Montecito: A Dual-Core, Dual-Thread Itanium Processor

Intel's Montecito is the first Itanium processor to feature duplicate, dual-thread cores and cache hierarchies on a single die. It features a landmark 1.72 billion transistors and server-focused technologies, and it requires only 100 watts of power.

Cameron McNairy
Rohit Bhatia
Intel

•••••• Intel's Itanium 2 processor series has regularly delivered additional performance through the increased frequency and cache as evidenced by the 6-Mbyte and 9-Mbyte versions.[1] Montecito is the next offering in the Itanium processor family and represents many firsts for both Intel and the computing industry. Its 1.7 billion transistors extend the Itanium 2 core with an enhanced form of temporal multithreading and a substantially improved cache hierarchy. In addition to these landmarks, designers have incorporated technologies and enhancements that target reliability and manageability, power efficiency, and performance through the exploitation of both instruction- and thread-level parallelism. The result is a single 21.5-mm ¥ 27.7-mm die[2] that can execute four independent contexts on two cores with nearly 27 Mbytes of cache, at over 1.8 GHz, yet consumes only 100W of power.

Beyond Itanium 2

Figure 1 is a block diagram of the Itanium 2 processor.[3] The front end—with two levels of branch prediction, two translation lookaside buffers (TLBs) and a zero-cycle branch predictor—feeds two bundles (three instructions each) into the instruction buffer every cycle. This eight-entry queue decouples the front end from the back and delivers up to two bundles of any alignment to the remaining six pipeline stages. The dispersal logic determines issue groups and allocates up to six instructions to nearly every combination of the 11 available functional units (two integer, four memory, two floating point, and three branch). The renaming logic maps virtual registers specified by the instruction into physical registers, which access the actual register file (12 integer and eight floating-point read ports) in the next stage. Instructions then perform their operation or issue requests to the cache hierarchy. The full bypass network allows nearly immediate access to previous instruction results while the retirement logic writes final results into the register files (10 integer and 10 floating-point write ports).

Figure 2 is a block diagram of Montecito, which aims to preserve application and operating system investments while providing greater opportunity for code generators to continue their steady performance push. This opportunity is important, because even three years after the Itanium 2's debut, compilers continue to be a source of significant performance improvement. Unfortunately, *compiler-*

optimization compatibility—which lets processors run each other's code optimally—limits the freedom to explore aggressive ways of increasing cycle-for-cycle performance and overall frequency.

Single-thread performance improvements

Internal evaluations of Itanium 2 code indicate that static instructions per cycle (IPC) hover around three and often reach six for a wide array of workloads. Dynamic IPC decreases from the static highs for nearly all workloads. The IPC reduction is primarily due to inefficient cache-hierarchy accesses, and to a small degree, functional unit asymmetries and inefficiencies in branching and speculation. Montecito targeted these performance weaknesses, optimizing nearly every core block and piece of control logic to improve some performance aspect.

Asymmetry, branching, and speculation

To address the port-asymmetry problem, Montecito adds a second integer shifter yielding a performance improvement of nearly 100 percent for important cryptographic codes. To address branching inefficiency, Montecito's front end removes the bottlenecks surrounding single-cycle branches, which are prevalent in integer and enterprise workloads. Finally, Montecito decreases the time to reach recovery code when control or data speculation fails, thereby lowering the cost of speculation and enabling the code to use speculation more effectively.[4]

Cache hierarchy

Montecito supports three levels of on-chip cache. Each core contains a complete cache hierarchy, with nearly 13.3 Mbytes per core, for a total of nearly 27 Mbytes of processor cache.

Level 1 caches. The L1 caches (L1I and L1D) are four-way, set associative, and each holds 16 Kbytes of instructions or data. Like the rest of the pipeline, these caches are in order, but they are also nonblocking, which enables high request concurrency. Access to L1I and L1D is through prevalidated tags and occurs in a single cycle. L1D is write-through, dual-ported and banked to support two integer loads and two stores each cycle. The L1I has dual-ported tags and a single data port to support simultaneous demand and prefetch accesses. The performance levels of Montecito's L1D and L1I caches are similar to those in the Itanium 2, but Montecito's L1I and L1D have additional data protection.

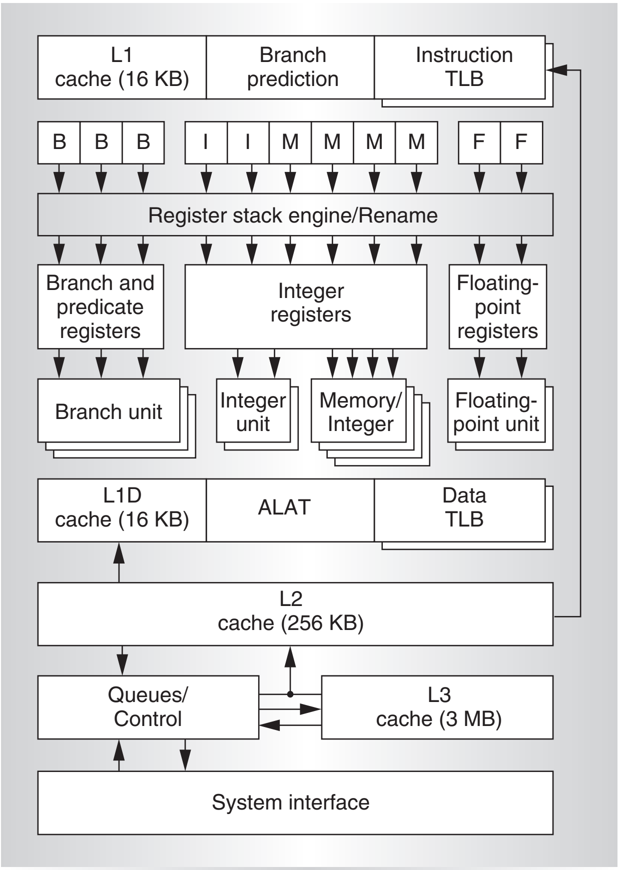


Figure 1. Block diagram of Intel's Itanium 2 processor. B, I, M, and F: branch, integer, memory, and floating-point functional units; ALAT: advanced load address table; TLB: translation look-aside buffer.

Level 2 caches. The real differences from the Itanium 2's cache hierarchies start at the L2 caches. The Itanium 2's L2 shares data and instructions, while the Montecito has dedicated instruction (L2I) and data (L2D) caches. This separation of instruction and data caches makes it possible to have dedicated access paths to the caches and thus eliminates contention and capacity pressures at the L2 caches. For enterprise applications, Montecito's dedicated L2 caches can offer up to a 7-percent performance increase.

The L2I holds 1 Mbyte; is eight-way, set associative; and has a 128-byte line size—yet has

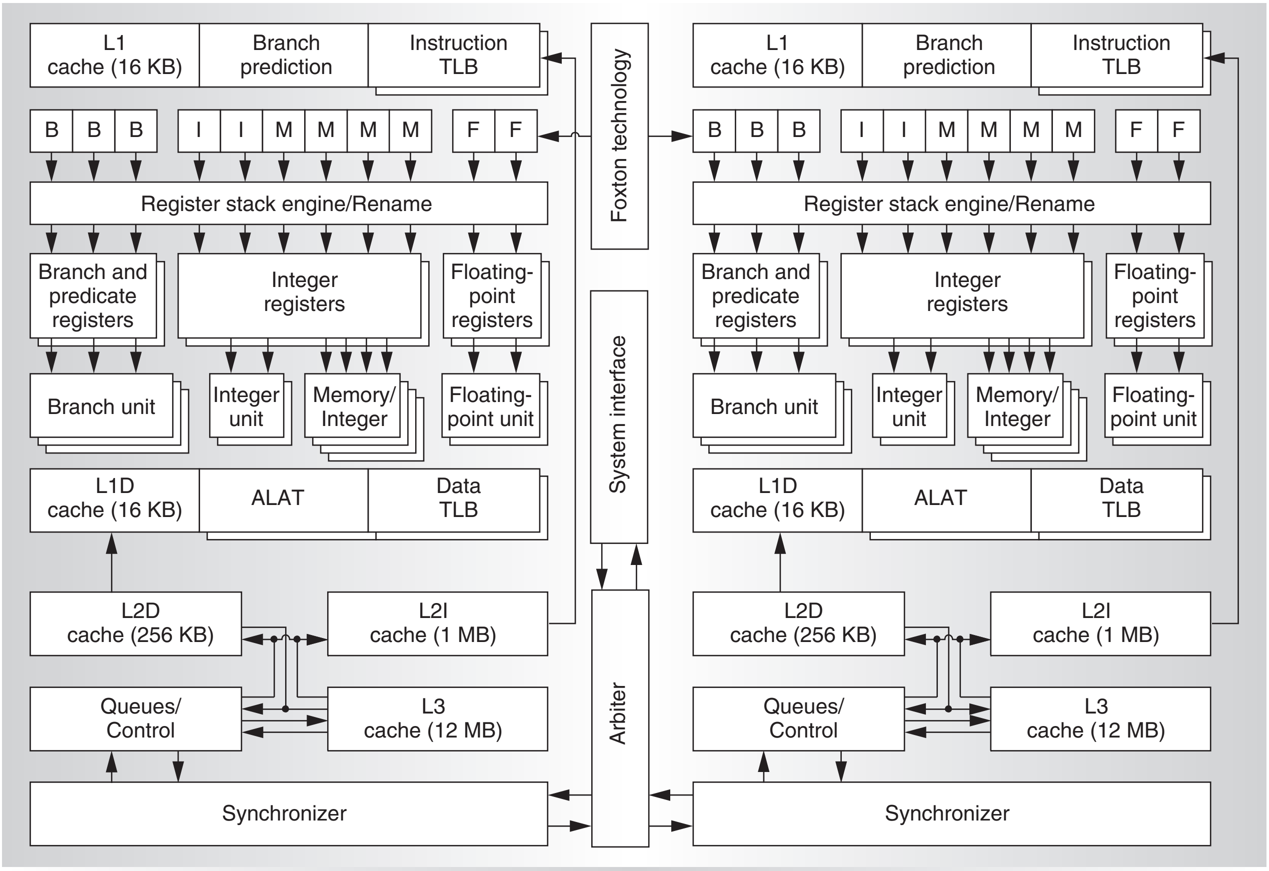


Figure 2. Block diagram of Intel's Montecito. The dual cores and threads realize performance unattainable in the Itanium 2 processor. Montecito also addresses Itanium 2 port asymmetries and inefficiencies in branching, speculation, and cache hierarchy.

the same seven-cycle instruction-access latency as the smaller Itanium 2 unified cache. The tag and data arrays are single ported, but the control logic supports out-of-order and pipelined accesses, which enable a high utilization rate.

Montecito's L2D has the same structure and organization as the Itanium 2's shared 256-Kbyte L2 cache but with several microarchitectural improvements to increase throughput. The L2D hit latency remains at five cycles for integer and six cycles for floating-point accesses. The tag array is true four-ported—four fully independent accesses in the same cycle—and the data array is pseudo-four-ported with 16-byte banks.

Montecito optimizes several aspects of the L2D. In the Itanium 2, any accesses to the same cache line beyond the first access that misses L2 will access the L2 tags periodically until the tags detect a hit. The repeated tag queries consume bandwidth from the core and increase the L2 miss latency. Montecito suspends such secondary misses until the L2D fill occurs. At that point, the fill immediately satisfies the suspended request. This approach greatly reduces bandwidth contention and final latency. The L2D, like the Itanium 2's L2, is out of order, pipelined, and tracks 32 requests (L2D hits or L2D misses not yet passed to the L3 cache) in addition to 16 misses and their associated victims. The difference is that Montecito allocates the 32 queue entries more efficiently, which provides a higher concurrency level than with the Itanium 2.

Level 3 cache. Montecito's L3 cache remains unified as in previous Itanium processors, but is now 12 Mbytes. Even so, it maintains the same 14-cycle integer-access latency typical of the L3s in the 6- and 9-Mbyte Itanium 2 family. Montecito's L3 uses an asynchronous interface with the data array to achieve this

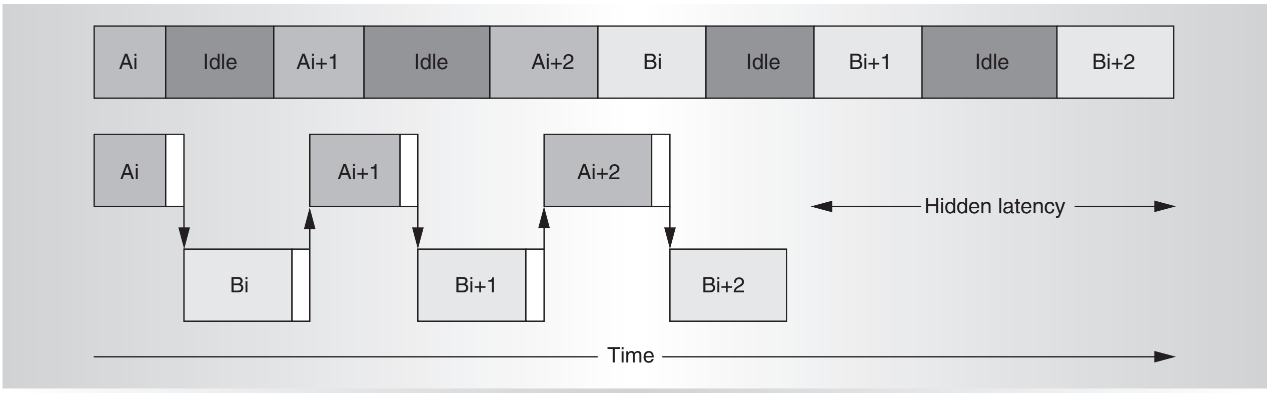


Figure 3. How two threads share a core. Control logic monitors the workload's behavior and dynamically adjusts the time quantum for a thread. If the control logic determines that a thread is not making progress, it suspends that thread and gives execution resources to the other thread. This partially offsets the cost of long latency operations, such as memory accesses. The time to execute a thread switch—white rectangles at the side of each box—consumes a portion of the idle time.

low latency; there is no clock, only a read or write valid indication.[5] The read signal is coincident with index and way values that initiate L3 data-array accesses. Four cycles later, the entire 128-byte line is available and latched. The array then delivers this data in four cycles to either the L2D or L2I in critical-byte order. The asynchronous design not only helps lower access time, but also significantly reduces power over the previous synchronous designs.

Montecito's L3 receives requests from both the L2I and L2D but gives priority to the L2I request in the rare case of a conflict. Conflicts are rare because Montecito moves the arbitration point from the Itanium 2's L1-L2 to L2-L3, which greatly reduces conflicts because of L2I and L2D's high hit rates.

Dual threads

Itanium 2 processors have led the industry in all three categories of performance: integer, floating-point, and throughput computing. However, the gap between processor speed and memory-system speed continues to increase with each semiconductor generation, and the additional memory latency typical of large systems only exacerbates the problem. For many integer and floating-point applications, instruction and data prefetching in combination with Montecito's fast and large cache hierarchy effectively bridge the processor-memory chasm. However, commercial transaction-processing codes, characterized by large memory footprints, realize only limited benefits from these techniques. In fact, although the L2I reduces instruction stream stalls, it increases the data stall component as a percentage of execution time.[6]

As Figure 3 shows, Montecito addresses the problem of ever-growing memory latency by offering two hardware-managed threads that reduce stall time and provide additional performance. The dual-thread implementation duplicates all architectural state and some microarchitectural state to create two logical processors that share all the parallel execution resources and the memory hierarchy. Resource sharing is a blend of temporal multithreading (TMT) for the core and simultaneous multithreading (SMT) for the memory hierarchy.

The cache hierarchy simultaneously shares resources, such as queue entries and cache lines, across the two threads, which gives each thread equal access to the resources. Montecito's designers chose not to pursue an SMT approach at the core level for several reasons. The most important is that internal studies show an average three and a maximum six IPC for integer and enterprise workloads, and an average five IPC for technical workloads before cache misses cause a degradation. Thus, the core utilization level is not the problem; rather it is decreasing the impact of memory accesses on IPC. Given the underlying in-order execution pipeline, the small potential return for supporting SMT at the core would not justify its cost. The improvements in the cache hierarchy, though intended to benefit instruction-level parallelism, accommodate the increased demand that dual threads create.

Other shared structures, such as branch-prediction structures and the advance load address table (ALAT), require special consideration to support the two threads. The branch-prediction structures use thread tagging so that one thread's predictions do not taint the other's prediction history. Each thread contains a complete set of return stack structures as well as an ALAT, since simple tagging would not be sufficient for these key performance structures.

Duplicating the complete architectural state represents the most significant change to the core and requires a core area (excluding the L3 cache) less than 2 percent larger than that for a single-thread implementation. Moreover, this growth, though mainly in the already large register files, does not affect the register file and bypass network's critical paths because Montecito includes a memory cell in which two storage cells share bit lines.[7]

Thread switch control

In-order core execution, coupled with the exclusive-resource-ownership attribute of temporal multithreading, lets Montecito incorporate two threads without affecting legacy software optimizations and with minimal changes to pipeline control logic. The competitively shared resources of the memory subsystem (caches and TLBs) required additional effort to ensure correctness and to provide special optimizations that avoid negative thread interactions. Part of this additional effort centered on thread-switch control.

The control logic for thread switching leverages the existing pipeline-flush mechanism to transfer control from one thread to the other. A thread switch is asynchronous to the core pipeline and is attached to the issue group of instructions in the exception-detection (DET) pipeline stage.[3] The switch becomes the highest priority event and causes a pipeline flush without allowing the issue group to raise faults or commit results. The thread-switch control logic asserts this pipeline flush for seven cycles for a total switch penalty of 15 cycles—seven more than the typical eight-cycle pipeline flush.

Five events can lead to a thread switch, many of which affect a thread's urgency—an indication of its ability to use core resources effectively:

- *L3 cache miss/data return.* L3 misses and data returns can trigger thread switches, subject to thread urgency comparison (described later). An L3 miss in the foreground thread is likely to cause that thread to stall and hence initiate a switch. Similarly, a data return to the L3 for the background thread is likely to resolve data dependences and is an early indication of execution readiness.
- *Timeout.* Thread-quantum counters ensure fairness in a thread's access to the pipeline execution resources. If the thread quantum expires when the thread was effectively using the core, thread-switch control logic switches it to the background and sets its urgency to a high value to indicate its execution readiness.
- *ALAT invalidation.* In certain circumstances, utilization and performance increase when a thread yields pipeline resources to the background thread while waiting for some system events. For example, spin-lock codes can allocate an address into the ALAT and then yield execution to the background thread. When an external access invalidates the ALAT entry, the action causes a thread switch back to the yielding thread. This event lets the threads share core-execution resources efficiently while decreasing the total time the software spends in critical sections.
- *Switch hint.* The Itanium architecture provides the `hint@pause` instruction, which can trigger a thread switch to yield execution to the background thread. The software can then indicate when the current thread does not need core resources.
- *Low-power mode.* When the active thread has transitioned to a quiesced low-power mode, the action triggers a thread switch to the background thread so that it can continue execution. If both threads are in a quiesced low-power state, an interrupt targeting the background thread awakens the thread and triggers a thread switch, while an interrupt targeting the foreground thread just awakens the foreground thread.

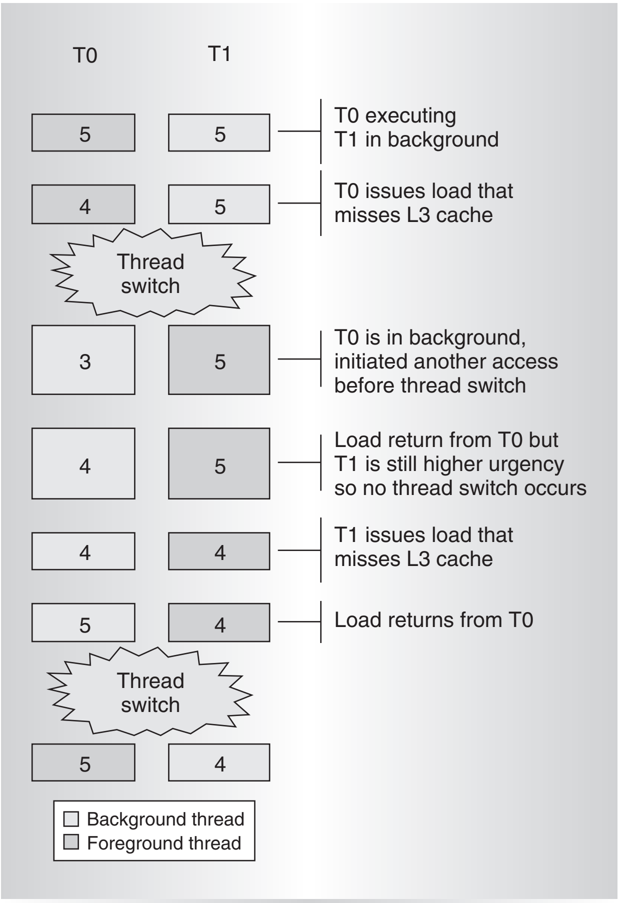
Given that reducing the negative impact of memory latency is the primary motivation for having two threads, the most common switch event is the L3 cache/data return. Other events, such as the timeout, provide fairness, while the ALAT invalidation and switch hint events provide paths for the software to influence thread switches.

Switching based on thread urgency

Each thread has an urgency that can take on values from 0 to 7. A value of 0 denotes that a thread has no useful work to perform. A value of 7 signifies that a thread was actively making forward progress before it was forced to the background. The nominal urgency value of 5 indicates that a thread is actively progressing.

Figure 4 shows a typical urgency-based switch scenario. Whenever the software experiences a cache hierarchy miss or a system interface data return (L3 miss/return event), the thread-switch control logic compares the urgency values of the two threads. If the urgency of the foreground thread is lower than that of the background thread, the control logic will initiate a thread switch. Every L3 miss decrements the urgency by 1, eventually saturating at 0. Similarly, every data return from the system interface increments the urgency by 1 as long as the urgency is below 5.

The thread-switch control logic sets the urgency to 7 for a thread that switches because of a timeout event. An external interrupt directed at the background thread will set the urgency for that thread to 6, which increases the probability of a thread switch and provides a reasonable response time for interrupt servicing.



T0
T1
5
5
T0 executing T1 in background
4
5
T0 issues load that misses L3 cache
Thread switch
3
5
T0 is in background, initiated another access before thread switch
4
5
Load return from T0 but T1 is still higher urgency so no thread switch occurs
4
4
T1 issues load that misses L3 cache
5
4
Load returns from T0
Thread switch
5
4
Background thread
Foreground thread

Figure 4. Urgency and thread switches on the Montecito processor.

Dual cores and the arbiter

As we described earlier, Montecito is a single die with duplicate dual-threaded cores and caches. The two cores attach to the system interface through the *arbiter*, which provides a low-latency path for each core to initiate and respond to system events.

Because the two cores share a system interface, the bandwidth demands for the shared-bus topology effectively double. Consequently, the system interface bandwidth increases from 6.4 gigabytes per second (GBps) with up to five electrical loads to 10.66 GBps with up to three electrical loads. Montecito systems can use the higher bandwidth three-load design while maintaining the computational density of the five-load configuration. The increased bandwidth easily absorbs the additional pressure that the dual threads put on the system interface.

Figure 5 is a block diagram of the arbiter, which organizes and optimizes each core's request to the system interface, ensures fairness and forward progress, and collects responses from each core to provide a unified system interface response. The arbiter maintains each core's unique identity to the system interface and operates at a fixed ratio to the system interface frequency. An asynchronous interface between the arbiter and each core lets the core and cache frequency vary as needed to support Foxton technology, which is key to Montecito's power-management strategy.

As the figure shows, the arbiter consists of a set of address queues, data queues, and synchronizers, as well as logic for core and system interface arbitration. (For simplicity, the fig-

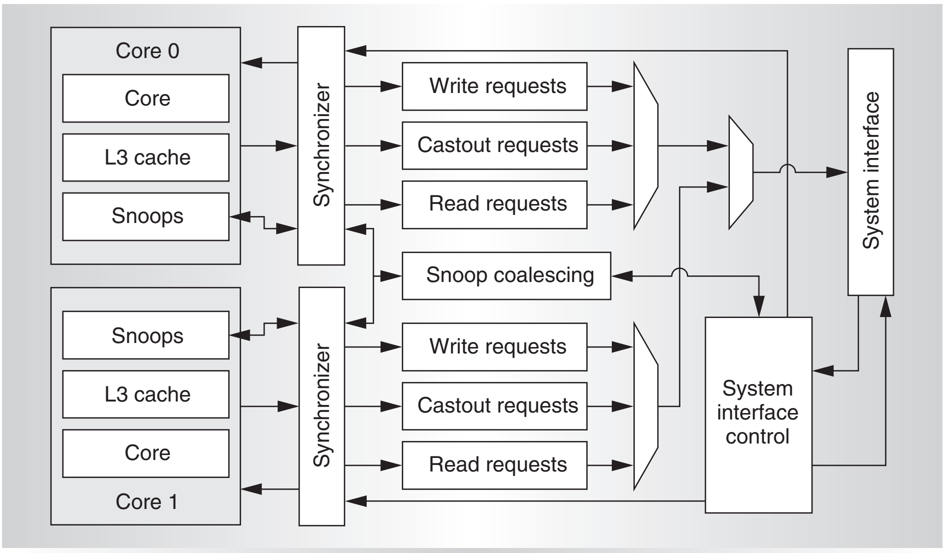


Figure 5. Montecito's arbiter and its queues.

ure omits the error-correction code (ECC) encoders/decoders and parity generators.) The arbitration logic ensures that each core gets an equal share of bandwidth during peak demand and low latency access during low demand.

The core initiates one of three types of accesses, which the arbiter allocates to the following queues and buffers:

- *Request queue.* This is the primary address queue that supports most request types. Each core has four request queues.
- *Write address queue.* This queue holds addresses only and handles explicit writebacks and partial line writes. Each core has two write address queues.
- *Clean castout queue.* This queue holds the address for the clean castout (directory and snoop filter update) transactions. The arbiter holds pending transactions until it issues them on the system interface. Each core has four clean castout queues.
- *Write data buffer.* This buffer holds outbound data and has a one-to-one correspondence with addresses in the write address queue. Each core has four write data buffers, with the additional two buffers holding implicit writeback data.

Other structures in the arbiter track and initiate communication from the system interface to the cores. The Snoop queue issues snoop requests to the cores as needed and coalesces the snoop response from each core into a unified snoop response for the socket.

The arbiter delivers all data returns directly to the appropriate core using a unique identifier provided with the initial request. It delivers broadcast transactions, such as interrupts and TLB purges, to both cores in the same way that delivery would occur if each core were connected directly to the system interface.

Fairness and arbitration

The arbiter interleaves core requests on a one-to-one basis when both cores have transactions to issue. When only one core has requests, it can issue its requests without the other core having to issue a transaction. Because read latency is the greatest concern, the read requests are typically the highest priority, followed by writes, and finally clean castouts. Each core tracks the occupancy of the arbiter's queues using a credit system for flow control. As requests complete, the arbiter informs the appropriate core of the type and number of deallocated queue entries. The cores use this information to determine which, if any, transaction to issue to the arbiter.

Synchronization

Given each core's variable frequency and the system interface's fixed frequency, the arbiter is the perfect place to synchronize the required communication channels. Montecito allocates all communication between the core and the arbiter to an entry in an array, or *synchronizer*, in that core's clock domain. The arbiter then reads from the array's entries in its own clock domain. These synchronizers can send data directly to the core without queuing when a synchronizer is empty.

With this design, the core can run much faster or much slower (within limits) than the arbiter depending on the system's and cores' power constraints. Communication from the arbiter to the core occurs in a similar, but reversed, manner using the synchronizers. A few signals do not need queuing, so simple synchronization for clock-domain crossing suffices.

Power efficiency

If the Montecito design team had taken the Itanium 2 power requirement of 130W as a baseline, implementing Montecito using a 90-nm process would have required power delivery and dissipation of nearly 300W.[2] And this excludes the additional power required from the increased resource use resulting from improvements targeting instruction- and thread-level parallelism.

Montecito, however, requires only 100W. Such a power constraint typically comes with severe frequency sacrifices, but as we described earlier, Montecito can execute enterprise applications using all the available cache, thread, and core features at over 1.8 GHz. Foxton technology is one of the key innovations that make this performance possible.

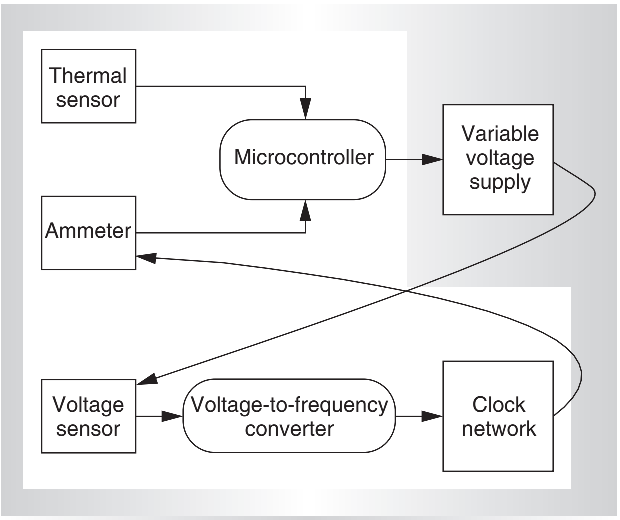


Figure 6. How Foxton technology controls voltage and frequency in Montecito. Foxton Technology uses two control loops, one for voltage and one for frequency. Components in the gray area are part of Montecito.

Dynamic voltage and frequency management

Foxton technology's goal and methods are significantly different from those of power-management technologies in other processors. Perhaps the greatest difference is that Foxton technology attempts to tap unused power by dynamically adjusting processor voltage and frequency to ensure the highest frequency[8] within temperature and power constraints. The 6-Mbyte Itanium 2, for example, consumes 130W for worst case code[1] and has power limits that keep it from using a higher voltage, and thus a higher frequency or performance point. But low-power applications, such as enterprise or integer workloads, often consume only 107W. Consequently, 23W of power—and hence performance—remain untapped.

When Foxton technology detects lower power consumption, it increases the voltage. The ensuing frequency boost for enterprise applications is 10 percent over the base frequency. This frequency change is nearly instantaneous and exposes the entire power and thermal envelope to every application and protects against power and temperature overages. Foxton technology also enables optimal support of demand-based switching (DBS) so that the operating system or server management software can lower the overall socket power requirement when saving power is important.

Voltage and frequency control loops. Figure 6 shows how Foxton technology can provide a near cubic reduction in power when needed. An on-chip ammeter feeds into an on-chip microcontroller that uses digital signal processing to determine power consumption. The microcontroller runs a real-time scheduler to support multiple tasks—calibrating the ammeter, sampling temperature, performing power calculations, and determining the new voltage level. The microcontroller will also identify when available control (voltage or frequency) cannot contain the over-power or over-temperature condition and indicates these situations to Montecito's platform for mitigation. If catastrophic failures occur in the cooling system, the microcontroller will safely shut down the processor.

When the microcontroller detects a need, it will change the voltage it requests of the variable voltage supply in 12.5 mV increments. The voltage control loop responds within 100 ms.

Voltage-frequency coordination. Because voltage directly impacts the transistors' ability to operate correctly at a given frequency, Montecito monitors the voltage throughout the chip with 24 voltage sensors.[9] The sensors respond to local voltage changes from current-induced droops and global voltage changes that Foxton technology has imposed. The sensors select an appropriate frequency for the given voltage and coordinate efforts to determine the appropriate frequency: If one sensor detects a lower voltage, all sensors decrease frequency. Likewise, the last sensor to measure a higher voltage enables all detectors to increase frequency.

A digital frequency divider (DFD) provides the selected frequency, in nearly 20-MHz increments and within a single cycle, without requiring the on-chip phase-locked loops (PLLs) to resynchronize.[10]

Thus, Foxton technology can specify a voltage, monitor the voltage at the cores, and have the clock network respond nearly instantaneously. The fine increments and response times in both the clock and the voltage planes mean that the application can use the entire power envelope. This contrasts to alternatives in which the frequency and voltage have specific pairs and often require PLL resynchronization at transitions, leaving parts of the power envelope untapped.

Demand-based switching

At times, it makes sense to sacrifice performance for lower power, such as when processor utilization is low. For these cases, Montecito provides DBS, which lets an operating system that is monitoring power to specify a lower power envelope and realize power savings without adversely affecting performance. The operating system can invoke DBS by selecting from a set of advanced configuration and power interface (ACPI) performance states.

Because Montecito varies both voltage and frequency, systems can realize a significant power savings through this option with a small performance impact; typically a 3 percent power change has a 1 percent frequency change. Additionally, Montecito lets original equipment manufacturers (OEMs) establish a maximum power level less than 100W, which means that Montecito will automatically adjust its voltage and frequency to fit comfortably in a 70W blade or a 100W high-performance server.

Other sources of power efficiency

Foxton technology and DBS are not the only innovations that enable Montecito's power efficiency. Many improvements throughout the core and cache increase power efficiency as well. Clock gating, however, a design strategy traditionally used to reduce average power, found limited application in Montecito. The idea behind clock gating is to suspend clocks to logic when that logic is idle. However, because Montecito starts with a highly utilized pipeline and adds improvements to boost performance, few areas of logic remain idle for an extended period. Moreover, turning the suspended clocks back on comes with a latency that either negatively affects performance or forces speculative clock enabling, which in turn lessens clock gating's benefit.

For these reasons, Montecito applies gating for blatantly useless signal transitions, but avoids it when clock gating would negatively affect overall performance. However, in the L3 cache, the Montecito designers took clock gating to the extreme by completely removing the clock for data-array accesses. The asynchronous interface to the L3 data array drastically lowers the power required to complete a read or write operation. A synchronous design, with a clock transitioning every cycle over a large area, is estimated to consume about 10W more than the asynchronous design. Montecito designers also replaced many power-hungry circuits with less aggressive and more power-conscious ones. All these power savings result in increased frequency and hence performance.

More "Ilities"

Itanium processors are already known for their outstanding reliability, availability, and serviceability (RAS), but Montecito expands the Itanium 2 RAS by providing protection for every major processor memory array from the register file[7] to the TLBs and caches.[11] Because Montecito parity protects even non-data-critical resources, when a pair of Montecito processors operate in lockstep, Montecito can indicate detected errors externally and avoid fatal lockstep divergences.

Error reporting

For Montecito, the key to providing reliability and availability is a partnership between the processor—with its array protection, error logging, and precision—and the software stack—which will correct, contain, and report the failure. Montecito provides robust error detection and correction for critical data. For software-based correction and data extraction, it provides hooks to support low-level array access and a robust set of information about the event so that the software stack can mitigate the errors.

Data protection

Montecito's variety of structures and arrays results in an approach tailored to the structure and its purpose. Some arrays use ECC; others use parity. Some error responses are immediate; some are deferred. Some errors are machine correctable; others require mitigation from the software stack.

Data criticality and structure size determined the method of structure protection. Montecito replaced Itanium 2's parity with ECCs for key structures, for example, while adding parity to many structures. The criticality and size also enabled the use of Pellston technology,[11] which maintains availability in the face of hard errors developed in the L3 cache while providing a mechanism for Montecito to scrub detected errors. Montecito also has parity protection for the register files (both integer and floating point),[7] temporary system interface data, and L2I data and tag bits. Parity protects some noncritical elements as well, such as branch-prediction structures and some cache-replacement structures. Many designs ignore these errors, but systems that use socket-level lockstep to provide higher reliability and data protection require reporting of such errors externally. If these systems know that an internally detected error caused a loss of lockstep, they can recover and so become available more often. System software can configure large arrays such as the L3 tag and data to correct errors in line, which does not induce a loss of lockstep.

Virtualization support

Customers continue to demand a better return on their computational technology investment, particularly ease of management. Ways to achieve that return include consolidated workloads, resource management, and optimized processor utilization.

Montecito's virtualization technology, a mixture of hardware and firmware, provides a conduit for efficient virtual machine monitor (VMM) implementations. VMMs address management requirements by allowing multiple operating system instances to utilize the underlying hardware at the same time without any knowledge at the operating-system level that the processor is shared.

Virtualization support in Montecito includes a fast mechanism to switch between guest and host modes, a virtual address protection scheme to isolate the guests' address space from that of the VMM host, and privileged resource-access interceptions with a new virtualization fault to abstract the processor from the OS. The huge amount of processing capability a single Montecito provides means that virtual machine support will allow multiple virtual partitions to coexist and share this power.

These features, when combined with key firmware support, let VMM software abstract the processor from the operating system such that a single processor can run multiple operating systems while each operating system thinks it has complete control of the processor.

Intel designed the Montecito processor to address the enterprise, technical, and integer workloads. Important attributes for success in these areas include reliability, power efficiency, and of course performance. While Foxton technology advances Montecito as a power-efficient design, the core, cache hierarchy, dual-thread and dual-core changes provide Montecito with top end performance.

Overall, supporting dual threads in Montecito cost relatively little and we expect performance to increase 8 to 30 percent over other Itanium processors, depending on the system and workload. Figure 7 shows relative performance for integer, technical, and enterprise workloads. The integer and technical workloads essentially scale with frequency for the previous Itanium processors, but Montecito's core changes should allow it to perform better than raw frequency would provide. Montecito's enterprise performance stands out over previous Itanium processors, in large part because of Montecito's dual threads and dual cores. Montecito, with its high performance, low power and many capabilities, is well prepared and suited for the server market. MICRO

Acknowledgments

We thank the Montecito design, verification and performance-evaluation teams for their

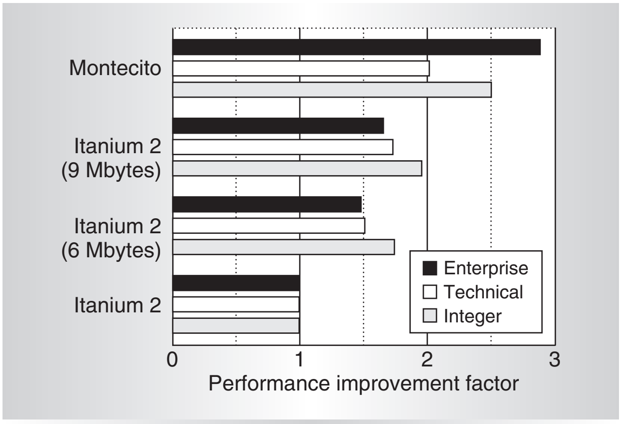


Figure 7. Estimated performance of Montecito relative to previous Itanium 2 processors.

contributions in successfully delivering the first dual-thread, dual-core Itanium processor. The work we describe in this article would not have been possible without the contribution of each individual on those teams.

References

1. S. Rusu et al., "Itanium 2 Processor 6M: Higher Frequency and Larger L3 Cache," *IEEE Micro*, vol. 24, no. 2, Mar.-Apr. 2004, pp. 10-16.
2. S. Naffziger et al., "The Implementation of a 2-core, Multi-threaded Itanium Family Processor," *Int'l Solid State Circuits Conf. Digest of Technical Papers,* IEEE Press, Feb 2005, pp. 182-183.
3. C. McNairy and D. Soltis, "Itanium 2 Processor Microarchitecture," *IEEE Micro*, vol. 23, no. 2, Mar.-Apr. 2003, pp. 44-55.
4. M. Mock et al., "An Empirical Study of Data Speculation Use on the Intel Itanium 2 Processor," to be published in *Proc. Workshop Interaction Between Compilers and Computer Architecture*, IEEE CS Press, 2005.
5. J. Wuu et al., "The Asynchronous 24 MB On-Chip Level 3 Cache for a Dual-core Itanium Architecture Processor," *Int'l Solid State Circuits Conf. Digest of Technical Papers,* IEEE Press, Feb 2004, pp. 488-487.
6. L. Spracklen et al., "Effective Instruction Prefetching in Chip Multiprocessors for Modern Commercial Applications," *Proc. High-Performance Computer Architecture*, IEEE CS Press, 2005, pp. 225-236.
7. E. Fetzer et al., "The Multi-threaded, Parity Protected, 128 Word Register Files on a Dual-core Itanium Architecture Processor," *Int'l Solid State Circuits Conf. Digest of Technical Papers*, IEEE Press, 2005, pp. 382-383.
8. C. Poirier et al., "Power and Temperature Control on a 90 nm Itanium Architecture Processor," *Int'l Solid State Circuits Conf. 2005 Digest of Technical Papers*, IEEE Press, 2005, pp. 304-305.
9. E. Fetzer et al., "Clock Distribution on a Dual-core, Multi-threaded Itanium Architecture Processor," *Int'l Solid State Circuits Conf. Digest of Technical Papers*, IEEE Press, 2005, pp. 292-293.
10. T. Fischer et al., "A 90 nm Variable Frequency Clock System for a Power-Managed Itanium Architecture Processor," *Int'l Solid State Circuits Conf. Digest of Technical Papers*, IEEE Press, 2005, pp. 294-295.
11. C. McNairy and J. Mayfield, "Montecito Error Protection and Mitigation," to be published in *Proc. Workshop High-Performance Computing Reliability Issues*, IEEE CS Press, 2005.

Cameron McNairy is an architect for the Montecito program at Intel. He was also a microarchitect for the Itanium 2 processor, contributing to its design and final validation. In future endeavors, McNairy plans to focus on performance; reliability, availability, and serviceability; and system interface issues in Itanium processor design. He received a BSEE and an MSEE from Brigham Young University and is a member of the IEEE.

Rohit Bhatia is an Itanium processor microarchitect at Intel. His research interests include computer architecture, Itanium processor microarchitecture, and the functional verification of microprocessors. Bhatia has an MSEE from the University of Minnesota and an MS in engineering management from Portland State University.

Direct questions and comments about this article to Cameron McNairy, Intel, 3400 East Harmony Rd., HRM1-1, Fort Collins, CO 80528; cameron.mcnairy@intel.com.