Intel Discloses New IA-64 Features

Rotating Registers Reduce Code Expansion; Merced Touted for Big Servers

by Linley Gwennap

In a series of talks at the recent Intel Developers Forum, the company tantalized industry watchers by dribbling out a few more details about its IA-64 instruction set and its first implementation, Merced. In a joint presentation by Intel's John Crawford and Hewlett-Packard's Jerry Huck, the two architects shed additional light on the IA-64 design. They provided further details on the architecture's support for predication and speculation and also described IA-64's branch architecture. A newly disclosed feature, rotating registers, provides an efficient way to unroll loops while minimizing code expansion.

In other talks, Intel disclosed that Merced and its first chip set, the 460GX, will support high-availability features required in large servers. The company asserts that four-processor Merced servers will deliver more performance on the TPC-C benchmark than four-way servers using 1-GHz Alpha 21264 processors or 750-MHz UltraSparc-3 processors, two key Merced rivals that are expected to ship next year. But it has yet to disclose any details about clock speed, bus bandwidth, or other metrics to support this position.

Register Renaming Implemented in Software

One of the key philosophies of IA-64 is the idea of moving complexity from the hardware to the software. Register renaming is one example. Most high-end processors map a small number (8–32) of logical registers onto a larger set of physical registers (up to 80 in the case of the 21264). Because software can access only the logical registers, the hardware must assign mappings and translate accesses using an associative lookup table. This complexity increases die size and often the pipeline depth as well.

IA-64 eliminates this hardware complexity with its large register file (128 integer, 128 floating-point) that is directly accessible by software. Specifying the physical register names in software works well except in the case of tight loops, a common occurrence. In these short code sequences, there may not be enough instructions in the loop to cover the latency of load instructions, resulting in unwanted stalls.

An out-of-order processor reorders instructions to cover the latency of the loads. The reordering naturally overlaps instructions from two or more iterations of the loop until enough instructions are found to overcome the latency (or the hardware runs out of resources). This overlap will cause register conflicts, since each loop iteration references the same registers, but these conflicts are resolved by hardware register renaming.

An IA-64 processor can address the latency problem by unrolling the loop in software. This common compiler technique duplicates the loop instructions, often several times, to generate enough instructions to cover the load latencies. Each duplicate set of instructions, however, must use a different set of registers to avoid collisions. IA-64 has plenty of registers available, but all of these duplicate instructions can create massive code expansion.

Rotating Registers Compact Code

To reduce code expansion, IA-64 uses its rotating registers. With this technique, the upper three-quarters of each register file (integer, FP, and predicates) rotates, leaving the lower registers for global variables. Accesses to these upper registers are offset by the value in the corresponding RRB (rotating register base) register. A special instruction, BR.CTOP, decrements each of the RRBs by one at the end of each loop iteration, allowing the next iteration to use a new set of physical registers. (With proper spacing, several variables can be rotated through the register file at once.)

The rotating predicate registers provide a simple way to handle loop setup (prologue) and termination (epilogue). If the prologue and epilogue instructions are appropriately predicated, and the predicate registers rotated, the prologue instructions are executed only during the initial iteration(s) of the loop, and the epilogue instructions are executed only

MEMCPY LOOP: **for (i=0; i<n; i++) {*b++ = *a++}**

(a) PA-RISC with hardware reordering

```
; Set up r2=loop count, r10=source addr, r11=destination addr

loop:   LDWM      r1, (r10)       ; Load into r1, inc addr
        STWM      (r11), r1       ; Store from r1, inc addr
        ADDIB,>   r2, -1, loop    ; Decr loop count and branch
```

(b) IA-64 with rotating registers

```
; Set up LC=loop count–1, r10=source addr, r11=destination addr
; Clear predicate registers, set p16, set EC=epilogue count

loop:   (p16) LD8       r34 = [r10], 8    ; Load into "r34," inc addr
        (p17) ST8       [r11] = r35, 8    ; Store from previous "r34," inc addr
              BR.CTOP   loop              ; Decr loop count and branch
```

Figure 1. In a simple memory-copy loop, a PA-RISC processor with hardware reordering will cover the latency of the first load by launching subsequent loads, creating multiple versions of "r1" using hardware renaming. Without adding instructions to the loop, an IA-64 processor will accomplish the same effect by rotating its registers; in this case, "r35" refers to the previous iteration of "r34."

during the final iteration(s) of the loop. Some setup is still required to properly initialize the predicates, but this can be done well in advance of beginning the loop, removing this setup from the critical path.

Eschewing an orthogonal register set, HP and Intel added several special registers to implement this process. The 64-bit LC (loop count) register performs its eponymous function. The 6-bit EC (epilogue count) register controls the execution of epilogue instructions. Three RRBs (each 6 or 7 bits) rotate the integer, FP, and predicate registers, as described above. The use of special registers allows the BR.CTOP instruction to specify several operations at once, but in the common case of nested loops, register rotation can be used in only one of the loops.

This method of register renaming allows a single copy of the loop code to be unrolled in hardware rather than software, eliminating most of the code expansion, as Figure 1 shows. Rotating the registers adds some complexity (a few 7-bit registers and adders) to the hardware, but it adds far less than the fully generic renaming hardware in a reordering CPU. The rotating register concept dates back to Cydrome's Cydra-5, one of the original VLIW processors; not coincidentally, its architect, Bob Rau, is now on staff at HP.

By handling epilogue and prologue issues in a simple fashion, IA-64's rotating registers are appropriate even for loops that iterate only a few times. Thus, this technique can be broadly applied. In current processors, loop unrolling is rarely used, except in scientific code, where iteration over long vectors amortizes prologue and epilogue overhead.

Static, Dynamic Prediction Combined

The basic IA-64 branch instruction uses a 21-bit relative offset. Branch targets must be the first instruction in a bundle, allowing branching within ±16M. Aligning the targets simplifies the branch hardware and extends the target range, but it will cause some code expansion by adding an average of one NOP instruction per branch target.

Because conditional branches use the predicate field to specify the condition, they have the same long offset. Indirect branches (including call/return) use a special set of eight BRs (branch registers), instead of the integer registers, to hold target addresses. These special registers are likely to be physically located near the fetch unit, not the ALUs, and thus they could obviate the call/return stack used in most high-end processors to fetch the target of subroutine returns.

The LC register improves branch prediction for loops. Most branch predictors mispredict the final (fall-through) iteration of a loop-closing branch, but IA-64 hardware can easily and accurately predict these branches by looking at the LC register. BR.CLOOP is a degenerate form of BR.CTOP that simply decrements LC and loops if LC $\neq$ 0. Again, this mechanism can be used for only one loop at a time, as there is only one LC register.

IA-64 branches include at least two bits to give the compiler more control over branch prediction. Like many RISC architectures, IA-64 provides a "hint" as to whether a branch is likely to be taken or not taken. The hardware can use this hint to initialize the dynamic branch predictor. Some branches, however, are easily predicted by software, as they nearly always branch the same way. The second bit indicates that software prediction should be used; the hardware predictor can ignore these branches, freeing entries for more difficult branches. (The hardware may enter static taken branches into its target-address predictor.) This combination of software and hardware prediction should provide more accuracy than today's hardware-only branch predictors.

Like PA-RISC, IA-64 can combine a comparison and a branch in a single cycle. PA-RISC uses a compact compare-and-branch instruction. IA-64 instead combines a predicate-generating CMP instruction and a branch predicated on that result into a single group for parallel execution.

A branch instruction must be at the end of a parallel-instruction group. As a special case, two or more branches can be placed together at the end of a group to form a multiway branch. All the branches can be processed in a single cycle (assuming the hardware has enough resources), as it is a simple matter to check the predicates and determine which, if any, branch should be taken. This construction is useful when several short code blocks have been combined using predication; all the exit cases can be processed at once.

Flexible Design Allows Massive Speculation

The recent disclosures indicate that IA-64's predication and speculation capabilities are more extensive than previously indicated (see MPR 10/27/97, p. 1). Speculation is used to hoist loads above branches, giving the compiler more flexibility to reorganize code. To handle exceptions, each IA-64 register is tagged with an associated NaT (not a thing) bit that is set when a LD.S (speculative load) encounters an exception. The actual exception is deferred until a CHK.S instruction is encountered.

The CHK.S instruction is simply a conditional branch that tests the NaT bit. If NaT is true, it branches to fixup code that reexecutes the load and handles the exception. The IA-64 architects did not take advantage of the target register (which is undefined when NaT is true) to store the load address; thus, the fixup code must have access to any registers needed to recreate the load address.

The NaT mechanism allows instructions that use speculatively loaded data to be hoisted as well. Any computation instruction sets the target register's NaT bit if any source is NaT. This NaT propagation allows entire routines to be executed speculatively, with exceptions later handled by a single CHK.S. Note that the recovery code must also redo any speculative calculations after reloading the correct data. HP's Huck estimates that half of the instructions in a typical program are likely to execute speculatively.

IA-64 also includes a mechanism for hoisting loads above stores. In a traditional architecture, the compiler can

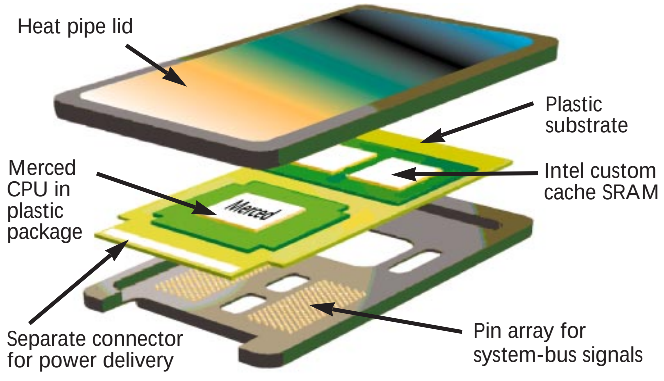
Heat pipe lid
Plastic substrate
Merced CPU in plastic package
Intel custom cache SRAM
Merced
Separate connector for power delivery
Pin array for system-bus signals

Figure 2. The Merced CPU is combined with up to 4M of SRAM in a module that attaches horizontally to the motherboard.

do this only if it can guarantee that the load and store use different physical addresses. With indirect addressing, however, this pointer disambiguation can be impossible at compile time. Reordering processors handle this task easily, since loads and stores are reordered at runtime, after addresses have been calculated.

To hoist a load above a store, IA-64 uses the LD.A (advanced load) instruction. In addition to performing a normal load, this instruction inserts the load address into the ALAT (advanced load address table). Subsequent store addresses are associatively checked against the ALAT; if a match is found, the offending entry is removed. Before using the data from an LD.A instruction, an LD.C is needed to see if the entry associated with that target register is still in the ALAT. If so, the LD.C is a zero-cycle NOP; if not, the LD.C simply reexecutes the load. A CHK.A instruction allows speculative computation based on the result of an LD.A.

The size of the ALAT is implementation dependent. If an LD.A bumps a "live" address from the ALAT, the LD.C (or CHK.A) will reload the data, causing a performance loss but no error. Similar structures, such as the P6's MOB (memory reorder buffer), are found in reordering processors. The MOB, however, is invisible to software, whereas the ALAT is directly manipulated by the IA-64 compiler.

Predicates Better Than Traditional CMOV

Predicates can be generated using CMP (compare) or TBIT (test bit) instructions. Special versions of CMP combine a comparison with a predicate value to create compound conditions (e.g., A = B or A = C). CMP always generates two predicates, the requested value and its complement, either of which can be stored in any predicate register (p0–p63). The first register, p0, is hardwired "true" and can be used as a target to discard unwanted predicates.

Many programs are limited by control flow; that is, they have many conditional branches that are hard to predict. By combining short routines, the compiler can use predicates to eliminate branches, avoiding costly mispredictions. Traditional architectures can eliminate branches using the simpler CMOV (conditional move) instruction, but this method requires longer code sequences and often results in lower performance than a fully predicated architecture such as IA-64.

Merced Targets Big Systems

In other presentations, Intel left no doubt that the primary target of Merced is high-end commercial servers. The first IA-64 processor and its associated system-logic chip set, the 460GX, will include a host of features to deliver the performance and reliability needed by these expensive systems.

For example, all of the caches and system buses are protected from data loss using ECC or other techniques. Corrupted data is corrected when possible; if not, it can be marked as bad and the affected process terminated without crashing the entire system. For a fully fault-tolerant system, two Merced CPUs can operate in lockstep and crosscheck each other. This feature is not likely to see significant use, as Tandem, the leading vendor of fault-tolerant systems, has chosen Alpha over Merced for its next-generation boxes.

The 460GX supports ECC on the system bus and in the main-memory subsystem and can map out failed DRAMs. It handles up to four Merced processors and can be used as a building block in larger systems, although several Intel customers are developing their own system logic to connect eight or more Merced processors. The 460GX supports hot plugging on up to four PCI buses, each at up to 64 bits and 66 MHz for extra bandwidth. The multichip set can also be used in workstations, as it includes an AGP 4× port.

Sources indicate the 460GX allows at least 16G of SDRAM interleaved four ways, as Direct RDRAM will not provide adequate density in 2000. This bandwidth may be wasted on Merced, however, as Intel says the processor's system bus will carry significantly less than 3.2 Gbytes/s.

As Figure 2 shows, the processor itself is housed in a module containing the CPU chip and custom cache chips. Using both sides of the substrate, the module appears to have room for four SRAMs. Using its 1-Mbyte SRAM, known as CK1 (see MPR 7/13/98, p. 1), Intel should be able to fit up to 4M of full-speed cache on the Merced module.

The module design is functionally similar to the Xeon module but mechanically quite different. The module lies horizontally above the motherboard rather than vertically and uses two sets of connectors. A pin array carries the bus signals and provides better electrical performance than the edge connector in the Slot 2 module; this method should enable faster bus speeds. Power delivery is handled through an edge connector (presumably connected to a wire harness) to efficiently deliver plenty of amps.

The module lid is hollow, forming a heat pipe. This design spreads the intense heat from the CPU across the entire lid, reducing heat density. The system maker must attach a large heat sink to the module to further dissipate the heat. Intel has not disclosed the power of the processor, but the package design clearly implies that it will be high; we estimate the Merced module will dissipate more than 70 W.

Merced Nearing Tapeout

The Merced design team has made much progress since the major slip announced last summer (see MPR 6/22/98, p. 1). The processor is finally nearing tapeout, and the company expects to receive first silicon around the middle of this year. The schedule allows about 12 months to bring up and verify the design before it is ready for system shipments. This schedule seems somewhat aggressive for a high-end processor implementing a new instruction set and aimed exclusively at multiprocessor-capable systems; we would not be surprised if system shipments slip toward late 2000.

Intel has been working hard in an attempt to ensure a smooth bringup process. Given the focus on multiprocessing, the company is already performing MP verification on a presilicon RTL model. Seven operating systems—64-bit Windows NT, SCO UnixWare, Novell Modesto (NetWare), Compaq Tru64 Unix, Silicon Graphics Irix, Sun Solaris, and HP-UX—have booted in MP mode using a system simulator and are expected to be ready for mid-2000 shipments. An eighth port, Linux, is in progress. Because the system designers were aiming for 1999 shipments, the Merced slip has put them ahead of the CPU. Some platforms are already being tested and await only the processor.

Intel has begun working with software vendors to ensure that key applications will be available when Merced ships or shortly after. Leading server applications from Oracle, Informix, SAS, Baan, SAP, and others are already on tap. Technical applications are in the pipe from vendors such as Cadence, Mentor, Synopsys, Softimage, Avid, and Adobe. These vendors have already received software development kits, including the system simulator; several applications are already running on the simulator.

Given that Merced is likely to be superseded by its successor fairly quickly, Intel is trying to build a smooth migration path to McKinley. Although McKinley will use a much faster system bus than Merced's, Intel hopes vendors will be able to reuse much of the system infrastructure. For example, Intel's 870 chip set for McKinley will support legacy memory and I/O from the 460GX. McKinley will also use no more power or board space than Merced, avoiding chassis redesign.

IA-64 Performance Debate Unsettled

The new details of IA-64 highlight its philosophy of moving complexity from the hardware to the compiler. With these new features, an IA-64 compiler can perform most of the code motions handled by hardware in a reordering processor. The compiler can perform these code motions across an arbitrarily large group of instructions, whereas reordering hardware is limited to a window of no more than 80 instructions in today's implementations. Without predication and access to large register files, RISC compilers cannot perform the same optimizations and must rely on the hardware.

Initial criticisms of IA-64 focused on its emphasis on static instruction scheduling, which ignores dynamic information available to the hardware at runtime. Some of the newly disclosed features address these issues and show how IA-64 combines static and dynamic scheduling. The ALAT, for example, allows loads to pass stores in a manner impossible in a purely static machine. IA-64's branch prediction is another example of a combination of static and dynamic methods.

The tradeoff is that dynamic features add hardware complexity. Initially, it appeared that an IA-64 design might be more compact than an out-of-order processor by eliminating the instruction-reordering and register-renaming logic. IA-64 processors, however, still require features such as dynamic branch prediction, rotating registers, and the ALAT, which consume die area. Although these particular features may be smaller than their RISC counterparts, when combined with other IA-64 features such as predication and the large register files, they are likely to limit any die-size advantage IA-64 might have over RISC.

Code size remains a concern. The rotating registers avoid the massive code bloat of loop unrolling, but speculation is another issue. Every speculative instruction must be duplicated in a fixup routine, although these infrequently executed routines aren't likely to do much, other than take up disk space. More critical are the aligned branch targets and the 41-bit instructions themselves, which are 33% larger than RISC instructions. An increase in code size reduces the effectiveness of the instruction cache and requires more bandwidth from the system bus and from main memory.

The large register file improves performance by reducing data-cache accesses, but it creates a problem on context switches. Saving state requires storing 128 integer registers (64 bits each), 128 FP registers (at least 80 bits each), 63 predicate bits, 256 NaT bits, 8 branch registers, 3 RRBs, the LC, EC, and Intel knows what else.

The upside of all these features is real, but it remains to be quantified. We expect the net performance advantage of IA-64 over RISC will be around 20–30%, significant but not impossible for competitors to overcome. If Intel delivers strong implementations, they should match or exceed the performance of the fastest competitive chips.

Merced may not be the best implementation of IA-64, but it should be very competitive, enough to establish the architecture in high-end servers and some workstations. The design is progressing well, and we expect first systems to appear in 2H00. Support from both system and software vendors is strong and unwavering. As we have seen with x86, this support, more than any technical merits or demerits, determines the fate of a new microprocessor. M

For More Information

For more information on IA-64, visit Intel's Web site at *http://developer.intel.com/design/processor/future/ia64.htm.*