IA-64 Architecture Innovations

February 1999

Joint Whitepaper between Intel & HP

Background

New workloads such as internet communications, e-commerce, Decision Support Systems, In Memory Databases, Digital Content Creation, and Compute intensive CAD etc. place heavy demands on high performance servers and workstations. While the performance of today's processors continues to improve, existing architectures based on an out-of-order execution model, require increasingly complex hardware mechanisms and are increasingly impeded by performance limiters such as branches and memory latency. Intel's IA-64 architecture is a unique combination of innovative features, which overcomes the performance limitations of traditional architectures. The IA-64 architecture is based on innovative techniques/features such as Explicit Parallelism, and Predication and Speculation, resulting in superior Instruction Level Parallelism (ILP) and increased instructions per cycle (IPC) to address the current and future requirements of these demanding Internet, high end server, and workstation applications. In addition, the IA-64 architecture provides headroom and scalability for continued future growth.

Continuing from the original IA-64 architectural disclosure in 10/97, Intel and Hewlett-Packard disclosed new details behind the innovative predication and speculation features and IA-64's unique way of handling software pipelining. Combining all of the above results in an architecture that goes well beyond today's traditional architectures. Since the original disclosure, RISC vendors have often unsuccessfully compared the "bolt –on " features offered in traditional processors to those of IA-64. Comparisons of features such as "cmove" and "non-faulting loads" generate overhead that often outweighs their benefits compared to the innovative IA-64 features.

Predication enhances parallelism

Branches are a severe performance challenge to out of order RISC architectures. Traditional architectures use a technique called branch prediction to predict the correct path. Mispredicts commonly occur with this technique about 5-10 % of the time resulting in rather severe performance penalties of as much as 30-40 %. IA-64 uses a technique called Predication to execute both paths in parallel to overcome the limitations of traditional architectures. 64 one-bit predicate registers are used to "switch off" a branch that is seen to be untrue. IA-64 also provides efficient support for multi-way branches and parallel compares, both features allow n-way branching in one machine cycle, significantly speeding execution. In all cases, the predicate indicates whether a particular branch and its associated data values are "turned on" or "turned off". Branches which are "turned on" continue to execute, and as soon as any branch is "turned off" execution ceases. Predicates, which are part of every IA-64 conditional, provide a very efficient way to "turn off" execution of a branch that is not taken, set up and exit from a loop, handle dynamic prediction and do n-way compares. Predicates help handle the complex control flow issues that spring up when the compiler pursues really aggressive instruction-level parallelism.

Predication reduces branches and associated mispredicts thereby increasing performance. Predication is particularly useful in applications with hard to predict branches like large database, data mining, data warehousing etc.

Lacking the mechanism of predication, it is not surprising that traditional RISC architectures have had little success with Instruction-level parallelism. Existing RISC architectures that use 'cmoves' and similar instructions may remove branches, but at the cost of adding so many instructions that the benefits are nearly outweighed by the code-bloat. When knowledgeable observers analyze the effort of traditional RISC architectures to increase Instruction Level Parallelism (ILP) through 'bolt-on' instructions such as 'cmove', they correctly conclude that the pursuit of ILP within these architectures is hardly worth the trade-off. The reason why ILP works with IA-64 is the use of completely new architectural constructs such as predicates that are not available to any existing RISC architecture.

Speculation minimizes the effect of memory latency

In the current environment, the CPU's are getting faster at a much higher rate than the memory. We expect this trend to continue in the future, possibly at even a faster rate thereby increasing the gap between the CPU and memory speeds. Thus the latency of memory is a big performance bottleneck in today's systems.

To reduce the impact of this limitation, traditional architectures allow compiler and the processor to schedule loads before the data is needed, but branches act as a barrier to load hoisting.

IA-64 architecture employs a technique called as speculation to initiate loads from memory earlier in the instruction stream – even before a branch. This allows early retrieval of information from the memory so it is available when the "use" is required. Thus speculation increases instruction level parallelism and reduces the impact of memory latency resulting in handsome performance gains.

Control speculation is a feature that runs deep within IA-64. Traditional RISC architectures can use a 'non-faulting load' to avoid costly error handling when loading data ahead of time which may not be valid. But if you want to turn off the errors, why have errors in the first place? Isn't it possible that some errors are valid and will lead to serious problems if ignored? Traditional RISC architectures face one of two alternatives: add extra error-checking code which, once again, cancels out the performance benefit of speculative execution ; or 'work without a net,' risking disastrous undetected errors due to turning off the error messages. IA-64 gets around both problems by offering a novel architectural approach to dealing with errors when loading data.

The basic idea is that data values are associated with a bit which tells whether an error was generated when the data was loaded or not. This error ("Nat") bit follows the data through arithmetical operations, moves, and conditionals until the data is checked through a check instruction (an instruction which has been designed not to take an additional machine cycle). Only when the check instruction is executed will the error recovery process be started, and at least 99% of the time, loads which triggered errors turn out to have been

loading data that is never used, and the time-consuming error recovery routines are not needed.

This means that with IA-64, the compiler can aggressively load data well ahead of time (speculation) with out paying unnecessary exception penalties. With traditional RISC architectures, you either get error rates that cancel out the performance advantages of speculative execution or you use non-faulting loads and add error-detecting instructions that cancel out the advantages of speculative execution, or you use non-faulting loads without error detection and risk disaster. But with IA-64, the architectural device of “Nat“ bits allows IA-64 compilers to make aggressive use of speculation without paying the penalties required for all RISC architectures.

A final benefit of “Nats” is that it provides for structured error handling as is found in C and C++. Because the compiler can schedule the “check” instruction whenever it wants, it is possible to safely defer errors until the best possible time to recover from them. The architectural support for structured error handling in IA-64 is an important feature promoting system reliability and performance.

An additional form of speculation is called Data Speculation. This allows the compiler to schedule a load even before a store (overcoming the store barrier) that might write to the same register that the load reads from. IA-64 keeps track of all loads that are scheduled before possibly conflicting stores in a table called the Advanced Load Address Table (ALAT). Whenever a store is encountered that actually conflicts with a load in this table, the entry are removed. When the load is checked in the ALAT, the absence of an entry indicates that the value needs to be reloaded to get the correct result. This feature is unique to IA-64 and allows loads to be scheduled far enough ahead that the impact of memory latencies is greatly reduced.

A 1998 study by August et al. demonstrates that predication, control speculation and data speculation provide a performance improvement of 79% over an architecture without these features. While Speculation has broad applications benefits, it is particularly effective for code with frequent cache accesses. This is typical of operating systems, large databases, high end EDA and Compute Server based applications.

Loop Unrolling and Rotation

Efficient handling of loops is one of the key determiners of performance for all computer architectures and IA-64 provides special features to support it. Simply treating a loop as a simple branch, of course, yields poor performance. Branching takes time and since the compiler knows ahead of time what the branches are, many branches can be avoided by simply duplicating the code in the loop two, four, or even more times to reduce the number of branches and allow greater parallelism. This is called “unrolling” the loop and is supported in both IA-64 and RISC architectures.

The problem with simple loop unrolling by the compiler is that it leads to significant code bloat. IA-64 introduces a unique feature called Register Rotation that allows pipelining of loops with optimal efficiency and without code bloat.

In IA-64, both registers and predicates can rotate under control of the compiler. This means that with each rotation step, register values that were, for example, in registers 40 would advance to 41. Similarly, the associated predicate values might advance from predicate register 30 to 31. Naturally, values cycle back when they get to the highest value, which is why they are said to be “rotating”. With rotation, the compiler can run one copy of the code inside the loop and the values are assigned automatically to successive registers as the code iterates. When the loop is finished, with the help of dedicated “loop count” and “epilog count” registers, the predicates are flipped to zero and the loop exits. IA-64 handles loops without the need of “prolog” or “epilog” overhead at the beginning and end of the loop to handle setup and cleanup, and without any performance-limiting branch mispredictions as you would see with many RISC architectures.

Finally, since the compiler is fully in charge of the loop pipelining, it can apply speculation, hoisting both loads and uses of data to provide the most efficient execution. Furthermore, with four times the number of registers provided with traditional RISC architectures, IA-64 has the data working area needed for high performance loop execution. The result is IA-64 can apply the benefits of loop pipelining much more widely- both to integer code, which typically contains many small loops, as well as huge floating point loops.

Massive Resources

Traditional architectures, in addition to restricted instruction level parallelism, also have relatively fewer hardware resources to support the parallel execution. In contrast, IA-64 architecture based processors include 128 general purpose integer registers, 128 floating point registers, 64 predicate registers and many execution units to ensure serving of today's/future's demanding workloads. The architecture is inherently scalable allowing for easy expansion of the number of hardware execution units and increased parallel execution thereby providing maximum headroom for scalability with full compatibility.

Conclusion:

The IA-64 implementation of EPIC design philosophy enables new levels of parallelism and breaks the sequential execution paradigm (one instruction at a time) that exists with traditional architectures. The innovative use of predication and speculation uniquely combined with explicit parallelism allows IA-64 to progress well beyond the limitations (like mispredicted branches and memory latency) of traditional architectures- enabling industry leading performance. The architecture has been designed with new levels of headroom to meet the needs of challenging workloads of the future. The inherently scalable nature of the architecture makes it very compelling for the high-end server and workstation market segments.

The key points of this disclosure have been that predication and speculation are more than simple features to improve performance—below them lie sophisticated mechanisms for control flow handling and error management which permit a level of parallelism and performance beyond traditional RISC processors. IA-64 architecture incorporates a unique technique called Register rotation that enables increased loop performance without the downside of code bloat.

RISC architectures claim to match many of the features of IA-64 with similar sounding instructions. However, just like a tank formed by bolting weapons and armor to an old truck, the benefits are limited to specific conditions, but fall short in the heat of battle.