Cahners
MICROPROCESSOR REPORT
www.MPRonline.com
THE INSIDER'S GUIDE TO MICROPROCESSOR HARDWARE

GUEST VIEWPOINT: IS OUT-OF-ORDER OUT OF DATE?

IA-64's Parallel Architecture Will Improve Processor Performance

By William S. Worley Jr., HP Labs, and Jerry Huck, IA-64 Architecture Lab {2/7/00-02}

Microprocessors are on a relentless path to higher performance. Every innovation in computing—data mining, Java programming, distributed computing on the Internet, multimedia data streams, and so on—invariably requires greater computing power. Even traditional database processing and technical computing have increasing problem sizes that drive demand for higher-performance microprocessors.

To meet these and other future requirements, new approaches to exploit improvements in IC processes are needed. Today, nearly all microprocessors exploit parallelism to accomplish more work in less time. We believe that parallelism can best be exploited with a computer architecture that is designed from the ground up to support instruction-level parallelism (ILP). We have termed this style of architecture EPIC, for explicitly parallel instruction-set computing. IA-64, developed jointly by HP and Intel, is such an architecture (see *MPR 5/31/99-01*, "IA-64: A Parallel Instruction Set").

IA-64 enables the compiler to express more parallelism to the machine than is possible with existing RISC or CISC architectures. As a result, IA-64 significantly reduces the hardware cost of detecting and scheduling the parallelism among instructions. The ability to specify this parallelism directly is one of IA-64's primary advantages.

Through the 1980s, RISC and CISC architectures were not designed primarily for high ILP. Instead, they were designed to make the best use of the technology that was available at the time. The RS/6000 and Alpha architectures adopted similar computing resources and instruction set formats. The inability of RISC and CISC architectures to express parallelism directly can be overcome to some degree by adopting complex, nonarchitectural approaches, principally out-of-order (OOO) dynamic superscalar hardware. Although preserving customer investments and supporting an installed system base are good business reasons for developing such processors for legacy architectures, the advent of IA-64 eliminates the performance of OOO hardware as a compelling technical reason to do so.

The growing market requirement for a higher-performance 64-bit architecture and absence of existing Intel 64-bit binaries gave HP and Intel an opportunity to create something new. The companies took advantage of this opportunity—along with lessons learned from the past 15–20 years of computing evolution—to create a new architecture with performance characteristics superior to those of existing RISC and CISC architectures.

Not Just for ILP

A microprocessor that minimizes computation cycles makes a better building block for high-performance systems. Such a microprocessor can, for example, be replicated to build multiprocessor systems. Fast CPUs in an MP configuration reduce queuing and contention, and they provide greater overall throughput than a larger

number of slower processors, a fact that we have seen many times in OLTP benchmarks in which small numbers of PA-RISC processors outperformed larger numbers of slower processors.

The view that parallelism is achieved most effectively through higher levels of multiprogramming is unproved, and, to the extent that it may be true, does not provide the complete picture. It fails to appreciate the fact that parallelism must be improved at all levels of a system. Providing parallelism solely through hardware-based multithreading, simultaneous multithreading (SMT), or chip-level multiprocessing (CMP) cannot compensate for the lack of parallelism in the basic processing element. This is obvious for single-threaded code, but it is true even for some multithreaded code, such as the encryption codes mentioned later.

SMT and CMP apply equally to RISC, CISC, and EPIC microprocessors. The first paper design of an EPIC machine at HP labs in 1991 envisioned integrated hardware multithreading as an orthogonal complement to EPIC architecture capabilities. But SMT has its downsides. The nonlinear nature of caches can be a problem for some workloads. Instead of one thread thrashing the cache on one processor, one thread on an SMT processor can thrash the cache for all threads on the processor. Finding the best design for multiple working sets to share a single cache is a research problem that has generated several papers but, as yet, no clear solutions.

Effective utilization of the hardware resources of a modern microprocessor is difficult. Historically, we have found that doubling the number of function units of a RISC processor has resulted in less-than-linear scaling. IA-64 was specifically designed to utilize additional function units effectively. Defenders of superscalar RISC architectures argue that out-of-order processing is the best means to achieve high function-unit utilization.

Building an out-of-order processor, however, is complex and difficult. The original PA-8000, for example, used as many transistors in its reorder buffer as were used in the entire previous-generation PA-7200 chip. Most current-generation OOO engines are four-issue implementations, and our studies indicate that the complexity of these machines will scale quadratically for 1.5× or 2× increases in issue width. In contrast, the first member of the IA-64 family—Itanium (nèe Merced)—is already a six-issue machine.

Architecture vs. Implementation

Architectural influences are determinative for many parts of an implementation, but not for all. The speed of an ALU, for example, is primarily a function of IC process and word width. The repertoire of operations that the ALU can perform, however, is a second-order issue. In addition, the data-cache hierarchy and the memory system of RISC and EPIC processors are largely independent of the architecture. Cache and memory-system interfaces must meet the demands of the processor—be it RISC, CISC, or EPIC. Criticizing the IA-64 architecture on the basis of an initial memory-system design, which was chosen to balance cost and performance, is a bit unfair.

Some assert that memory systems can be more fully utilized by OOO RISC designs. Our analysis of contention, cache behavior, buffer queuing, processor affinity, memory interleaving, and other factors indicates that one can find better approaches to use available memory technology if one is willing to accept additional cost.

The IC process, the number of registers, the number of register ports, the bypass network, and the number of cache ports are the principal factors in determining the cycle time of an IA-64 processor. Any RISC or CISC design faces similar challenges. The critical path in many modern microprocessors, IA-64 processors included, is found in the function units and their bypass networks. But IA-64 processors distinguish themselves by higher utilization of this fundamental structure. As with all designs, a balance was sought among the clock rate, pipeline depth, and execution width.

The first IA-64 processors will be used in high-end servers and workstations. Over time, designs will broaden out to span a wide range of markets. An HP Labs study for high-performance embedded controllers has confirmed that EPIC-like machines are exceptionally effective.

IA-64's Parallelism Features

Several IA-64 capabilities express and enhance parallel execution. The first is predication, which reduces the number of encountered branches, mispredicted branches, and other obstacles to finding parallelism. Our studies conclude that the gain from branch reduction more than compensates for any extra instructions that might be executed (Note: Do not think of a single if clause, think of merging three to five different basic blocks into a single, branchless, critical-path-limited sequence of code.). A mispredicted branch disrupts the pipeline. The lost opportunity can be measured as the width of the machine times the length in cycles of the mispredict penalty. For newer RISC processors, this is often more than 20 instruction slots.

IA-64 specifies a large register set (128 GP registers and 128 FP registers), a feature that allows algorithm design to be fundamentally changed. Matrix operations, finite-element analysis, and many other technical algorithms can be restructured to take better advantage of the large register space. OOO superscalar advocates argue that their internal register-renaming hardware can bring to bear just as many register resources as IA-64. Rename registers, however, are not as effective as real registers for either the programmer or the hardware.

Having only 32 architecturally visible registers, as most RISC architectures do, requires that a programmer structure his or her code in such a manner that, at any point in time, the registers containing program state do not exceed 32. For problems such as 1,024-bit RSA encryption, one can effectively use many more than 32 general registers and 32

floating-point registers. Holding just two 512-bit operands and one 1,024-bit intermediate result fills 32 64-bit registers. For a RISC or CISC processor, the RSA program would require many housekeeping load and store instructions whose only productive functions are to limit the instantaneous general- and floating-point-register state. Although the underlying hardware may have many more internal rename registers, the programmer has no direct means to use these registers to hold program state. Two of the AES-study codes, mentioned later, used over 60 general registers. Having this many architecturally visible resources, and the IA-64 register rotation, enables coding strategies that simply are not possible with an architected set of only 32 registers.

On the hardware side, even though OOO superscalar implementations normally have extra internal registers, during every cycle they must make visible only the 32 registers of program state. Furthermore, in the event of an interruption, the hardware must be prepared to lose (and later perhaps partially reconstruct) all the nonvisible internal state. Thus, as is the case for the programmer and for the executable code, the hardware's use of additional register resources is handicapped by the need to maintain the fiction that the sole register state is that constituted by the 32 registers.

Most significant, IA-64 introduces a collection of features to deal with memory latencies, which continue to increase relative to processor speeds. IA-64's control and data speculation capabilities enable compiler-directed access to variables at points much earlier than they are needed for computation. This capability permits a greater degree of concurrency between executing instructions and memory accesses. OOO engines achieve a similar effect with dynamic hardware, but they are restricted to fixed hardware algorithms for correctly predicting the execution path and for triggering memory fetches. HP's analysis of the PA-8000 shows that the primary benefit of OOO operation in commercial workloads lies in initiating multiple earlier cache misses. IA-64 enables such acceleration on an even broader scale.

Involving the compiler in the process of identifying speculative load candidates opens a bigger window into the program than can practically be achieved by an OOO superscalar processor. Data profiling makes the compiler even more accurate at selecting variables for speculative handling. The IA-64 compiler has heuristics to control the degree of speculation, and the programmer has control over the compiler's heuristics.

Another important feature of IA-64 is its register stack engine (RSE). One might consider this feature a built-in asynchronous hardware thread that runs when there would otherwise be idle memory ports. This feature reduces the cost of procedure calls and returns and increases the utilization of the register file. It is especially valuable for accelerating call-intensive object-oriented code. The reduction in the time to spill and fill the general register file is significant for many applications. On one database benchmark, for example, RISC processors spend about 30% of their memory references for procedure entry/return housekeeping. Most of this overhead is eliminated by IA-64's RSE. The IA-64 architecture has been crafted carefully to make the hardware design of the RSE straightforward. The RSE does not add to the critical path of the machine and is a relatively small part of the Itanium and McKinley designs.

Putting It All Together

All of the IA-64 elements mentioned above combine synergistically to minimize the critical code path through a program. The size of the resulting program binary image may be larger, but IA-64's instruction stream is more linear, i.e., it contains fewer branches. Itanium and McKinley both compensate for this code growth with special mechanisms that efficiently deliver instructions to the processor. These mechanisms eliminate the effects of the increased code size with only modest area and design costs.

In a paper submitted to the NIST AES3 (advanced encryption standard) Conference by HP Labs researchers, the five final AES algorithms were analyzed for both PA-RISC and IA-64. This study shows that IA-64's register file and its wide parallel architecture are very effective. The full issue width of the machine was utilized by some of these codes. Not surprisingly, the two algorithms with the greatest theoretical parallelism—Twofish and Rijndael—showed the greatest function unit utilization. Eight of the 15 IA-64 codes (encryption, decryption, and keying for each of the five finalists) used more than 32 registers. Six of the 15 IA-64 codes had smaller code sizes than PA-RISC, due to the compact modulo-scheduling support in IA-64. In two cases, the code was more than four times smaller. Overall, the IA-64 code size was only 27% larger than PA-RISC, even though no explicit effort had been made to minimize code size on either architecture. (A quick look at the IA-64 code showed that the difference could have been reduced to the 10% range.)

Although these are not typical codes, they illustrate how RISC code compares with IA-64 code using identical goals and algorithms. Equivalent comparisons using compiled code are not yet available, because IA-64 compilers are still maturing. The AES study goes into detail on mapping the final AES algorithms onto PA-RISC and IA-64 machines and architectures. Features like predication and rotation played important parts in reducing IA-64 critical code paths and exposing parallelism. In one simple example, recurrence was trivially handled by referencing back into the rotating-register region—no extra copy or unrolling was needed.

IA-64 Compilers Find Parallelism

IA-64 builds on proven compiler techniques to extract parallelism from applications. Many of these techniques are used by existing compilers to gain the best performance for today's OOO RISC systems. These techniques include data prefetch,

branch hints, loop unrolling, and profile-based path structuring, as well as other well-established optimizations.

As an example, OOO RISC compilers generate better code when using profile results. On the PA-8000, reducing the number of taken branches, through profiling, improves branch prediction. This is especially valuable in large commercial codes, where branch prediction tables are not very effective.

Historically, every major improvement in instruction-level parallelism has required the use of new code-generation techniques. Only with such techniques can the compiler realize significant performance gains. The PA-8000 and Alpha 21264, for example, used new binaries to get optimal performance. The initially-hoped-for transparent performance gains from OOO superscalar machines have not materialized. Meticulous code scheduling by the compiler has proved just as essential for OOO engines as it will be for IA-64.

Code profiling is only slightly more important for IA-64 code generation than it is for an OOO processor. Without profiling, an OOO engine can easily get lost down the wrong path, due to branch mispredictions and false dynamic speculation—especially in large-footprint applications. Instructions that will not be executed are cached, and in-flight instructions are canceled.

As a further example, the performance of the specFP95 benchmark is greatly improved by inserting prefetch instructions. The OOO engine is not effective in automatically triggering the proper prefetches. OOO queues are generally not deep enough and do not understand which data will be needed. On the other hand, the compiler is able to analyze the data layout and trigger the best prefetches.

Compiler writers, and those who have hand-coded for OOO machines, talk of the frustration in understanding how to second-guess the limitations of the OOO hardware and work around them to achieve full utilization of the function units. The job usually boils down to trial and error. This process actually occurred for PA-RISC codes during the AES study.

Another significant issue with OOO design is sustaining the most critical memory references in flight. To the OOO engine, everything is equal. The IA-64 compiler, on the other hand, is able to locate the critical path through the code and to ensure that the important long-latency operations are started first. Memory buffer and other limitations will always mandate executing critical path instructions first. As noted in an earlier paragraph, the compiler will issue prefetches to ensure early initiation of the most-likely cache misses.

To complement the compiler's expanded ability to avoid cache misses, hardware resources still can be brought to bear in an IA-64 implementation. As an example, we have developed several strategies to improve the handling of cache misses. An in-order machine has options beyond a simple stall when a cache miss occurs. Running ahead with rollback, predictive address buffers, implicit prefetch, buddy prefetch, and other dynamic approaches can all be effective. Quantitatively, these techniques are far simpler than those used in OOO designs.

Static code generation is just one aspect of producing efficient code. The recent announcement of the Crusoe processor by Transmeta hints at the benefits of dynamic code generation. In many venues, researchers have been examining the significant performance improvements that can be achieved by dynamic measurement and tuning of code.

HP, for example, implemented a simple mechanism that sampled the current instruction during the normal timer interrupt. If the instruction was a kernel branch, the branch hint was rewritten to match the actual program flow. This one dynamic mechanism resulted in a 5% improvement for database applications.

More aggressive approaches are possible. The Itanium processor is able to watch a program's cache misses, and software can apply a simple heuristic to set prefetch hints correctly. This type of explicit approach, driven by runtime information, is extremely accurate and selective. Better performance feedback enables greater tuning accuracy and better utilization of the machine resources. IA-64 has hint fields in most branch and memory-reference instructions. Armed with runtime information for the executing program, hint fields can be rewritten on the fly to match the needs of the current workload.

These approaches allow software to tune code dynamically for performance. With such techniques, and without recompilation, performance can continue to evolve and improve, even after a machine and application has been put into production. IA-64 provides a greater range of tuning options than previous architectures.

Future Directions

The first IA-64 microprocessor, Itanium, exhibits ILP beyond that achievable by any OOO superscalar microprocessor in existence. Extensive study and analysis went into the EPIC architectural ideas, and they have been found to work. Itanium is just the initial implementation of the IA-64 architecture. It delivers the powerful EPIC innovations while providing complete binary compatibility with IA-32 and PA-RISC.

Future designs will be even more powerful. We are now at just the beginning of the IA-64 hardware and software implementation learning curves. As was the case for RISC and CISC architectures, the IA-64 architecture will evolve and become even more powerful. As was the case for RISC and CISC implementations, IA-64 implementations will mature and evolve through successive generations. And, as was the case for RISC and CISC software, the EPIC compilers will become better and better at exploiting the full capabilities of the architecture.

Any initial implementation of a new architecture will be conservative and will concentrate on the most important architectural elements. Itanium and McKinley are not

exceptions to these rules. The initial version of the IA-64 architecture by no means encompasses all the innovations and ideas developed by HP and Intel. The initial hardware implementations by no means embody all the techniques and designs envisioned by HP and Intel. Since the early 1990s, HP Labs has been evaluating scalability for high ILP, multiprocessing, hardware multithreading, and high-bandwidth memory systems. HP has also considered many static and dynamic compilation and simulation techniques.

IA-64 Will Deliver More ILP and Performance

IA-64 will deliver on its promise of expressing, enhancing, and exploiting instruction-level parallelism to improve performance. The IA-64 architecture will not remain static or fixed, and successive generations of processors will each introduce innovations. By the time we have third- and fourth-generation chips, we are confident that the present architectural controversy will have passed, and EPIC will have proved its superiority.

Bill Worley is a principal architect on two HP architectures: PA-RISC and PA Wide-Word, the later of which became the basis for HP's collaboration with Intel on IA-64. In 1995, Bill was named a distinguished contributor, the highest technical position at HP. Bill can be contacted at worley@hpl.hp.com.

Jerry Huck manages processor architecture in HP's computer products organization. These days his time is split between working with Intel on the IA-64 architecture and managing the team responsible for processor simulator and platform architecture definition. Jerry's team was also responsible for the evolution of the PA-RISC architecture. Jerry can be reached at jerry_huck@hp.com.

To subscribe to Microprocessor Report, *phone 408.328.3900 or visit* www.MDRonline.com