Microarchitecture and Performance Analysis of a SPARC-V9 Microprocessor for Enterprise Server Systems

Mariko Sakamoto[†], Akira Katsuno[†], Aiichiro Inoue[‡], Takeo Asakawa[‡], Haruhiko Ueno[‡], Kuniki Morita[‡], and Yasunori Kimura[†]

[†] *FUJITSU LABORATORIES LTD.*
{s.mariko, katsuno.akir-02, ykimura}@jp.fujitsu.com

[‡] *FUJITSU LIMITED*
{inoue.aiichiro, asakawa.takeo, ueno.haruhiko, morita.kuniki}@jp.fujitsu.com

Abstract

We developed a 1.3-GHz SPARC-V9 processor: the SPARC64 V. This processor is designed to address requirements for enterprise servers and high-performance computing. Processing speed under multi-user interactive workloads is very sensitive to system balance because of the large number of memory requests included. From many years of experience with such workloads in mainframe system developments, we give importance to design a well-balanced communication structure. To accomplish this task, a system-level performance study must begin at an early phase. Therefore we developed a performance model, which consists of a detailed processor model and detailed memory model, before hardware design was started. We updated it continuously. Once a logic simulator became available, we used it to verify the performance model for improving its accuracy. The model quite effectively enabled us to achieve performance goals and finish development quickly. This paper describes the SPARC64 V microarchitecture and performance analyses for hardware design.

1. Introduction

The enterprise server system (EPS) market had been dominated by mainframe servers for many years. The situation changed because of the explosive growth of information technology. These days, open server system vendors are targeting the EPS market, and this seems to be part of a growing trend in the market. The primary EPS operations are database processing and online transaction processing. They require computer systems that provide both high reliability and high performance. Mainframes have been known as systems that satisfy these requirements. However, with the widening use of Internet technology, demand from end users for non-stop processing on Internet servers is increasing. This demand led to development of open servers with a guaranteed availability of 99.999 percent. Mainframes have been also capable of achieving this high level of availability.

We have conducted many experiments as part of mainframe development [8]. Using our mainframe expertise while aiming for high reliability, high availability, and high performance, we developed a SPARC[1] V9 [9] processor that is called SPARC64 V. The processor is designed to address requirements for enterprise servers and high-performance computing. We designed it from scratch because we decided to make good use of mainframe thinking and needed to avoid inheriting any limitation. But then this disable to estimate final performance based on an existing machine.

The SPARC64 V operates at a frequency of 1.3 GHz and uses 0.13-µm CMOS technology. The first of these silicon processors was manufactured in December 2001.

The key features are throughput, multiprocessor performance and RAS (reliability, availability and serviceability.)

We use a software performance simulator (performance model) for the performance estimation and performance studies. We develop a performance model before hardware design is started and we use it until the end of the project. We continuously improve its rigidity concurrent with hardware development moreover we use verification results which from execution of performance test programs on logic simulator for improving the model accuracy. Using a single performance model throughout the project results in evaluation data consistency and the consistency is effective for architects when they verify model data. Result of these works, in the end of the project, the accuracy of the performance model can be a quite high.

---

[1] SPARC is a registered trademark of SPARC International, Inc.

Section 2 covers the hardware development strategy and performance model features. Section 3 describes the SPARC64 V microarchitecture. Section 4 explains some of our performance analyses on the performance model. Performance model accuracy is discussed in section 5.

2. Strategy

We devised a strategy to achieve performance goals and finish this development in a short and limited period. We realize it by a close use of such performance model and such steady efforts for accuracy improvement as we describe below.

- The performance model consists of detailed processor model and detailed memory system mode because we emphasize well-balanced communication structures on our hardware design.
- At an early stage, we carry the accuracy improvement out by continuous rigidity of the performance model. And once a logic simulator is available, we use it to verify the performance model for improving its accuracy, iteratively.

This aims that hardware architects can be confirm their decisions as needed during design development and fix performance bugs before an actual machine is built by estimating final system-level performance.

This section explains how the proposed strategy was executed. Figure 1 outlines development of the performance model and the hardware design. A close relationship between them is essential and the "Mutual feedback" arrow in Figure 1 indicates it. The horizontal axis is the time scale.

- Hardware development begins with a basic design phase, moves to a detailed design phase, and then moves to a verification phase.
- An initial performance model is available at the beginning of the basic design phase. We developed the model according to fundamental specifications.
- During hardware development, performance architects carry out performance studies, discuss the results with hardware architects, and update the performance model for subsequent studies. In parallel, hardware architects combine information from performance studies and from other examinations. They then update the hardware design accordingly. Experimental ideas are implemented in the model whenever necessary. In parallel with the studies, as hardware architects fix parts of the design, performance architects are informed about the choice and implement the specifications as precisely as necessary to improve performance model accuracy. The performance model is updated iteratively and continuously because the hardware design is improved step by step.
- At the end of the detailed design phase, a logic simulator that is based on the actual hardware logics available. First, it processes logic verifications. Next, it starts performance testing. Some of the performance test programs are generated from instruction traces by "Reverse Tracer [11]." Each of the original traces can be an input of the performance model. We know that individual execution results of each of these programs on the logic simulator is a detailed match of output from the performance model that inputs the original trace. Use of this feature can clarify bugs in the hardware logic and the performance model.

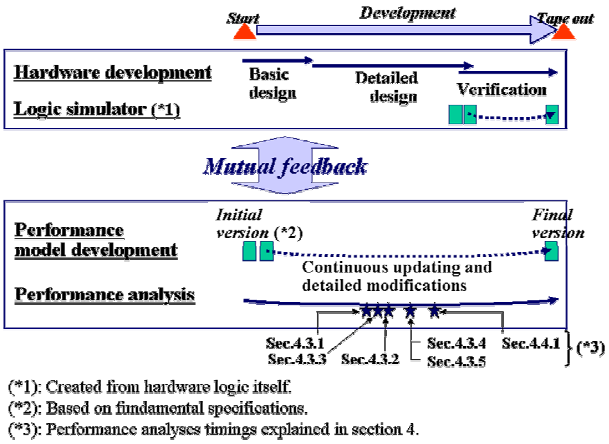

Figure 1. Development strategy.

Figure 2 illustrates accuracy improvements in the performance model. The horizontal axis is the time scale, and the vertical axis is an accuracy scale. The dotted line represents the performance of a physical machine, and we obtained this value after a physical machine is available.

- For verification of the initial performance model, we used a highly accurate performance model for a mainframe system that had been verified with a physical machine. We set up the initial model and the mainframe model with similar structures, and checked whether the models exhibit similar trends.
- The accuracy of the performance model is verified based on comparisons with physical machine performance. So we could not determine the actual accuracy of the model before a physical machine is built.
- However, in the later part of the verification phase, architects can estimate benchmark performance with the performance model because the model is verified based on comparisons with the logic simulator which is cycle accurate. It's not practical to estimate multiprocessor performance with the logic simulator. The performance model has advantages in flexibility, cost effectiveness, and portability.

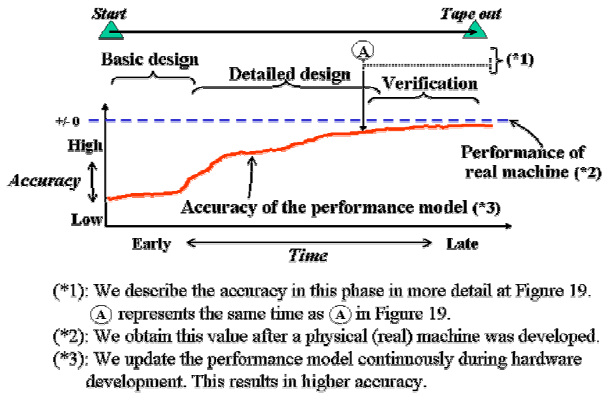

Figure 2. Accuracy improvements.

2.1. Performance model

The performance model was created based on actual hardware logic. It is a trace-driven software simulator written in C. Input traces are instruction traces, and some of them include information from both application execution and kernel execution. The source size is about ninety thousand steps. It has about five hundred parameters.

Requirements for the model are a capability to estimate system-level performance and multiprocessor system performance, flexibility for changing the model structure, and high accuracy.

The performance model consists of a detailed processor model and detailed memory system model. Generally, a performance model for performance evaluations consists of a detailed processor model and a rather rough memory system model, such as a latency model. But because of a large frequency gap between memory systems and processor cores, a system-level study on such a model would mislead hardware architects when they design an EPS.

From the beginning until the end of development, we improved each version of a *single* performance model step by step. Such continuity provided consistency of output from the performance model. In spite of the iterative improvements, features of different versions were closer than features of different tools. Once a logic simulator was available, we started to verify the performance model by comparing it with output from the logic simulator, and changed points found to be incorrect in the model. The verification was iterative.

The final version of the performance model has the following features. A processor can be modeled in detail at the register transfer level. Memory access resources between the CPU and memory, including buffers, queues, and pipelines, are the same as those in a physical machine. Moreover, a cache protocol used to access, read, write to, and invalidate an internal cache can be modeled. A bus network connecting chips between caches and memory, and data and request flows can be modeled in detail with the same concepts as those of actual systems. Such details include a request queue, bus conflict, bandwidth, and latency. In addition, requests between L2 caches can be modeled for MP system performance models.

When we set up the model with UP structure and run a multi-user interactive workload trace, it operates 7.8 K instructions per second on the Intel Pentium III processor which frequency is 1 GHz.

2.2. Evaluation and verification environment

Figure 3 illustrates the evaluation and verification environment.

- "Trace" is created on a physical machine. Using a workload running in a normal operating system environment, we wait until it reaches a steady state, and then start trace.
- Section 2.1 explains the "software performance model (performance model)."
- Output from the performance model ((1) in Figure 3) is used to determine hardware specifications.
- The "logic simulator" is created with hardware logic, so it is a truly accurate representation of the memory system and processor core design. We can verify the logic of both the memory system and microarchitecture.
- Test programs that can run on the logic simulator are roughly divided into two groups: logic test programs and performance test programs.
- Performance model accuracy is verified based on comparisons with output from the logic simulator ((2) in Figure 3). We execute a test program that is created from an instruction trace [11]. For the comparisons, we input the trace to the model.
- The final phase of development is a study of the full workload-driven performance on a developed machine. We evaluate performance model accuracy by comparing physical machine execution output with performance model output ((3) in Figure 3).

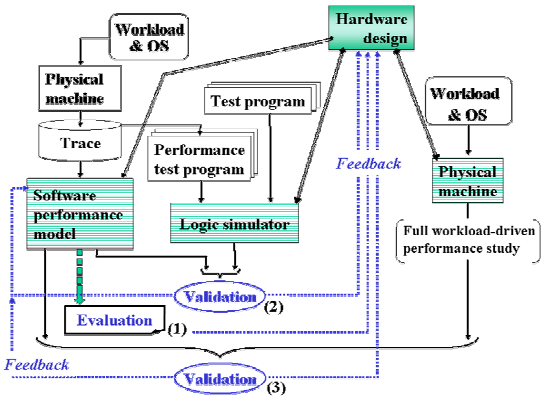

Figure 3. Evaluation and verification environment.

3. Microarchitecture

This section first provides an overview of the SPARC64 V microarchitecture. That is followed by detailed descriptions of techniques for improving throughput. Related basic studies on microarchitecture design are discussed in section 4.

The processor is a 64-bit microprocessor based on the SPARC-V9 architecture. Its operating frequency is 1.3 GHz, and it contains 191 million transistors fabricated using 0.13µm CMOS technology with eight-layer copper metallization. The die size is 18.1 by 16.0 mm.

The microarchitecture of the core has an out-of-order superscalar execution design. Up to four instructions can be issued per cycle. There are out-of-order resources, including a 64-entry instruction window, and renaming registers. Up to 32 floating-points and 32 integer results can be kept in the renaming registers. There are four kinds of reservation stations: RSA, RSE, RSF, and RSBR. Each RSA and RSBR consists of 10 entries of a buffer. RSA is for address generation operations, and RSBR is for branch operations. Each RSE and RSF consists of two sets of buffers; and eight entries are in each buffer. There are two sets of integer execution units, two sets of floating-point execution units, and two sets of address generation units. Having two set of floating-point multiply-add execution unit is effective for HPC performance. Accordingly, up to six instructions can be dispatched in a cycle. A unique execution unit is connected directly to each RSE and RSF.

The processor cache hierarchy consists of two levels of a non-blocking cache. There is a level-one (L1) instruction cache and L1 operand cache. Each type of L1 cache is a 128-KB, 2-way set associative cache. The instruction fetch width is 32 bytes, from which up to eight instructions can be fetched. The level-two (L2) cache is a unified, 2-MB, 4-way set associative, on-chip cache. We decided to install medium-size L1 caches and a large on-chip L2 cache in the processor to obtain a performance advantage in multiprocessor systems.

A 4-way branch history table is used for branch prediction, and it has 16K entries. To enable non-blocking memory access operations, the processor core has 16 entries of a load queue and 10 entries of a store queue. Table 1 itemizes these microarchitecture numbers.

Figure 4 is a high-level block diagram of the SPARC64 V. The processor consists of four units. Using information in the branch history table, the instruction control unit (I-unit) fetches up to eight instructions per cycle. It also decodes up to four instructions per cycle. The execution unit (E-unit) consists of out-of-order execution resources. Up to six instructions per cycle can be dispatched to execution units, and up to 64 instructions can be processed at a time. The storage unit (S-unit) consists of TLB and an L1 cache. The secondary-cache-and–external-access-unit (SX-unit) consists of an L2 cache and an external access unit, and it communicates with other processors and the memory system through system bus interface. We implement three techniques for improving throughput. A subsequent part of this section discusses these techniques.

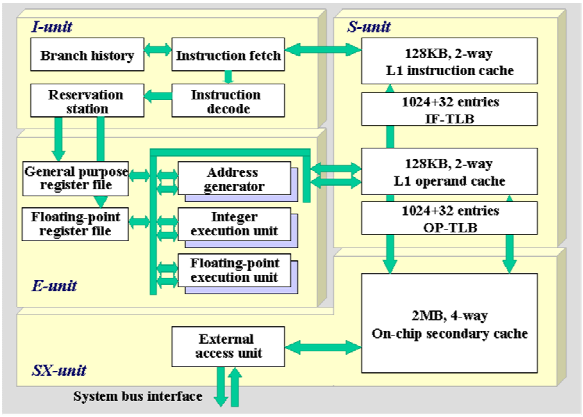

Figure 4. High-level block diagram of SPARC64 V.

Table 1. Microarchitecture.

Instruction set architecture	SPARC-V9
Clock rate	1.3 GHz
LSI technology	0.13 µm, 8 Cu layers
Transistor number	191 millions
Power consumption	57 W @ 1.3 GHz
Chip size	290mm^2 (18.1 by 16.0 mm)
LSI signal pin	269
Level 1 cache (I/D)	2-way, 128 KB
Level-2 cache	On-chip 4-way 2 MB
Processor core	
Execution control method	Out-of-order superscalar
Issue number	4-way
Instruction window	64 instructions
Instruction fetch width	32 bytes
Branch history table	4-way, 16K-entry
Execution unit	Fixed-point: 2
	Floating-point: 2 (Multiply-add)
	Address generator: 2
Reservation station	RSE: 16(8/8) for fixed-point
	RSF: 16(8/8) for floating-point
	RSA: 10 for address generator
	RSBR: 10 for branch
Reorder buffer	Fixed-point: 32
	Floating-point: 32
Load/Store queue	16/10 entries

3.1. Pipelines using speculative dispatch and data forwarding

The SPARC64 V is designed to support high-frequency CPUs and provide high throughput via several deep pipelines. There are 10 pipelines for six different purposes. Instruction fetch and instruction control have one pipeline each. Address generation, integer execution, and floating-point execution have two pipelines each.

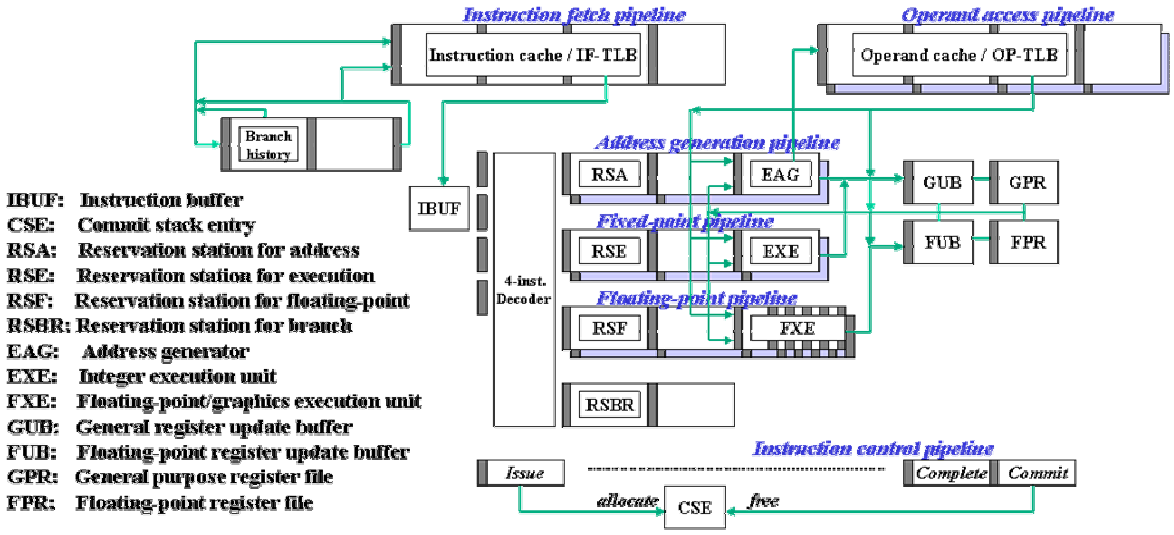

Figure 5. Pipelines and data flow.

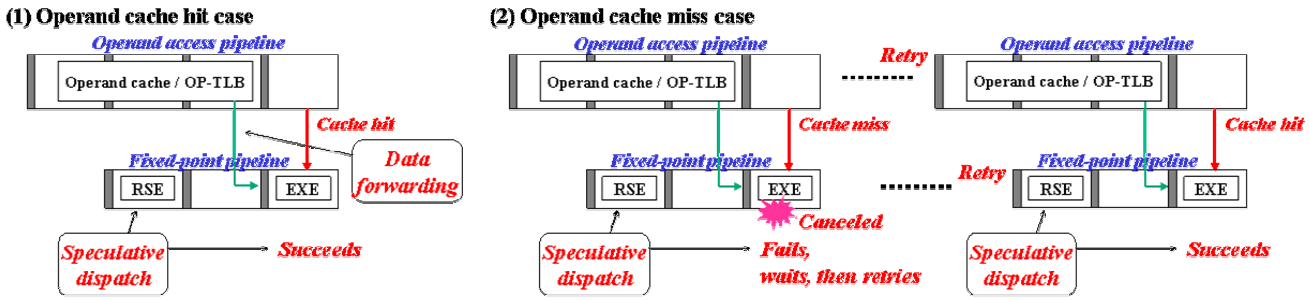

Figure 6. Speculative dispatch and data forwarding.

Figure5 shows data flow between units and pipelines. The instruction fetch pipeline has five stages; that is, one cycle to obtain priority, three cycles to fetch instructions from the L1 cache, and one cycle to validate a fetched value. The number of stages of an execution pipeline depends on the operation attribute.

The minimum is three stages. In the first stage, entries in a reservation station are examined, and only one instruction can be dispatched from a reservation station. Appropriate register data is read from a register file or renaming register in the second stage and results are then executed in the third stage.

The deeper the pipeline is, the higher the possibility of pipeline bubbles is. This might affect the throughput rate. The SPARC64 V dispatches instructions speculatively from reservation stations and thereby avoids the disadvantage of deep pipelines. We name this technique “speculative dispatch.” An instruction can be dispatched if source data of the instruction is not ready but would be ready before the dispatched instruction reaches the execution stage. The SPARC64 V predicts that a request of the prior instruction to the L1 operand cache should hit the cache line. It then speculates about the cycle number at which the corresponding data loaded from the L1 operand cache would be ready. Furthermore, a “data forwarding” technique accelerates the effect of speculative dispatch. Figure 6 illustrates speculative dispatch and data forwarding. There are data paths between each execution unit and every other execution unit, and also between every execution unit and operand access pipeline. They can usually make source data available in the next cycle after execution is completed or after the L1 operand cache returns it.

If the prediction were unsuccessful because a request causes a cache miss, all instructions that have read-after-write dependency must be cancelled at every stage of the execution pipelines.

3.2. Non-blocking dual operand access

The SPARC64 V can pass up to two requests between an operand access pipeline and the L1 operand cache.

This communication capability can be considered rather abundant in proportion to a 4-way issue processor. However, the processor is primarily targeting for database and transaction workloads, which generate more memory access requests than workloads for other purposes.

In an instruction decode stage, every memory access request is queued sequentially in a load queue or store queue (load/store queues). Concurrently, the request enters a reservation station (RSA). Up to two requests can be sent from RSA to address generation units (EAG) per cycle, and calculated memory addresses are then sent to specific load/store queue entries derived from the same instruction. Unless existing requests in the queues are passed to the L1 operand cache, the calculated addresses can be passed directly to the L1 cache.

The processor implements a non-blocking cache mechanism. The L1 operand cache is organized as eight banks, each of which is four bytes. Two requests can be accepted per cycle unless they cause a bank conflict. If they conflict, execution of a lower priority request is aborted and retried in a later cycle. However, a request that causes an L1 operand cache miss stays in load/store queues until its requested line become ready in the L1 cache.

3.3. Two level cache hierarchy

The cache hierarchy consists of a medium-capacity level-one (L1) copy-back cache and a large-capacity on-chip level-two (L2) cache. The cache hierarchy of many processors currently available on the market consists of three levels of cache: a small-capacity and fast-access level-zero (L0) cache, an L1 cache, and an L2 cache. In these processors, a store-through cache is a popular mechanism; it is also an effective method of operation for them. In spite of that trend, the SPARC64 V adopts a two-level cache hierarchy. Because (1) large-scale of interactive workloads would create a high rate of L0 cache misses, causing accesses to L1 cache and thereby deteriorating performance, and because (2) a deeper hierarchy increases the cost of move-out requests from other CPUs, two levels of cache are fine for the processor.

3.4. On-chip secondary cache with hardware prefetching

The SPARC64 V has a unified, 2-MB, 4-way set associative, on-chip L2 cache. We design the L2 cache to have high throughput and high reliability, while considering the trade-off between on-chip and off-chip designs. Such considerations include latency, cache size, and the number of set associativities. An on-chip design has an advantage in throughput because communication with an L1 cache is done within a chip. Since we emphasize high throughput and low latency, we choose an on-chip design. High reliability is achieved by using fewer physical parts and interface lines in the on-chip design.

The size, 2 MB, is a result of discussions about LSI technology, chip size, and manufacturing cost. We utilized a hardware prefetching technique to improve the cache-hit ratio, and determined the size accordingly. The hardware prefetch provides data in L2 cache for expected fetch requests in the near future. The prefetch is triggered by a L1 cache miss that is demanded by a memory request in a workload.

4. Performance analysis

This section contains information from some representative performance studies. We used the information to evaluate trade-offs in the SPARC64 V microarchitecture. Base model parameters of the studies are listed in Table 1. Workloads are SPEC CPU95, SPEC CPU2000, and TPC-C[2]. We use a uni-processor model to evaluate the SPEC and TPC-C benchmarks. Otherwise, we use an SMP model, and the results shown are for "TPC-C (16P)."

First, we design a data path structure for operation under specific frequencies. Next, we design cache structures that have a large influence on chip size. Then, we design detailed control logic. We show the timing of the studies in Figure 1.

4.1. Workload and trace generation

We used the SPEC CPU95 and SPEC CPU2000 benchmarks to study the processor and TPC-C workloads to study system-level behavior.

A Forte compiler and Shade, both of which are products from Sun Microsystems, Inc., were used to generate SPEC benchmark traces. The SPEC traces cover only application code.

We generated TPC-C workload traces by our original kernel tracer that was involved in kernel debugger and written by C and machine language. These traces cover both operating system code and transaction application code. We followed TPC guidelines during system setup in order to generate realistic traces and sampled these traces.

4.2. Benchmark characteristics

We characterize workloads based on a breakdown by execution time. We modeled a perfect L2 cache, a perfect L1 cache, perfect TLB, and perfect branch prediction, and

[2] TPC-C is a complex on-line transaction processing workload that is provided by Transaction Processing Performance Council.

then evaluate several models to find out the penalty of stalls. Figure 7 shows the results, where "sx" is stalls caused by L2 misses, "ibs/tlb" is stalls caused by L1 misses and TLB misses, "core" is execution time in the I-unit and the E-unit, and "branch" is stalls caused by branch prediction failures.

- SPECint95 spends 30 percent of execution time on stalls for branch miss prediction. This percentage is higher than that of SPECfp95 (3 percent). It agrees with the fact that SPECint95 includes more branch instructions than SPECfp95 as well as patterns that are difficult to predict.
- SPECfp95 spends 74 percent of execution time on execution in the processor core. This might reflect the depth of the floating-point execution pipeline.
- TPC-C has a large penalty, 35 percent of execution time, because of stalls caused by L2 misses. We determine that the L2 cache structure is a key to improving TPC-C performance.

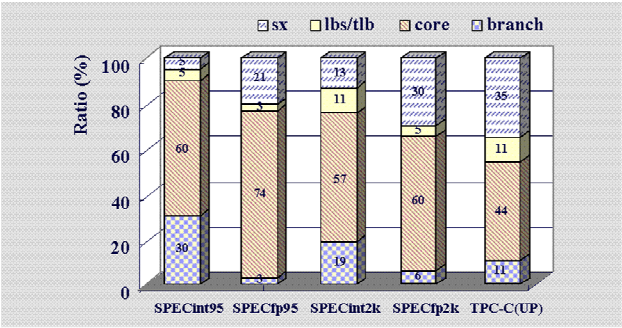

Figure 7. Benchmark characteristics.

4.3. Data paths

4.3.1. Superscalar. Out-of-order superscalar execution and deep pipelines are fundamental specifications of the SPARC64 V microarchitecture. We establish them as the base of our design. Next, we adopt "speculative dispatch and data forwarding" (see section 3.1) in the design. Lastly, we do performance studies to determine an appropriate instruction issue width because it is strongly associated with many processor design parameters.

Four-way instruction issue is the maximum setting that can be adopted in a trade-off between the fundamental specifications. We have to limit the complexity of the issue stage because of the high CPU frequency and because we do not want to divide the issue process into two stages.

Figure 8 shows the performance impact of 4-way issue execution compared with 2-way issue execution. These numbers are the IPC ratio. We infer that the 2-way issue width is too small, so it must be a performance bottleneck. SPECint95 and SPECint2000 improve performance more than the others do because they have high cache-hit ratios.

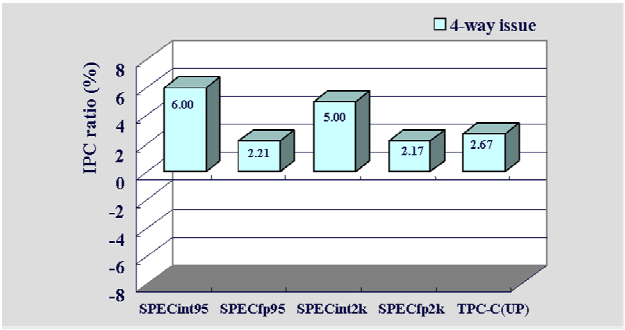

Figure 8. Issue width --- 4-way vs. 2-way.

Trade-offs between physical size and performance improvement are considered. The physical size of 4-way issue is more than twice that of 2-way issue. We noted that all earlier generations of SPARC processors accept 4-way issue. Our conclusion is that 4-way issue is better.

4.3.2. Branch prediction. Figure 7 shows that SPECint and TPC-C spend a lot of execution time on stalls for branch prediction failures. We therefore study performance by comparing two branch history tables. One is a 16K-entry, 4-way set associative, 2-cycle access table ("16k-4w.2t"), and the other is a 4K-entry, 2-way set associative, 1-cycle access table ("4k-2w.1t"). These two table structures are based on the values of fundamental properties such as RAM size and access latency. There are trade-offs between "16k-4w.2t" and "4k-2w.1t." "4k-2w.1t" has an advantage of fetch latency because it generates one bubble in a pipeline before it fetches a target instruction while "16k-4w.2t" generates two bubbles. In contrast, "16k-4w.2t" has an advantage of table size. "4k-2w.1t" would be useful for workloads that include high percentages of taken branch instructions where all of the taken branches have entries in a branch history table.

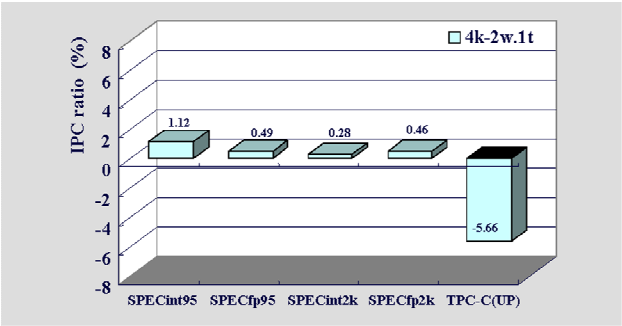

Figure 9. Branch history table --- latency vs. size.

Figure 9 shows the IPC ratio of "16k-4w.2t" to

"4k-2w.1t" as a percentage whose base is "16k-4w.2t." Figure 10 shows rates of branch prediction failure. SPEC benchmark programs benefit slightly from the advantage provided by "4k-2w.1t," and they have no difference between the prediction failures rates of "16k-4w.2t" and "4k-2w.1t." In contrast, TPC-C exhibits a different characteristic in its rates. Its prediction failure rate of "4k-2w.1t" is 60 percent greater than that of "16k-4w.2t." The "4k-2w.1t" IPC ratio shows a decrease of 5.6 percent. Since we place weight on improving TPC-C performance, we adopt the "16k-4w.2t" structure.

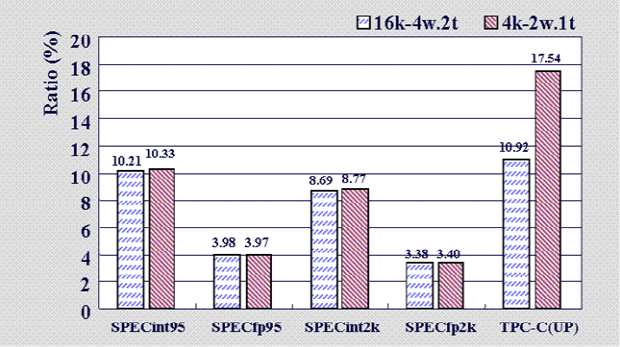

Figure 10. Branch prediction failures.

4.3.3. Level-one cache. We study the performance impact between a 32-KB, direct mapped, 3-cycle access L1 cache ("32k-1w.3c") and a 128-KB, 2-way set associative, 4-cycle access L1 cache ("128k-2w.4c".) Figure 11 shows the IPC ratio of "32k-1w.3cw.4c" to "128k-2w.4cw.3c" as a percentage whose base is "128k-2w.4c," and Figure 12 and Figure 13 show related cache miss ratios. For a TPC-C workload, the instruction miss rate of "32k-1w.3c" is 99 percent greater than that of "128k-2w.4c," and the operand miss rate of "32k-1w.3c" is 64 percent greater. This results in an IPC ratio decrease of 2.0 percent. A point of consideration is whether a two-percent advantage in the IPC ratio balances a fourfold increase in cache size. We conclude that it is worth having a larger L1 cache because the two percent does not provide the maximum possible benefits. We have fundamental understandings that EPS workload capacity would increase and L1 cache would become a bottleneck. Therefore, the number of benefits resulting from our TPC-C traces, which are a good reflection of real world behavior, can be considered a conservative number. Moreover, we predict that the direct mapped cache would easily cause thrashing during its processing of large workloads.

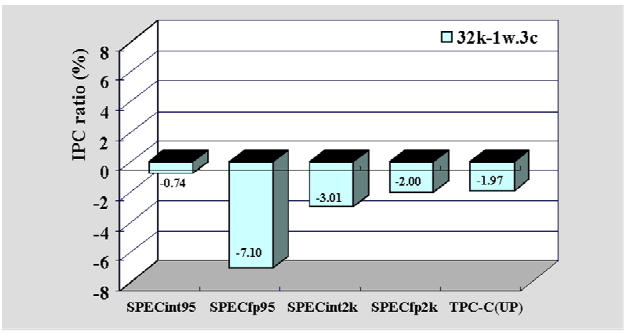

Figure 11. L1 cache --- latency vs. volume.

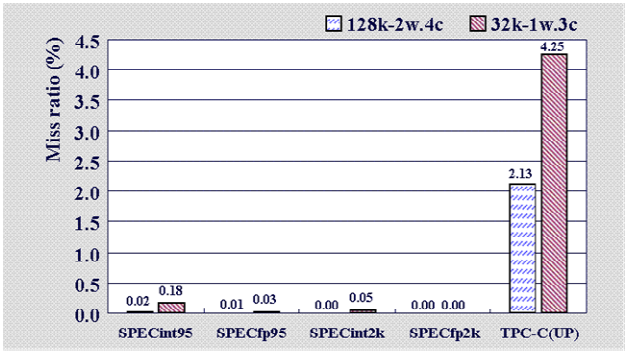

Figure 12. L1 instruction cache miss.

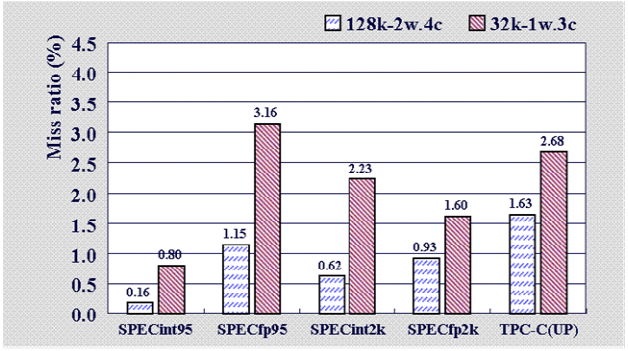

Figure 13. L1 operand cache miss.

4.3.4. On-chip level-two cache. We started design with the idea to use an off-chip 8-MB L2 cache because an earlier processor in the same series had an off-chip L2 cache. Then, the design changed to an on-chip design based on consideration of a speed gap between intra-chip communication and inter-chip communication. Since the processor being designed has a frequency of more than 1 GHz, the gap becomes unacceptable large. An MCM structure could reduce the gap, but is quite expensive.

We study on-chip L2 cache performance by comparing it with off-chip L2 cache performance. The on-chip L2 cache is a 2-MB, 4-way set associative cache

("on.2m-4w".) Due to limitations on chip size, 2 MB is the maximum size that can be adopted. The L2 off-chip models are of an 8-MB, 2-way set associative cache ("off.8m-2w") and an 8-MB, direct mapped cache ("off.8m-1w".) Access latency is derived from CPU frequency. We estimate the cost of communication between chips and we add 10τ to the latency of off-chip. The number of set associative is derived from the number of pins.

Figure 14 shows the IPC ratios among "on.2m-4w", "off.8m-2w" and "off.8m-1w". The ratios are expressed as percentages whose base is "on.2m-4w". TPC-C (16P) is a 16 SMP model. Figure 15 shows the L2 cache miss ratio. Compared with "on.2m-4w", "off.8m-1w" has IPC ratio decreases of 14 percent for TPC-C (UP) and 12.4 percent for TPC-C (16P). Thereby, we concluded that "off.8m-1w" offers no advantage over "on.2m-4w". In comparison, "off.8m-2w" has a slight IPC ratio increase.

Physical implementation of the on-chip cache of "on.2m-4w" requires about 270 signal pins. The off-chip cache of "off.8m-2w" requires about 560 signal pins. That is an approximate difference of two times, and it results in differences in signal switching noise level that cannot be ignored. "on.2m-4w" has an advantage of less noise. Moreover, since chip connections with signal wires have a higher chance of causing malfunction, "on.2m-4w" has an advantage in reliability.

After examining this trade-off, we adopt "on.2m-4w" in our design.

4.3.5. Hardware prefetching. Because we realized that an L2 cache size of 2 MB is too small, we needed to find some way to compensate for it. We started studying hardware prefetch algorithms. It's not preferable to increase the number of pipeline stages for the prefetching. We decided against using a buffer that stores data from a fetched line temporarily.

From our study of hardware prefetching (see section 3.3), Figure 16 shows the performance effect of prefetch compared with performance in a non-prefetch model. Results for SPECfp benchmarks indicate the IPC ratio improved by more than 13 percent. We find that our prefetch algorithm fits the chain access pattern of memory addresses. Figure 17 shows the L2 cache miss ratio. The left-side bar with the tag name of "*with*" is for a model with prefetching, and the ratio reflects all kinds of requests. The center bar with the tag name of "*with-Demand*" is for a prefetch model, but the ratio reflects only requests that are in the original workload; in other words, the ratio does not reflect prefetch requests. The right bar with the tag name of "*without*" is for a model without prefetching. We can determine the prefetch effect from a comparison between "*with-Demand*" and "*without*". Use of prefetch result in fewer cache misses. Differences between "*with*" and "*with-Demand*" indicate the negative effect of unnecessary prefetch requests. We find that FP programs receive a good degree of prefetch benefits.

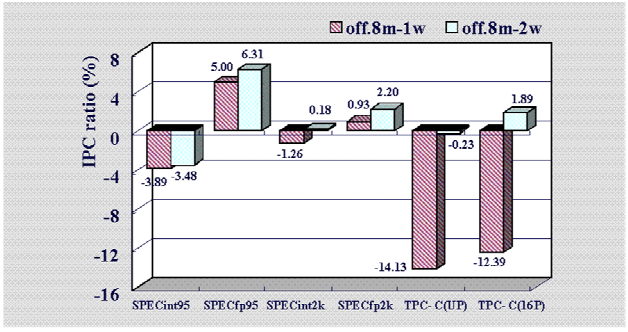

Figure 14. L2 cache --- latency vs. volume.

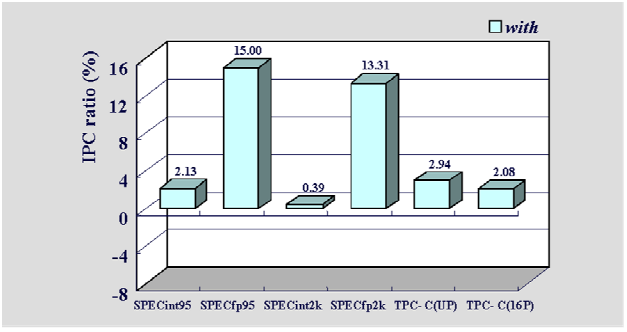

Figure 16. Hardware prefetching impact.

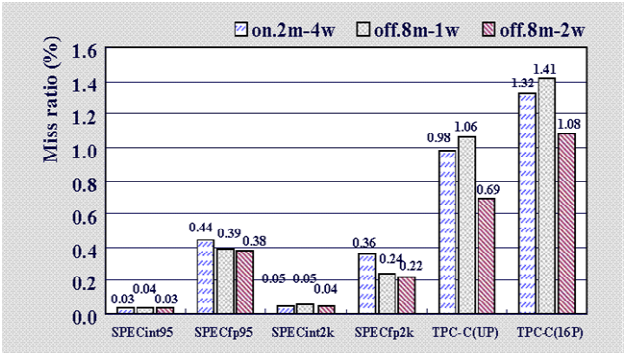

Figure 15. L2 cache miss.

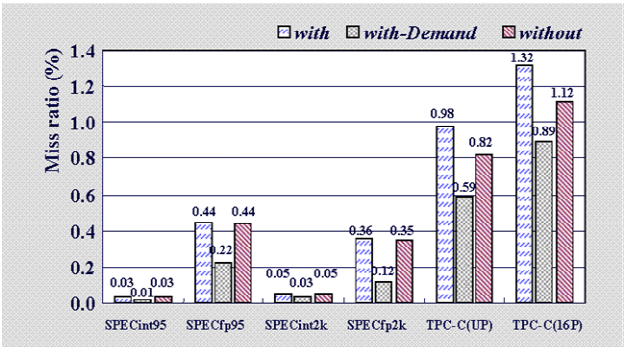

Figure 17. Hardware prefetching --- L2 cache miss.

4.4. Control logic

4.4.1. Reservation stations. To improve instruction execution parallelism, there are two sets of integer execution units and two sets of floating-point execution units. The SPARC64 V gains the greatest benefit from the dual execution units if it has one reservation station and dispatches up to two requests per cycle, because this would realize flexible out-of-order dispatch. However, it would increase implementation complexity compared with a structure where two reservation stations are connected to a unique execution unit and only one request can be dispatched respectively from each reservation station per cycle. To take full advantage of two dispatches from a single reservation station, we cannot set the two dispatch operations to run in parallel because the second selection may depend on the former operand. We must limit the complexity of the dispatch stage to keep consistency for high-frequency CPUs.

We study performance effects by comparing a structure having one reservation station ("1RS") with a structure having two reservation stations ("2RS"). "1RS" dispatches up to two operations per cycle from one reservation station. "2RS" dispatches up to one operation per cycle from a reservation station that is connected to a unique execution unit. Figure 18 shows the IPC ratio of "2RS" to "1RS" as a percentage whose base is "1RS." Use of "2RS" results in a slightly decrease IPC ratio. After examining the trade-off between IPC benefits and the complexity involved, we adopted "2RS" in our design.

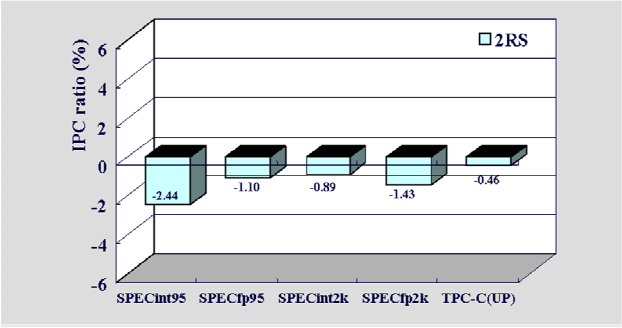

Figure 18. Reservation station --- 1RS vs. 2RS.

5. Performance model accuracy

The two graphs in Figure 19 show the accuracy of our performance model. During the verification phase, architects clarified and fixed logic bugs and performance bugs in their design ((2) in Figure 3). Concurrently, the accuracy of the performance model rose because hardware design was reflected more precisely on the model. The horizontal axis in the figure is a time scale. At "A" in the figure, performance tests began on the logic simulator. There was a time lag between memory system design completion and processor design completion. This is shown by A, B and C in Figure 19. We worked on processor validation mostly between A and B, and we worked on memory system validation mostly between B and C. Before B, we finished creating an accurate processor model. Before C, we updated the memory system model. The A in Figure 19 represents the same time as "A" in Figure 2.

Each graph has a different time scale, so we indicated the same time in the two graphs by using a dotted line to connect corresponding points.

The upper graph in Figure 19 shows estimated performance changes related to model rigidity. The horizontal axis of the figure is a time scale, and the vertical axis denotes the performance ratio. Labels such as "v1" and "v2" on the horizontal axis are version names of the performance model. We labeled the model to indicate major updates, so they do not represent equal time intervals. For each execution, we set up the model with the same parameters except for any parameters that were newly added before the execution, and the model used the same input traces generated from SPEC CPU2000 benchmark suits. We compared performance results based on "v8" performance values. Except for those of "v5," the performance estimates were always decreasing. There is no problem with this downward trend: a performance model generally has decreasing performance estimates as model rigidity improves. The exception at "v5" is the result of more-precise modeling of special instructions. Until "v4," we set an experimental penalty to each special instruction instead of modeling it in detail.

The lower graph in Figure 19 shows performance model accuracy based on a comparison with execution results on a physical machine. The horizontal axis of the figure is a time scale, and the vertical axis denotes the performance ratio. The time scale is uniform. The performance results reflect both a difference in the model version and a difference in the parameters. At the same time as hardware design parameters were being updated, we were reflecting the updated parameters to the performance model. Differing from lines in the upper graph, lines in the lower graph have many abrupt upward and downward changes. Many of these remarkable changes in tendency were the result of changes in memory system parameters such as memory access latency, bus width, and outstanding numbers. After C, we continued validation by changing the compiler optimization level. The final accuracy figure for SPECfp2000 is 3.9 percent, and it is 4.2 percent for SPECint2000.

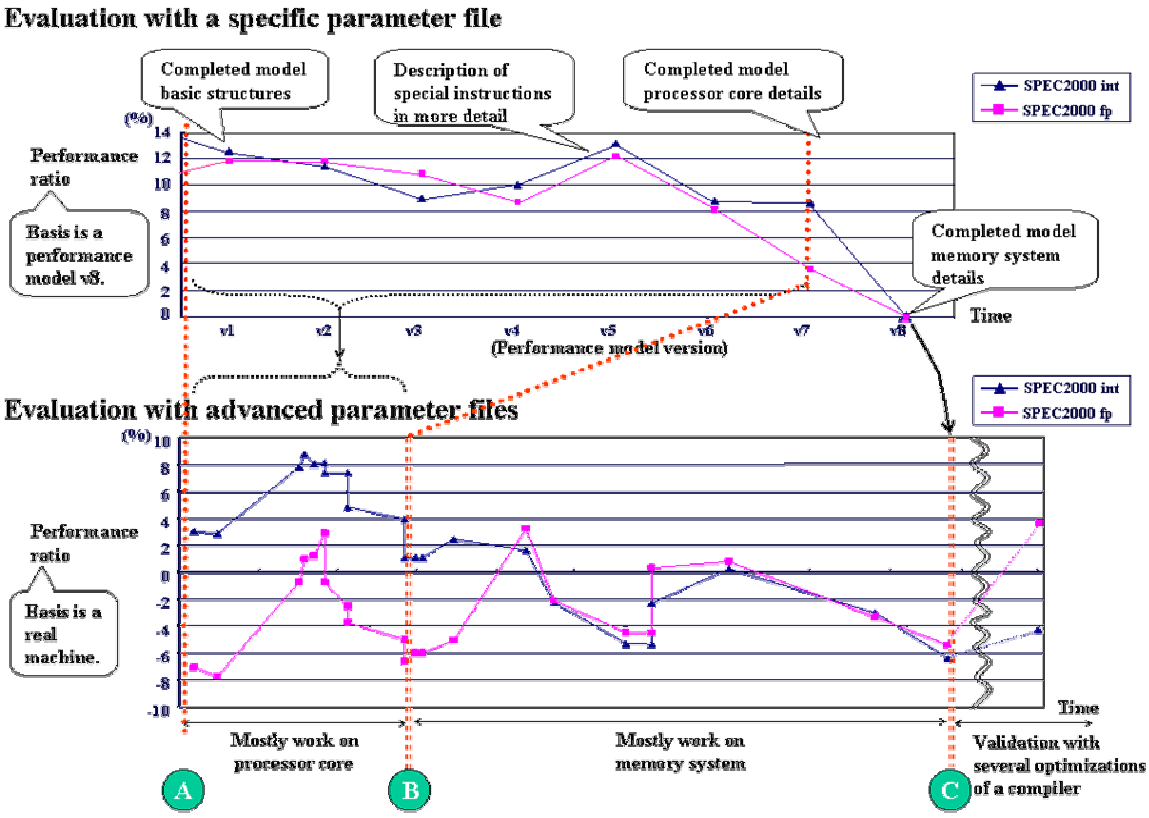

Figure 19. Performance model accuracy.

6. Related work

In this section, we discuss about related works that mention relations between processor development and performance evaluations.

For surviving marketplace competition, architects use performance evaluation models of a proposed processor to design new processor. Techniques to deal with problems in the performance evaluation are discussed in [12]. Performance works for commercial server design is focused in [13]. Design issues that relate to optimizing the performance of processor, I/O subsystems and software are described. In [14], a performance model that guides a processor design project is described. How Digital's architects did and what they learned related to their performance works are reported. In [15], a real system performance-tuning project across several generations of product line is described. Discussions are focused on system software tunings for large online transaction processing workloads at post-hardware phase.

There are no reports of performance models that consisted of detailed processor model and detailed memory system model, that were developed for using in a real processor development project, and that could be used to evaluate multiprocessor system.

7. Conclusion

We have developed a SPARC-V9 processor called SPARC64 V that is designed to address requirements for enterprise servers and high-performance computing. The processor operates at a frequency of 1.3 GHz and uses 0.13-µm CMOS technology. The first of these silicon processors was manufactured in December 2001.

The enterprise server system market had been dominated by mainframe servers for many years, and we conducted many experiments as part of mainframe development activities. Using our EPS expertise, we designed the processor from scratch since we recognized that the execution speed of multi-user interactive workloads is very sensitive to system balance; furthermore, we had to avoid any limitation that might be inherited from past designs.

Key features for addressing EPS requirements were throughput, MP performance, and RAS. Additionally, a key for HPC performance was instruction execution capability. To satisfy those requirements, our design had pipelines using speculative dispatch and data forwarding, non-blocking dual operand access, an on-chip secondary cache with hardware prefetch, a two-level hierarchy cache, and an increase in the number of execution units.

We needed to check what our EPS expertise acquired from work in mainframe development were common to the SPARC V9 platform and to enhance ideas. We developed a software performance simulator before the actual start of design. We emphasized system-level performance evaluation; therefore, in spite of using a latency model for the memory system, we created a memory model and processor model that were equally detailed. For the same reason, we made the performance model so that it could be used to evaluate multiprocessor systems.

We updated the performance model continuously during development. That is, we added experimental ideas for subsequent studies and described fixed specifications in greater detail to improve accuracy. Once a logic simulator was available, we started comparing output from the simulator with output from the performance model. This gave us remarkable opportunities to make the model highly accurate. At the end of development of the processor, we verified the accuracy of the model by running SPEC CPU2000 benchmark suits. The error rate was less than five percent.

Throughout development, hardware architects and performance architects worked closely with one another. As a result, we completed SPARC64 V development quickly, and the processor can be utilized in high-performance enterprise server systems.

8. Acknowledgements

Many people have been involved in the development of the SPARC64 V and the authors would like to thank all of them for the tremendous efforts they have put in to make this processor possible. We also wish to thank Yoshiro Ikeda and Masazumi Matsubara for working with us to develop the performance model; Masahiro Doteguchi and his colleagues for their commitment to the performance studies; Akira Asato and his staff for generating the performance test programs; and Noriyuki Toyoki, Yuuji Oinaga, Eizo Ninoi, Kimio Miyazawa, and Hiroshi Muramatsu for their support of this research. We would also like to thank our colleagues for their many insightful comments on this work.

9. References

[1] Gene Shen *et.al.*, "A 64-bit 4-Issue Out-or-Order Execution RISC Processor," In Proceedings of the ISSCC 1995.

[2] FUJITSU LIMITED, "FUJITSU/HAL SPARC64-III User's Guide," http://www.sparc.com/standards.html.

[3] FUJITSU LIMITED, "SPARC64 GP," http://primepower.fujitsu.com/en/sparc64.html.

[4] Tim Horel, and Gary Lauterbach, "UltraSPARC-III: Designing Third-Generation 64-Bit Performance," IEEE Micro May/June 1999, pp. 73-85.

[5] Sun microsystems, Inc., "UltraSPARC III Cu User's Manual," http://www.sun.com/processors/manuals/usIII_um.pdf, May 2002.

[6] Joel M. Tendler, Steve Dodson, Steve Fields, Hung Le, and Balaram Sinharoy, "POWER4 System Microarchitecture," Technical White Paper, October 2001.

[7] R. E. Kessler, "The Alpha 21264 Microprocessor," IEEE Micro March/April 1999, pp. 24-36.

[8] Fujitsu Technology Solutions, Inc., "Millennium Family of Servers," http://www.ftsi.fujitsu.com/services/products/compat/millennium/index.html.

[9] David L. Weaver, and Tom Germond editors, "The SPARC Architecture Manual, Version 9," Prentice Hall, ISBN 0-13-099227-5, 1994.

[10] Eric Sprangle, and Doug Carmean, "Increasing Processor Performance by Implementing Deeper Pipelines," In Proceedings of ISCA 2002.

[11] Mariko Sakamoto, Larry Brisson, Akira Katsuno, Aiichiro Inoue, and Yasunori Kimura, "Reverse Tracer: A Software Tool for Generating Realistic Performance Test Programs," In Proceedings of the HPCA8, pp. 81-91, 2002.

[12] Pradip Bose, and Thomas M. Conte, "Performance Analysis and Its Impact on Design," IEEE Computer May 1998, pp. 41-49.

[13] S.R.Kunkel *et al.*, "A performance methodology for commercial servers," IBM J. Res. & DEV. VOL. 44 NO. 6, pp. 851-872, Nov. 2000.

[14] Matt Reilly, and John Edmondson, "Performance Simulation of an Alpha Microprocessor," IEEE Computer May 1998, pp. 50-58.

[15] Steven Kunkel, Bill Armstrong, and Philip Vitale, "System Optimization for OLTP Workloads," IEEE Micro May/June 1999, pp. 56-64.