mhz	160 / 275Mhz
Level 1 Instruction cache (L1IC)	16KB predictive 2-way	8KB direct-mapped	4KB direct-mapped	external 1MB direct-mapped
L1IC thruput/latency	1 / 1	1 / 1	1 / 1	1 / 2
Icache bandwidth	16B per cycle	16B per cycle	16B per cycle	16B per cycle
branch history	2048-entry 2-bit BHT in L1IC	2048-entry 2-bit BHT	1024-entry 2-bit BHT	256-entry 3-bit BHT
branch target address	1024 11-bit NFA in L1IC	computed and stored in L1IC on miss	computed 1 cycle after L1IC fetch	32-entry BTB
return addr stack	4-entry	none	4-entry	none
taken branch latency	none	1 cycle	none if hit L1 Icache	none
mispredict penalty	4 cycles	5 cycles	3 to 6 cycles	5 cycles

Table 1: Summary of instruction fetch unit in recent microprocessors [1,4,12].

	IBM PPC620	SGI R10000	AMD 5K86	Intel Pentium Pro
Clock Frequency	133 Mhz	200 Mhz (estimated)	90 Mhz	133 / 200 Mhz
Level 1 Instruction cache (L1IC)	32KB effective 8-way	32KB 2-way	16KB direct-mapped	8KB 4-way
L1IC thruput/latency	1 / 2	1 / 2	1 / 1	1 / 3
L1IC bandwidth	16B per cycle	16B per cycle	16B per cycle	16B per cycle
branch history	2048-entry 2-bit BHT	512-entry 2-bit BHT	1024-entry 1-bit BHT in L1IC	512-entry 4-way 4-bit BHT
branch target address	256-entry BTB	computed 1 cycle after L1IC fetch	1024-entry BTB in L1IC	512-entry, 4-way BTB
return addr stack	none	4-entry branch stack	none	16-entry
taken branch latency	1 cycle	1 cycle	none	1 cycle
mispredict penalty	2 cycles	> 4 cycles	> 3 cycles	> 10, 15 cycles (typ)

Table 1: Summary of instruction fetch unit in recent microprocessors [1,4,12].

3. Instruction prefetch and dispatch

Features	UltraSPARC (US-I,-II) PDU implementation
Cycle time	single cycle instruction cache access. 6ns in US-I; 4ns in US-II
Capacity, Associativity	16KB, predictive 2-way set-associative, 32B block size
Physical organization	direct-mapped, 32 bytes per line, delivers up to 4 out of 8 instructions in a cache line each cycle
Set prediction	1024-entry 1-bit dynamic set predictors (SP), 1 for 4 instructions
Branch prediction	2048-entry 2-bit dynamic history (PRED), 1 for 2 instructions
Address prediction	1024-entry 11-bit next-fetch-address (NFA), 1 for 4 instructions, 4-entry subroutine return stack
Decoded instruction bits	One 4-bit instruction class field (ICLASS) per instruction
Address translation	single cycle access, 64-entry fully associative TLB. single cycle check, one 1-entry micro-TLB, 41-bit physical address
Instruction buffer	12-entry instruction buffer
Transistors, Area	1.6 million transistors; <10% chip area

Table 2: Summary of the UltraSPARC PDU.

To limit the number of design choices for the UltraSPARC PDU, the following goals were set [17]:

- single cycle cache access sets the cycle time.
- contributes to less than 30% of total CPI cost.
- sustains instruction issue rate above three for integer and four for floating point and graphic code.
- instruction cache miss rate below 2% per instruction.
- branch and target prediction accuracy above 85% per branch with minimal misprediction penalties.
- consumes less than 20% of chip area.

All the PDU goals were achieved in the UltraSPARC design. Highlights of the UltraSPARC PDU are summarized in Table 2. Two key PDU design decisions are its cache organization and branch prediction technique. In the remainder of this paper, the choices made in the UltraSPARC design, predictive set-associative cache and in-cache prediction, are evaluated.

4. Evaluation metric

4.1. Methodology

Two approaches were taken in the design of the UltraSPARC prefetch and dispatch unit. Results from three analytical models set the six PDU goals. A cycle accurate simulator finalized the UltraSPARC organization.

Three analytical models were coded in a spread sheet program. The CPI model estimates CPI costs for a wide range of superscalar pipelines with different numbers and types of functional units, branch predictors, and cache and memory organizations. Two sets of inputs used by the CPI model are processor-independent statistics such as cache miss rates and dynamic instruction and dependency counts, and pipeline and cache configurations for the target design. A speed model was created to estimate cycle time by calculating worst case delay of critical paths using a delay file for the target process technology. The delay file contains estimation of delays, produced by HSPICE, for the most commonly used components such as the ALU, registers and cache array. An area model estimates die size by summing the estimated cell sizes for each component with consideration given to cache array efficiency, routing and input/output overhead. The results from the analytical models set the six PDU goals.

Although the six goals limited the number of choices, the remaining PDU alternatives needed a thorough and detailed analysis. A cycle accurate simulator, *spitfire_per*, was written which contains 26,000 lines of C code [16] (Spitfire was the code name of UltraSPARC). The simulator includes parameterized components such as pipelines, multi-level cache, load and store buffers, instruction prefetch and dispatch front-end, and a simple memory and bus model. The final UltraSPARC organization was based on the results from *spitfire_per*.

The results reported in this paper are based on a faster PDU simulator, *Cachesim7* [21], and confirmed with *spitfire_per* results. *Cachesim7* has been generalized for evaluation of a wider spectrum of alternatives based on in-cache prediction schemes. Results of alternative branch designs are based on *pred1* [21], a generic BHT and BTB simulator.

4.2. Benchmarks

SPEC92Int was one of the benchmarks used to evaluate architectural alternatives for the UltraSPARC. Its analysis formed the basis of this paper. The results from the SPEC95Int benchmark was also included, with the hope that it is more representative of the programs we use today. What we found was that the two sets of benchmarks behaved similarly and most of their programs did not exercise the instruction cache nor the branch prediction mechanism typical in a modern microprocessor.

All the performance results were obtained from trace-driven simulations for the six SPEC92Int and eight SPEC95Int benchmark programs [2,3]. *The results from gcc, the only program in the benchmark suite that exercises a substantial number of static branches similar to larger programs [13], are presented along with the geometric means of the six SPEC92Int and eight SPEC95Int programs.* The SPEC92Int and SPEC95Int programs were compiled with the SunSoft compiler SC3.0 and SC4.2 respectively. The binaries were traced with *shade* [7] running on Solaris 2.5. *Shade* produces dynamic instruction traces for the simulators described above. Due to *shade's* limitations, only user level instructions were simulated. Branch statistics for the benchmarks are shown in Table 3.

Benchmarks	programs	Dynamic instructions (excluding annulled instruction)	Dynamic Branches (% dynamic instruction)			static instr.	Static Branches (% static instruction)			num. of static branches constituting 90% of dynamic branches
			uncond	return	cond		uncond	return	cond	
SPEC92Int	espresso	931,034,624	1.6%	0.7%	17.9%	74,848	9.3%	1.3%	8.9%	191
	li	4,764,766,826	4.3%	2.4%	16.3%	53,506	10.1%	1.6%	9.4%	120
	eqntott	579,705,590	2.8%	2.0%	19.8%	26,212	5.9%	1.9%	12.5%	77
	compress	126,826,068	1.3%	0.7%	14.4%	42,968	8.3%	1.7%	9.1%	17
	sc	315,481,376	3.5%	1.1%	18.8%	67,579	9.3%	1.4%	9.4%	235
	gcc	1,127,468,030	2.9%	1.8%	15.7%	208,283	9.3%	2.0%	11.1%	2427
SPEC95Int	go	94,271,363,517	2.3%	0.9%	12.7%	104,537	6.1%	1.0%	10.0%	1137
	m88ksim	65,618,372,090	2.1%	1.4%	13.9%	53,098	8.1%	2.2%	10.3%	170
	gcc	11,926,786,390	2.5%	1.8%	19.0%	326,220	9.0%	1.7%	12.8%	4240
	compress	14,048,170,148	2.9%	0.0%	13.4%	29,030	8.1%	1.4%	10.7%	6
	li	51,761,814,700	4.5%	2.0%	17.0%	42,799	9.9%	1.7%	10.0%	114
	ijpeg	82,707,620,515	5.3%	5.3%	12.3%	80,675	5.6%	2.8%	9.4%	63
	perl	35,450,862,596	2.9%	2.4%	16.4%	99,313	9.8%	1.2%	10.6%	188
	vortex	165,162,278,043	2.8%	2.1%	16.7%	150,792	9.2%	1.3%	9.5%	267

Table 3: Branch statistics for the SPEC92Int and SPEC95Int benchmarks.

Note: Espresso and sc SPEC92Int results are based on tial.in and loada1 respectively. Gcc SPEC95Int results are based on inputs amtjp.i, gcc.i, integrate.i, and varasm.i. All results based on the average of the first one billion dynamic instructions of each input. Unconditional branches (uncond) include call, branch_always and branch_never. Returns are JMPL used as returns. Conditional branches (cond) are PC-relative conditional branches.

5. Performance evaluation

In this section, predictive set-associative cache and in-cache prediction techniques are described. Miss rates and CPI costs for each are presented and analyzed.

5.1. Predictive set-associative cache (PSAC)

A direct-mapped cache has a higher miss rate than a set-associative cache but is faster because instructions from a cache access can be used while the tag comparison takes place, thus reducing cache cycle time. A set-associative cache reduces conflict misses, and overall cache misses, by providing two or more sets to which a cache block can be stored. The disadvantages of a set-associative cache are higher area cost and longer access time than a direct-mapped cache of the same size. If a set-associative cache were implemented, UltraSPARC would have a 30% increase in cycle time, or else two-cycle cache accesses.

A PSAC enables a direct-mapped cache to achieve the same higher cache hit rate of a multi-way set-associative cache with minimal area overhead. A PSAC is divided into N logical sets (N=2 in the UltraSPARC) implemented with a direct-mapped SRAM array. Each cache line contains one or more dynamic set predictors (SP). When a cache line is fetched, the corresponding SP selects the set for the next fetch. If a set prediction is correct, the access time of a PSAC is the same as a direct-mapped cache. In the event that an SP misses, the needed set is refetched in a subsequent cycle and the incorrect SP is updated. During a cache miss, a most recently used history bit, mru, is used to decide the placement or replacement of a cache line, just as in a set-associative cache. If the set size does not exceed the smallest page size, a PSAC enables a further speed improvement by allowing a cache access to proceed in parallel with TLB translation. Although the translation outcome is needed to determine a cache hit or miss, the likely cache entry can be used in the interim.

The UltraSPARC uses a 2-way PSAC and there is a one-bit SP for every four instructions in the UltraSPARC instruction cache. The instructions of only the predicted set are accessed in each four-instruction fetch, thus saving power and cache cycle time. To minimize SP miss cost to one cycle, tags of both sets are read in parallel.

The efficiency of a PSAC depends on miss rate differences between a direct-mapped and a set-associative cache of equal size, and set prediction accuracy. If the working set of programs fits in a direct-mapped cache, PSAC does not help. A PSAC is useful when conflict misses dominate cache misses. Figure 1 to Figure 4 show cache miss rates for direct-mapped and predictive 2-way PSAC, and SP miss rates. Three reasons for set prediction misses are explained below and their breakdowns are shown in Table 4. Table 5 shows the CPI costs for several possible UltraSPARC cache organizations.

Fetch size is the number of instructions, four in the UltraSPARC, read in each cache access. Although fetch size does not affect the cache miss rate, it affects the SP miss rate in two ways. First, there is a great increase in SP sharing misses, discussed below, if the number of instructions per fetch is less than the number of instructions per SP. Figure 1 to Figure 4 show that eight instructions per SP is always much worse than four or fewer instructions per SP because the UltraSPARC is a four instruction per fetch design. Second, larger fetch sizes reduce SP miss rates because there are more instructions in each fetch and fewer set predictions are needed for a given number of instructions fetched. Another indirect contributor to SP misses is a cache's block size. The UltraSPARC has a 32 byte cache block. As the block size increases, there are fewer blocks in a given sized cache which causes an increase in conflict misses.

During a cache fill, SPs in the current cache line are initialized to the current set. ***First use misses*** occur when the initial set predictions are wrong. The SP of the last cache line leading to the missed fetch may be updated to avoid a future miss. If a cache line has multiple callers, each of the callers' accesses may result in a first use miss. Table 4 shows that first use misses decrease with larger caches. ***Branch misses*** happen because conditional branches change between taken and not taken directions during execution. If only a single SP is used to hold set history for a branch, changes in branch direction may cause a future set prediction miss. A design that has separate or *dual SPs*, one for each of taken and not taken directions, can avoid this class of misses but requires branch prediction information before set selection for the next fetch. ***Sharing misses*** are a major component of total set prediction misses. Sharing misses occur when an SP is shared between two or more instructions and one of the earlier instructions is a branch. The latter instruction may be the delay instruction of the branch. For example, if the set of a CALL target address is different from the set of the sequential address, an SP can hold either, but not both of them. Sharing misses also occur when a fetched block has one or more entry points after a branch. A branch miss is a form of sharing miss.

As cache size increases, total cache misses decrease and conflict misses dominate over capacity and compulsory misses. Because mispredictions are irrelevant during cache misses, set prediction miss rates increase initially for small cache sizes as cache hits increase. Despite these initial increases in set prediction miss rates, Table 5 shows that a PSAC with fewer than eight instructions per SP has a lower or comparable CPI than the alternatives. When working sets fit in the larger caches, set prediction miss rates drop as well.

The UltraSPARC design point, 16KB cache size, four instructions per SP, is highlighted in Table 4 and Table 5. Figure 1 shows that a 16KB direct-mapped cache has only a 0.6% higher cache miss rate than a 2-way PSAC in gcc92, but the CPI, shown in Table 5, is increased by 0.03 or 3% for a design with 1.00 original CPI. Similarly, Figure 3 showed that a 0.7% higher direct-mapped cache miss rate in gcc95 increased the CPI by 0.02. A 2-cycle, 2-way design is worse than the others because of its multi-cycle access. Table 5 also shows that the SPEC92Int and SPEC95Int CPI differences between a direct-mapped and a 2-way PSAC design are minimal because many programs fit in small, direct-mapped caches and a PSAC offers little benefit in CPI reduction.

The UltraSPARC design could be improved by having one instruction per SP and most importantly, by reducing the instruction cache miss cost. A one predictor per instruction design would decrease set prediction miss cost by 0.02 for SPEC92Int and 0.01 for SPEC95Int.

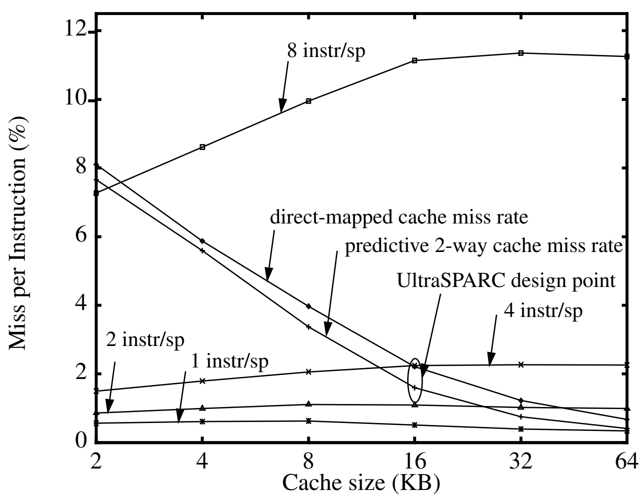

Figure 1. Icache and SP miss rates (gcc92)

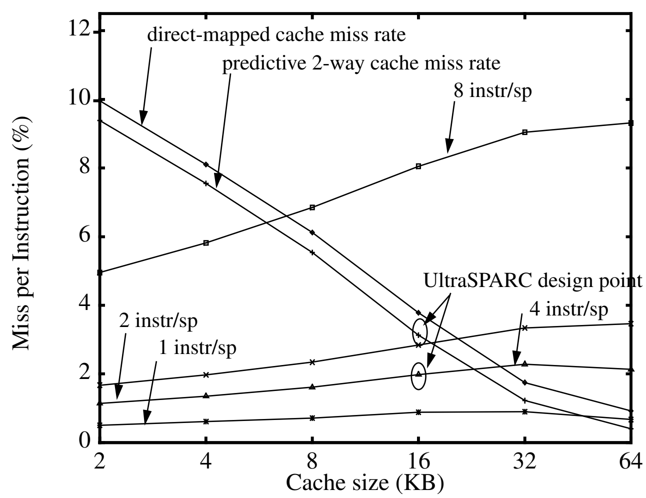

Figure 3. Icache and SP miss rates (gcc95)

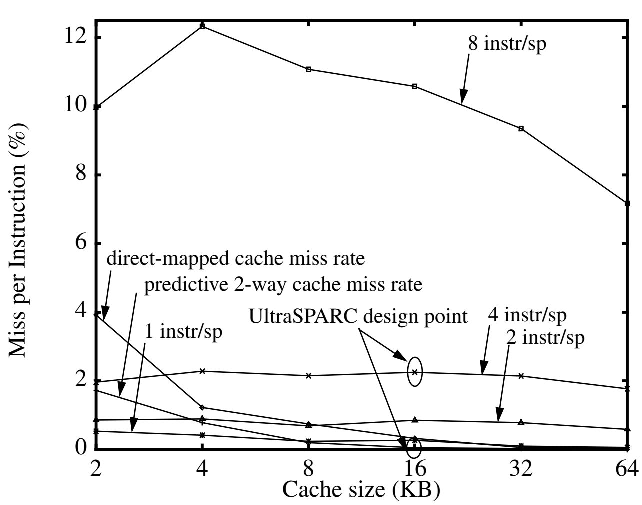

Figure 2. Icache and SP miss rates (SPEC92Int)

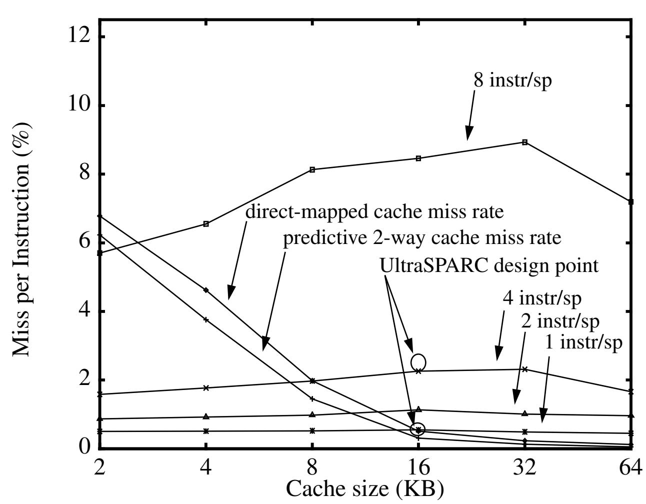

Figure 4. Icache and SP miss rates (SPEC95Int, excluding 129.compress. Icache and SP miss rates for 129.compress are nearly zero.)

cache size	Predictive-2-way, single sp							
	1 instr/sp		2 instr/sp		4-instr/sp		8 instr/sp	
	1st_use	brmiss + sharing	1st_use	brmiss + sharing	1st_use	brmiss + sharing	1st_use	brmiss + sharing
	gcc92							
2KB	0.36% 64%	0.20% 36%	0.32% 37%	0.54% 63%	0.29% 19%	1.21% 81%	0.11% 2%	7.16% 98%
4KB	0.34% 56%	0.27% 44%	0.30% 30%	0.69% 70%	0.27% 15%	1.52% 85%	0.10% 1%	8.51% 99%
8KB	0.28% 44%	0.35% 56%	0.26% 23%	0.85% 77%	0.22% 11%	1.84% 89%	0.08% 1%	9.87% 99%
16KB	0.16% 31%	0.35% 69%	0.14% 13%	0.95% 87%	**0.13% 6%**	**2.11% 94%**	0.05% 0%	11.08% 100%
32KB	0.07% 18%	0.32% 82%	0.06% 6%	0.96% 94%	0.05% 2%	2.22% 98%	0.02% 0%	11.33% 100%
64KB	0.04% 12%	0.30% 88%	0.03% 3%	0.96% 97%	0.02% 1%	2.24% 99%	0.00% 0%	11.25% 100%
	gcc95							
2KB	0.26% 52%	0.24% 48%	0.23% 20%	0.91% 80%	0.21% 13%	1.46% 87%	0.10% 2%	4.85% 98%
4KB	0.26% 43%	0.35% 57%	0.23% 17%	1.12% 83%	0.21% 11%	1.76% 89%	0.09% 2%	5.73% 98%
8KB	0.28% 39%	0.43% 61%	0.25% 16%	1.36% 84%	0.23% 10%	2.11% 90%	0.09% 1%	6.76% 99%
16KB	0.26% 30%	0.62% 70%	0.23% 12%	1.75% 88%	**0.22% 8%**	**2.62% 92%**	0.08% 1%	7.97% 99%
32KB	0.13% 14%	0.77% 86%	0.12% 5%	2.16% 95%	0.11% 3%	3.23% 97%	0.04% 0%	9.00% 100%
64KB	0.04% 6%	0.63% 94%	0.03% 1%	2.10% 99%	0.03% 1%	3.43% 99%	0.01% 0%	9.30% 100%

Table 4: Types of set prediction misses. The first percentage in each pair is the SP miss rate per instruction. The second percentage is the portion of total SP misses for the miss type.

cache size	Direct-mapped	2-way, 2 cycles	Predictive-2-way, single sp				Direct-mapped	2-way, 2 cycles	Predictive-2-way, single sp			
			1 instr/ sp	2 instr/ sp	4-instr/ sp	8 instr/ sp			1 instr/ sp	2 instr/ sp	4-instr/ sp	8 instr/ sp
	gcc92						SPEC92Int					
2KB	0.67	0.87	0.64	0.64	0.65	0.71	0.33	0.40	0.16	0.16	0.17	0.25
4KB	0.49	0.71	0.48	0.48	0.49	0.56	0.11	0.32	0.08	0.08	0.10	0.20
8KB	0.34	0.53	0.30	0.30	0.31	0.39	0.07	0.28	0.03	0.04	0.05	0.14
16KB	0.20	0.40	0.15	0.16	**0.17**	0.26	0.04	0.27	0.02	0.03	**0.04**	0.12
32KB	0.12	0.33	0.09	0.09	0.10	0.20	0.02	0.27	0.02	0.02	0.04	0.11
64KB	0.08	0.30	0.06	0.06	0.08	0.17	0.02	0.26	0.01	0.02	0.03	0.08
	gcc95						SPEC95Int					
2KB	0.84	1.02	0.79	0.80	0.81	0.84	0.34	0.67	0.32	0.32	0.32	0.35
4KB	0.69	0.87	0.65	0.66	0.66	0.70	0.24	0.54	0.20	0.21	0.22	0.26
8KB	0.53	0.72	0.49	0.50	0.51	0.55	0.12	0.41	0.10	0.10	0.11	0.18
16KB	0.34	0.53	0.30	0.31	**0.32**	0.37	0.06	0.34	0.05	0.06	**0.06**	0.12
32KB	0.18	0.38	0.15	0.16	0.17	0.23	0.04	0.30	0.03	0.03	0.05	0.10
64KB	0.11	0.32	0.08	0.09	0.11	0.16	0.03	0.28	0.02	0.03	0.03	0.07

Table 5: Icache miss CPI costs. Miss costs are 8 cycles for Icache, 30 cycles for Ecache and 1 cycle for SP. The 512KB unified Ecache miss rates are 0.11% and 0.06% for gcc92 and SPEC92Int, and 0.19% and 0.01% for gcc95 and SPEC95Int respectively.

5.2. Branch History: PRED field vs. BHT

The PRED and BHT results are based on single cycle access, predictive 2-way PRED and direct-mapped BHT with single level 2-bit predictors and four cycle miss penalty. The UltraSPARC 16KB instruction cache contains 2048 2-bit dynamic PRED fields, or one PRED field for every two instructions. The PRED fields do not directly affect next instruction fetches but are used by the PDU to determine whether to update the NFA and SP fields when PRED misses occur. The breakdown of three PRED miss types are shown in Table 6. During a cache fill, PRED fields are initialized to likely-not-taken state unless there is a likely-taken software hint. A ***first use miss*** occurs the first time a PRED field is used and the initial prediction is wrong. First use misses decrease with larger caches and can also be reduced by preserving history in the next level cache. ***Direction misses*** occur when a branch changes direction and constitute the majority of PRED misses. ***Sharing misses*** occur when a PRED field is shared between two or more branches. Although fewer than 30% of all instructions are branches, some branches are clustered together which results in sharing misses.

Several observations can be drawn from the miss rates and CPI results shown in Figure 5 to Figure 8, Table 6 and Table 7. First, as the number of predictors or cache size increases, PRED miss rates decrease because more branch history is maintained. Second, most SPARC branches have a delay slot and the delay slot rarely contains another branch instruction. The delay slot is the reason for the negligible miss rate differences between one instruction per PRED with 2N predictors and two instructions per PRED with N predictors. Third, BHT, with 2048 or fewer entries, has lower miss rates than PRED because victimization of a BHT entry affects only one entry but victimization of a cache line may lose several PREDs. With more than 2048 entries, the working set fits in the PRED as evidenced in the small first use miss rates in Table 6. Fourth, the optimal design point, the lowest point in each curve, shifts from four to two instructions per PRED as the number of PREDs increases. With fewer PREDs, PRED misses are dominated by cache misses; with more PREDs, direction and sharing misses are the major component of the PRED misses. Fifth, Table 7 shows that a design with fewer than eight instructions per PRED has similar miss rates and CPI costs as a BHT but a PRED implementation uses less area. The UltraSPARC design is at or near the optimal.

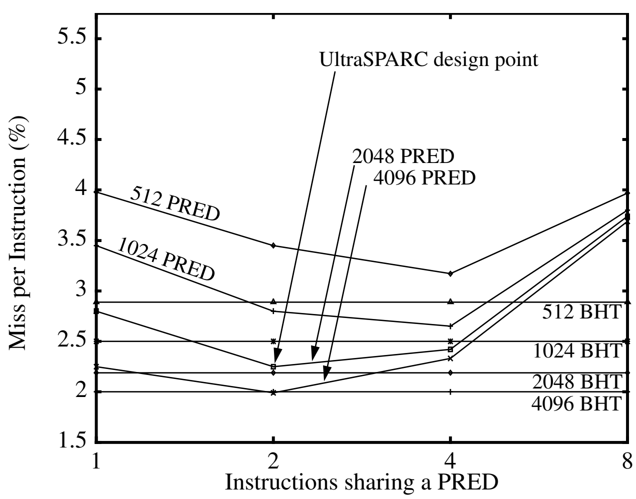

Figure 5. PRED and BHT miss rates (gcc92).

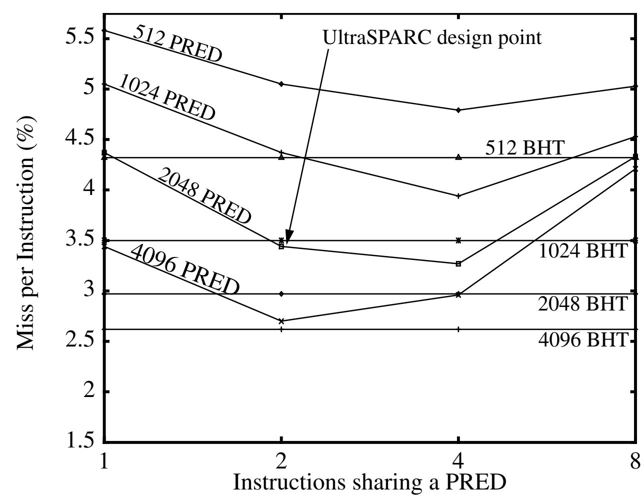

Figure 7. PRED and BHT miss rates (gcc95).

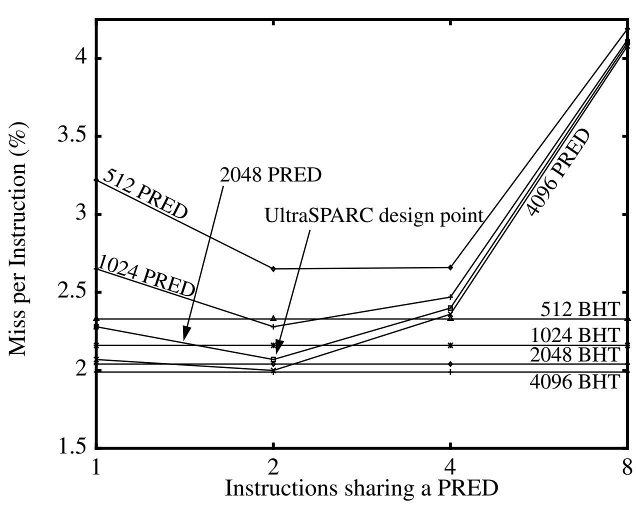

Figure 6. PRED and BHT miss rates (SPEC92Int).

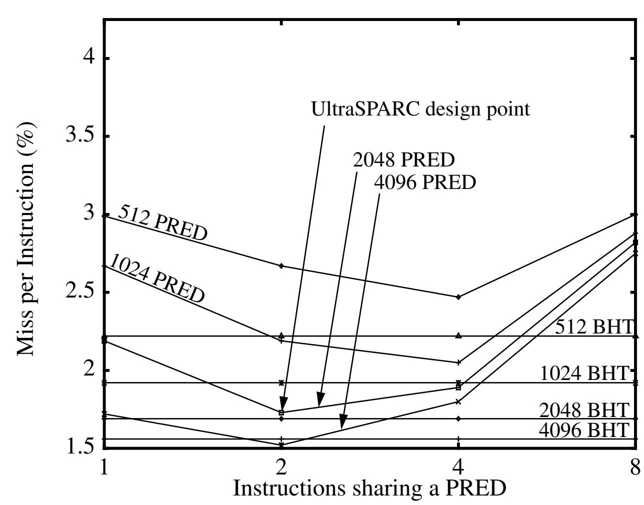

Figure 8. PRED and BHT miss rates (SPEC95Int).

number of PRED	1 instr/pred		2 instr/pred		4-instr/pred		8 instr/pred	
	1st_use	direction + sharing	1st_use	direction + sharing	1st_use	direction + sharing	1st_use	direction + sharing
	gcc92							
512	2.72% 68%	1.26% 32%	2.03% 59%	1.42% 41%	1.14% 36%	2.03% 64%	0.43% 11%	3.54% 89%
1024	2.03% 59%	1.42% 41%	1.23% 44%	1.57% 56%	0.53% 20%	2.12% 80%	0.20% 5%	3.60% 95%
2048	1.23% 44%	1.57% 56%	**0.58% 26%**	**1.67% 74%**	0.25% 10%	2.17% 90%	0.11% 3%	3.63% 97%
4096	0.58% 26%	1.67% 74%	0.28% 14%	1.71% 86%	0.14% 6%	2.19% 94%	0.04% 1%	3.65% 99%
	gcc95							
512	4.16% 75%	1.42% 25%	3.44% 68%	1.61% 32%	2.40% 50%	2.39% 50%	1.05% 21%	3.98% 79%
1024	3.44% 68%	1.61% 32%	2.58% 59%	1.79% 41%	1.35% 34%	2.59% 66%	0.41% 9%	4.12% 91%
2048	2.58% 59%	1.79% 41%	**1.46% 42%**	**1.98% 58%**	0.54% 17%	2.73% 83%	0.13% 3%	4.20% 97%
4096	1.46% 42%	1.98% 58%	0.58% 21%	2.12% 79%	0.17% 6%	2.79% 94%	0.01% 0%	4.20% 100%

Table 6: Breakdown of PRED misses by type for gcc. The first percentage in each pair is the PRED miss rate per instruction. The second percentage is the portion of total PRED misses for the miss type.

# history entries	BHT	PRED				BHT	PRED			
		1 instr/pred	2 instr/pred	4-instr/pred	8 instr/pred		1 instr/pred	2 instr/pred	4-instr/pred	8 instr/pred
	gcc92					SPEC92Int				
512	0.12	0.16	0.14	0.13	0.16	0.09	0.13	0.11	0.11	0.17
1024	0.10	0.14	0.11	0.11	0.15	0.09	0.11	0.09	0.10	0.16
2048	0.09	0.11	**0.09**	0.10	0.15	0.08	0.09	**0.08**	0.10	0.16
4096	0.08	0.09	0.08	0.09	0.15	0.08	0.08	0.08	0.09	0.16
	gcc95					SPEC95Int				
512	0.17	0.22	0.20	0.19	0.20	0.09	0.12	0.11	0.10	0.12
1024	0.14	0.20	0.17	0.16	0.18	0.08	0.11	0.09	0.08	0.12
2048	0.12	0.17	**0.14**	0.13	0.17	0.06	0.09	**0.07**	0.07	0.11
4096	0.10	0.14	0.11	0.12	0.17	0.06	0.07	0.06	0.07	0.10

Table 7: Branch history miss CPI costs: BHT vs. PRED.

5.2.1. Branch Target Address: NFA vs. BTB

In the absence of a branch misprediction, the PDU uses the next fetch address (NFA) field in the current cache line for the next fetch without examining the instruction types or branch direction history in the current cache line. Because the NFA is part of the instruction cache, next fetch address prediction does not lengthen the cycle time of the cache or the PDU. The NFA also enables the PDU to follow one or more branches in the next fetch or ***1-cycle branch following***, which is crucial in wide-issue designs. An NFA design does not have the cycle time and branch following disadvantages of a BTB. The branch following disadvantage may greatly degrade the efficiency of a wide-issue instruction fetch unit design with frequent back-to-back taken branches.

Predicted NFAs are verified with either the sequential address of an actually-not-taken branch or the target address of an actually-taken branch. If an NFA misprediction occurs, the NFA is updated according to an update policy and the corrected address is refetched within two cycles. Updating an NFA only on PRED misprediction and branch direction change yields the lowest NFA miss rates [22] and is assumed in this paper. There are three types of NFA misses. ***First use misses*** are caused by initial predicted-not-taken but actually-taken branches. A ***PRED miss*** happens when branch history is incorrect; a PRED miss most likely leads to an NFA miss. Misprediction penalties for these two NFA miss types are accounted for in the PRED miss cost and are not included in Table 8. ***Sharing misses*** occur when an NFA is shared between two or more instructions and one of the earlier instructions is a branch or the delay slot of a branch.

To improve target address prediction accuracy for subroutine returns, a BTB or an NFA may be supplemented with a subroutine return address stack. There is a four-entry return address stack implemented in the UltraSPARC. When a return instruction is fetched, the next fetch based on the NFA is discarded. The PDU refetches with the return address at the top of the return stack. Refetching causes a one cycle penalty for subroutine return but improves the overall CPI.

The PRED and BHT results are based on single cycle access, predictive 2-way NFA and direct-mapped BTB. Several observations can be drawn from Figure 9 to Figure 12 and Table 8. First, NFA miss rates are dominated by sharing misses; NFA sharing should be avoided. Second, unlike PRED miss rates which decrease with more PRED entries, the NFA miss rates may increase with more NFA entries, as shown near the UltraSPARC design point in the figures. Despite higher NFA misses, the combined CPI cost for a PRED and an NFA decreases with more PRED and NFA entries. The NFA misses caused by PRED misses are irrelevant and are not included in the figures and Table 8. Third, miss rates increase rapidly when the fetch size, four instructions per fetch, is less than the number of instructions per NFA.

Table 8 shows the CPI impacts of NFA and BTB misses. An NFA is a better choice than a BTB for the UltraSPARC design because a BTB design would have a two cycle taken branch latency or else a longer cycle time. The NFA miss cost may be reduced for a given sized cache at the expense of increased area if there are less sharing between instructions. For example, a design with one NFA field per instruction would have 0.05 and 0.08 lower CPI costs for gcc92 and SPEC92Int, and 0.07 and 0.08 lower CPI costs for gcc95 and SPEC95Int than the UltraSPARC design point respectively.

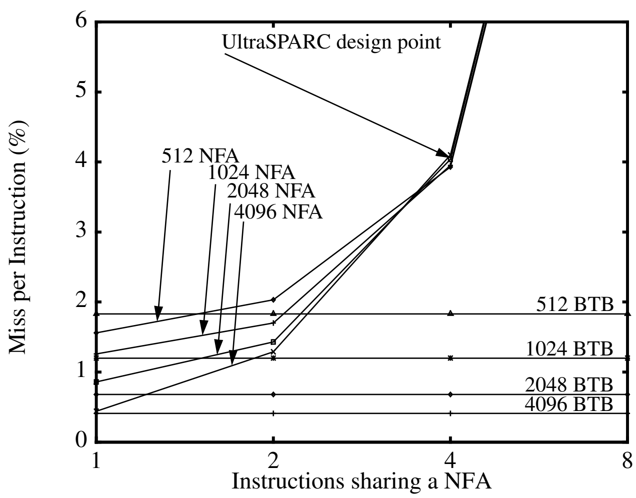

Figure 9. NFA and BTB miss rates (gcc92).

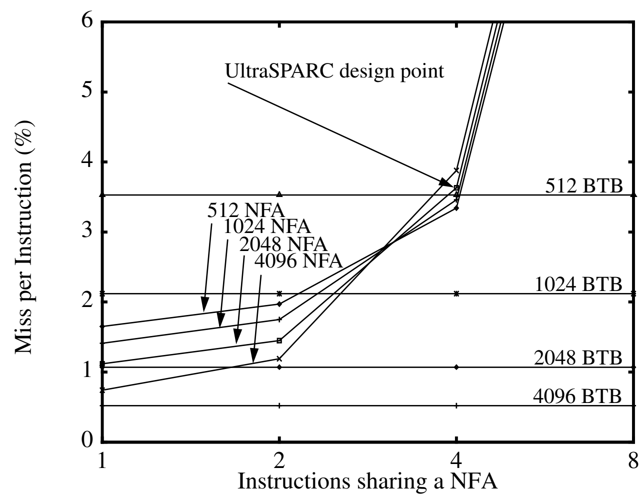

Figure 11. NFA and BTB miss rates (gcc95).

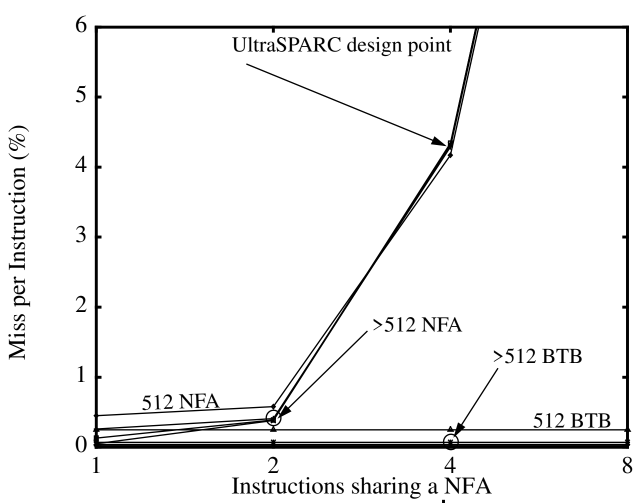

Figure 10. NFA and BTB miss rates (SPEC92Int).

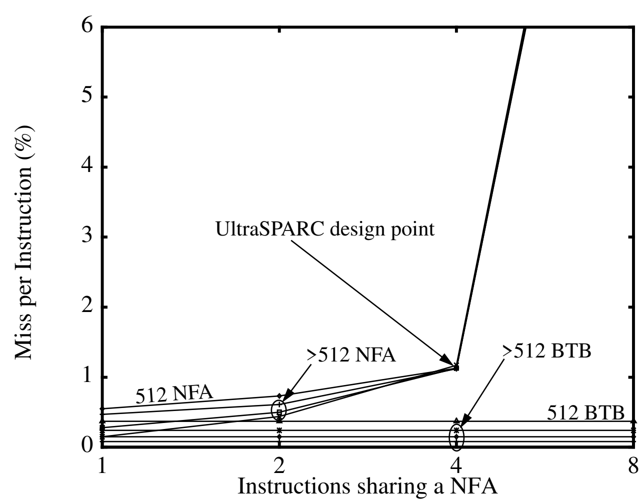

Figure 12. NFA and BTB miss rates (SPEC95Int).

num of predictors	BTB	NFA+ret_stack				BTB	NFA+ret_stack			
		1 instr/ nfa	2 instr/ nfa	4-instr/ nfa	8 instr/ nfa		1 instr/ nfa	2 instr/ nfa	4-instr/ nfa	8 instr/ nfa
	gcc92					SPEC92Int				
512	0.19	0.05	0.06	0.10	0.30	0.14	0.02	0.03	0.10	0.33
1024	0.16	0.05	0.06	**0.10**	0.30	0.13	0.02	0.02	**0.10**	0.33
2048	0.14	0.04	0.05	0.10	0.30	0.12	0.01	0.02	0.10	0.33
4096	0.13	0.03	0.05	0.10	0.30	0.12	0.01	0.02	0.10	0.33
	gcc95					SPEC95Int				
512	0.17	0.05	0.06	0.09	0.29	0.14	0.04	0.04	0.06	0.29
1024	0.16	0.05	0.06	**0.09**	0.29	0.14	0.04	0.04	**0.06**	0.29
2048	0.16	0.04	0.05	0.09	0.30	0.14	0.03	0.04	0.06	0.30
4096	0.16	0.04	0.05	0.10	0.31	0.14	0.03	0.04	0.06	0.30

Table 8: CPI costs for NFA and BTB misses. There are 1.8%, 1.3%, 1.8%, 1.8% subroutine returns per instruction in gcc92, SPEC92Int, gcc95 and SPEC95Int respectively. The return stack hit rates are 92%, 95% 93%, 94% for gcc92, SPEC92Int, gcc95, and SPEC95Int respectively.

6. Area Estimates

To evaluate the merits of architectural ideas in a processor design, it is necessary to analyze the cycle time, cycle-per-instruction and area impacts together. In the early stage of the UltraSPARC design, area estimates for many cache and branch prediction organizations were obtained from the formula, *area = num_SRAM_bits * um²_per_bit / efficiency + random logic*. A 6-transistor SRAM cell, based on the Texas Instrument 0.5um CMOS process used for UltraSPARC fabrication, is 63 um^2. Efficiency factors for SRAM arrays are provided to account for overhead such as decoders, word line drivers, sense amps, comparators and other control logic. For example, excluding the Iclass field in Table 2, there are 162048 6-transistor SRAM cells in the UltraSPARC 16KB, predictive 2-way in-cache prediction design. The estimated area for the SRAM array is 10.21 mm^2; total area estimates including overhead for the entire ICP design is 21.27 mm^2, which has an efficiency factor of 48%. The actual area measured from the layout is 22.3 mm^2. The high correlation between the estimate and the layout shows that the above formula provides an accurate first order estimation of area cost.

Table 9 illustrates area estimates for various possible PDU organizations of the UltraSPARC design, assuming a 41-bit physical address and a 32-byte cache block size. The UltraSPARC design point is highlighted in Table 9. The area required for the UltraSPARC PDU design, the predictive 2-way ICP, and its alternative are estimated to be 21.27 mm^2 and 32.79 mm^2 respectively, or a 54% saving for the ICP design. Most of the savings in the in-cache prediction design are due to the sharing of existing decoder, word lines and tags in the cache array made possible by the incorporation of the prediction information in the instruction cache. Because the contents in a BHT and a BTB are predictions, it may be possible to eliminate their tags, at an expense of higher CPI costs, to save area.

cache size	branch history entries	branch target entries	Direct-mapped+BHT+BTB		Predictive-2-way ICP	
			# bits	mm^2	# bits	mm^2
2KB	256	128	31,424	4.36	20,064	2.81
4KB	512	256	62,208	8.55	40,256	5.51
8KB	1024	512	123,136	16.74	80,768	10.83
16KB	2048	1024	243,712	32.79	162,048	21.27
32KB	4096	2048	482,304	64.23	325,120	41.80
64KB	8192	4096	954,368	125.82	652,288	82.19

Table 9: Area estimates for direct-mapped, BHT and BTB vs. predictive 2-way ICP.

7. Conclusions and future work

In the UltraSPARC prefetch and dispatch unit (PDU) design, we examined a technique that combined two prediction methods: predictive set-associative cache and in-cache prediction. This combination was compared with alternative designs such as direct-mapped and set-associative caches, and branch history table and branch target buffer. Two possible organizations for the UltraSPARC PDU are shown in Table 10. They were chosen for three reasons. They both met the cycle time, area budget and were the highest performance techniques known to the UltraSPARC design team in 1991. We chose the In-cache Prediction (ICP) design for its fast cycle time, lower cycles-per-instruction (CPI) and lower area costs.

A predictive set-associative cache was a better choice than a direct-mapped or a set-associative cache for the high clock rate design of UltraSPARC. A Predictive Set Associative Cache (PSAC) is more effective when cache misses are dominated by conflict misses. The sharing of set predictors in a PSAC should be avoided for any size cache. For example, the UltraSPARC design could have a lower CPI if there were one set predictor (SP) for each instruction. This change would have negligible increase in area and would not impact cycle time. We would have incorporated this change if we had learned about its impact early in the UltraSPARC design.

Branch prediction is the focus of much micro-architectural research today and most of the efforts are concentrated on improving branch history prediction accuracy, as in [15]. This alone is not adequate in optimizing a PDU design. Other factors that may impact PDU performance are branch target prediction, taken branch latency, misprediction costs, and most importantly, cycle time of the design. We chose the in-cache prediction technique to perform branch prediction in the UltraSPARC design. This technique offers high accuracy in history and target address prediction, no branch following latency and fast cycle time. Moreover, in-cache prediction can directly support multi-branch following per cycle with no cycle time or area impact. In-cache prediction is sensitive to sharing misses when a next fetch address is shared between two or more instructions. The UltraSPARC design would have lower CPI, at the expense of increased area, if additional NFA fields were added to minimize sharing misses.

Results in this paper show that interactions between cache size, sharing of predictors and update policy are complex. Further studies, in particular with a larger set of programs, are needed to better understand their interactions, to improve prediction accuracy and to reduce miss penalty. It will also be an interesting and possibly rewarding exercise to apply multi-level and collating schemes to the integrated prediction techniques.

Advances in process technology offer more transistors on an integrated circuit and enable higher clock speed. As the transistor budget increases, a microprocessor may incorporate a second level cache on the same die. To meet the faster cycle time, the first level cache size may be reduced. A smaller cache size has negative impact on both the PSAC and the ICP because of higher CPI costs due to first use misses. One way to minimize the impact is to save the prediction information in the next level cache when a cache line is victimized. If the victim is needed in the future, the saved information can be used to initialize the cache line and minimize first use misses. This organization deserves further investigation.

cache size	branch history entries	branch target entries	direct-mapped cache, BHT, BTB, 4-entry return stack					ICP predictive 2-way cache, 2 instr/PRED, 4 instr/NFA, 4 instr/SP (single predictor), 4-entry return stack				
			mm^2	CPI cost				mm^2	CPI cost			
				gcc92	SPEC92Int	gcc95	SPEC95Int		gcc92	SPEC92Int	gcc95	SPEC95Int
8KB	1024	512	16.74	0.63	0.30	0.84	0.34	10.83	0.52	0.24	0.77	0.21
16KB	2048	1024	32.79	0.45	0.25	0.62	0.26	**21.27**	**0.36**	**0.22**	**0.55**	**0.18**
32KB	4096	2048	64.23	0.34	0.22	0.44	0.24	41.80	0.28	0.22	0.37	0.15

Table 10: Area estimates and CPI costs for ICP and an alternative.

8. Acknowledgments

There is a popular quote that says "Failure is an orphan...but success has many fathers." The UltraSPARC is a success by any industry standard. If this paper turns out to be useful to the readers, it will be due to the vision and support of many. First, Dave Patterson and Neil Wilhelm were strongly supportive of this work. They encouraged the author to tell the what, how and why, and what we would have done differently if we had the present knowledge in 1991. Hindsight is useful if we learn from it.

The author acknowledges the contributions of Kit Tam, Alfred Yeung, and Bill Joy in defining the UltraSPARC instruction fetch architecture. When we started the UltraSPARC program, popular opinion was that it could not be done and let alone be on time. Special thanks go to the UltraSPARC-Spitfire-team whose hard work and dedication not only brought us the UltraSPARC microprocessor, but disproved the myth.

The author thanks his colleagues, Neil Wilhelm, John Ousterhout, Ivan Sutherland, Charles Molnar, Tom McWilliams, Barton Sano, Brent Welch, Jon Bertoni, James Rose, Jon Lexau, Bill Coates, C.C. Lee and Bob Coldwell, and the anonymous reviewers, for reading and commenting on early drafts of this paper. Special thanks go to Dave Patterson who has done an unenviable job of reading and commenting on many early drafts.

Finally, the author would like to thank Sun Microsystems Laboratories and Sun Microelectronics for their support.

9. Bibliography

[1] *Microprocessor Report* (Sept. 12, 1994 - Mar 6, 1995)

[2] SPEC CINT92, Release V1.1, Dec. 1992

[3] SPEC CINT95, Release V1.0, Sept. 1995

[4] R10000 Microprocessor User's Manual, Release v1.1, Jan. 22, 1996

[5] A. Agarwal and S.D. Dydar, "A Technique for Reducing the Miss Rate of Direct-mapped Caches", Proceeding of the 20th International Symposium on Computer Architecture, May 1993

[6] B. Calder and D. Grunwald, "Next Cache Line and Set Prediction", Proceeding of the 22nd International Symposium on Computer Architecture, May 1995

[7] R. Cmelik R. and D. Keppel, "Shade: A Fast Instruction-Set Simulator for Execution Profiling", Sun Microsystems Laboratories Technical Report TR-93-12

[8] D. Greenley et al, "UltraSPARC: The Next Generation Superscalar 64-bit SPARC", The Proceedings of COMPCON-95, March 1995

[9] J. Lee, A. Smith, "Branch Prediction Strategies and Branch Target Buffer Designs", *IEEE Computer,* 17:6-22, January 1984

[10] M. Hill, "A Case for Direct-mapped Caches", *Computer 21*, 12 (Dec 1988)

[11] M. Johnson, "Superscalar Microprocessor Design", Prentice Hall. Inc., Englewood Cliffs, NJ

[12] N. Patkar et al, "Microarchitecture of HAL's CPU", The Proceedings of COMPCON-95, March 1995

[13] S. Sechrest, C. Lee, T. Mudge, "Correlation and Aliasing in Dynamic Branch Predictors", To appear in the Proceeding of the 23nd International Symposium on Computer Architecture, May 1996

[14] A. Smith, "Cache Memories", *ACM Computer Survey 14*, 3 (Sept. 1982), 473-530.

[15] T-Y Yeh, and Y. Patt," A Comparison of Dynamic Branch Predictors that use Two Levels of Branch History", Proceedings of the 20th International Symposium on Computer Architecture, 257-266, May 1993.

[16] R. Yung, A. Yeung, M. Maturana, *Spitfire_per* simulator, Sun Microsystems Inc, Apr. 1991.

[17] R. Yung, "In-Cache Prediction: Instruction Cache And PDU Design To Shorten Sequential and Target Fetch Delay", Technical Memo, Sun Microsystems, Inc., Oct. 23 1991

[18] R. Yung, "Instruction Dependencies of the SPEC92 Integer Programs", Sun Microsystems Laboratories Technical Report #94-0059, March 1994

[19] R. Yung, A. Yeung, K. Tam, W. Joy, "Rapid Instruction Prefetching and Dispatching Using Prior Prefetching Predictive Annotations", U.S. Patent, Jun. 1992

[20] R. Yung, "Rapid Data Retrieval From Data Storage Structures Using Prior Access Predictive Annotations", U.S. Patent, Oct. 1994

[21] R. Yung, *Cachesim7* and *Pred1* simulators, Sun Microsystems Laboratories, Nov. 1995.

[22] R. Yung, "Design of the UltraSPARC's Instruction Fetch Unit", Sun Microsystems Laboratories Technical Report, In Preparation.