Cahners
MICROPROCESSOR REPORT
VOLUME 13, NUMBER 15
NOVEMBER 15, 1999
THE INSIDERS' GUIDE TO MICROPROCESSOR HARDWARE

Hal Makes Sparcs Fly

Sparc64 V Employs Trace Cache and Superspeculation for High ILP

MICROPROCESSOR FORUM 1999

by Keith Diefendorff

Disproving the adage that all interesting microarchitectural techniques have already been implemented in microprocessors, Hal Computer Systems is developing an innovative 1-GHz, eight-issue, out-of-order superscalar design it calls Sparc64 V. Speaking at last month's Microprocessor Forum, Mike Shebanow, vice president and CTO of Hal's Microprocessor Division, described the new processor, which is aimed at the same large-scale enterprise servers that giants Compaq, HP, IBM, Intel, and Sun all covet. Hal expects to tape out Sparc64 V by 3Q00 and to place chips into Fujitsu GranPower SMP servers by late 2001.

Agreeing with Intel Itanium architects, Shebanow—one of the principal architects of HPS, the High-Performance Substrate developed at UC Berkeley under Yale Patt—is also a believer in instruction-level parallelism (ILP). Not surprisingly then, Sparc64 V uses its 65 million transistors to exploit ILP, as opposed to the thread-level parallelism (TLP) that IBM's Power4 (see MPR 10/6/99, p. 11) is chasing with chip multiprocessing and that Compaq's Alpha EV8 (see MPR 11/15/99, p. 13) is pursuing with simultaneous multithreading.

On the other hand, like Compaq, IBM, and Sun architects, Shebanow does not believe that changing instructions sets is necessary, and he believes that dynamic instruction scheduling can fully exploit the available ILP. Shebanow admits, however, that coaxing a wide-issue out-of-order SPARC V9-compatible processor to very high frequency is a significant challenge.

Sparc64 V uses two advanced techniques not found in any current mainstream microprocessor: a trace cache, as Figure 1 shows, and superspeculative execution (execution past memory data dependencies). It can issue up to six integer instructions per cycle, and Shebanow expects his 1-GHz monster to deliver more than 70 SPECint95 (base). On floating-point code, the chip can issue up to eight instructions per cycle, two of them floating-point multiply-add (FMA) instructions that Hal added to SPARC V9. Dual-issuing FMA instructions gives Sparc64 V a maximum throughput of 4 GFLOPS and a SPECfp95 (base) score of more than 130.

These stats are a huge improvement over the 14.7 int/27.7 fp scores of the current Sparc64 III-300. With Sparc64 V, Hal is determined not to sacrifice as much frequency for ILP as it has in previous processors.

Trace Cache Untangles Code

Sparc64 V has its roots in a class of processors computer architects call trace processors (see "Trace Processors," Rotenberg et al., *Proc. 13th IEEE/ACM Int'l Symposium on*

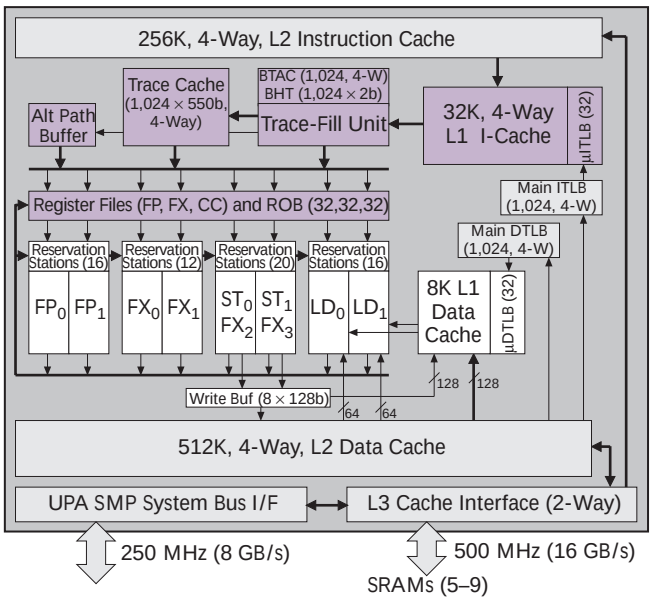

Figure 1. In the front end of the Sparc64 V pipeline (purple), the predicted code stream is sliced into eight-instruction trace packets that are stored in a trace cache and later dispatched to reservation stations in order. The back end of the pipeline (white) issues and executes individual instructions out of order. The cache hierarchy (gray) includes separate on-chip L2 instruction and data caches, separate TLBs, and an external unified L3.

Microarchitecture, 1997, p. 140). In trace processors, the instruction-fetch hardware breaks the code stream into segments called traces that follow the predicted flow of control. Instructions in traces are predecoded, and knowledge about data-dependence relationships and hardware resource requirements is kept with the trace. Traces are stored in a trace cache, one trace per entry, and become the basic unit for fetching and execution in the machine. By exploiting the preprocessed instruction-stream information as well as the spatial and temporal locality of traces, trace processors strive to execute one complete trace per cycle.

By performing much of the work of decoding instructions, analyzing data dependencies, and routing instructions to execution units up front, the trace technique can take pressure off the performance-critical front end of the execution pipeline. In a sense, the trace-creation hardware performs some of the same functions as the back end of a VLIW or EPIC compiler. As in an EPIC design, taking these functions out of line of the execution pipeline allows the machine to have both a reasonably short pipeline and a fast cycle time.

One Gigahertz, Eight-Issue, and a Short Pipe

As Figure 2 shows, the Sparc64 V pipeline is nine stages, which compares favorably with Itanium—a six-issue EPIC design with a ten-stage pipeline (see MPR 10/6/99, p. 1). Keeping the pipeline short—without compromising speed—was a key design goal for Sparc64 V; Hal says it has learned from experience that long pipelines suffer too much from pipeline inefficiencies.

In Sparc64 V's implementation, a trace packet can include up to eight instructions, or at most two basic blocks. Two block-termination branches can be resolved in a single cycle. Each of the eight 32-bit instructions in a packet is heavily preprocessed and the resulting information appended to the trace packet.

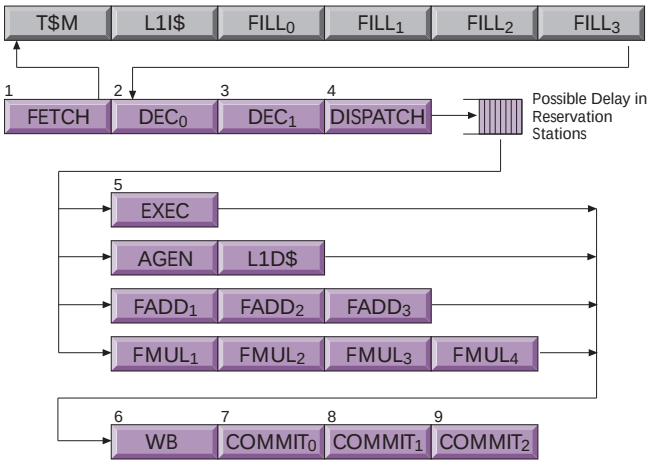

Figure 2. By moving the complexities of decoding and dependency analysis out of line into a separate trace-fill pipeline (gray), Hal was able to keep the performance-critical execution pipeline (purple) short without sacrificing cycle time.

Once formed, packets enter the execution pipeline and are stored in the trace cache, possibly to be fetched again later. The trace cache holds 1,024 packets and is four-way set-associative. It is virtually indexed by eight low-order bits of the program counter ($PC_{2–9}$) and is physically tagged.

After entering the execution pipeline, traces are dispatched to reservation stations and the reorder buffer (ROB) in program order. From there, individual instructions are issued to the execution units out of order. Instructions execute and complete out of order but are reassembled into packets before results are committed to the register file in order. One full packet can be committed every cycle.

Recovery from speculative execution past mispredicted branches is via checkpoints. As packets are dispatched, they are assigned a checkpoint ID, and a snapshot of the machine state is recorded. The processor can track 16 checkpoints, allowing up to 128 instructions to be in flight at one time.

Sparc64 V supplies instructions and data using an unusual cache hierarchy consisting of separate on-chip L1 and L2 instruction and data caches and a unified off-chip L3. The unusual aspects are a very small 8K direct-mapped L1 data cache and the separate instruction and data L2s. Most processors with on-chip L2s, such as Pentium III (see MPR 10/25/99, p. 1), PowerPC G4 (see MPR 10/25/99, p. 10), the 21364 (see MPR 10/26/98, p. 12), and Power4, use a unified L2. Hal separated the L2s to minimize interference, which can be excessive in heavy TPC and numeric workloads, and to improve the chip layout topology.

Much Work Done out of Line

The process of creating traces in the trace-fill unit is a complex one. The fill-unit pipeline must accomplish many tasks while remaining short enough to prevent trace-cache misses from appreciably degrading performance. The first cycle (T$M) of the fill pipeline is required just to detect that a trace-cache miss has occurred and to determine which block to fetch from the L1 instruction cache. The second cycle (L1I$) fetches a 32-byte block from the L1. The L1 is 32K in size, is four-way set-associative, and has a 64-byte line size. The L1 I-cache is backed by an on-chip 256K four-way set-associative L2 instruction cache with a three-cycle latency from address.

In the third stage of the fill-unit pipeline, $FILL_0$, instructions in the L1-cache block are translated from memory order to packet order. If more instructions are needed to fill a packet, the fill unit recycles through L1I$. Once enough instructions have been accumulated, $FILL_0$ packs the newly fetched instructions into packets of up to eight instructions, which will be cracked by subsequent stages. The fill unit collects instructions into packets according to their expected execution order by predicting branches and following the predicted path.

To predict the path, Hal uses a conventional agrees-mode-encoded Gshare dynamic branch predictor (see MPR 11/17/97, p. 22). The predictor's 8-bit global-branch-history

register is hashed (XORed) with eight low-order bits of the branch address to create an index into the branch-history table. The BHT contains 4,096 entries, each a two-bit saturating up-down counter. The most-significant bit of the selected entry is XORed with the static prediction from the branch opcode to create the final prediction. SPARC V9 branches have explicit static predictions encoded in the opcode; for SPARC V8 branches, static predictions are inferred from the direction of the branch. Shebanow says the BHT gives a 93% prediction accuracy on the SPECint95 benchmarks.

Branch address are predicted via a 32-entry return stack and a large 1,024-entry four-way set-associative branch-target-address cache that is indexed in the same manner as the trace cache. To communicate prediction information to later stages and to avoid PC computations at fetch time, 62 bits of branch-prediction information are carried forward with the trace packet, including the taken address, the not-taken address, and a variety of other bits of information.

A feature Hal toyed with but rejected was automatic predication. This technique assigns predicates to instructions at trace-formation time rather than branching around them. Unfortunately, performance studies showed only a tiny performance gain, even with unrealistically large hardware configurations.

It is not clear how this finding reflects on full architectural predication, such as that in IA-64. A compiler has a far greater scope and more freedom to rearrange code than any hardware trace-fill unit. But Shebanow, who was initially enamored of predication, says that even though their hardware predication covered most of the common cases that a compiler could exploit, it still failed to show a performance boost. The problem, Hal discovered, was that predication is a bet against the branch predictor, which is all too frequently a bad bet.

MICHAEL MUSTACCHI

Mike Shebanow, CTO of Hal's Microprocessor Division, describes the Sparc64 V trace cache at the Forum.

Packet Breaks Limit IPC

In $FILL_1$, the fill unit resolves intrapacket dependencies (RAW, WAR, and WAW) and applies the packet-break rules. To simplify sequencing, Hal requires all packets in the trace cache to leave the machine in a conforming state, i.e., a state in which the next-program-counter (NPC) is equal to the current-program-counter plus 4 (NPC = PC + 4). A packet ending with a delayed branch, for example, would be nonconforming. In such cases, packets are broken to enforce conformance.

Normally, trace processors would terminate a packet on a predicted-taken conditional branch (forward or backward). With an innovation introduced by Hal, however, in many cases two basic blocks can be contained in a single packet, significantly improving packing efficiency. To implement this feature, Hal divides branches into two categories: D and S branches. D branches are simple conditional branches that can dual-issue. S branches are unconditional branches, such as long jump (JUMPL) and subroutine call (CALL), which must be single-issued. S branches must be executed in the first fixed-point ALU (FX_0), which has special hardware to handle this type of branch. D branches can be executed by any of the four fixed-point ALUs.

Two D branches can be allocated in a packet, subject only to the rule that the first (the internal branch) cannot jump backward within the packet to anything other than the first instruction; if it does branch to the first instruction, the instructions packed up to the first branch must be able to be replicated in the second half of the packet. This seemingly odd rule allows important cases, such as the four-instruction string-copy loop, to be packed two iterations per packet. At the same time, this rule prevents pollution of the trace cache that would occur in pathological cases, such as, for example, a seven-instruction loop producing seven separate traces, all containing the same instructions.

To prevent multiple address translations for a single fetch, all instructions in a packet are required to fall within a single 8K virtual page. There is also a restriction requiring all instructions in a packet to fall within two consecutive 512-byte memory blocks. Together, these rules make it feasible to enforce coherence and inclusion between the trace cache, which is PC aligned, and the caches, which are memory-address aligned. Without this restriction, the entire contents of the trace cache would have to be flushed every time a cache block is invalidated from the bus.

Other packet-break rules require certain instructions, such as SAVE, which alters the SPARC V9 register-window pointer (CWP), to be the first instruction in a packet. Other instructions, such as RESTORE, must be the last instruction. Instructions that are "microcoded" (cracked into multiple simpler instructions), such as LDD, which loads two integer registers, must occur in their own packet. $FILL_1$ applies a total of 29 such packet-break rules.

In the $FILL_2$ stage, register-file ports and execution slots are allocated for every instruction in the packet. A compromise Hal made for frequency was to limit the number of register-file ports. Theoretically, Sparc64 V could execute two loads and four three-source-operand fixed-point instructions per cycle, requiring 16 read ports in the fixed-point register file (FXRF). Statistically, however, because of two-source-operand instructions, immediate operands, instructions that read the same register, and results being forwarded from other instructions in the packet, 16 ports are rarely needed. Hal took advantage of this fact to implement

only eight FXRF read ports. The floating-point register file (FPRF) was given six ports, and the condition code register file (CCRF), which includes the general-status register and the four floating-point condition registers, was given four.

Similarly, six integer results could theoretically be written per packet, but Hal was able to safely limit the FXRF to four write ports, because neither stores nor branches write back, and because only one instruction in a packet needs to write to any given register. The FPRF also has four write ports, and the CCRF has two.

Because the number of read and write ports is less than can be required by certain instruction sequences, packets must sometimes be broken to avoid oversubscribing to the available ports. Shebanow says, however, that this type of packet breakage doesn't occur often enough to significantly impact performance. More common is breaking the packet because the instruction mix doesn't match execution-unit mix: there are two load, two store, two fixed-point-ALU, and two floating-point-ALU units; the store units perform double duty as auxiliary fixed-point ALUs. But even this structural hazard doesn't pose a serious performance problem.

Shebanow says that simulations indicate a packing efficiency, after all packet-break rules have been applied, of between four and five instructions per packet. The average number of instructions executed per cycle (IPC), however, will be fewer due to pipeline stalls.

No Time to Waste on Housekeeping

The tasks of scheduling register ports and operand-forwarding paths and aligning instructions onto the appropriate execution units are completed by the fill unit in stage $FILL_3$. Removing these tasks from the execution pipeline saves precious execution-pipeline stages, but it requires a significant amount of information to be kept, in decoded form, with each packet, as Figure 3 shows. Each trace packet is a total of 550 bits wide, nearly twice the size of the raw instructions.

Register read-port assignments are made by creating a read-port identifier for each read port in each of the three register files. Each identifier indicates which register that port will read. The source-operand specifiers in each instruction are then recoded to point to one of the read-port identifiers. This level of indirection allows the registers to be read quickly in the execution pipeline, with no decoding or shuffling required at that time.

The source-operand specifiers are recoded as a two-bit type field and a three-bit ID field. The two-bit type field indicates which of four sources will supply the respective source operand. Operands can come from a register, from the result of another instruction, from an immediate operand, or from a short-constant generator. In the case of a register source, ID indicates the read-port identifier. For a forwarded result, ID indicates which instruction in the packet will generate the result. For immediate operands, ID selects one of four possible 13-bit literals stored separately in the trace packet. And for short constants, ID encodes one of several frequently used values, e.g., 1, 4, 8, and –8. The floating-point source-operand specifiers and condition-register source-operand specifiers are recoded in a similar manner.

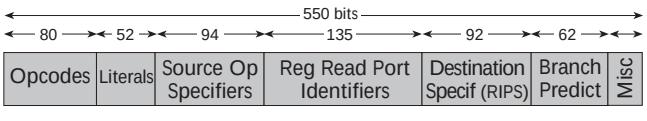

Figure 3. The eight instructions in a trace packet are blown apart and reassembled into like fields, with nearly 300 bits of decode information added.

Destination-register operand specifiers are also recoded and placed into structures Hal calls reorder-buffer-issue packets, or RIPs. Each register file has its own RIP, which contains a specifier for each write port in the file. These RIP specifiers point to the instruction slot that will produce the result for the associated write port, as well as indicating which register will be updated (the RD field) when the result is committed. In assigning write ports, the write-after-write (WAW) dependence is honored by letting only the last write to a given register in a packet actually update the destination register. Fields in the RIP specifiers form the basis of ROB entries, one of which is created for each RIP specifier at the time the packet is dispatched.

Ready, Set, …

With the control-flow-graph of the program straightened out by the trace-formation process, trace execution begins in the DEC_0 and DEC_1 stages of the execution pipeline. Traces are delivered to DEC_0 from one of three sources: the trace cache, the trace-fill pipe, or the alternate-path buffer (APB). Initially, traces are dispatched from the trace cache until a miss occurs. At that point, the dispatcher switches to streaming mode, supplying traces from the fill pipe until another trace-cache hit is encountered.

Currently, Sparc64 V allocates only the first four packets of each new stream into the trace cache. This limit ensures that there are enough traces in the cache to cover the startup time of the fill pipeline while at the same time preventing a loop from polluting the cache. For loops, traces are supplied directly from the fill pipeline; the additional fill stages have no impact on performance in steady-state loop operation. Shebanow says that in the final design the restriction on the number of packets allocated into the trace cache at the beginning of a stream may be increased or eliminated altogether, depending on simulation results.

In the case of a trace-cache hit on the predicted path and a miss on the alternate path, the fill unit is directed to create the alternate trace, which it stores in the APB. In the case of a hit on both the predicted path and the alternate path, the fill unit creates the next sequential trace, again storing it in the APB. The APB is indexed by the checkpoint ID of the predicted trace, so the alternate trace can be supplied immediately from the buffer on a branch misprediction, reducing the mispredict penalty.

In DEC_0, traces are transported from the trace cache, the fill unit, or the APB, and instructions are decoded. In DEC_1 the processor reads source operands (or tags) from the register files (or ROB) and renames destination registers. It also creates new entries in the ROB to hold the results from each instruction in the packet. As part of the register-renaming process, the five-bit RD destination-register identifiers in the RIPs are flattened to eight bits, uniquely identifying them in SPARC's register-window space of 157 total registers.

The ROB is a content-addressable memory (CAM) with 32 entries for integer results, 32 for floating-point results, and 32 for condition register results. The size of the ROB was limited by cycle-time considerations. In addition to the information from RIP specifiers, each ROB entry contains a field for result data, a tag that can be supplied in lieu of data, and an assortment of status bits that track the state of the entry and its data, e.g., written, speculative, and visible (most recent). By manipulating these status bits, the instruction scheduler can perform operations such as flushing invalid speculative results and replaying instructions.

When operands are read, the register identifiers index the register file and simultaneously access the ROB. A hit in the ROB causes the register-file data to be ignored and ROB data to be supplied, if it is present. If the result is not yet in the ROB, the tag is supplied instead.

At the end of DEC_1, the full packet of instructions with operands (or tags) is dispatched to the reservation stations. The tags tell the reservation stations which slots and which checkpoint IDs to watch for operands that weren't available at the time the packet was dispatched.

… Go

In the DISPATCH stage, instructions and operands (or tags) are distributed to the reservation stations. The two fixed-point execution slots share 12 reservation stations; the load slots have 16, as do the floating-point slots; and the store slots share 12 reservation stations for instructions and 20 for store data.

Each instruction waits in a reservation station until all its tags have been resolved, i.e., until all its operands are either present or are being forwarded from another instruction. At this point, individual instructions are issued to the execution units in oldest-ready-first order. If the reservation stations are empty when an instruction is dispatched, they are bypassed, and the instruction is issued directly into execution without a delay cycle.

The execution latency of most simple fixed-point instructions is one cycle. Floating-point add latency is three cycles, and floating-point multiply latency is four. Sparc64 V's floating-point slots are symmetric; each of the two FP slots has a full double-precision adder as well as a full double-precision multiplier. VIS instructions (SPARC's SIMD visual instruction set) use only one of the floating-point slots.

Load latency is dependent on where in the memory hierarchy the data is found. In the best case of an L1 hit, the load pipeline requires one stage for address generation and one stage to access data in the cache. Thus, the minimum load-use penalty is two cycles.

Results are written to the reorder buffer in the WB stage. At this point, execution is complete as far as any direct effect on performance goes, i.e., all results are available to other instructions. But three more stages, $COMMIT_{0–2}$, are required to reassemble results into packets, make the final decision to commit the results, broadcast the decision, commit results from the ROB to the register files, and retire the packets and checkpoints from the machine. Results are committed and instructions retired as a full packet, which occurs only after a packet is the oldest outstanding packet with no exceptions pending, i.e., packets are retired in order.

When in Doubt, Superspeculate

Most modern processors speculate on control-flow decisions so they can make progress, despite uncertainty about the path through a program. But Sparc64 V takes speculation a step further, speculating on data as well. This technique, for which the term superspeculation was coined by Mikko Lipasti and John Shen, solves two significant performance problems in Sparc64 V.

First, Hal was able to grease the straight path through the L1 data cache to be very fast. The L1 is small (8K), direct mapped, write through, and sum addressed for maximum speed. (A sum-addressed cache computes the cache index as part of the cache-address-driver circuits in a way that eliminates the time required to do a full base-plus-index addition.) It takes considerably longer, however, to prove that the data in the cache is actually valid: the address must be translated through the TLB, which could involve a TLB miss, and it must be tested for memory dependencies before it can be qualified as valid. In the best case, this validation process takes two cycles longer in Sparc64 V than accessing L1 data. Superspeculation permits Sparc64 V to proceed as soon as the data is available, backing it out later if the data proves invalid.

The second problem solved by superspeculation is that Solaris, by far the most popular OS used on SPARC processors, requires the processor to run in TSO (total store ordering) mode. In this mode, load operations must be performed in program order with respect to other loads, to protect against the outside chance that an address-alias conflict or a coherency action from another processor may change the data values of a subsequent memory operation in an unexpected way. Superspeculation allows Sparc64 V to proceed as if these unlikely events will not occur and to run memory operations in a more performance-favorable order. In effect, it allows Sparc64 V to operate in SPARC V9's relaxed-memory-ordering (RMO) mode while still adhering to the stricter TSO semantics.

In essence, the processor simultaneously executes two parallel data-flow graphs, a data data-flow graph and a confirmation data-flow graph. The data graph executes at normal memory latency with relaxed ordering, possibly running

ahead of confirmation, which is subject to longer latencies. Once data is confirmed, however, a confirmation data-flow graph executes in TSO mode, but at a uniformly single-cycle rate, allowing it to again catch up with data flow.

Implementing superspeculation required that reservation-station valid bits be extended from one bit to two, so that three states, invalid (I), valid speculative (SV), and valid nonspeculative (NSV) could be represented. Either SV or NSV allows an instruction to be scheduled for issue.

SV results are initially generated by load instructions, and they are propagated by instructions that use SV data. The load unit (LDU) distributes SV data on its distribution bus and confirms validity to the central instruction scheduler on a separate confirmation bus. For every SV data value generated, the LDU will eventually either confirm the data was valid or rebroadcast NSV data on the distribution bus.

When the LDU confirms valid data, the central scheduler propagates the confirmation to all instructions that were either directly or indirectly dependent on the SV data. If the LDU rebroadcasts NSV data instead, the central scheduler, replays all instructions that executed with bad data. Since instructions are not deallocated from the reservation stations until they are confirmed, and they cannot be confirmed until they are nonspeculative, the scheduler can easily replay instructions by making bad results in the ROB invisible and resetting the reservation-station status bits, causing the instructions to be rescheduled and reissued.

A feature initially implemented in Sparc64 V but later removed was load-value prediction (LVP)—the ultimate form of superspeculation. LVP predicts the result of a load at dispatch, allowing a zero-cycle load-use penalty when the prediction is correct. Even though Hal verified that data could indeed be predicted with high accuracy, it found no net performance gain. Since the feature cost 5 mm^2 of silicon and had no performance value, it was removed.

Like automatic predication, Hal discovered that LVP required a bet that often went against the branch predictor. The primary need for short load latency comes from the need to reduce control-flow-decision latency (and thereby mispredict penalties) in, for example, load-compare-branch sequences. The problem Hal ran into was that it was never clear which to trust, the LVP or the branch predictor. In the case where the branch predictor was right and the LVP was wrong, but the hardware bet on the LVP, a substantial amount of good work was unnecessarily discarded.

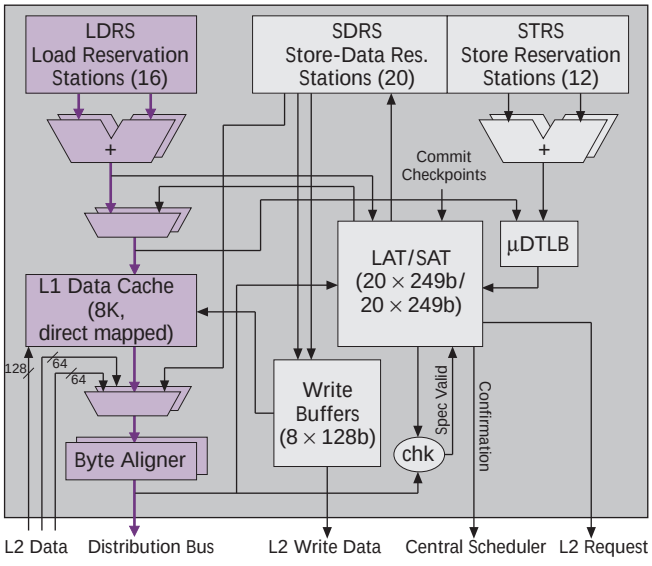

Figure 4. Sparc64 V's load unit (LDU) can issue, execute, and complete two loads and two stores per cycle. The primary load path (purple) has been tuned to be as fast as possible (two cycles). The LAT/SAT (load-address table/store-address table) maintains all data structures and information pertinent to loads and stores that are in flight (dispatched but not committed).

Load/Store Bandwidth Galore

In typical programs, it is common for 25% of the dynamic instruction mix to be loads and 15% to be stores. The percentages can be even higher in certain loops. Therefore a processor needs a high load/store bandwidth to keep other instructions from stalling. Because load/store execution units are inherently complex beasts, however, it is tempting for designers to skimp on these resources. Sparc64 V does not appear to have fallen into this trap: the processor can dispatch, issue, execute, and complete two 64-bit loads and two 64-bit stores every cycle.

As Figure 4 shows, the LDU is fronted by 16 load reservation stations (LDRS), 12 store-address reservation stations (STRS), and 20 store-data reservation stations (SDRS). The critical path through the unit is from LDRS through the L1 to the distribution bus. Hal put an enormous effort into making this path as fast as possible; in the normal case of an L1 hit, it requires two cycles. Although 8K is small for an L1 cache, Shebanow says that any larger L1 would have required the path to be three cycles. Simulations proved, however, that the small two-cycle design was a better performance tradeoff, given the existence of a large on-chip L2 backing up the L1.

But load accesses don't always flow smoothly through the fast path; when something goes wrong it's the load-address-table/store-address-table unit (LAT/SAT) that straightens things out. The LAT/SAT holds all of the primary control and data structures for the LDU, and it maintains information on all pending loads that have been dispatched but not yet committed. A LAT/SAT entry is established when each load or store is dispatched, and the entry is filled in when the load or store is issued. There are 20 LAT entries and 20 SAT entries, each 249 bits wide.

The LAT/SAT schedules "check loads" into the pipeline whenever a load is boosted above another load or store. Check loads execute in program order with respect to other loads to verify that the speculative data sent by the boosted load has not changed in the meantime. If it changes, the LAT/SAT replays the load and rebroadcasts corrected NSV data.

Check loads, because they access the L1 like normal loads, can have a deleterious effect on performance. There-

fore the LDU aggressively searches for opportunities to purge unnecessary check loads from the pipeline before they access the L1. Check loads can be removed whenever the LAT/SAT determines that, for a given load, all older loads are committable, i.e., have already accessed the L1, don't themselves require a check load, and have nonspeculative operands. Check loads need not be generated in RMO mode.

Before the LDU can boost a load above a stalled store, the load must be checked for dependencies against all older stores in the store reservation stations. Loads are never boosted above stores on which they are dependent. Loads that hit on older stores with their data ready can have that data forwarded to the distribution bus, assuming the load request is for data the same size or shorter than the store data. Data forwarding from the SDRS is scheduled as a load replay. The age of loads and stores is determined by their checkpoint ID, which is tracked by the LAT/SAT. The LDU always commits stores in program order, so no dependency check is required between stores.

Two of the major things that can go wrong in the fast path are a microDTLB miss and an L1-cache miss. In the case of a microDTLB miss, the LAT/SAT unit queues and saves the virtual address, then schedules a microDTLB update from the main TLB. After the microDTLB is updated, the LAT/SAT unit replays the load by reinserting the virtual address back into the load path. The microDTLB has 32 entries and is fully associative, while the main DTLB has 1,024 entries and is four-way set-associative.

Large On-Chip L2 Cache Allows Small L1

In the event of an L1 load miss, the LAT/SAT captures the physical address and schedules an access to the on-chip L2 data cache. The L2 is 512K, is four-way set-associative, uses a copyback write policy, and is fully pipelined so one access can be scheduled every cycle. The L2 has an eight-cycle latency (including the L1 access) and has eight 64-bit-wide banks. The L2 data path provides a peak bandwidth of 32 GBytes/s at 1 GHz. Cache lines fetched from the L2 are allocated into the L1 on read misses (but not on write misses), and critical load data from the L2 is multiplexed directly into the byte-aligner for broadcast on the distribution bus.

The on-chip L2s are backed by an external 16–64M two-way set-associative unified L3 cache. The path to the L3 is 128 bits wide and operates at 500 MHz into double-data-rate (DDR2) SRAMs, yielding 16 GBytes/s of L3 bandwidth.

L3 misses cause a bus transaction to main memory across Hal's modified Ultra Port Architecture (UPA) bus, which can operate at up to 250 MHz double-data-rate and be up to 128 bits wide, delivering 8 GBytes/s of main-memory bandwidth. Hal's modified UPA bus is not electrically compatible with Sun's version, but it preserves enough of the MP protocol to be 100% compatible with Solaris software.

Stores write their data to an eight-entry write buffer. The 128-bit write buffers are collapsing buffers and can gather multiple stores into a single transaction before writing data into the L2 and, on a hit, into the L1. Data normally lingers in the write buffers for only a short period of time, so it is considered transitory and therefore not forwarded from the write buffers to the distribution bus on a load hit. Instead, a load hit on a write-buffer entry causes the load to be replayed after the data has been written into the L2.

Definitely Intended for Big Iron

Sparc64 V provides an unusually robust set of RAS (reliability, availability, and serviceability) features, even for a processor targeting the high-end continuous-availability server market. The on-chip L2 caches are protected by ECC on both data and tags. The L3 is protected by ECC on data and by parity and redundancy on the tags. All of the processor's internal data paths are either ECC or parity protected, and all function-unit operations are checked by dual-rail-logic inconsistency checks or by redundancy and voting.

Utilizing its checkpoint-recovery mechanism, Sparc64 V can retry instructions that experience RAS failures in an attempt to get past transient errors, such as those caused by inopportune alpha-particle strikes. All RAS errors, recovered from or not, are logged, and the log can be interrogated by either the operating system or, in the case of fatal errors, the JTAG scan logic.

Hal obviously didn't put much emphasis on getting the Sparc64 V chip to fit in a PC chassis. Shebanow projects that the part will dissipate 100 W at 1 GHz in its target process. While this is roughly on a par with what we expect from competing chips, taming this fire-breathing dragon with conventional air cooling won't be easy. To achieve this feat, Hal will use wind-tunnel airflow (800 CFM at 4 m/s) over a massive copper heat sink with internal heat pipes to efficiently spread the heat load. The part will be delivered in a 2,063-contact glass-ceramic LGA with thin-film copper wiring layers; only 663 of the contacts carry signals.

Hal estimates the 65-million-transistor Sparc64 V chip will occupy about 380 mm^2 of silicon in Fujitsu's 1.5-V CS85 process. This process uses a tungsten local interconnect layer topped by six layers of copper interconnect and C4 array bumps. The contacted pitch of the first three interconnect layers is 0.6 micron and the transistors have a 0.12-micron physical gate length (L_g) and a 0.09-micron L_{eff}. These parameters put CS85 slightly ahead of IBM's 0.18-micron CMOS-8S (see MPR 9/14/98, p. 1). By 2H01, however, IBM will have moved on to CMOS-8S2 and SOI wafers, which it is planning to use in Power4.

Not Seeking an Uncontested Niche

With fuel being tossed on the fire by the Internet, the server market is heating up. Intel and HP are making a major push into this space with Itanium; IBM is parrying with Power4; Compaq is readying EV7 and 8; and Sun is working on UltraSparc 4 and 5. From a competitive point of view, this is an elite group with deep pockets and unquestioned commitment. Fujitsu, Hal's parent company, also has deep pockets,

For More Information

The Sparc64 V presentation Hal made at Microprocessor Forum can be found on its Web site at *http://mpd.hal.com/products/index.html*.

but its future is less tightly linked to the server market than these other companies, making Fujitsu's commitment less certain.

Hal and Fujitsu receive some minimal protection from these competitors by defining their target market as the Solaris-compatible server market, making Sun their only serious opponent. To some extent, this makes for easier pickings. In the past, Sun's UltraSparcs have trailed noticeably behind other processors in performance. But Sun's system-level efforts have managed to compensate for this shortcoming, making the Solaris-compatible subsegment quite large and financially attractive.

Perhaps Fujitsu sees an opportunity to poach this market by offering a higher-performance processor than it expects Sun to deliver. It is also possible that Sparc64 V is simply a defensive move. Fujitsu may feel that it can't trust Sun to design, or to sell it, processors with sufficient floating-point performance to stay competitive in high-end technical servers, where Fujitsu is focused today, or processors with high enough reliability to penetrate further into the commercial-server space. Although Sun is mum on its plans for future UltraSparc chips, other than a vague long-range roadmap (see MPR 10/5/98, p. 15), we doubt the company is consciously planning to cede the high-performance, high-reliability segments of the market to Fujitsu.

Ultimately, however, Solaris compatibility does not define a defensible segment. Fujitsu will eventually have to do battle with the other heavies: Compaq, HP, IBM, and Intel. Even though Hal has a top-notch design team, it will be a challenge for this relatively small, 200-person, team to match the efforts of these giants—regardless of the amount of money that Fujitsu is willing to invest in them.

In this broader market, Sparc64 V looks competitive, but not compellingly superior to the chips these other companies are developing. It will compete most effectively on technical workloads, where single-thread instruction-level parallelism is abundant. In this arena, Intel's McKinley, which we expect to have single-thread SPECint/fp95 ratings as high as 90/150, may outperform Hal's chip, but Sparc64 V will have the advantage of backward compatibility (at full speed).

For commercial servers, Sparc64 V will find the going tougher. In these markets, where multithread performance is more important, approaches like Power4's chip multiprocessing (CMP) and Alpha EV8's simultaneous multithreading (SMT) come into play. A Power4 chip, for example, which will ship at about the same time as Sparc64 V, will be in a more aggressive process and have about the same die size, twice the on-chip cache, twice the number of CPUs, and several times the memory bandwidth. Shebanow dismisses the memory-bandwidth advantage of Power4, however, saying that any more than the 8 GBytes/s provided by Sparc64 V is nothing but overkill for any processor in this generation.

Shebanow, while a believer in ILP, freely acknowledges that SMT or CMP is the way of the future. For this generation, however, he defends Sparc64 V's single-thread approach for two reasons. First, Solaris doesn't currently support multiple physical (CMP) or virtual (SMT) processors on a single chip. Sun has not said if or when this deficiency will be remedied, but it probably won't be soon, as OS-kernel modifications usually take a long time. Second, he is unsure how rapidly applications will be rewritten to take advantage of explicit multithreading and shared on-chip caches.

Complexity? What Complexity?

Although Sparc64 V's competitive position in 2001 may be questionable, the part will surely be an impressive technical accomplishment if Hal can pull it off. Although trace processing and superspeculation are ideas that have been kicking around in the literature for a few years, they have not yet been put into practice. Intel has said that Willamette, due late next near, will use a trace cache, but little else is known about that design.

A striking characteristic of Sparc64 V is its daunting complexity. Shebanow shrugs off this criticism, saying that the overall concepts are actually quite simple and the dataflow control not really that difficult. Mostly, he says, it is the need to attain a gigahertz clock rate while issuing eight instructions per cycle that introduces the complexity. Shebanow says he fears the kind of complexity in a Pentium III with its convoluted x86 instruction set far more than that in Sparc64 V.

Indeed, from an objective point of view, the core does not seem unmanageably complex. Of Sparc64 V's 65 million transistors, only 15 million are logic. This isn't that much more than the eight million in a Pentium III core, considering that Sparc64 V is 64 bits wide and has twice as many execution units. Still, the complexity is significant and gives one pause to empathize with the poor Willamette engineers who may be trying to implement similar techniques on top of the x86 architecture. Perhaps this explains why Intel is trying with IA-64 to foist complexity back onto the software. It may also explain why IBM with Power4 and Compaq with EV8 have switched their attention from ILP to the untapped store of TLP.

Shebanow insists, however, that for this generation, ILP is the way to go, especially for Fujitsu's primary market target of Unix technical servers. He also says that compatibility is critical to the server market and does not believe that static scheduling—à la IA-64—is sufficiently better than dynamic scheduling to justify switching to a new instruction set. If he is correct, and if Hal can really achieve 1-GHz speeds with Sparc64 V, Hal and Fujitsu will be in a good position to make hay in at least the Solaris-compatible server market, and perhaps in the broader server market as well. M