THE PRESIDENT'S VIEW

IA-64 and Merced—What and Why

Everyone Talks About Merced, But No One's Doing Anything About It

by Peter Christy

It's very weird. 1997 may well be the year of first silicon for what is likely to be the most significant new microprocessor family the world has seen in a long time: Merced, the first implementation of the Intel/HP architecture collaboration called IA-64. Yet we hear almost nothing about IA-64 or Merced. From the computer-user side you might guess there would be more curiosity about what these new processors might do to the performance curve. From the software and system side, you might expect a lot more discussion about how disruptive the evolution from the x86 family will be.

It is easy to understand why things are so quiet—Intel is suppressing discussion as a condition of learning about IA-64 and Merced. Intel is neither stupid nor mean-spirited. It just sees more liability than benefit from such discussions at this time. We're sympathetic to Intel's concerns, but we feel a broad IA-64 discussion is appropriate among the community that will feel the impact.

The development of IA-64 signals a fundamental shift; it's the first of several next-generation architectures designed to compensate for the complexity of today's CPUs with tighter links between a smarter, more advanced compiler and a simpler, faster microarchitecture.

The Potential Value of a New Architecture

A new architecture can redefine the performance roadmap, and the name of this game is performance. Today's highly superscalar microprocessors use an ever-decreasing percentage of their transistors to do useful work. Of the millions of transistors in a modern CPU, remarkably few directly contribute to adding numbers or moving data under the direction of the programs being executed. Far too many are spent rearranging instructions and keeping track of their original order. An architecture that reversed this trend and invested more in real application performance would be a significant. change. There are enough transistors available to implement many more functional units than CPUs have today, if only we could put them to productive use.

An increasing percentage of the logic on a microprocessor (the CPU less the cache) is dedicated to management functions such as control of out-of-order and speculative execution. Such logic is difficult to design and largely implementation specific, increasing the complexity and time-to-market of the processors. This logic also tends to span a lot of die area, creating speed-limiting paths. A new architecture could greatly reduce the investment in these overhead functions.

The dominant consumer of transistors in microprocessors is cache memory. Cache is a good and easy way to use silicon, but at best, a cache lets the CPU work at speed; it does not by itself do useful work. Architectural enhancements can make cache fundamentally more effective.

Looking at Intel's problems specifically, a new architecture could fix the most glaring problems in IA-32 (x86), including its 32-bit address limitations and the awkward floating-point stack architecture.

Although we're not yet privy to any of the specific details of IA-64, we believe that the new architecture attacks all of these problems. In short, we think an innovative architecture can do more useful work per clock cycle and run at faster clock rates, thereby getting onto a new and better performance curve. Any processor that redefines performance expectations (Digital's Alpha, for example) will have a significant impact. Such a processor from the dominant market leader will have a profound impact.

Is IA-64 RISC, VLIW, or Something Else?

We tend to think of IA-64 as a modern-day RISC machine in some sense. The earliest view of RISC, from John Cocke in IBM's 801 design, was based on the idea of trading compile-time optimization against CPU complexity. Instructions in the 801 were simple and chosen for their ability to be implemented in high-speed logic. A large register set replaced complex addressing modes. Delayed branch instructions reflected the nature of branch processing and let a compiler optimize for it. The term RISC was coined, after the nature of the instructions. For lack of a better term, I'll use RISC here, but much more in the sense of Cocke's original concept: rely on the compiler.

The transition to next-generation architectures such as IA-64 can be understood as another transfer of complexity from the hardware to the compiler. The early commercial RISC processors (the 801 was never brought to market) took advantage of simplified instructions to pipeline instruction execution and thus increase performance. (Contrary to popular belief, RISC chips did not have faster clock rates than contemporaneous CISC chips; they just introduced pipelining sooner.) Pipelined execution more than made up for the decrease in "power" of the simplified instructions. Today's RISC—and CISC—microprocessors have become complex again, so it's time for another dose of the same medicine: more compile-time optimization and simpler hardware.

We expect IA-64 to include more compile-time instruction sequencing. For this reason, some will call it a VLIW

(very long instruction word) machine. Today's superscalar microprocessors have extensive logic to detect parallelism in the instruction stream and initiate concurrent operations when possible. We expect IA-64 to pass much of this burden back to the compiler. To a much greater extent than today, the CPU will just execute the instruction stream it is given as fast as it can, knowing the instruction interactions are safe because the compiler made it so.

The IA-64 architecture is also likely to include predicated execution and a larger register set. Today's microprocessors work hard to unearth data dependencies (i.e., whether the contents of a register are stable or still subject to unfinished calculations). With more registers, a compiler can use different register subsets to support execution on multiple branch paths, alleviating the need to analyze dependencies because there is less need to reuse the registers.

To get the performance benefit of multi-issue designs without the complexity of dynamic control logic, the compiler needs the ability to create parallel branch-path logic. That means some enhancements to the notion of condition codes and the use of earlier test conditions to control execution (predicated execution) are needed. With a smart enough compiler, the result is the same as today's out-of-order or speculative execution, except that the mechanism is moved from the CPU to the compiler. The CPU logic gets simpler, and a source of slow paths is removed. The processor runs faster and does more work.

Changing Roles for Cache and Compiler

Better prefetch, store, and branch hinting are also likely. Ideally, the compiler and processor will cooperate to anticipate the use of data, so the cache can be managed most effectively, and to anticipate the flow of the program, minimizing processor stalls due to unexpected branches.

At the time the program is compiled, the compiler can develop a comprehensive model of the program logic and flow (this is used today for comprehensive optimization). The compiler can determine quite a bit from the logic of the program, and even more if the language supports advisories (e.g., pragmas) from the programmer regarding the program's expected behavior.

With today's instruction sets, the compiler has only limited ability to make use of this information in a way that speeds execution. Much of the analysis is thrown away and unavailable to the CPU. If the compiler can generate hinting instructions, compiler/processor efficiency can be improved. Based on the program flow analysis, the compiler can annotate a load or store with a measure of urgency ("we will need this data soon" or "if you have load bandwidth available, get this data"). Similarly, the compiler can generate a compile-time view of branch prediction and pass this data to the CPU to preload the branch-prediction tables.

There are more than enough transistors to provide many more function units on a microprocessor than what we see today. Their utilization can be scheduled at compile time if there is enough parallelism in a program to take advantage of the hardware.

In summary, today's instruction sets make it difficult for the compiler to advise the CPU on what is likely to happen, so processor designers add lots of clever but bulky logic to determine program behavior on the fly. IA-64 has much room to improve this compiler/processor cooperation.

Many IA-64 Ideas Are Not New

Many of the ideas in IA-64 have been used already, but never in a mainstream microprocessor, and certainly not in a product from a market leader.

We expect IA-64 to take full advantage of all that has been learned about computer architecture, but many of the key concepts aren't really new. Similar architectures were developed in the 1980s. Cydrome and Multiflow, among others, sold VLIW minicomputers (see **080205.PDF**). The clock speed of these minicomputers was limited by the low-integration component technology and by the large physical size of the computer.

VLIW machines promised more work per clock cycle, and thus greater execution power, than more conventional designs of the same era. These machines worked, but soon minicomputers as a whole started to fade away, in part because they couldn't keep up with VLSI CPU performance progress, and because small firms couldn't afford to produce VLSI versions of their unique architectures.

These early VLIW machines also suffered because they pushed the requirements of compiler technology; compiling code for them could be painfully slow. It will take at least as much work to compile a program efficiently for IA-64 as it did for a Cydrome or Multiflow computer, but the compiler will be running on a CPU at least 100 times faster, so we don't expect compile time to be a particular hurdle.

Parallelism Provides High Value

Microprocessor performance increases with clock speed and with the amount of work done per cycle. We've already discussed how IA-64 helps increase clock speed and makes caches more effective. We also believe IA-64 will bend the price/performance curve by permitting more internal parallel execution.

Microprocessors already benefit from parallelism. Many CPUs dynamically schedule multiple function units based on observed data dependencies in the instruction stream. This multiple-issue mechanism clearly yields incremental performance up to a point, but it's not obvious that going beyond today's four-way issue is worth the effort.

We believe IA-64 will enable explicit parallelism well beyond simultaneous four-way execution, although not with existing binaries. The parallelism will be achieved by compilers that more fully extract parallelism from the source code, the addition of explicit parallel notations in programming languages, new compilers for those languages, and explicitly parallelized programs.

Today's programming languages reflect the nature of conventional computers. An intrinsically parallel problem (e.g., matrix operations or the processing of a multimedia data stream) is represented as a cascade of iterated loops, with simple, scalar computations at the center of the loops. Smart compilers for machines with parallel hardware (e.g., supercomputers) work hard to abstract these loops back into the parallel operations they represent to generate efficient code for parallel or vector hardware.

But as supercomputer vendors learned, it's a lot easier to generate good parallel code by starting with programs in which the parallelism is clearly expressed, rather than obscured by the programmer in iterative loops that force the compiler to rediscover it. The same wisdom will apply to IA-64: the programs that benefit most from parallel capabilities will be the ones written to make the parallelism clear. This will often require some thought and effort—effort that will be rewarded with exceptional performance.

Intel and HP will provide excellent support for existing binary code through some combination of direct hardware support and software preprocessing. We don't expect these binaries to take much advantage of additional IA-64 parallelism. Source code that is recompiled to run natively on IA-64 should do considerably better, but we expect some explicit parallelization of algorithms will be needed to take full advantage of IA-64's parallel capabilities.

The MMX additions to IA-32 (x86) are similar. MMX is a parallel-instruction enhancement limited to operations suitable for media processing. I believe the impact of MMX will be larger than many expect. Few programs have obvious MMX needs, but the potential for parallelism in media algorithms is greater than one might expect. MMX can accelerate printer drivers and rendering engines as well as MPEG decoding. IA-64's parallelism will support a wider set of data types and be more broadly usable than MMX, appearing in data-access and searching applications. We don't expect such parallel processing to accelerate most programs, but we do expect it to accelerate many of those that need it the most.

Running Existing Binaries on IA-64

IA-64 will execute x86 and PA-RISC binaries faithfully. HP is expert at moving binary code forward transparently, given its experience moving from two CISC architectures to PA-RISC. Intel and Microsoft have learned a lot about compatibility with existing x86 binaries as their CPU and OS designs evolve. Apple's emulation of 68K code on the PowerPC and Digital's clever FX!32 emulation clearly demonstrate the value and practicality of supporting existing binaries.

The question of performance on existing code is an interesting one. When Apple introduced its PowerPC Macs with 68K emulation, PowerPCs were so much faster than 68Ks that emulated programs ran as quickly as on a fast 68K. That was more than fast enough for Apple's customers. The same will apply to Merced: almost all binaries will run as fast on Merced as they did on the x86 processor for which they were written. Although x86 code will present a tougher challenge, improvements in emulation technology will boost Merced's performance on old applications.

Merced will have the additional benefit of being a mainstream processor. Digital had to develop FX!32 from scratch because there isn't a great incentive for people to recompile Windows applications to run natively on Alpha. As IA-64 becomes broadly deployed by Intel, the incentive to produce a native version will be much higher.

HP will be concerned with supporting PA-RISC binaries and with making x86/Windows code run. HP has a long and distinguished history in assuring binary compatibility as generations move forward, so it knows how to take care of existing customers. The ability of IA-64 to provide good x86 execution is just icing on the cake for HP.

IA-64 Has Limitations

Despite the innovations and improvements in IA-64, it won't sweep away all competitors instantly. Intel has an institutionalized way of introducing new processors at the price and performance top and then letting them trickle down over time. We expect the introduction of IA-64 to be no different. Merced will be big and expensive, but will get smaller and cheaper over time. It will take time for the software industry to understand how to take full advantage of the new capabilities; it always does. And for most people, a low-cost x86 PC will be more than enough by the time IA-64 is introduced.

These performance improvements will probably come with some penalty in code density, as was the case for RISC compared with CISC. Where code density is a key issue (e.g., ROM-based applications or very inexpensive computers) the acceptance of IA-64 will be slow.

Finally, IA-64 programs will probably have to be reoptimized for each different implementation of the architecture, continuing on the processor-specific binary path the industry has been on since the System/360 or VAX promoted universal instruction codes throughout the family. The technical issues this raises can be handled transparently (for instance, by using a distribution format that is optimized for a specific processor at the time the software is loaded), but it will require additional system technology to manage.

Learning from the Past, Designing for the Future

Merced and IA-64 are coming, and their impact will be profound. There is little doubt that significant architecture advances can and will occur. IA-64 processors will get more work done on each clock cycle through increased on-chip parallelism and will run at faster clock speeds because of reduced complexity.

It seems the prevailing wisdom regarding microprocessor architecture is like a pendulum, swinging first one way and then another. Once-simple designs have become ever more complex. The advent of IA-64 heralds the beginning of the swing back, shifting complexity to the compiler to create a simpler and faster microprocessor. M