A Fast and Flexible Performance Simulator for Micro-Architecture Trade-off Analysis on UltraSPARC™-I

Marc Tremblay, Guillermo Maturana, Atsushi Inoue and Les Kohn

SPARC Technology, Sun Microsystems, Inc.
2550 Garcia Avenue, Mountain View, CA 94043

Abstract - Over one hundred micro-architecture features were analyzed and simulated in order to determine if they should be included in UltraSPARC-I. A fast and flexible performance simulator was developed in order to model these features. In this paper, we describe UPS (UltraSPARC-I Performance Simulator), and show how it was used to do trade-off analysis.

I. Introduction and Motivation

Advances in process technology have resulted in the availability of a large transistor budget for the current generation of superscalar RISC processors. UltraSPARC-I, a 64-bit 4-way superscalar processor developed by Sun's SPARC Technology, makes use of over five million transistors in order to deliver performance three to five times faster than the previous generation[1][2][3][4].

The large number of transistors available allows many of the latest micro-architecture advances to be implemented on-chip. On UltraSPARC-I, there are over 100 micro-architecture features influencing the performance of the processor. Each single feature must be analyzed carefully so that its inclusion is quantitatively justified. The gain in performance must be weighed against:

- cycle time
- die size
- design time

The complexity of these features as well as their inter-dependency makes it challenging for architects to evaluate their effectiveness. In most cases, an accurate simulation of a feature in the context of the whole processor is the only way to quantify its impact. Traditionally, some features, for example, the size of internal caches, could be simulated on a simple cache simulator and results could be applied to a simple analytical performance model. That is no longer the case for high performance processors such as UltraSPARC-I. Non-blocking caches mandate that caches be simulated along with the rest of the processor since the miss latency may possibly be covered (at least partially) by executing other independent instructions. Also, because of the inter-dependency between the various features, it becomes important to be able to "tweak" a few features simultaneously, so that the global impact can be measured.

This paper describes the methodology that we used for performance trade-off analysis for UltraSPARC-I. We first briefly describe the architecture of UltraSPARC-I in Section II. The performance simulator is covered in Section III.and Section IV. Examples of architecture trade-off analysis are described in Section V.

II. Overview of UltraSPARC-I

UltraSPARC-I is a high-performance, highly integrated superscalar processor implementing the SPARC V9 64-bit RISC architecture[5]. UltraSPARC-I can execute four instructions per cycle even in the presence of conditional branches and cache misses. This is mainly due to the de-coupled aspect of the units feeding instructions and data to the rest of the pipeline. Instructions predicted to be executed are issued in program order to multiple functional units, execute in parallel, and can complete out of order.

The instruction set includes several graphics instructions that provide the most common operations related to two-dimensional image processing, two and three-dimensional graphics, and image compression algorithms.

UltraSPARC-I is decomposed into 8 functional blocks (Figure 1): Prefetch and Dispatch Unit (PDU), Integer Execution Unit (IEU), Floating-point/Graphics Unit (FGU), Load/Store Unit (LSU), Instruction and Data Memory Management Units (IMMU and DMMU), External Cache Unit (ECU), and the System Interface Unit (SIU). An UltraSPARC-I module typically contains the processor, some external cache RAMs, and two custom data buffer chips.

The PDU presents a stream of instructions to the IEU and FGU through a 12 entry instruction buffer. Instructions are fetched from a 2-way pseudo set associative 16K instruction cache in a pipelined manner. Dynamic branch prediction allows the PDU to efficiently fetch instructions across conditional branches. Misses in the instruction cache are forwarded to the ECU.

32nd ACM/IEEE Design Automation Conference ®

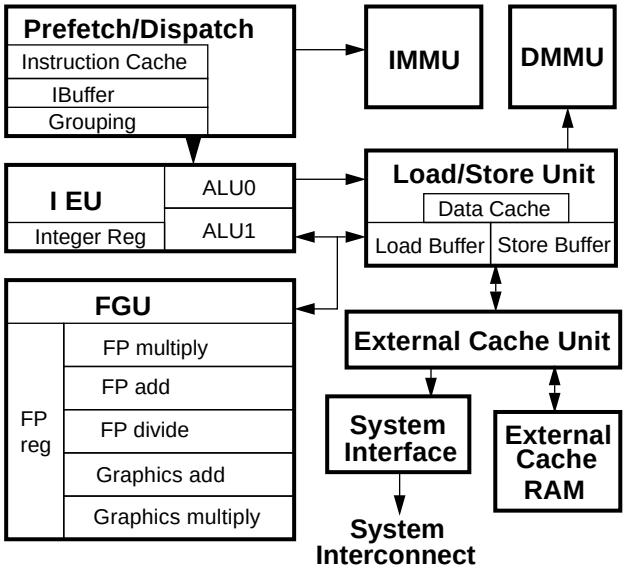

Figure 1: UltraSPARC-I Block diagram

The IEU groups and dispatches up to four instructions from the instruction buffer. The IEU features two ALU pipelines, one load/store pipeline, one branch pipeline, and a windowed register file. A non-pipelined early-out multi-cycle integer multiplier and integer divider are also part of the IEU. Floating-point and Graphic instructions are sent to the FGU and load/store instructions are sent to the LSU to be executed. The IMMU's and the DMMU's primary task is to provide a physical address from the virtual address provided by the Program Counter (PC) and the virtual address adder respectively.

The FGU contains three floating-point units, 2 graphics units, a completion unit, and a register file. All floating-point and graphics instructions [6] (except for divide and square root), have a one to three cycle latency and are fully pipelined, allowing the issuance of two floating-point graphics instructions per cycle. The completion unit allows divide and square root instructions to complete out of order with respect to other floating-point and graphics instructions.

The LSU contains an 8 entry store buffer, and 9 entry load buffer, and a direct mapped write-through 16K data cache. The store and load buffers provide for non-blocking stores and loads. They complete out of order with respect to other instructions in the execution pipelines. A load miss doesn't stall the execution pipelines until a dependent instruction is dispatched. Memory references that miss in the data cache are forwarded to the ECU.

The ECU contains a 2nd-level cache controller supporting 512K to 4Meg of external cache and a bus interface unit. It supports fully pipelined, single cycle accesses to the external cache. A MOESI (modified, owned, exclusive, shared, invalid) protocol is used to maintain coherency across the system.

III. UltraSPARC-I Performance Simulator (UPS)

The UltraSPARC-I Performance Simulator (UPS) consists of approximately 45,000 lines of C code. It is a trace driven simulator which takes a "shade" [7] trace as input and generates several files based on the processor model being simulated. Among others, the following files are generated:

- Statistics file: all statistics related to micro-architecture features are gathered in this file.
- Data references trace file: all activity related to loads, stores, writebacks, and snoops is traced in this file.
- Execution trace file: the progress of all instructions is graphically represented in this file. For each cycle, it shows which instructions are dispatched and which instructions have reached subsequent stages in the pipeline.
- Instruction buffer file: all activity regarding instruction fetching, branch prediction, branch target prediction, instruction buffer utilization, etc. is traced in this file.

All "trace" files generate their output for a specified cycle range or for a specified function call. This effectively provides observability to hot regions that architects may want to analyze.

A buffer containing a list of sequentially executed instructions, the effective address of memory operations, and the direction taken by branches, forms the link between the trace generator and UPS. From this information, UPS can fully simulate a processor without generating all data information (e.g. there is no need to generate 64-bit multiplications/divides, etc.).

Several C modules partitioned along the same way as the block diagram in Figure 1, simulate each part of the processor and interact with each other through simple data structures. This partitioning allows several designers to work on the simulator in parallel.

UPS is able to simulate around 6200 instructions/sec on a 60 MHz SPARCStation 20. At this speed UPS is fast enough to simulate some of the SPEC92 benchmarks. For some of the larger benchmarks, such as 013.spice2g6, which execute more than 15 billion instructions, the simulation latency is in the order of a month. Since the main goal of UPS is to do trade-off analysis and make quick design decisions, such a long latency is unacceptable (notice that this latency may be fine for final performance prediction). For this reason we use a sampling methodology [8], which speeds up simulation by over 300 times, bringing the simulation time down to 2 1/2 hours on a single machine for the longest SPEC92 benchmarks. Samples can also be distributed across machines, speeding up simulation time even more. The same methodology was used for simulating real world applications such as database programs, Verilog, Synopsys, VCS from Chronologic, Hspice, and others.

IV. Events measured

In order to be effective, the performance simulator must report numerous statistics enabling the measurement of the impact of each feature. UPS keeps track of the following events (among others):

- All possible combination of data and control dependences (e.g. add-to-shift, load-to-fmul, etc.)
- All resource conflicts (e.g. running out-of-ALUs or having more than 2 floating-point instructions to dispatch when only 2 FP units are available, etc.)
- All cache effects. This includes stalls due to instruction cache misses, data cache misses, external cache misses. All important cache statistics are monitored including writebacks and displaced blocks.
- All TLB effects, including the average number of cycles spent in the TLB miss handler.
- Branch prediction statistics.
- Load buffer statistics.
- Store buffer information, including the effectiveness of store compression.
- Dynamic instruction mix.
- External cache bus arbitration/utilization.

The statistics file is processed by a Perl script which combines information across samples and arranges it in a format appropriate for a spreadsheet. Graphical information is then generated from the statistics.

V. Architecture trade-off analysis

One of the main goals for UPS was to offer enough flexibility for architects to be able to evaluate hundreds of variations of the micro-architecture features. Examples of the parameters that can be specified at compile time are:

- Scalarity (width) of the superscalar processor
- Pipeline depth and bypass restrictions
- Functional units: how many and what kind (ALUs, shifter, loads, stores, FPadd, FPmul, FPdiv, FPsqrt, etc.)
- Cache organization: size, associativity, line size, writethrough, writeback, replacement algorithm, levels, latency, throughput, etc.
- TLB: size, associativity, page size
- Depth of store buffer, load buffer, instruction buffer
- Bus widths
- Main memory latency
- Floating-point operations: latency and throughput.

By adjusting these parameters and by comparing different runs of the simulator, micro-architecture trade-off analysis can be conducted effectively. We discuss some examples in the following subsections.

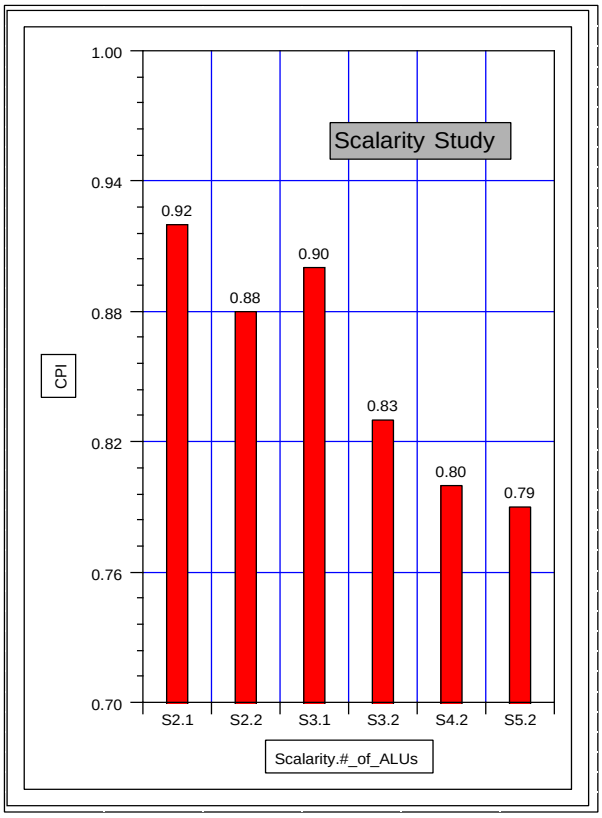

Figure 2. Impact of Scalarity

A. Scalarity Study

One of the main characteristics influencing performance is the scalarity of the processor. In Figure 2, we show an example on how the scalarity and the number of ALUs influence the SPECint92 CPI. Each column is labeled SX.Y where X is the scalarity and Y is the number of ALUs. Even though we used binaries compiled for a uni-scalar processor for this experiment, which do not fully take advantage of the additional functional units, the following conclusions can still be drawn from the graph:

- The combination (ALU, load/store, branch) is not common (S3.1 does not perform significantly better than S2.1).
- A 2-scalar processor with 2 ALUs performs better than a 3-scalar with 1 ALU.
- A second ALU for the 3-scalar machine buys 8% in performance.
- Going to 4-scalar buys around 4% while going to 5-scalar only shows 1.3% improvement.

It is important to keep in mind that the differences between each column in Figure 2 are significantly larger for code generated from a compiler which has knowledge of the target processor, specifically the number of functional units and their latency.

B. On-chip Caches

UltraSPARC-I has a 16Kbyte data cache and a 16Kbyte instruction cache on-chip. Process shrinks or smaller SRAM cells could allow larger caches to be implemented. In Figure 3, we show the impact of increasing each cache from 16K to 32K and 64K. From the graph we observe:

- Beyond 16K, increasing the instruction cache does not improve floating-point code much (0.2% better for 32K I-cache and 0.3% better for 64K). This code is typically very "loopy" with loops easily fitting in a 16K cache.
- Increasing the data cache size results in interesting performance improvements for both integer and fp benchmarks (e.g. 7.1% better for integer and 5.6% better for FP for an I-cache and d-cache of 64Kbytes).
- 64 Kbytes of total on-chip cache, more specifically a 32 Kbyte I-cache and a 32 Kbyte D-cache, performs as well as a 16 Kbyte I-cache plus a 64 Kbyte D-cache (80 Kbytes of total cache).

We ran these simulations using binaries compiled for a processor with blocking caches, so no attempt was made by the compiler to schedule dependences away from loads. Doing so would reduce the impact of having a larger D-cache. On the other hand, better compilers reduce overall CPI which increase the impact (as a percentage) of on-chip caches.

C. Floating-point divide and square root

The capability of setting the latency of each unit on UltraSPARC-I allowed us to determine the performance difference between a 3 and 4-bit per cycle algorithm. The 4-bit per cycle algorithm requires more levels of logic and could have slowed down the cycle time by at least 10%. If going to 3-bits per cycle cost less than 10%, then it is the better choice (especially since integer performance and database performance would also suffer from the lower clock frequency).

Figure 4 shows the impact of going to a 3-bit, 2-bit and 1-bit per cycle scheme (vs. 4-bit). Going to a 3-bit per cycle algorithm only reduces floating-point performance by 3% on average. The "Ora" (15.3%) and "Doduc" (8%) benchmarks are affected the most. These results influenced our decision to choose the 3-bit per cycle algorithm and not impact cycle time.

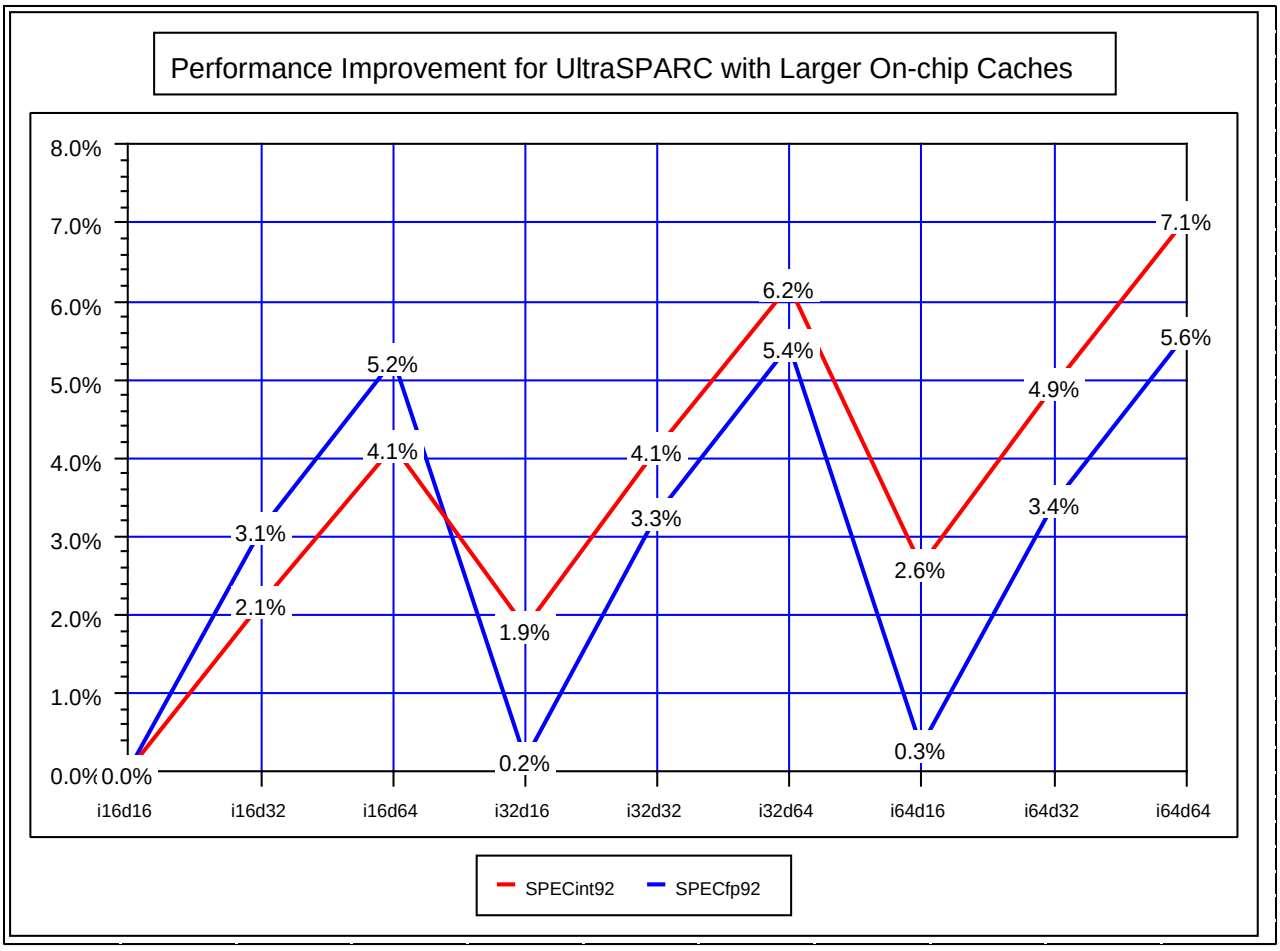

Figure 3. Impact of increasing on-chip cache sizes.

VI. Conclusions

The design of modern processors requires extensive analysis and sophisticated simulation techniques so that architects can allocate die area to features that benefit performance the most and optimize the CPI vs. clock rate trade-off. The simulator used for UltraSPARC-I (UPS) provided both fast turnaround time and the required flexibility for us to validate, in terms of performance, the impact of dozens of micro-architecture features on UltraSPARC-I.

The authors would like to thank Robert Yung, Shing Kong and many others from the UltraSPARC-I design team, who have contributed significantly to writing the code for UPS.

References

[1] D. Greenley, *et. al.*, "UltraSPARC: The Next Generation Superscalar 64 bit SPARC", 40th annual Compcon, 1995.

[2] Larry Yang, "System design methodology of UltraSPARC", 32nd Design Automation Conference Proceedings.

[3] James Gateley, et. al., "UltraSPARC I Emulation",2nd Design Automation Conference Proceedings (in press)

[4] Ariel Cao, et. al., "CAD Methodology for the Design of UltraSPARCMicroprocessor at Sun", 32nd Design Automation ConferenceProceedings.

[5] D. L. Weaver and T. Germond, "The SPARC Architecture Manual", Version 9, Prentice Hall, Englewood Cliffs, New Jersey, (1994).

[6] L. Kohn, *et. al.*,"The Visual Instruction Set (VIS) in UltraSPARC", 40th annual Compcon, 1995.

[7] Bob Cmelik and David Keppel, "Shade: A Fast Instruction-Set Simulator for Execution Profiling", Proc. ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems, May 1994, pp 128-137.

[8] Gary Lauterbach, "Accelerating Architectural Simulation by Parallel Execution of Trace Samples", Sun Microsystems Laboratories, Inc. Technical Report SMLI TR-93-22 (Dec 1993).

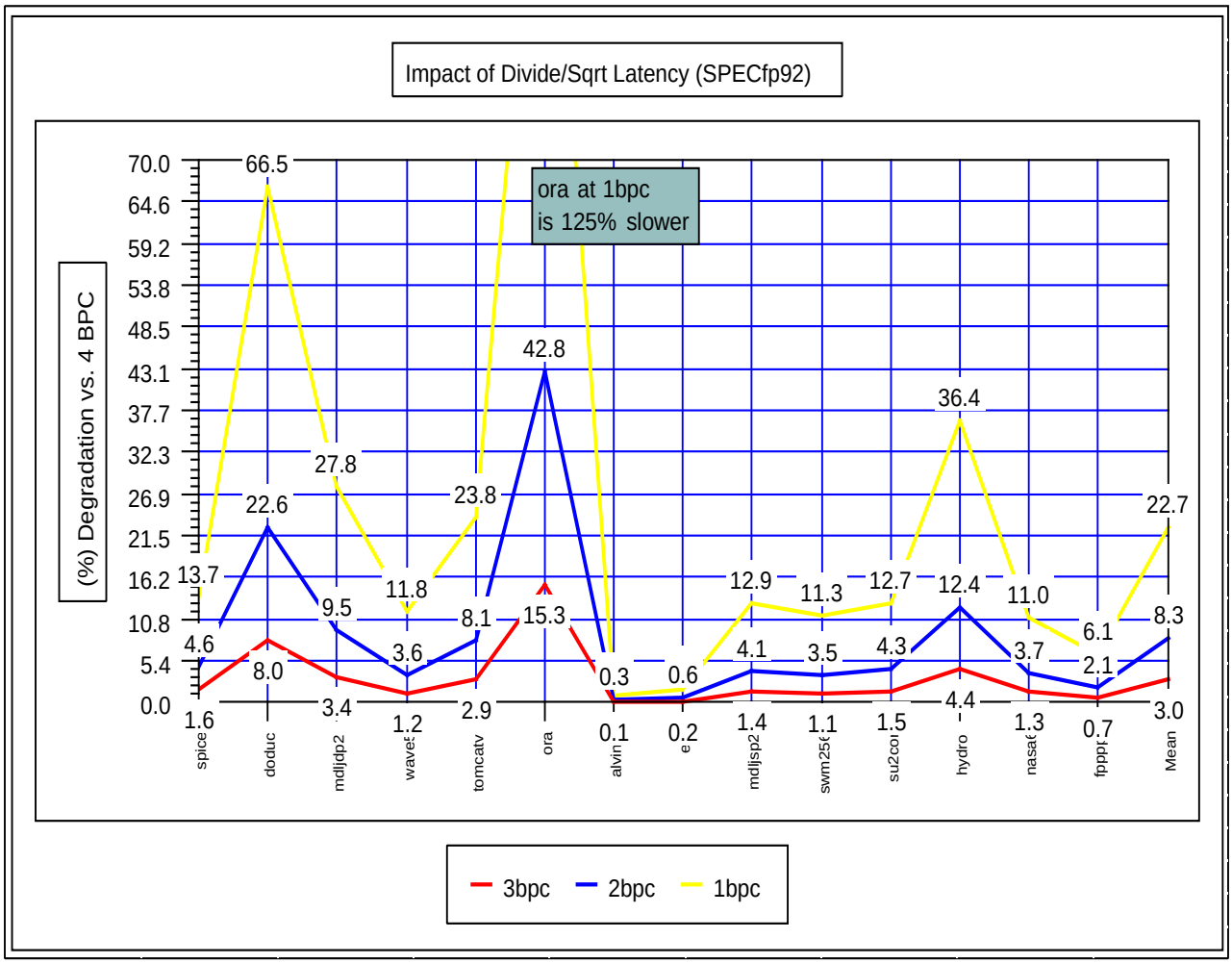

Figure 4. Impact of the floating-point divide/square root algorithm.