Intel® Technology Journal

Intel® Pentium® 4 Processor on 90nm Technology

Intel® Pentium® 4 processors built on the 90-nanometer process retain the multitasking capabilities of Hyper-Threading (HT) Technology, and add new features including enhanced Intel NetBurst® microarchitecture, a larger, 1 MB Level 2 (L2) cache and 13 new instructions.

Inside you'll find the following papers:

The Microarchitecture of the Intel® Pentium® 4 Processor on 90nm Technology

LVS Technology for the Intel® Pentium® 4 Processor on 90nm Technology

Support for the Intel® Pentium® 4 Processor with Hyper-Threading Technology in Intel® 8.0 Compilers

Library Architecture Challenges for Cell-Based Design

Performance Analysis and Validation of the Intel® Pentium® 4 Processor on 90nm Technology

Full Hold-Scan Systems in Microprocessors: Cost/Benefit Analysis

THIS PAGE INTENTIONALLY LEFT BLANK

Intel® Technology Journal

Intel® Pentium® 4 Processor on 90nm Technology

Articles

THIS PAGE INTENTIONALLY LEFT BLANK

Preface

By Lin Chao, Publisher

At Intel Corporation, we use "roadmaps" to help set the direction for each new microprocessor. The microprocessor's roadmap includes features, performance, power, frequency, number of transistors and market segments. Included in this roadmap is a mapping to the semiconductor process technology on which the microprocessor will be built. It requires a delicate balance to match each new processor's specifications and features to underlying circuit devices, which physically must be manufactured using a semiconductor process. The semiconductor process technology is what determines the microprocessor's features and capabilities.

The Intel® Pentium® 4 processor on 90-nanometer (nm) technology is the first Intel processor to be manufactured on 90nm semiconductor process technology. This new process offers smaller dimensions and more transistors on the same area, allowing us to double the number of transistors while reducing the chip size by over 15 percent. The smaller the chip size, the lower the manufacturing costs per chip. Typically, the higher the number of transistors, the higher the performance and capabilities offered. So 90nm is a winning solution.

Among the many new features of the 90nm process is "strained silicon." Strained silicon has silicon atoms spaced so the lattice of the silicon lines up with one another. This stretches and "strains" the silicon, which means the resistance to the flow of electrons is reduced. Electrons can flow up to 70 percent faster in this strained lattice. The strained silicon remains strained, and transistors made from it get a 35 percent performance benefit over unstrained transistors of the same size.

The six papers in this issue of *Intel Technology Journal* (Vol. 8, Issue 1, 2004) explore Intel engineers' talented work on our Pentium 4 processor on 90nm technology and the innovations used to design, manufacture and test it. The papers discuss its microarchitecture including the thirteen new instructions referred to as SSE3 and the 31-stage pipeline; a compiler supporting the new features for writing high-performance software; use of Low-Voltage Swing circuit logic; use of standard library cells; a new testing system named Full Hold-Scan; and insights into the complexities of modeling for validation purposes. All the papers reflect the creative and innovative work by the team of engineers who made the newest Intel Pentium 4 processor a reality.

® Intel and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

THIS PAGE INTENTIONALLY LEFT BLANK

Foreword

By William M. Siu
Vice President
General Manager, Desktop Platforms Group
Intel Corporation

The role of a today's PC is expanding. The boundary between PCs and Consumer Electronics devices is blurring. Whether it is decoding and displaying high-definition content to a plasma TV or sharing your personal content wirelessly to other rooms in the home, PCs require horsepower and multitasking capabilities. A single PC in the future will likely be required to manage vast media libraries, time shift live video, and share multiple video streams to other devices around the home and it may need to do this all at once. Corporations are looking for PCs that enhance employee productivity by supporting collaborative tools such as Microsoft Office 2003, videoconferencing or instant messenger. They are looking for PCs that offer high responsiveness to applications while running security applications or database queries in the background.

These usage models are becoming relevant even to mainstream users. Intel® Pentium® 4 processors built on the 90-nanometer (nm) process provide improved responsiveness for today's corporate and home applications, and offer headroom for the next wave of technologies. Intel Pentium 4 processors with Hyper-Threading Technology,[1] together with Intel's next-generation chipset, which delivers high-definition audio, integrated wireless capabilities, and PCI Express among other advances, provide the market place with the most compelling platform to meet user needs.

Intel Pentium 4 processor on 90nm technology contains several microarchitecture enhancements such as a larger 1 MB Level 2 (L2) cache, larger 16 KB Level 1 (L1) cache and 13 new instructions. The processor's hardware prefetcher and branch predictor have also been improved to keep the execution units busy and avoid processor stalls. A new integer multiplication unit and fast shift/rotate features improve latency and help encryption and decryption. The processor also has eight write-combining buffers compared to six in the previous generation. Four entries were added to the floating-point schedulers to enhance floating-point and media application performance, especially in HT Technology configurations. These architectural enhancements in the processor are expected to fuel office and digital home usage models. For example, a home user can play an immersive game while encoding audio or video, compressing images or compositing special effects. An IT department can run background applications such as continuous virus scanning, encryption or compression simultaneously, while minimizing disruption for other business users in the same computing environment.

[®] Intel and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

[1] Hyper-Threading Technology requires a computer system with an Intel® Pentium® 4 processor supporting HT Technology and an HT Technology-enabled chipset, BIOS and operating system. Performance will vary depending on the specific hardware and software you use. For more information including details on which processors support HT Technology, see http://www.intel.com/products/ht/hyperthreading_more.htm.

Apart from the architectural changes in the processor, Intel Pentium 4 processor on 90nm technology has the distinction of being the world's first high-volume processor on the new technology. Intel's 90nm process technology is the most advanced semiconductor manufacturing process in the industry, built exclusively on 300mm wafers. This new process combines high performance, low-power transistors, strained silicon with gate length scaling down to 50nm, 7 layers of high-speed copper interconnects, and a new low-k Carbon Doped Oxide dielectric material. This is the first time all of these technologies have been integrated into a single manufacturing process. The 90nm process technology reduces die size by more than 15% compared to the previous 130nm process, while more than doubling the number of transistors. Given the manufacturing efficiency of the new 90nm process, Pentium 4 processor on 90nm technology is expected to be the fastest ramp in Intel's history.

We faced new challenges for the verification and validation of the Pentium 4 processor on 90nm technology. The processor embodied a new microarchitecture, new features, new circuit designs, and it's done on the leading 90-nanometer technology. It's the most complex design we've had since Willamette, the first Intel Pentium 4 processor. Intel's validation and design teams developed a new methodology called Design for Validation (DFV). DFV features allow debuggers to reproduce failures deterministically on a system and to dump key state data from the CPU. DFV features allow us to reduce debug time significantly. Engineers from the validation and design teams analyzed the I/O buffer behavior in detail, both modeling and simulating the GTL Reference Voltage data and improved them in the final release of the product.

Will Intel Pentium 4 processor with HT Technology on 90nm process succeed in the market place? Certainly! Traditional computing requirements have not gone away. A usage model evolution is taking place that requires new platforms with rich features for both consumers and corporate computer users. Intel Pentium 4 processor on 90nm process, with its multi-tasking capability and headroom for the next wave of technologies, and Intel's next-generation chipset, with its rich feature set, provide the most compelling platform for users. In addition, the processor's die size benefits from 90nm technology process and 300mm size wafers, thereby increasing our flexibility to more readily meet demand in market place.

The Microarchitecture of the Intel® Pentium® 4 Processor on 90nm Technology

Darrell Boggs, Desktop Platforms Group, Intel Corporation
Aravindh Baktha, Desktop Platforms Group, Intel Corporation
Jason Hawkins, Desktop Platforms Group, Intel Corporation
Deborah T. Marr, Desktop Platforms Group, Intel Corporation
J. Alan Miller, Desktop Platforms Group, Intel Corporation
Patrice Roussel, Desktop Platforms Group, Intel Corporation
Ronak Singhal, Desktop Platforms Group, Intel Corporation
Bret Toll, Desktop Platforms Group, Intel Corporation
K.S. Venkatraman, Desktop Platforms Group, Intel Corporation

Index words: Pentium® 4 processor, Hyper-Threading Technology, microarchitecture

ABSTRACT

This paper describes the first Intel® Pentium® 4 processor manufactured on the 90nm process. We briefly review the NetBurst® microarchitecture and discuss how this new implementation retains its key characteristics, such as the execution trace cache and a 2x frequency execution core designed for high throughput.

This Pentium 4 processor improves upon the performance of prior implementations of the NetBurst microarchitecture through larger caches, larger internal buffers, improved algorithms, and new features. This processor also implements Hyper-Threading Technology, which is the ability to simultaneously run multiple threads, allowing one physical processor to appear as two independent logical processors. This technology is another means of providing higher performance to the end user. We discuss how this processor not only maintains support for this key technology but also increases the benefit seen due to Hyper-Threading Technology.

We also describe 13 new SSE3 instructions that have been added to the IA-32 instruction set and are implemented for the first time on this processor. These instructions can be used in multimedia algorithms, such as motion estimation, and for complex arithmetic. Additionally, two new instructions are added for improving thread synchronization. To conclude, performance data are presented that show the benefit of this Pentium 4 processor over prior implementations on key applications and benchmarks.

INTRODUCTION

The first Intel Pentium 4 processor manufactured on the 90nm manufacturing process contains 125 million transistors with a die size of 112mm^2. It builds upon the NetBurst microarchitecture that forms the foundation of prior Pentium 4 processors. Like its predecessors, this processor is designed to provide the end user with new levels of performance, enabling compute-intensive tasks to be undertaken by conventional desktop processors. One means of achieving this performance is by designing the processor to run at a high frequency. The frequency of a processor is a key component to determining overall performance, as the frequency determines the rate at which the processor can process data. We have extended the original Pentium 4 processor pipeline to enable this processor to reach

---

higher frequencies than is possible with the original pipeline. Additionally, as the frequency of the processor continues to increase, the amount of time spent waiting for data to be retrieved if they are not located in the processor's caches is becoming a larger and larger percentage of overall execution time. This effect reduces the performance impact of continually increasing the

processor frequency. To alleviate this problem, several features are implemented to increase the number of times that data will be present in the caches. With these and other features, including a set of new instructions, the Pentium 4 processor is able to achieve new heights in performance.
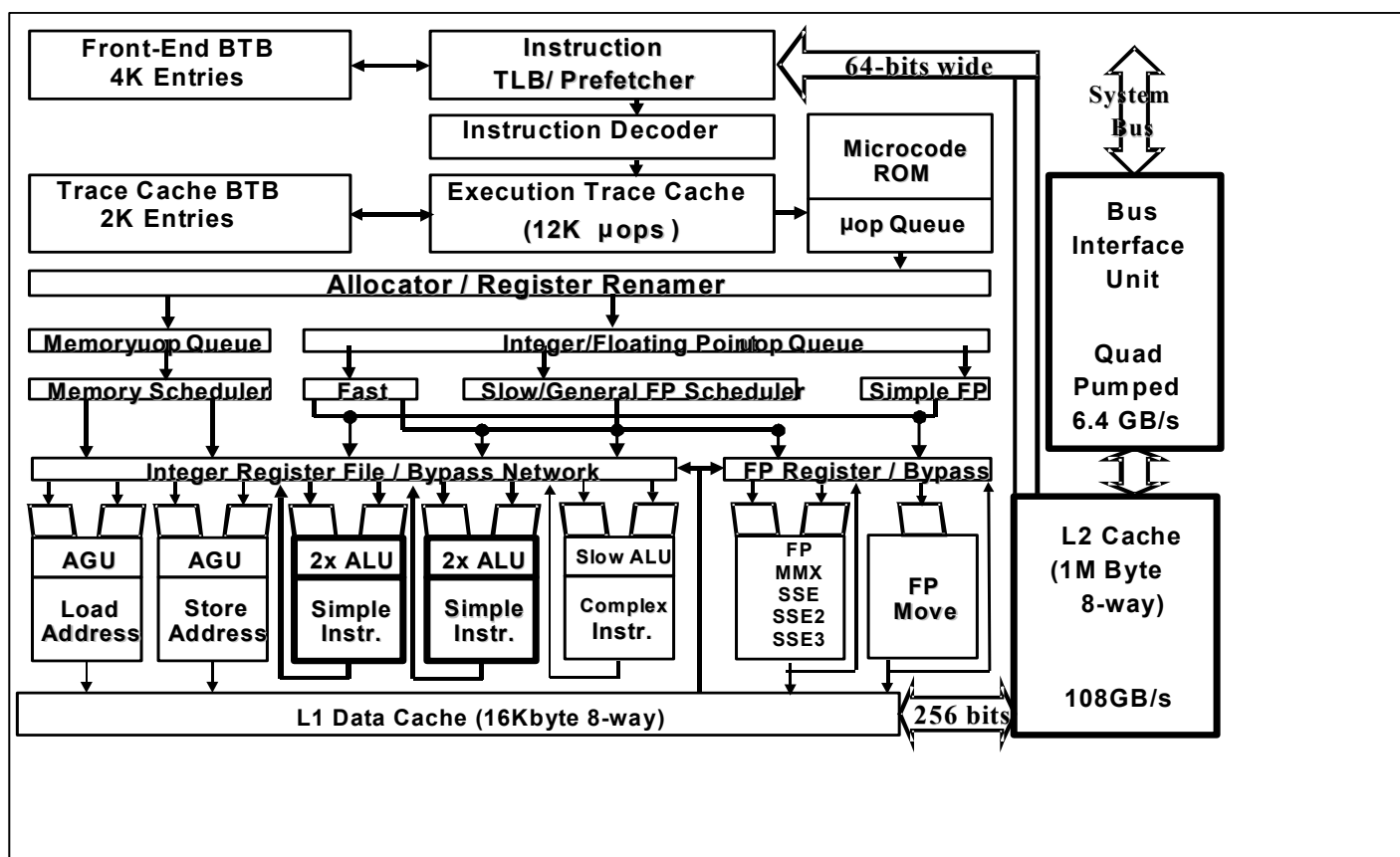

Figure 1: Block diagram of the Intel® Pentium® 4 processor

NETBURST® MICROARCHITECTURE OVERVIEW

The NetBurst microarchitecture is the basis for the latest version of the Intel Pentium 4 processor. Elements of this microarchitecture include an Execution Trace Cache, an out-of-order core, and a Rapid Execution Engine [1]. This implementation also contains store-to-load forwarding enhancements that were introduced in previous implementations. Figure 1 depicts the block diagram for the Pentium 4 processor.

Execution Trace Cache

The NetBurst microarchitecture has an advanced instruction cache called an Execution Trace Cache. This cache stores decoded instructions in the form of uops rather than in the form of raw bytes such as are stored in more conventional instruction caches. Once stored in the trace cache, uops can be accessed repeatedly just like a conventional instruction cache. Storing uops instead of bytes allows the complicated instruction decoding logic to be removed from the main execution loop.

In addition to removing the cumbersome decode logic from the main execution loop, the Execution Trace

Cache takes the already decoded uops from the instruction decoder and assembles or builds them into program-ordered sequences of uops, called traces. It packs the uops into groups of up to six uops per trace cache line and these lines are combined to form traces. These traces consist of uops from the sequentially predicted path of the program execution. This allows the target of a branch to be included in the same trace cache line as the branch itself, even if the branch and its target instruction are thousands of bytes apart in the program. Thus, both the branch and its target instructions can be delivered to the out-of-order core at the same time. Conventional instruction caches typically provide instructions up to and including a taken branch in a given clock cycle but no instructions following the branch. If the branch is the first instruction in a cache line, only the single branch instruction is delivered that clock cycle. Conventional instruction caches also often add a clock delay getting to the target of the taken branch due to delays getting through the branch predictor and then accessing the new location in the instruction cache. The trace cache avoids both of these instruction delivery delays.

The trace cache is able to deliver up to three uops per clock cycle to the out-of-order core. Most instructions in a program are fetched and executed from the trace cache. Only when there is a trace cache miss does the machine fetch and decode instructions from the unified second-level (L2) cache. The Execution Trace Cache on the Pentium 4 processor can hold up to 12K uops and has a hit rate similar to an 8 to 16 kilobyte conventional instruction cache.

Out-of-Order Core

The Execution Trace Cache provides the out-of-order core with a stream of uops to prepare for the Rapid Execution Engine to consume. The main responsibility of the out-of-order core is to extract parallelism from the code stream, while preserving the correct execution semantics of the program. It accomplishes this by reordering the uops to execute them as quickly as possible.

The out-of-order core will schedule for execution as many ready uops as possible each clock cycle, regardless of their original program order. By considering a larger number of uops from the program, the out-of-order core can usually find many independent uops that are ready to execute. The maximum number of uops that the out-of-order core can contain is 126, of which 48 can be load operations and 32 can be store operations.

At the heart of the out-of-order core are the uop schedulers. The schedulers determine when a uop is ready to execute by tracking its input register operands. When the input operands have been produced, the uop is considered to be ready to execute. The scheduler will then schedule the uop to execute when the execution resources required by the uop are available. Thus, uops are allowed to schedule and execute in what is called data-dependent order. In many code sequences, there are independent streams of execution. The scheduler identifies the streams of execution and allows these streams to execute in parallel with each other, regardless of their original program order.

There are five different schedulers connected to four different dispatch ports. On two of these ports, up to two uops can be dispatched each clock cycle. The fast Arithmetic and Logic Unit (ALU) schedulers can schedule on each half of a clock cycle, while the other schedulers can only schedule once per clock cycle. One fast ALU scheduler shares a dispatch port with the floating-point/media move scheduler, while the other fast ALU shares another dispatch port with the complex integer/complex floating-point/media scheduler. These schedulers arbitrate for a dispatch port when multiple schedulers have uops ready to execute at the same time. The remaining two dispatch ports allow one load and one store address uop to be dispatched every cycle. The collective dispatch bandwidth across all of the schedulers is six uops per clock cycle. This is twice the rate at which the out-of-order core can receive uops from the Execution Trace Cache and allows higher flexibility to issue ready uops on the different ports.

Rapid Execution Engine

The Rapid Execution Engine of the NetBurst microarchitecture executes up to six uops per main clock cycle. These uops are executed by several execution units: two double-speed integer ALUs, a complex integer unit, load and store Address Generation Units (AGUs), a complex floating-point/media unit, and a floating-point/media move unit. These highly tuned and optimized execution units are designed for low latency and high throughput.

The double-speed integer ALUs are able to execute at a rate of two uops per clock cycle, providing for a very high ALU throughput. Being able to execute these uops at twice the rate of the main core clock enables application performance to be increased relative to running the ALUs at the main clock rate.

The NetBurst microarchitecture is also able to execute one load and one store address uop every clock cycle through the AGUs. The AGUs are very tightly coupled to the low-latency first-level (L1) data cache. On this processor, the cache is 16 kilobytes in size and is used for both integer and floating-point/media loads and

stores. It is organized as an 8-way set associative write-through cache containing 64-byte cache lines.

The low latency of the L1 cache is very hard to achieve. This cache uses unique access algorithms to enable its low latency. The algorithms leverage the fact that almost all accesses hit the L1 data cache and the Data Translation Lookaside Buffer (DTLB). Generally, the schedulers assume that loads will hit the L1 data cache and will schedule dependent uops before the parent load has finished executing. Allowing these dependent uops to dispatch prior to knowing if the load has hit the cache is a form of data speculation. If the load misses the L1 data cache, the dependent uops will already be well into their execution and will temporarily be bound to incorrect data. Using a mechanism known as replay, the processor tracks and re-executes instructions that received incorrect data. Only the dependent operations are replayed; all independent operations are allowed to complete. Using this form of data speculation allows more parallel execution streams to be extracted from the program and increases the performance of the processor.

Floating-Point (x87), MMX, SSE (Streaming SIMD Extension), SSE2 (Streaming SIMD Extension 2), and the new SSE3 (Streaming SIMD Extension 3) operations are executed by the two floating-point execution blocks. One of the execution blocks is used for simple operations, such as SSE register-to-register moves and x87/MMX/SSE/SSE2 store data uops. The other execution block is used for more complex operations.

Store-to-Load Forwarding Enhancements

In all implementations of the NetBurst microarchitecture, stores are written to the L1 data cache in programmatic order and only after the store is guaranteed to be non-speculative. This requires that all operations older than the store must be completed before the store's data are committed to the cache. The forwarding mechanism implemented enables a load dependent on a store's data to have its data "forwarded" prior to the commitment of the store's data into the L1 cache. Forwarding is accomplished by doing a partial address match between the load and all older stores in the Store Forwarding Buffer (SFB) in parallel with the load's L1 data cache access. If the load's partial address matches that of an older store in the SFB, then the load gets its data from the SFB instead of the cache. The forwarding mechanism is optimized for speed such that it has the same latency as a cache lookup. To meet this latency requirement, the SFB cannot afford to do a full address and access size check. This function is accomplished by the Memory Ordering Buffer (MOB) later in the pipeline. The role of the MOB is to ensure that the forwarded load got the correct data from the most recent dependent store. In the event that the forwarding from the SFB was incorrect, the load in question must be re-executed after the dependent store writes to the L1 cache. The load can then pick up its data from the cache.

The latency from when a store has valid data to when these data are written into the cache can be high because of the deep pipeline of the NetBurst microarchitecture. So in cases where a load must wait for a store to commit its data for the load to complete, a significant reduction in performance can occur. Most of these cases are rare in real-world applications. However, there a few instances where applications do see a performance loss:

- Forwarding disabled due to address misalignment.

- Wrong forwarding due to a partial address match.

Mechanisms have been implemented on recent implementations of the Intel Pentium 4 processor to improve the performance in the above cases.

Force forwarding is a mechanism that allows the MOB to control the forwarding in the SFB. Figure 2 shows the block diagram for this mechanism. Two new selection points were added to the existing store-forwarding path. The forwarding-entry-selection mux allows the MOB to override the SFB's partial address match-based entry selection, while the data alignment mux allows for misaligned data to be rotated, based on the shift information provided by the MOB.

When a load first executes, the SFB detects a dependency with older stores based on a partial address match. When this load comes to the MOB to determine its "true" dependencies, the MOB can either agree with the SFB's decision to forward or it can cause the load to be re-executed. The load can be re-executed because the SFB detected either an incorrect dependency or because it failed to detect a dependency when a dependency did exist. If the SFB's dependency check is wrong, the MOB can correct the forwarding logic when the load re-executes by directing the SFB in one of two ways: forward to the load from the right entry and rotate the data as necessary or disable forwarding to the load if there is no dependent store in the SFB.
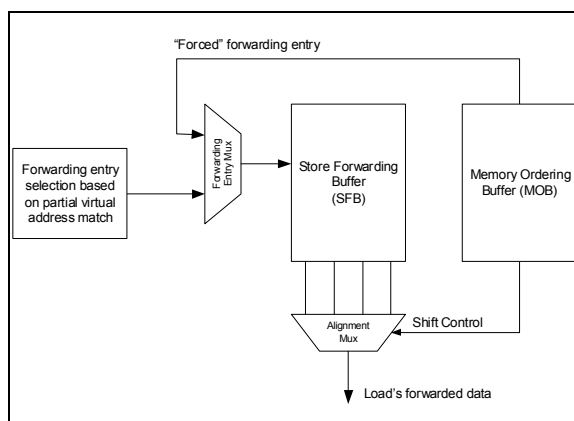
Figure 2: Force forwarding block diagram

The misaligned address cases that are fixed by the force forwarding mechanism are shown in Figure 3. In the figure, for each load at a given starting address, the data access sizes for which force forwarding is supported are listed. These cases can be categorized as follows:

- DWord/QWord Store forwarding to Byte/Word loads whose data are fully contained in either the lower or upper DWord.

- QWord Store forwarding to DWord Load to the upper DWord of the Store.

For each of these cases, the MOB "forces" the SFB to forward from a specific store by a given shift amount in order to align the store's data to the load.
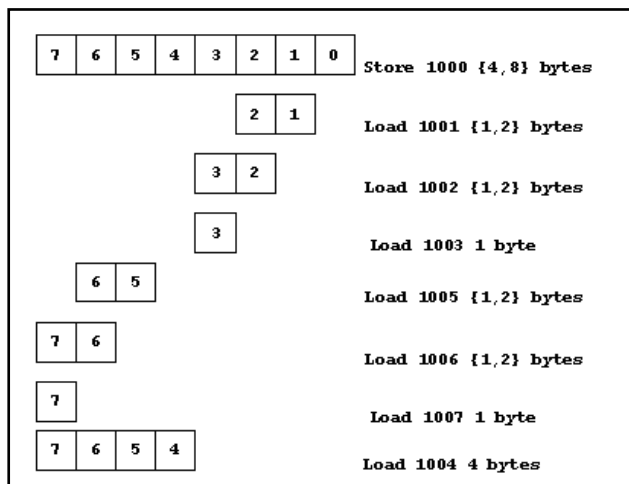
Figure 3: Supported cases of misaligned forwarding

False forwarding occurs when the SFB detects a partial address match between a load and a store, but their full addresses do not match. The MOB detects the false forward condition and determines if there exists another store that the load should have forwarded from. If a store exists that can be forwarded, then the MOB will direct the SFB to forward from this store entry using the

force forwarding mechanism when the load re-executes. If the MOB detects that there is no dependent store in the forwarding buffer, then the MOB instructs the SFB to not forward to this load. When the load is re-executed, it can then pick up its data from the cache instead.

NEW MICROARCHITECTURAL FEATURES AND ENHANCEMENTS

The 90nm Intel Pentium 4 processor improves performance over prior processor implementations through increasing the sizes of key resources, while also improving existing algorithms and introducing new microarchitectural features. These changes were made throughout the various parts of the processor as detailed below.

Front End

The instruction fetch and decode portions of this Intel Pentium 4 processor remain largely unchanged from previous implementations, but some performance enhancements have been made.

The simple static branch prediction scheme that is used when the Branch Target Buffer (BTB) has no prediction for a conditional branch has been enhanced. At the time the instruction decoder realizes that an instruction is a branch that was not predicted by the BTB, a static branch prediction is made. Making this prediction at decode time allows for a faster restart, and therefore better performance, rather than waiting for the normal execution time detection of a mispredicted branch.

In prior Pentium 4 processor implementations, the static prediction algorithm was to predict that a branch was taken if the branch direction was backwards and to predict that the branch was not taken if the branch jumps forward. This helped by correctly predicting *taken* for the first iteration of most loops. This works well for backwards branches that are in loops, but not all backwards branches are loop-ending branches.

We can try to ascertain the difference between loop-ending branches and other backwards branches by looking at the distance of the branch and the condition on which the branch is dependent. Our studies showed that a threshold exists for the distance between a backwards branch and its target; if the distance of the branch is larger than this threshold, the branch is unlikely to be a loop-ending branch. If the BTB has no prediction for a backwards branch, the Intel Pentium 4 processor will then predict *taken* for the branch only if the branch distance is less than this threshold.

We also discovered that branches with certain conditions were more often not taken, regardless of their

direction and distance. The conditions that they used are not common loop-ending conditions, so for branches with these conditions and no BTB prediction, the static prediction algorithm predicts them as *not taken*.

In addition to these changes in the static prediction algorithm, we also enhanced the dynamic branch prediction algorithms to reduce the number of times that a branch is mispredicted. Each time a branch is mispredicted, the pipeline must be flushed. Thus, large performance gains can be had by reducing the number of branch mispredictions. To this end, one of the dynamic branch predictor enhancements we made was to add an indirect branch predictor. This was motivated by results from the Intel Pentium® M processor team, who saw good performance improvements on some applications [3]. Table 1 compares the number of branch mispredictions per 100 instructions on the 90nm version of the Intel Pentium 4 processor versus the 130nm version of the processor on the components of SPECint*_base2000. The data were collected using the performance counters available on each processor, and they show the reduction in mispredictions on almost all components, due to the algorithmic enhancements.

Table 1: Comparison of mispredicted branches per 100 instructions

	130nm	90nm
164.gzip	1.03	1.01
175.vpr	1.32	1.21
176.gcc	0.85	0.70
181.mcf	1.35	1.22
186.crafty	0.72	0.69
197.parser	1.06	0.87
252.eon	0.44	0.39
253.perlbmk	0.62	0.28
254.gap	0.33	0.24
255.vortex	0.08	0.09
256.bzip2	1.19	1.12
300.twolf	1.32	1.23

Another performance enhancement was to expand the set of instructions where the processor detects that dependence chains can be broken. A common technique to zero a register is to xor the register with itself, rather than to move an immediate of 0 into the register. This technique is preferred because of the smaller resulting code size. The result is logically equivalent, but the xor method adds a dependency on the previous contents of the register. In an out-of-order machine, this extra dependency can result in a performance loss. Previous processor implementations recognized when the xor, pxor, and sub instructions were used in this manner, and they removed the dependency on the source register, since the same answer is arrived at regardless of the value of the sources. On this Intel Pentium 4 processor, additional instructions that are used for the same purpose are now detected. Among these are the SSE instruction xorps and the SSE2 psub and xorpd instructions.

We can also now encode more types of uops inside the trace cache than could be encoded in prior processors. If an instruction uses a uop that cannot be encoded in the trace cache, then the uops for the entire instruction have to be sequenced from the Microcode ROM. This enhancement allows for higher average uop bandwidth from the front end of the machine to the execution core by removing transitions to the Microcode ROM. Indirect calls with a register source operand and software prefetch instructions are the best examples of instructions that can now be encoded in the trace cache.

Execution Core

The execution core of the Intel Pentium 4 processor is similar to previous implementations in that the two integer ALUs run at 2x the frequency of the rest of the

---

® Pentium is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

* Other names and brands are the property of their respective owners.

processor, allowing for high throughput of common arithmetic and logical operations. An enhancement we implemented in this processor was to add a shifter/rotator block to one of the ALUs. This block allows the most common forms of shift and rotate instructions to be executed on a fast ALU. On prior Pentium 4 processor implementations, these operations were executed as complex integer operations that took multiple cycles to execute.

Another key operation whose latency has been reduced on this processor is integer multiply. Previously, the Intel Pentium 4 processor executed integer multiplies using the floating-point multiplier. This introduced latency by paying the cost of moving the source operands to the floating-point side and then moving the result back to the integer side. On this processor, we added a dedicated integer multiplier to service these operations.

On top of the changes to the execution units, we also changed the L1 data cache. As with all implementations of the NetBurst microarchitecture, the cache is designed to minimize the load-to-use latency by using a partial virtual address match to detect early in the pipeline whether a load is likely to hit or miss in the cache. On this processor, we significantly increased the size of the partial address match from previous implementations, thus reducing the number of false aliasing cases. More importantly, we increased the size of the cache. Previously, the L1 data cache was 8 kilobytes in size and 4-way associative. Now the size of the cache has been increased to 16 kilobytes by increasing the associativity to 8-ways.

The schedulers in the NetBurst microarchitecture are critical, as they must run at a high speed in order to continually feed the high-speed execution core. The schedulers in this implementation of the microarchitecture remain largely the same, as the rate at which they can feed the core is unchanged from prior implementations. In all implementations, the schedulers are capable of scheduling up to six uops per clock cycle.

Even though the rate of scheduling remains the same, we made several enhancements to the schedulers to improve performance on the implementation. The two schedulers that are used to hold uops used in x87/SSE/SSE2/SSE3 instructions were increased in size. By increasing the size of these schedulers, the window of opportunity to find parallelism in multimedia algorithms is increased. And we increased the effective size of the queues that feed all the schedulers, such that more uops can now be buffered between the allocator and the scheduler before the allocator has to stall. This allows the allocation and renaming logic to continue to

look ahead in the instruction stream even when the schedulers are full.

Additionally, we changed the mechanism used to schedule load uops to improve performance. As on prior implementations, store instructions are broken up into two pieces: a store address and a store data uop. In the previous implementations, loads were scheduled asynchronously to store data uops. Thus, if a load needed to receive forwarded data from a store, it was possible that the load would execute before the store data uop. If this occurred, the load would have to be re-executed after the store data uop had finally executed. Because of this, latency could be introduced because the minimum latency between a store data uop and a dependent load was not the common case latency for loads that had been re-executed. On top of that penalty, having to re-execute the load meant that precious load bandwidth was being wasted on loads that executed more than once. To alleviate both of these issues, we added a simple predictor to the processor that marks whether specific load uops are likely to receive forwarded data, and, if so, from which store they are likely to forward. Given this information, the load scheduler now holds a load that is predicted to forward in the scheduler until the store data uop that produces the data it depends on is scheduled. In doing so, both of these performance penalties are reduced significantly.

We also added a performance feature to enhance applications that use the SSE/SSE2/SSE3 instructions. On the x87 side, the Floating-Point Control Word (FCW) is often modified as the programmer wants to change the rounding mode and precision of the data that are being worked with. To avoid serializing the processor each time that the FCW is modified, a simple prediction scheme was implemented on the NetBurst microarchitecture to capture common renaming cases. This same idea is now extended on this implementation of the microarchitecture to also handle the MXCSR, which is the corollary of the FCW for instructions that use the SSE registers. On prior implementations, changes to the MXCSR would serialize the machine. On this processor, the common case modifications of MXCSR will not incur a serialization.

Memory System

In the memory subsystem of the processor, we made a number of changes to increase overall performance. The changes made focus on trying to reduce the amount of time spent waiting for data to be fetched from DRAM and on increasing the size of critical resources so as to limit the number of times the processor is forced to stall because of a resource shortfall.

One mechanism to reduce the amount of time spent waiting for data to be returned from DRAM is to increase the size of the caches. Previous implementations of the Intel Pentium 4 processor contained unified L2 caches of either 256 or 512 kilobytes. On the 90nm version of the Intel Pentium 4, we implemented a 1MB L2 unified cache. Similar to the previous implementations, the cache is a writeback 8-way set associative cache and contains 128-byte lines.

A second way to reduce the time waiting for DRAM is by using software prefetch instructions that are inserted by the programmer to bring data into the cache before the data are actually used. On all Pentium 4 processors, software prefetch instructions bring in data from DRAM into the L2 cache. These instructions opportunistically look up the L2 cache and on a miss, initiate a data prefetch cycle on the front-side bus. The data are filled only to the L2 cache so as not to pollute the much smaller L1 data cache.

On previous Pentium 4 processor implementations, these operations were dropped on a DTLB miss. The Pentium 4 processor adds a mechanism to allow the software prefetch instructions to initiate page table walks and allow data TLB fills if the prefetch access is to a page currently not cached in the TLB. We added special fault-handling logic to handle cases where page faults were detected on the software prefetch instructions. These instructions are dropped silently without reporting the fault to the operating system, and the prefetch operation is not performed. In effect, the 90nm version of the Pentium 4 processor allows software prefetch instructions to not only prefetch data, but also to prefetch page table entries into the DTLB. As we previously mentioned, the cost of software prefetch instructions has been greatly reduced on this processor, as software prefetches can now be cached in the trace cache; they used to have to be fetched from the Microcode ROM.

A third mechanism used to reduce the time waiting for DRAM is through a hardware prefetching scheme. The hardware prefetcher looks for streams of data and tries to predict what data will be needed next by the processor and proactively tries to fetch these data. This mechanism can be superior to software prefetching, as it requires no effort from the programmer and can improve performance on code that has no software prefetch instructions. All Intel Pentium 4 processors contain a hardware prefetcher that can prefetch both code and data streams, where the data stream can be accessed by loads and/or stores. This implementation of the processor improves upon the previous implementations in its ability to detect when to prefetch data and what data needs to be prefetched. Figure 4 shows the effect of the hardware prefetcher. We show the performance of this processor with the hardware prefetcher enabled versus the hardware prefetcher disabled on the most hardware prefetcher-sensitive components in the SPECint_base2000 and SPECfp*_base2000 benchmarks[1]. These are the components that gain more than 10% in performance by enabling the hardware prefetcher.

Addressing resource constraints was the other means of improving performance in the memory system. On previous Intel Pentium 4 processors, only 24 stores could be simultaneously outstanding in the processor. This number has now been increased to 32. Additionally, the number of write-combining buffers that are used to track streams of stores was increased from 6 to 8, which also alleviates pressure on the number of stores that can be in the machine simultaneously by allowing stores to be processed faster. Finally, the number of unique outstanding loads that have missed the L1 data cache and can be serviced has been increased from 4 to 8.
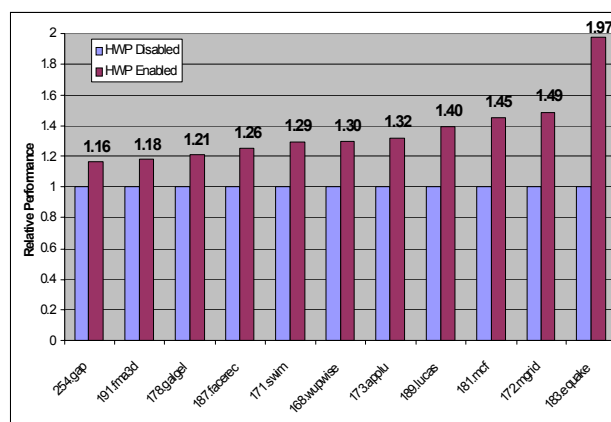

Figure 4: Effect of the hardware prefetcher

HYPER-THREADING TECHNOLOGY

Hyper-Threading Technology was introduced on previous implementations of the Intel Pentium 4 processor and is also present on many versions of this latest processor. Hyper-Threading Technology allows one physical processor to appear to the operating system as two logical processors [2]. This allows two program software threads, either related or unrelated, to execute

---

* Other names and brands are the property of their respective owners.

[1] Estimated performance through measurements on non-production hardware.

simultaneously throughout the processor. Prior to Hyper-Threading Technology, only one thread could be executed at a time on a processor, and each switch to a different thread would incur a context-switching overhead penalty.

Many of the changes mentioned previously were motivated mainly by Hyper-Threading Technology performance. For instance, increasing the number of outstanding loads that miss the L1 data cache from 4 to 8 has very little performance impact on the majority of single-threaded applications. This resource, however, is more important when two threads are being executed. Increasing the size of the resource that controls this behavior provides for better threaded performance while also slightly enhancing single-threaded performance. Similarly, the size of the queue that sits between the front end of the processor and the allocation/rename logic was also increased in this processor implementation. Again this change was motivated by the need for increased performance when running multiple threads, as the size increase provides minimal benefit when only running a single thread.

Other changes that were made in this processor implementation to help support Hyper-Threading Technology performance include additions to the type of operations that can be conducted in parallel. For instance, on previous implementations, the processor could either work on a page table walk or on handling a memory access that splits a cache line, but not on both simultaneously. For single-thread performance, this limitation was rarely seen as a bottleneck. However, when running multiple threads, the effect of this bottleneck becomes much more acute as the behavior of one thread can have a significant negative impact on the other thread. In this processor, this bottleneck has been fixed such that a page table walk can occur at the same time as a memory access that splits a cache line is being handled. Similarly, on prior implementations, if a page table walk missed all the caches and had to go to DRAM, no new page table walks could be started. This again was very rarely seen as a bottleneck for single-threaded performance but was detrimental when running multiple threads as one poorly behaving thread could effectively stall both threads. Now, in this implementation, a page table walk that misses all of the caches and goes to DRAM does not block other page table walks from being initiated.

Changes were also made to some of the thread selection points in this version of the Pentium 4 processor in order to improve overall bandwidth. For example, the trace cache now responds faster to stalling events in the core, dedicating all of its resources to the thread that is not stalled, thereby generating better overall performance.

In addition to these changes, this processor also contains an enhancement for Hyper-Threading Technology performance known as the context identifier that was included in some prior processor implementations. With Hyper-Threading Technology, the partial virtual address indexing scheme used for the L1 cache creates conflicts when each logical processor's access pattern matches the partial virtual tag even when accessing separate regions of physical memory. For example, this situation can occur if the stacks of the two threads are offset by a fixed amount that is greater than the size of the partial match, such that these two addresses, although different, alias to the same partial tag. This causes contention in the cache, leading to a reduced cache hit rate. In order to reduce the likelihood of contention, a context identifier bit is added to the partial virtual tag for each logical processor. This bit is dynamically set or reset based on the page-table structure initialization for each logical processor and serves as an indication of data sharing intent across logical processors.

For example, assume that two logical processors share the same page directory base in physical memory. This gives a strong indication that data are intended to be shared between the logical processors. In such a case, the additional context-identifier bit for each logical processor is set to the same value, allowing for sharing of the L1 data cache. Conversely, if the page-directory bases are different, it is likely that both logical processors are working on separate data regions. In such a case, sharing of the L1 data cache is disallowed by keeping the context-identifier bit different across logical processors.

There may be uncommon cases where logical processors use different page directory bases but still share the same physical memory region through page-table aliasing. These arise when two different page table entries across logical processors point to the same physical page frame. The processor detects such cases and implements a reservation mechanism to prevent repetitive L1 cache access conflicts among different logical processors.

SSE3 INSTRUCTIONS

The Intel Pentium 4 processor extends the IA-32 ISA with a set of 13 new instructions. With the exception of three (`fisttp`, `monitor`, `mwait`), these instructions use the SSE registers. These new instructions are designed to improve performance in the following areas:

- x87 to integer conversion (`fisttp`)

- Complex arithmetic (`addsubps`, `addsubpd`, `movsldup`, `movshdup`, `movddup`)

- Video encoding (`lddqu`)

- Graphics (`haddps`, `hsubps`, `haddpd`, `hsubpd`)

- Thread synchronization (`monitor`, `mwait`)

Improved x87 Conversions to Integer

`Fisttp` has been added to provide the ability of IA-32 to ignore the value of the Floating-Point Control Word (FCW) when converting a value from x87 to an integer. Currently on IA-32, a conversion to integer is done with the convert-store instruction `fistp`. The rounding mode used for the conversion is taken from the FCW. In order to meet Fortran and C/C++'s specifications for conversion to integer, the rounding mode has to be set to *chop*, whereas the default rounding mode is usually set to *even* to minimize rounding errors. Because `fistp` gets its rounding mode from FCW, the user has to create a new FCW that is equal to the default one, but with the rounding mode changed to chop. Once FCW is changed, `fistp` can be used to do the conversion. Finally, the user has to restore the default value of FCW. The whole operation involves changing FCW twice, and since `fldcw` is a relatively slow instruction, it can degrade the performance of an application. To alleviate this problem, `fisttp` has been added. It is a new `fistp` instruction that ignores FCW and always uses chop as its rounding mode.

As shown below, the benefit of `fisttp` is two-fold: fewer instructions are needed and there is no need to modify FCW. The instruction is available in three precisions: Word (16-bit), DWord (32-bit), and QWord (64-bit).

Code without `SSE3`:
```
fstcw <old FCW>
movw ax, <old FCW>
or ax, 0xc00
movw <new FCW>, ax
fldcw <new FCW>
fistp <INT>
fldcw <old FCW>
```

Code with `SSE3`:
```
fisttp  <INT>
```

Complex Arithmetic

Complex arithmetic usage is ubiquitous, as it is used in Discrete/Fast Fourier Transform (DFT/FFT), Discrete Multi Tone (DMT) modulators, frequency domain filtering, etc. A typical example of the importance of complex arithmetic in a multimedia context is given by the implementation of an Acoustic Echo Canceller (AEC). In an AEC, a long Finite Impulse Response (FIR) filter is used to model the inverse of the acoustic channel. It is not uncommon for this filter to have 1024 or more taps. The operation done by a FIR filter is called a convolution, and its execution time is $O(n^2)$. With filters of such large length, and with the quadratic cost of a convolution, the operation of filtering in the time domain can be prohibitive, to the point of not meeting, for example, a real-time constraint. By moving from the time domain to the frequency domain, the execution time can be significantly reduced. Because the execution time of a DFT is also $O(n^2)$, moving to the frequency domain does not appear to have saved anything. But DFT has fast implementations with execution time $O(n\log n)$. Such fast implementations of DFT are collectively called FFT. In the frequency domain, a convolution ($O(n^2)$) is simply a point-product ($O(n)$). For a filter with fixed coefficients, the n-element input array can be transformed into the frequency domain in $O(n\log n)$ operations; the point-multiplication (with the frequency domain transformed set of coefficients) takes $O(n)$ operations; the conversion of the result back to the time domain (using an inverse FFT) takes also $O(n\log n)$ operations. For large n, the complexity behaves as $O(n\log n)$, significantly faster than $O(n^2)$.

Three benchmarks out of SPEC[*] CPU2000[*] make heavy use of complex arithmetic: 168.wupwise (BLAS3 ZGEMM – complex matrix multiply), 189.lucas (FFT_SQUARE – a FFT-based function to square large integer numbers), and 187.facerec (FFT).

Five instructions have been added to significantly accelerate complex arithmetic. Two instructions (`addsubps` and `addsubpd`) perform a mix of floating-point addition and subtraction, hence removing the need for changing the sign of some operands. The

three others (`movsldup`, `movshdup`, `movddup`), in their memory version, combine loads with some level of duplication, hence saving the need for a shuffle instruction on the loaded data.

Code without SSE3:

```
movapd     xmm0, <mem_X>
movapd     xmm1, <mem_Y>
movapd     xmm2, <mem_Y>
unpcklpd xmm1, xmm1
unpckhpd xmm2, xmm2
mulpd      xmm1, xmm0
mulpd      xmm2, xmm0
xorpd      xmm2, xmm7
shufpd     xmm2, xmm2, 0x1
addpd      xmm2, xmm1
movapd     <mem_Z>, xmm2
```

Code with SSE3:

```
movapd     xmm0, <mem_X>
movddup    xmm1, <mem_Y>
movddup    xmm2, <mem_Y+8>
mulpd      xmm1, xmm0
mulpd      xmm2, xmm0
shufpd     xmm2, xmm2, 0x1
addsubpd xmm2, xmm1
movapd     <mem_Z>, xmm2
```

The code sequence above shows how to implement a double-precision complex multiplication using SSE2 only or with the new SSE3 instructions, where mem_X contains one complex operand and mem_Y the other; mem_Z is used to store the complex result; and xmm7 is a constant used to change the sign of one data element. Since the main speed limiter of this code is the number of execution uops (7 for SSE2, 4 for SSE3), the new instructions can improve complex multiplications by up to 75%. On SPEC CPU2000, the compiler is able to use SSE3 to improve 168.wupwise by 10-15%.

Video Encoding

The most compute-intensive part of a video encoder is usually Motion Estimation (ME) where blocks from the current frame are checked against blocks from the previous frame to find the best match. Many metrics can be used to define the best match. The most common is the L1 metric: the sum of absolute differences. Due to the nature of ME, loads of the blocks from the previous frame are unaligned whereas loads of the blocks from the current frame are aligned. Unaligned loads suffer two penalties:

- cost of handling the unaligned access

- impact of the cache line splits

The NetBurst microarchitecture does not support a uop to load 128-bit unaligned data. For that reason, 128-bit

unaligned load instructions, such as `movups` and `movdqu`, are emulated with microcode, using two 64-bit loads whose results are merged to form the 128-bit result. In addition to the cost of the emulation, unaligned loads are penalized by the cost of handling cache line splits if the access crosses a 64-byte boundary.

SSE3 adds `lddqu` to solve the cache line split problem on 128-bit unaligned loads. The instruction works by loading a 32-byte block aligned on a 16-byte boundary, extracting the 16 bytes corresponding to the unaligned access. Because the instruction loads more bytes than requested, some usage restrictions apply. `Lddqu` should be avoided on Uncached (UC) and Write-Combining (USWC) memory regions. Also, by its implementation, `lddqu` should be avoided in situations where store-load forwarding is expected. In load-only situations, and with memory regions that are not UC or USWC, `lddqu` can advantageously replace `movdqu`/`movups`/`movupd`.

The code below shows an example of using the new instruction. Both code sequences are similar except that the load unaligned (`movdqu`) is replaced by the new unaligned load (`lddqu`). With the assumption that 25% of the unaligned loads are across a cache line, the new instruction can improve the performance of ME by up to 30%. MPEG* 4 encoders have demonstrated speedups greater than 10%.

Motion Estimator without `SSE3`:

```
movdqa xmm0, <current>
movdqu xmm1, <previous>
psadbw xmm0, xmm1
paddw  xmm2, xmm0
```

Motion Estimator with `SSE3`:

```
movdqa xmm0, <current>
lddqu  xmm1, <previous>
psadbw xmm0, xmm1
paddw  xmm2, xmm0
```

Graphics

Most (graphics) vertex databases are organized as an array of structures (AOS), where each vertex structure contains data fields such as the following:

- x, y, z, w: coordinates of the vertex

- n_x, n_y, n_z, n_w: coordinates of the normal at the vertex

- r, g, b, a: colors at the vertex

- u0, v0: 1st set of 2D texture coordinates

- u1, v1: 2nd set of 2D texture coordinates

---

* Other names and brands are the property of their respective owners.

By its very nature, SSE does not deliver optimal results when operating on vertex databases organized as an AOS. SSE is much better at handling vertex databases organized as a structure of arrays (SOA), where the first array contains the x of all the vertices; the second array, the y of all the vertices; etc. Because AOS is the favored way vertex databases are organized, in order to use SSE, the data have to be loaded and reorganized using shuffle instructions.

The most common operation performed in a vertex shader is the scalar product, where 3 (or 4) pairs of single-precision data elements are multiplied and the 3 (or 4) results summed. Due to the AOS organization of the vertex database, evaluating the scalar product can be challenging with SSE because of the lack of horizontal instructions. We have added horizontal floating-point addition/subtraction instructions to speed up the evaluation of scalar products.

The code sequence below illustrates how a scalar product of four single-precision pairs of elements can be evaluated with and without the new instructions:

Code without SSE3:

```
        mulps   xmm0, xmm1
        movaps  xmm1, xmm0
        shufps  xmm0, xmm1, 0xb1
        addps   xmm0, xmm1
        movaps  xmm1, xmm0
        shufps  xmm0, xmm0, 0x0a
        addps   xmm0, xmm1
```

Code with SSE3:
```
        mulps   xmm0, xmm1
        haddps  xmm0, xmm0
        haddps  xmm0, xmm0
```

Thread Synchronization

Monitor and mwait instructions provide a solution to address Hyper-Threading Technology performance of the operating system idle loop and other spin-wait loops in operating systems and device drivers. Software can use the monitor and mwait instructions to hint that a thread is not doing useful work (e.g., spinning and waiting for work). The processor may then go into a low-power and performance-optimized state. Monitor and mwait provide a way for software to wake up the processor from this low-power/performance-optimized state via a store to a specified memory location (e.g., a store to the work queue).

Monitor sets up hardware to detect stores to an address range, generally a cache line. The monitor instruction relies on a state in the processor called the *monitor event pending flag*. The monitor event pending flag is either set or clear and its value is not architecturally visible except through the behavior of the mwait instruction. The monitor event pending flag is set by multiple events including a write to the address range being monitored and reset by the monitor instruction.

The monitor instruction sets up the address monitoring hardware using the address specified in EAX and resets the monitor event pending flag. A store to the address range will set the monitor event pending flag. Other events will also set the monitor event pending flag, including interrupts or any event that may change the page tables. The content of ECX and EDX are used to communicate other information to the monitor instruction.

Mwait puts the processor into the special low-power/optimized state until a store, to any byte in the address range being monitored, is detected, or if there is an interrupt, exception, or fault that needs to be handled. There may also be other time-outs or implementation-dependent conditions that may cause the processor to exit the optimized state. The mwait instruction is architecturally identical to a nop instruction. It is effectively a hint to the processor to indicate that the processor may choose to enter an implementation-dependent optimized state while waiting for an event or for a store to the address range set up by the preceding monitor instruction in program flow. For example, a Hyper-Threading Technology-capable processor may enter a state that allows the other thread to execute faster, or it may enter a state that allows for lower power consumption, or both.

The monitor and mwait instructions must be coded in the same loop because execution of the mwait instruction will clear the monitor address range. It is not

possible to execute `monitor` once and then execute `mwait` in a loop. Setting up `monitor` without executing `mwait` has no adverse effects.

Typically the monitor/mwait pair is used in a sequence like this:

```
EAX = Logical Address(Trigger)
ECX = EDX = 0                          // Hints
While ( !trigger_store_happened) {
        MONITOR EAX, ECX, EDX
        If ( !trigger_store_happened ) {
                MWAIT EAX, ECX
        }
}
```
The above code sequence makes sure that a triggering store does not happen between the first check of the

trigger and the execution of the `monitor` instruction. Without the second check that triggering store would go un-noticed.

It is expected that operating systems will use the `monitor` and `mwait` instructions to significantly improve the performance of idle loop handling and allow the system to provide higher performance at lower power consumption.
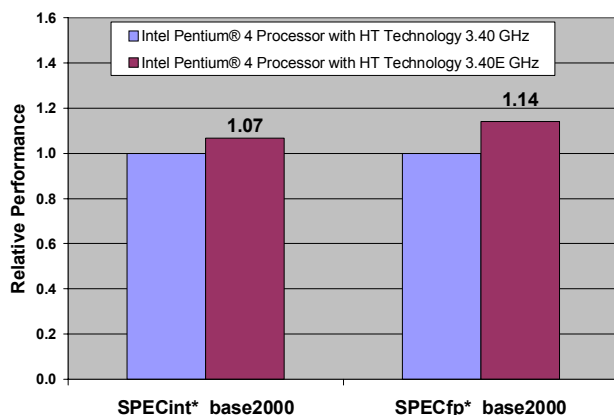
PERFORMANCE

Given all of these changes in the 90nm version of the Intel Pentium 4 processor, the real question is how much performance benefit will be realized on applications from making these changes.

Table 2: Detailed system configuration for results shown in Figure 5

| Processor	Pre-production Intel® Pentium® 4 processor 3.40 GHz supporting Hyper-Threading Technology	Pre-production Intel® Pentium® 4 processor 3.40E GHz supporting Hyper-Threading Technology
Motherboard	Intel Desktop Board D875PBZ AA-204	Intel Desktop Board
Motherboard BIOS	BZ87510A.86A.0041.P09	Pre-production BIOS
Cache	512KB full-speed Advanced Transfer Cache	1MB full-speed Advanced Transfer Cache
Memory Size	1 GB (2x512MB) PC3200 DDR400 (Samsung* PC3200U-30331-B2 M368L6423ETM-CCC CL3 Double-Sided DDR400 memory)	1 GB (2x512MB) PC3200 DDR400 (Samsung* PC3200U-30331-B2 M368L6423ETM-CCC CL3 Double-Sided DDR400 memory)
Hard Disk	Seagate* ST3160023AS 160 GB Serial ATA (SATA) (7200 RPM, 8MB cache)	Seagate* ST3160023AS 160 GB Serial ATA (SATA) (7200 RPM, 8MB cache)
Hard Disk Driver	Intel Application Accelerator RAID Edition 3.5 with RAID ready	Intel Application Accelerator RAID Edition 3.5 with RAID ready
Video Controller/Bus	ATI* Radeon* 9800 Pro 8x AGP	ATI* Radeon* 9800 Pro 8x AGP
Video Memory	128 MB DDRAM	128 MB DDRAM
Operating System	Microsoft* Windows* XP Professional, Build 2600, Service pack 1 on NTFS Default Microsoft DirectX* 9.0b	Microsoft* Windows* XP Professional, Build 2600, Service pack 1 on NTFS Default Microsoft DirectX* 9.0b
Video Driver Revision	ATI Catalyst* 3.5 Driver Suite: display driver version: 6.14.10.6360	ATI Catalyst* 3.5 Driver Suite: display driver version: 6.14.10.6360
Graphics	1024x768 resolution, 32-bit color	1024x768 resolution, 32-bit color
SPEC* CINT2000	Intel C++ Compiler Plug-in V8.0 Microsoft Visual Studio* .NET V7.0 (for libraries)	Intel C++ Compiler Plug-in V8.0 Microsoft Visual Studio* .NET V7.0 (for libraries)
SPEC* CFP2000	Intel C++ Compiler Plug-in V8.0 and Intel FORTRAN Compiler Plug-in V8.0 Microsoft Visual Studio .NET V7.0 (for libraries)	Intel C++ Compiler Plug-in V8.0 and Intel FORTRAN Compiler Plug-in V8.0 Microsoft Visual Studio .NET V7.0 (for libraries)

Table 3: Detailed system configuration for results in Figure 6

Processor	Pre-production Intel® Pentium® 4 processor 3.40E GHz supporting Hyper-Threading Technology
Motherboard	Intel Desktop Board
Motherboard BIOS	Pre-production BIOS
Cache	1MB full-speed Advanced Transfer Cache
Memory Size	512MB (4x128MB) Samsung PC3200U-30330-C3 M368L1624DTM-CCC 128MB DDR PC3200 CL3 Single-Sided DDR400 memory
Hard Disk	IBM 120GXP 80 GB IC35L080AVVA07-0 ATA-100
Hard Disk Driver	MS default UDMA-5
Video Controller/Bus	ATI Radeon 9700 Pro AGP graphcis
Video Memory	128 MB DDRAM
Operating System	Microsoft* Windows* XP Professional, Build 2600, Service pack 1 on NTFS Default Microsoft DirectX* 9.0b
Video Driver Revision	ATI CATALYST 6.13.10.6166 driver
Graphics	1024x768 resolution, 32-bit color

Figure 5: Performance comparison (estimated SPEC* CPU2000* performance as measured on pre-production hardware)

Figure 5 compares the performance of this new processor with the performance of the 130nm version of the Intel Pentium 4 processor with a 512kb L2 cache on SPEC CPU2000, as estimated on pre-production hardware. Detailed system configuration information is shown in Table 2. As can be seen here, the performance enhancements that have been described in this paper do have a noticeable effect on overall performance.
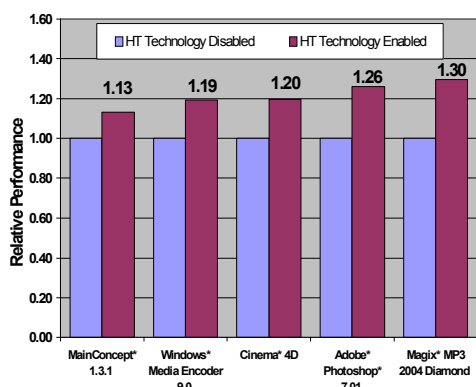
Figure 6: Performance benefit of Hyper-Threading Technology

Hyper-Threading Technology on this processor also shows significant benefits on popular consumer applications and for various multi-tasking scenarios. Figure 6 compares the performance on some of these applications and scenarios when Hyper-Threading Technology is enabled and disabled on this processor.

Table 3 lists the detailed system configuration for these results.

CONCLUSION

The NetBurst microarchitecture that was introduced with the Intel Pentium 4 processor brought unprecedented levels of performance to the end user through its unique features such as the Execution Trace Cache and an execution core that ran at 2x the core frequency. Now, we are building upon the strength of those previous processors with the new Intel Pentium 4 processor manufactured on the 90nm process. With these new performance features and enhancements, the performance of desktop processors continues to reach new heights. With capabilities like Hyper-Threading Technology and a set of new instructions, building blocks are being provided for software to be created to take advantage of this power and deliver to users a new level of functionality on their desktop.

ACKNOWLEDGMENTS

The authors thank all of the architects, designers, and validators around the world who collaborated in the creation of this product.

REFERENCES

[1]. Hinton, G.; Sager, D.; Upton, M.; Boggs, D.; Carmean, D.; Kyker, A.; Roussel, P., "The Microarchitecture of the Pentium® 4 Processor," *Intel Technology Journal* Q1, 2001.

[2]. Marr, D.; Binns, F.; Hill, D.; Hinton, G.; Koufaty, D.; Miller, J.; Upton, M., "Hyper-Threading Technology Architecture and Microarchitecture: A Hypertext History," *Intel Technology Journal*, Q1, 2002. http://developer.intel.com/technology/itj/2002/volume06issue01/

[3]. Gochman, S.; Ronen, R.; Anati, I.; Berkovits, A.; Kurts, T.; Naveh, A.; Saeed, A.; Sperber, Z.; and Valentine, R., "The Intel® Pentium® M Processor: Microarchitecture and Performance," *Intel Technology Journal*, Q2, 2003. http://developer.intel.com/technology/itj/2003/volume07issue02/

AUTHORS' BIOGRAPHIES

Darrell Boggs is a senior principal engineer/architect with Intel Corporation and has been working as a microarchitect for 12 years. He graduated from Brigham Young University with a M.S. degree in Electrical Engineering. Darrell played a key role on the Intel® Pentium® Pro processor design, and was one of the key

---

® Pentium is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

architects of the Pentium 4 processor. Darrell holds many patents in the areas of register renaming; instruction decoding; events and state recovery mechanisms; speculative architectures; and Hyper-Threading Technology. His e-mail address is darrell.boggs at intel.com.

Aravindh Baktha has been with Intel for 12 years. He worked on the design and microarchitecture of the Pentium 4 processor. Prior to joining the Pentium 4 processor team, Aravindh worked on the design of the 80960HA processor in Arizona and the Itanium® processor in California. Aravindh received his undergraduate degree from the University of Zambia and his M.S. degree in Electrical and Computer Engineering from Illinois Institute of Technology. His e-mail address is aravindh.baktha at intel.com

Jason M. Hawkins received his B.S. degree in Electrical and Computer Engineering from Brigham Young University. He joined Intel in 1997 and has focused on validation and microarchitecture of the Pentium 4 family of processors. His e-mail address is jason.hawkins at intel.com.

Deborah T. Marr is the CPU architect responsible for Hyper-Threading Technology in the Desktop Products Group. Deborah has been at Intel for over thirteen years. She first joined Intel in 1988 and made significant contributions to the Intel 386SX processor, the Pentium Pro processor, and the Pentium 4 processor. Her interests are in high-performance microarchitecture and performance analysis. Deborah received her B.S. degree in EECS from the University of California at Berkeley in 1988 and her M.S. degree in ECE from Cornell University in 1992. Her e-mail address is debbie.marr at intel.com.

John (Alan) Miller has worked at Intel for over seven years. During that time, he has worked on design and architecture for the Pentium 4 processor. Alan obtained his M.S. degree in Electrical and Computer Engineering from Carnegie Mellon University. His e-mail address is alan.miller at intel.com.

Patrice Roussel graduated from the University of Rennes in 1980 and L'Ecole Supérieure d'Electricité in 1982 with a M.S. degree in signal processing and VLSI design. Upon graduation, he worked at Cimatel, an Intel/Matra Harris joint design center. He moved to the USA in 1988 to join Intel in Arizona and worked on the 960CA microprocessor. In late 1991, he moved to Intel in Oregon to work on the Pentium Pro processor. Since 1995, he has been the floating-point architect of the Pentium 4 processor. His e-mail address is patrice.roussel at intel.com.

Ronak Singhal received his B.S. and M.S. degrees in Electrical and Computer Engineering from Carnegie Mellon University. He subsequently joined Intel in 1997 and has spent the majority of his time focused on microarchitecture performance analysis and verification for the Pentium 4 processors. His e-mail address is ronak.singhal at intel.com.

Bret Toll received his B.S. degree in Electrical Engineering from Portland State University and M.S. degree in Computer Science and Engineering from Oregon Graduate Institute. He joined Intel in 1993 and has focused on microcode, machine check architecture, and instruction decode microarchitecture for the Pentium 4 processor. In his spare time he likes to tinker with cars and is currently building a 1965 Ford roadster from a kit of components and hand-picked items from junkyards. His e-mail address is bret.toll at intel.com.

K. S. Venkatraman received his B.S. degree from Birla Institute of Technology and M.S. degree from Villanova University. He joined Intel in 1997 and has focused on microarchitecture as well as post-silicon performance analysis for the Pentium 4 processor. In his spare time, he enjoys amateur radio and riding his motorcycle. His e-mail address is k.s.venkatraman at intel.com.

® Itanium is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

THIS PAGE INTENTIONALLY LEFT BLANK

Support for the Intel® Pentium® 4 Processor with Hyper-Threading Technology in Intel® 8.0 Compilers

Kevin B. Smith, Enterprise Platforms Group, Intel Corporation
Aart J.C. Bik, Enterprise Platforms Group, Intel Corporation
Xinmin Tian, Enterprise Platforms Group, Intel Corporation

Index words: Compilers, Intel Pentium 4 processor, Optimization, OpenMP, Hyper-Threading Technology, Vectorization

ABSTRACT

Intel's 8.0 compilers enable software developers to take advantage of the new architectural and micro-architectural features of the latest Intel® Pentium® 4 processor with Hyper-Threading Technology. This paper describes the support for both automatic optimization techniques and programmer-controlled methods of achieving high performance using the Intel 8.0 C++ and FORTRAN compilers. Details of both the automatic and programmer-controlled optimization techniques are presented. Results show that use of this compiler can significantly speed up software running on this new processor.

INTRODUCTION

The latest Intel Pentium 4 processor with Hyper-Threading Technology contains new features, both architectural and micro-architectural, which Intel's 8.0 compiler family uses to significantly increase software performance. This paper shows how the Intel 8.0 C++ and FORTRAN compilers enable software developers to take advantage of the new features of this latest Intel processor.

The latest Intel Pentium 4 processor implements a set of new instructions called the Streaming-SIMD-Extensions 3 (SSE3). We present an overview of these new instructions. Similarly, we discuss the new micro-architectural features and changes that affect the compiler.

Intel's compilers support both automatic optimization techniques and programmer-controlled methods to achieve high-performance software. The compiler provides two automatic optimization techniques to gain performance from recompilation: vectorization and advanced instruction selection. We describe new vectorization capabilities focusing on how these improve performance on the new processor. Then we present a section on advanced instruction selection providing details on both the implementation of complex data operations using the new SSE3 instructions, and on how micro-architectural changes affect instruction selection for other operations. The mapping of complex operations onto SSE/SSE2/SSE3 instructions is also shown, and we contrast this with the implementation of complex data types when generating code for Intel processors that do not support the SSE3 instructions. Next, we discuss how changes to the compiler's advanced instruction selection are motivated by the processor's micro-architectural changes. Experimental results show that together these improvements to automatic optimization techniques can speed up software by up to 25%.

Intel's compilers also offer programmers the ability to leverage performance features of Intel's processors by making changes to their source code. There are two kinds of source-level changes that the user can perform to take advantage of this new processor's performance features: direct insertion of SSE3 instructions and insertion of OpenMP[*] [7, 8] directives. The direct insertion of SSE3 instructions can be done either with intrinsic functions, which map directly to SSE3 instructions, or with inline assembly code containing SSE3 instructions. Both methods are discussed in this paper. Insertion of OpenMP

---

[*] Other brands and names are the property of their respective owners.

directives allows the programmer to take advantage of improved Hyper-Threading (HT) Technology support in this latest Intel Pentium 4 processor. Results are presented showing that our OpenMP implementation works with the improved HT Technology to produce significant performance improvements.

We conclude by summarizing the ways in which Intel's 8.0 compilers enable programmers to extract maximum performance from this latest Intel processor, and we review the performance improvements that have been achieved using these methods.

Throughout the paper, assembly examples follow the conventional format (see [13]).

NEW FEATURES

The latest Intel Pentium 4 processor contains both architectural and micro-architectural changes that the compiler uses to increase software performance.

New Instructions

The Streaming-SIMD-Extensions 3 supports 11 new instructions that can be used by the compiler to boost software performance: *addsubpd*, *addsubps*, *haddpd*, *haddps*, *hsubpd*, *hsubps*, *movddup*, *movshdup*, *movsldup*, *fisttp*, and *lddqu*. The new addsubpd and addsubps instructions allow one vector element to be subtracted while its next vector element is added. The haddpd, haddps, hsubpd, and hsubps instructions provide the capability to add or subtract horizontally within a vector, which enables more efficient clean-up code at the end of vectorized reduction loops. The movddup, movshdup, and movsldup instructions allow duplication of certain data elements into a vector. The lddqu instruction is a more efficient form of movups, useful when a memory load is likely to cross a cache-line boundary. Finally, the fisttp instruction provides a more efficient implementation of floating-point to integer conversion.

Micro-Architectural Changes

In addition to providing new instructions, the latest Intel Pentium 4 processor also has micro-architectural changes that the compiler can use to boost software performance. The latency of the multiply, shift, rotate, and some SSE/SSE2 instructions has been decreased. This reduced latency makes it more desirable than it used to be for the compiler to use these instructions. The processor's ability to forward stored data to a subsequent load that overlaps the store has also been improved. This allows more aggressive vectorization, since there is less probability that inefficiencies due to store mis-forwarding occur as a side effect. There are also many micro-architectural changes, which significantly improve Hyper-Threading Technology performance, and increase the opportunities

for software performance improvement as a result of threading of an application.

AUTOMATIC OPTIMIZATION TECHNIQUES

New compiler options enable the generation of SSE3 instructions and tuning specifically for the micro-architectural changes in the latest Intel Pentium 4 processor. On Microsoft Windows* platforms, the /QxP option [11] directs Intel's compilers to use SSE3 and to tune for the new processor (for Linux* platforms the –xP option [12] has the same effect). Additional options are available (/QaxP [11] for Windows and –axP [12] for Linux) that direct the compiler to generate special high-performance copies of functions that can be sped-up using automatic techniques targeted for the new processor. These high-performance function copies will only be executed on the new processor, while on older Intel processors a less optimized version will be executed. When these options are specified, two classes of automatic optimizations are used to improve software performance: vectorization and advanced instruction selection.

Vectorization

Multimedia extensions provide a convenient way to exploit fine-grained parallelism in an application. Because manually rewriting sequential software into a form that exploits multimedia extensions can be rather cumbersome, vectorizing compilers have proven to be necessary tools for making multimedia extensions easier to use. Details of the vectorization methodology used by the Intel® C++ and FORTRAN compilers are given elsewhere [2][3]. This section briefly discusses aspects of vectorization that are specific to exploiting SSE3 instructions.

Vectorization of Single-Precision Complex Data Types

A complex number $c \in \mathbb{C}$ has the form

$$c = x + y \cdot i$$

where $x, y \in \mathbb{R}$ denote the real part and imaginary part, respectively. The C99 standard [5] and FORTRAN have built-in single-precision and double-precision complex data types that simplify programming with complex numbers by assigning the usual complex semantics to operators like + (addition), - (subtraction), * (multiplication), and / (division) when applied to operands with a complex data type. Some SSE3

---

* Other brands and names are the property of their respective owners.

instructions are particularly useful to vectorize these complex operations in an efficient manner. Consider, for example, the FORTRAN code shown below.

```
complex a(10), b(10), c(10)
do i = 1, 10
   a(i) = b(i) * c(i)
enddo
```

The data layout of the single-precision complex arrays interleaves the 4-byte real parts and 4-byte imaginary parts of all complex elements in the 80-byte chunk of memory allocated for each array. Since complex multiplication is defined as

$$c \cdot c' = (x \cdot x' - y \cdot y') + (x \cdot y' + y \cdot x') \cdot i$$

the Intel compiler can translate the DO-loop shown above into the following sequence of SSE and SSE3 instructions.

```
L:
movaps     xmm0,    XMMWORD PTR b[eax]
movsldup   xmm2,    XMMWORD PTR c[eax]
mulps      xmm2,    xmm0
movshdup   xmm1,    XMMWORD PTR c[eax]
shufps     xmm0,    xmm0, 177
mulps      xmm1,    xmm0
addsubps   xmm2,    xmm1
movaps     XMMWORD PTR a[eax], xmm2
add        eax,     16
cmp        eax,     80
jb         L
```

Since vector iterations process two complex data elements at one time, the loop only iterates five times. Furthermore, fine-grained computational overlap between operations on the real and imaginary parts of each complex data element is obtained.
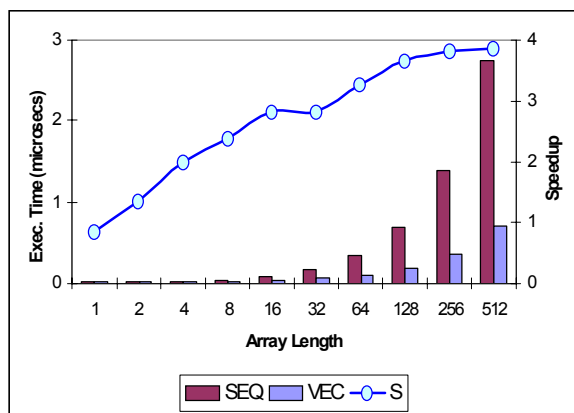
Figure 1: Single-precision complex vectorization speedup

Figure 1 plots the execution times (in microseconds) of the FORTRAN loop shown above using a sequential FPU-based implementation (SEQ) and a vectorized implementation using SSE3 instructions (VEC) for various array lengths using a 2.8GHz Intel Pentium 4 processor with HT Technology. Execution times were obtained by running the kernel repeatedly and dividing the total runtime accordingly. The corresponding speedup (S) is shown in the same figure using a secondary y-axis. Clearly, vectorizing single-precision complex operations by means of SSE3 instructions already pays off for arrays of length two or more, with the speedup going up to almost four.

Vectorization for SSE3 Idioms

Some SSE3 instructions enable a slightly more efficient implementation of constructs that were already vectorized by previous versions of Intel's compilers. The instruction haddps, for instance, simplifies accumulating partial sums after a vectorized reduction. The following C example

```
float a[100], red;
…
red = 0;
for (i = 0; i < 100; i++) {
   red += a[i];
}
```

is vectorized as shown below, where the loop body exploits four-way SIMD parallelism to implement the sum-reduction. The post loop clean-up code uses two horizontal add instructions to reduce the four partial sums back into one final sum again (formerly, this required six instructions).

```
    pxor      xmm0, xmm0
L:  addps     xmm0, XMMWORD PTR a[eax]
    add       eax,  16
    cmp       eax,  400
    jb        L
    haddps    xmm0, xmm0
    haddps    xmm0, xmm0
    movss     DWORD PTR red,  xmm0
```

Other SSE3 instructions provide a more compact way to implement some frequently occurring data rearranging instruction sequences.

Improved store-to-load forwarding in the latest Intel Pentium 4 processor also allows the Intel 8.0 compilers to become more aggressive in exploiting fine-grained SIMD parallelism in straight-line code. As an example, the structure shown below

```
struct {
  double x;
  double y;
} v;

...
v.x  += 1.0;
v.y  += 2.0;
```

can be mapped to only a few SSE2 instructions.

```
    movapd    xmm0, XMMWORD PTR v
    addpd     xmm0, const_1.0_2.0
    movapd    XMMWORD PTR v, xmm0
```

Such straight line code vectorization offers the compiler many more opportunities to improve the performance using SSE/SSE2/SSE3 instructions than more traditional loop-oriented vectorization. Collapsing fine-grained parallelism into enclosing loops exposes more iterations to a vector loop. This increases the probability that loops can be effectively vectorized and enables the compiler to apply more advanced methods, such as dynamic loop peeling for alignment or dynamic data dependence testing (see [3] for details). The two statements in the following loop-body, when taken in isolation, do not expose enough parallelism to enable use of 4-way parallel SSE instructions.

```
struct {
  float x;
  float y;
} a[100];

for (i = 0; i < 100; i++) {
   a[i].x = 0;
   a[i].y = 0;
}
```

However, when collapsed into the loop the full potential of SSE instructions can be exploited, as shown below.

```
    pxor      xmm0, xmm0
    xor       eax, eax
L:  movaps    XMMWORD PTR a[eax], xmm0
    add       eax, 16
    cmp       eax, 800
    jb        L
```

Interprocedural Alignment Analysis

Like most instructions, multimedia instructions operate more efficiently when memory operands are aligned at their natural boundary, i.e., 64-bit memory operands should be 8-byte aligned and 128-bit memory operands should be 16-byte aligned. In order to obtain aligned access patterns in vector loops, the Intel compilers perform advanced static and dynamic alignment analysis and an enforcement method, as described elsewhere [2][3]. The Intel 8.0 compilers further extend this support with interprocedural alignment analysis. This analysis consists of finding maximal values 2^n in a mapping ALIGN such that for a function f(), the value

$$ALIGN(f, p) = <2^n, o> \quad \text{with } 0 \le o < 2^n$$

denotes that all actual arguments that are associated with formal pointer/call-by-reference argument p evaluate to an address A that satisfies A mod 2^n = o.

By defining an alignment lattice of the form

```
<16,0>…                                      … <16,15>
  <8,0><8,4> <8,2><8,6> <8,1><8,5> <8,3><8,7>
     <4,0>  <4,2>          <4,1>  <4,3>
        <2,0>                  <2,1>
              <1,0>
```

with the meet operator going toward bottom element <1,0> (i.e., arbitrary alignment), and jump functions that use modulo arithmetic to correlate incoming formal arguments with any outgoing expression used as actual argument, the problem can be solved similar to the algorithm given in [4] for interprocedural constant propagation. More details can be found in [3]. A similar approach to propagating alignment information within a function was proposed by Larsen et. al. in [6].

More precise knowledge on the alignment of data structures can substantially increase the effectiveness of using SSE3 instructions. The compiler's ability to perform interprocedural alignment analysis is partially responsible for the large performance improvement that is seen in the results section for the 168.wupwise benchmark.

Advanced Instruction Selection

As stated earlier, both FORTRAN and C99 support basic data types used for representing complex numbers. Advanced instruction selection uses the new SSE3 instructions to implement these complex types' basic operations. Additionally, micro-architectural changes create opportunities for advanced instruction selection optimizations to tune for the latest Intel Pentium 4 processor. Both of these forms of instruction selection will be covered in the following sections.

Implementing Complex Operations with SSE3

As described previously, the representation of the complex data types interleaves the real and imaginary parts in memory. The real portion occupies memory at the lowest address and the imaginary portion occupies memory immediately above the real portion. With the addition of SSE3 instructions, operations on complex data map well onto the vector instruction sets provided on the latest Intel Pentium 4 processor.

The C99 Standard [5] defines many operations and library routines that perform computation on complex data items. In addition to the basic operations mentioned previously, other complex operations are provided by library routines. These additional operations include complex conjugate (conj), extract real part (creal), and extract imaginary part (cimag). In the Intel compilers' intermediate representation, both single- and double-precision complex data types and operations are directly represented and optimized when generating code targeting the latest Intel Pentium 4 processor. At the transition from machine-independent optimization to machine-specific code generation, the complex operations are translated directly

into SSE, SSE2, and SSE3 instructions. This occurs for the basic operations and for the simpler library routines. We discuss how these operations are translated in the next section.

Complex Loads and Stores

The most basic of operations on complex data are loading and storing. For single-precision complex data, the movlps instruction is used to move the data into the lower two single-precision vector elements of an xmm register. This same instruction is also used for storing single-precision complex data. For double-precision complex data, the movapd instruction will be used to load and store the data to/from an xmm register. This instruction can only be used if the effective address for the load/store is known to be 16-byte aligned. In the event that the compiler cannot prove alignment of the data, then a sequence of movlpd, movhpd instructions will be used to perform the load/store. The ability to use the aligned forms of these instructions has a significant impact on the performance of the resultant code, as discussed previously.

Creation of Complex from Real, Imaginary Parts

Complex numbers are often created from separate expressions for the real and imaginary parts. This is done using the unpcklps or unpcklpd instructions. For the double-precision complex creation operation, if both expressions are loaded directly from memory, then this can be more efficiently done with the movlpd, movhpd instructions. A common operation of creating a double-precision complex with an imaginary part of 0.0 from a real expression is done with an xorpd instruction, followed by a movsd instruction. When the real value is a memory expression, the xorpd is unnecessary as movsd from memory zeros the upper vector element.

Complex Addition and Subtraction

The operations of complex addition and subtraction are defined as adding or subtracting the real and imaginary portions of the data. These operations map straightforwardly onto the addps/subps and addpd/subpd instructions, respectively, for single- and double-precision complex data.

Complex Conjugate

The complex conjugate operation is defined as leaving the real portion of the complex number unchanged, while negating the imaginary portion. The fastest implementation of this is done using the xorps/xorpd instructions with a constant having only the sign bit of the imaginary part set. This toggles the sign bit of the imaginary part, while leaving the real portion unchanged. The implementation of complex conjugate for a double-

precision complex data object that resides in the xmm0 register is shown below.

```
_floatpack.1: DWORD 0x80000000
              DWORD 0x00000000
              DWORD 0x00000000
              DWORD 0x00000000
…xorpd xmm0, XMMWORD PTR _floatpack.1
```

Complex Multiplication

The definition of complex multiplication was given above in discussing vectorization of single-precision complex data types. Complex multiplication of two operands A and B of double-precision complex type is handled quite similarly. The code sequence is shown below.

```
movapd    xmm0, XMMWORD PTR A
movddup   xmm2, QWORD PTR   B
mulpd     xmm2, xmm0
movddup   xmm1, QWORD PTR   B+8
shufpd    xmm0, xmm0, 1
mulpd     xmm1, xmm0
addsubpd  xmm2, xmm1
movapd    XMMWORD PTR C, xmm2
```

As can be seen above, the movddup and addsubpd SSE3 instructions make the implementation of complex multiplication quite efficient. In contrast, the code generated for this same operation when targeting older Intel Pentium 4 processors is shown below.

```
movsd     xmm3, QWORD PTR A
movapd    xmm4, xmm3
movsd     xmm5, QWORD PTR A+8
movapd    xmm0, xmm5
movsd     xmm1, QWORD PTR B
mulsd     xmm4, xmm1
mulsd     xmm5, xmm1
movsd     xmm2, QWORD PTR B+8
mulsd     xmm0, xmm2
mulsd     xmm3, xmm2
subsd     xmm4, xmm0
movsd     QWORD PTR C, xmm4
addsd     xmm5, xmm3
movsd     QWORD PTR C, xmm5
```

As can be seen this sequence uses the scalar SSE2 instructions and isn't as efficient in either code size or execution time.

Complex Real and Imaginary Part Extraction

Both the creal and cimag operations are generated using the vector instruction sets. The creal operation requires no instructions at all. The compiler simply uses scalar SSE/SSE2 instructions to operate on the low vector element in future computations. For the cimag operation, if the complex value is in memory, the compiler simply adjusts the memory address and size to refer to the imaginary part of the value. When the value is in an xmm register, then a movshdup or unpckhpd instruction is used to copy the imaginary value into the lowest vector element of the register. Thereafter, scalar SSE/SSE2

operations will be performed using the resulting value in the lowest vector element.

Partial Constant Propagation and Folding

The expansion of complex operations into vector instructions can result in some partially redundant operations. For example in the code below, a double-precision complex value A is multiplied by a scalar B that has been cast to a double-precision complex value. After the cast the complex representation of B has an imaginary part of 0.0.

```
C = A * (double _Complex)B;
```

The expansion of this multiplication would be as shown below.

```
movapd    xmm0, XMMWORD PTR A
movsd     xmm2, QWORD PTR   B
movapd    xmm1, xmm2
movddup   xmm2, xmm2
mulpd     xmm2, xmm0
unpckhpd  xmm1, xmm1
shufpd    xmm0, xmm0, 1
mulpd     xmm1, xmm0
addsubpd  xmm2, xmm1
movapd    XMMWORD PTR C, xmm2
```

The unpckhpd in the above sequence serves to duplicate the imaginary part into both high and low vector elements. However, the upper vector element of xmm1 can be proven to be zero, so the unpckhpd instruction just creates a vector of 0.0. This zero vector will be used as an operand to a mulpd; therefore, the mulpd will also produce a zero vector. The result of the mulpd is an operand of addsubpd. A source operand of a zero vector is an identity operand for addsubpd, passing through the destination operand unchanged. So after partial constant propagation and arithmetic simplification, the above code has been optimized into the code below.

```
movapd    xmm0, XMMWORD PTR A
movddup   xmm2, QWORD PTR   B
mulpd     xmm2, xmm0
movapd    XMMWORD PTR C, xmm2
```

Partial constant folding and arithmetic simplification allows extremely efficient generation of complex operations involving scalar values converted to complex, a frequent occurrence. In a micro-benchmark containing only complex multiplication by a scalar, this optimization improves execution time by 55%.

Maximal Use of SSE/SSE2/SSE3 Instructions

The latest Intel Pentium 4 processor has improved latency for some of the frequently used operations in SSE and SSE2 instructions. For example, cvtps2pd latency is improved from seven cycles in earlier Intel Pentium 4 processor implementations to three cycles. In addition, micro-architectural improvements have raised the overall performance of the SSE/SSE2 instruction sets. The Intel 8.0 compilers now use SSE/SSE2/SSE3 instructions for all possible floating-point operations when generating code targeted for the latest Intel Pentium 4 processor. This improves performance of many floating-point-intensive applications, particularly those where performance was limited by denormal exception processing. The improvement in denormal processing is a result of using the flush-to-zero (FTZ) and denormals-are-zero (DAZ) modes that are available only with the SSE, SSE2, and SSE3 instruction sets.
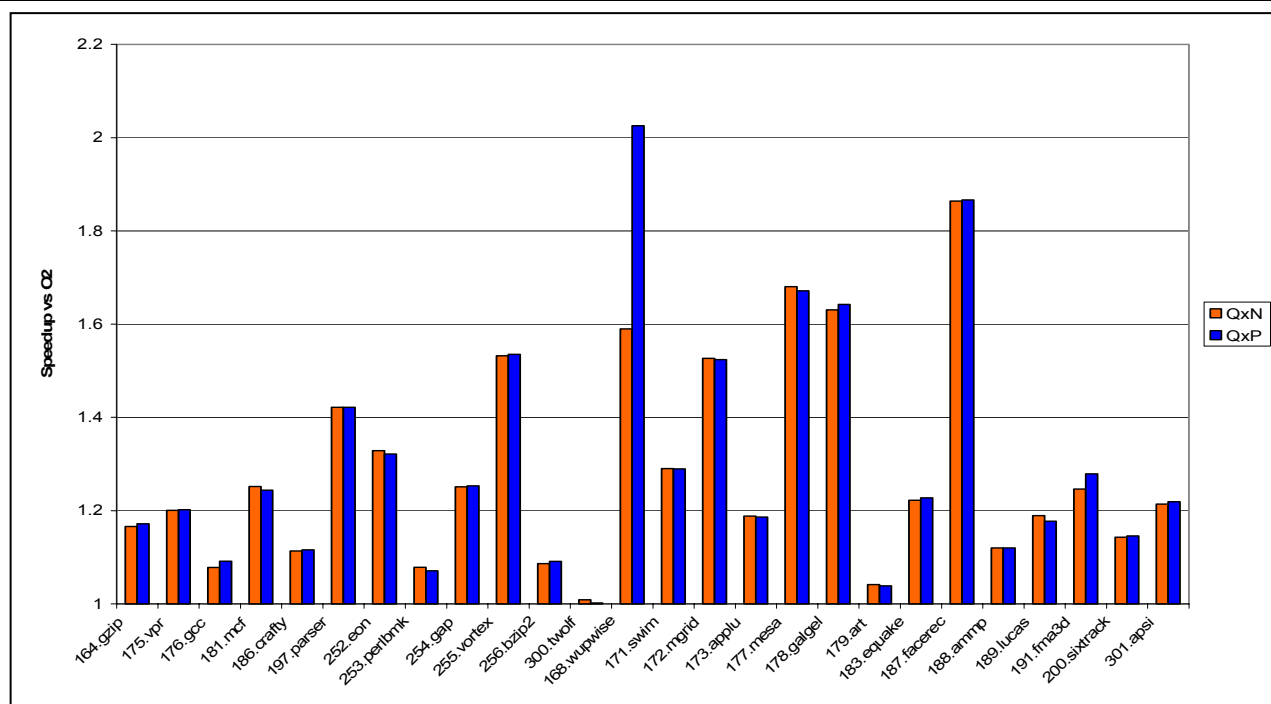
Figure 2: Processor-specific speedup of SPEC* CPU 2000 estimates (based on measurements on Intel internal platforms)

Lower Multiply/Shift Latency

The latency of the imul and shl instructions has been improved in the new Intel Pentium 4 processor. This caused changes to the compiler's heuristics for expansion of multiplication by compile-time constants. These changes result in more compact code at the same performance level. For example, when generating code for older Intel Pentium 4 processors, a multiply of edx by 68 produces

```
lea    ecx, DWORD PTR [edx+edx]
add    ecx, ecx
lea    eax, DWORD PTR [ecx+ecx]
add    eax, eax
add    eax, eax
add    eax, eax
add    eax, ecx
```

However, using compiler option /QxP, the code below is produced for the same operation.

```
mov    ecx, edx
shl    ecx, 6
lea    eax, DWORD PTR [ecx+edx*4]
```

The resulting code is smaller and has slightly better performance than the earlier sequence.

---

* Other brands and names are the property of their

respective owners.

Fisttp Instruction Usage

The SSE3 fisttp instruction is useful for converting floating-point data to integer. For conversion from floating-point format to 32-bit integer or smaller data types the compiler will use the cvtsd2si or cvtpd2dq instructions. However, for converting from floating-point format to 64-bit signed or unsigned integer format, the fisttp instruction is most efficient, and the compiler will use the instruction in that circumstance. This is illustrated by the following snippet of C code.

```
__int64 llfunc(double in)
{
    return (__int64)(in);
}
```

The code generated for this by the Intel 8.0 C++ Compiler using the –QxP –O2 options follows.

```
_llfunc PROC NEAR
        sub      esp, 20
        fld      QWORD PTR [esp+24]
        fisttp   QWORD PTR [esp]
        mov      eax, DWORD PTR [esp]
        mov      edx, DWORD PTR [esp+4]
        add      esp, 20
        ret
```

For comparison, the code below is generated for the same code when using the –QxN –O2 options. Note the use of the fnstcw and fldcw instructions that are necessary to assure that rounding is done towards 0 as required by the C language.

```
_llfunc PROC NEAR
        sub        esp, 20
        fld        QWORD PTR [esp+24]
        fnstcw     [esp+8]
        movzx      eax, WORD PTR [esp+8]
        or         eax, 3072
        mov        DWORD PTR [esp+16], eax
        fldcw      [esp+16]
        fistp      QWORD PTR [esp]
        fldcw      [esp+8]
        mov        eax, DWORD PTR [esp]
        mov        edx, DWORD PTR [esp+4]
        add        esp, 20
        ret
```

The code using the new fisttp instruction is 3 times faster than the code using the older code sequence as measured on a micro-benchmark doing only conversion of double-precision floating-point values to signed 64-bit integers.

SPEC* CPU2000 Performance Results

To give a flavor of the impact of processor-specific optimizations, some performance results are given for SPEC* CPU2000 [9]. This industry-standardized benchmark suite consists of 14 floating-point and 12 integer C/C++ and FORTRAN benchmarks that are derived from real-world applications. The graph in Figure 2 shows the speed-ups obtained on a 2.8GHz Intel Pentium 4 processor with HT Technology for each of the SPEC CPU2000 benchmarks. The bars denoted QxP represent the ratio of the performance of executables obtained using high-level, interprocedural, profile-guided, and processor-specific optimizations for the latest Intel Pentium 4 processor with HT Technology processor (-O3 -Qipo -Qprof_use -QxP) compared with the performance of executables that result when using default optimizations (-O2). Similarly, the bars denoted QxN represent the ratio of the performance of executables obtained using high-level, interprocedural, profile-guided, and processor-specific optimizations for older Intel Pentium 4 processors (-O3 -Qipo -Qprof_use -QxN) compared with the performance of the default executables. The results reveal the advantages of the latest processor-specific optimizations particularly for programs such as 168.wupwise, where performance is highly dependent on the speed of complex arithmetic. For 168.wupwise, this results in a 25% improvement compared to older Intel Pentium 4 processor-specific optimizations, and a 2X performance improvement compared to default (-O2) level of optimization. The results also show that significant performance improvements can be obtained using the latest Intel Pentium 4 processor even when using processor-specific optimizations (-QxN) that are targeted at older Intel Pentium 4 processors.

PROGRAMMER-GUIDED OPTIMIZATION TECHNIQUES

In addition to automatic optimization techniques, the Intel 8.0 compilers also support programmer-controlled methods of improving software performance for the latest Intel Pentium 4 processor. The Intel C/C++ compilers support intrinsic functions, which the compiler maps directly to the Streaming-SIMD-Extensions 3 (SSE3). These intrinsics can be fully optimized by the compiler. The C/C++ compilers also support inline assembly code, and the SSE3 instructions are fully supported in inline assembly. Both the C/C++ and FORTRAN compilers support OpenMP. The programmer can insert OpenMP directives into the source programs to allow their applications to be threaded, thus taking advantage of the improvements to Intel Pentium 4 processor Hyper-Threading (HT) Technology [10].

SSE3 Intrinsics

The C/C++ compiler provides a set of intrinsic functions that the compiler maps directly into SSE3. Intrinsics allow the programmer to write low-level code using SSE, SSE2, and SSE3 without having to worry about issues such as register allocation or instruction scheduling, for which compilers are well suited. This allows programmers to concentrate on mapping their algorithms efficiently to these instructions, and it allows the compiler to fully optimize these instructions. Table 1 lists the intrinsics supported for the SSE3 and the instruction that the intrinsic maps to.

Table 1: SSE3 intrinsic to instruction mapping

Intrinsic Name	Instruction Generated
_mm_addsub_ps	addsubps
_mm_hadd_ps	haddps
_mm_hsub_ps	hsubps
_mm_moveldup_ps	movsldup
_mm_movehdup_ps	movshdup
_mm_addsub_pd	addsubpd
_mm_hadd_pd	haddpd
_mm_hsub_pd	hsubpd
_mm_loaddup_pd	movddup xmm, m64
_mm_movedup_pd	movddup reg, reg
_mm_lddqu_si128	lddqu

Inline Assembly Using SSE3

The Intel C/C++ compiler also supports inline assembly code in C/C++ source. All of the SSE3 instructions are supported in inline assembly. Examples of the inline assembly supported for each of the new instructions are shown below.

```
__asm {
    addsubpd  xmm0, xmm1
    addsubpd  xmm3, XMMWORD PTR mem
    addsubps  xmm5, xmm1
    addsubps  xmm4, XMMWORD PTR mem
    haddpd    xmm2, xmm7
    haddpd    xmm3, XMMWORD PTR mem
    haddps    xmm5, xmm6
    haddps    xmm0, XMMWORD PTR mem
    hsubpd    xmm2, xmm7
    hsubpd    xmm3, XMMWORD PTR mem
    hsubps    xmm5, xmm6
    hsubps    xmm0, XMMWORD PTR mem
    lddqu     xmm2, XMMWORD PTR mem
    movddup   xmm0, xmm1
    movddup   xmm2, QWORD PTR   mem
    movshdup  xmm1, xmm0
    movshdup  xmm3, XMMWORD PTR mem
    movsldup  xmm1, xmm0
    movsldup  xmm3, XMMWORD PTR mem
}
```

OpenMP-Based Multi-Threading

Due to the simplicity of OpenMP model, it has become the dominant high-level programming model to exploit the Thread-Level Parallelism (TLP) in various applications for shared-memory multi-threaded architectures. The Intel 8.0 C++/Fortran95 compilers support OpenMP [7,8] directive-guided parallelization [9], which significantly increases the domain of applications amenable to thread-level parallelism. An application example, shown below, uses OpenMP parallel sections to exploit the TLP of Audio-Visual Speech Recognition (AVSR) through functional decomposition.

```
...
#pragma omp parallel sections default(shared)
{
  #pragma omp section
    { DispatchThreadFunc( &AVSRThData; } // data input and dispatch
  #pragma omp section
    { AudioThreadFunc( &AudioThData ); }        // process audio data
  #pragma omp section
    { VideoThreadFunc( &VideoThData );  }       // process video data
  #pragma omp section
      {  AVSRThreadFunc( &AVSRThData ); } // perform avsr
}
...
```

From the AVSR code sample shown above, we can see the *omp section*-1 invokes the call for data input and dispatching, the *omp section*-2 invokes the call to process the audio data, the *omp section*-3 invokes a call to process video data, and the *omp section*-4 invokes a call to do AVSR, so the performance gain is obtained by mapping four *sections* onto different logical processors to fully utilize processor resources based on HT Technology. In the next two sub-sections, we present optimizations used by Intel 8.0 compilers to tune performance for HT Technology.

Preloading with Aggressive Code Motion

When two threads are sharing data, it is important to avoid false sharing. An example of false sharing occurs when a private and a shared variable are located within the cache line size boundary (64 bytes) or sector boundary (128 bytes). When one thread writes the shared variable, the "dirty" cache line must be written back to memory and updated for each processor sharing the bus. Subsequently, the data are fetched into each processor 128 bytes at a time, causing previously cached data to be evicted from the cache on each processor. False sharing incurs a performance penalty when two threads run on different physical processors, due to cache evictions required to maintain cache coherency, or on logical processors in the same physical processor package, due to memory order machine clear conditions. In this section, we present an optimization-aggressive code motion that preloads all read-only shared memory references into register temps from inside of a region/loop/section to outside of a region/ loop/section, if a memory reference is proven to be a read-only memory reference based on the compiler's load-store analysis and memory disambiguation. The preloading code is moved right after the T-entry. For example, in Figure 3, the memory references of the dope-vector base-address, array lower bound, and stride are lifted to the outside of the loop and pre-loaded into a register temp *t0*, *t1*, and *t2*. Additionally, the memory references of *dv_ptr*-

>*baseaddr*, *dv_ptr->lower*, and dv_*ptr->stride* are replaced by register temp *t0, t1*, and *t2*, respectively.

The benefit of this optimization is that it reduces the data false sharing and avoids the performance penalty. This reduces the overhead of memory de-referencing, since the value is preloaded into a temporary register for the frequent read operations. The second benefit is that it enables advanced optimizations such as vectorization, if the memory de-references in array subscript expressions are lifted outside the loop. For example, in Figure 3, the address computation of array involves the memory de-references of the member *lower* and *extent* of the dope-vector, the compiler lifts the memory de-references of *lower* and *extent* outside the *m*-loop, because the compiler is able to prove that all references to members of the array's dope-vector are *read-only* within the parallel *do* loop. In general, this aggressive code motion enables a number of high-level optimizations such as loop unroll-and-jam, loop tiling, and loop distribution as well. This has resulted in good performance improvements in real applications.

```
real allocatable:: w(:,:)
… …
!$omp parallel do shared(x), private(m,n)
do m=1, 1600            !! Front-End creates a dope-vector for allocatable
  do n=1, 1600          !! array w
    w(m, n) = …    ➔   dv_baseaddr[m][ n] = …
  end do
end do
… …
T-entry(dv_ptr …)  !! Threaded region after multithreaded code generation
  … …
  t0 = (P32 *)dv_ptr->baseaddr   // dv_ptr is a pointer that points
  t1 = (P32 *)dv_ptr->lower      // to the dope-vector of array w
  t2 = (P32 *)dv_ptr->extent
  … …
  do prv_m=lower, upper
    do prv_n=1, 1600             //  EXPR_lower(w(prv_m, prv_n)) = t1
      t3[prv_m][prv_ n] = …      //  EXPR_stride(w(prv_m, prv_n)) = t2
    end do
  end do
T-return
… …
```

Figure 3: An example of code motion

Auto-Dispatching Fast Lock Routines

Efficient reduction of bus traffic and resource contention can significantly impact the overall performance of threaded applications on systems using the Intel Pentium 4 processor with HT Technology. One of the optimizations implemented in the 8.0 compilers reduces memory and bus contention by dispatching to fast lock routines in the Intel OpenMP runtime library based upon an auto-cpu dispatching mechanism. This allows generated code to query the thread *id* only once for each thread and re-use the thread *id* for invoking fast lock routines instead of the generic version of OpenMP lock routines in the source. The Intel compiler generates code that dynamically determines which processor the code is

running on and chooses which version of the function will be executed accordingly. This runtime determination allows the library to take advantage of architecture-specific tuning without sacrificing flexibility by allowing execution of the same binary on older Intel IA32 processors that do not support some of the newer instructions. See the sample code in Figure 4, the *omp_set_lock* and *omp_unset_lock* routines are called 200,000 times inside a parallel loop.

```
… …
#pragma omp parallel for shared(x)
for (i=1, i<200000; i++) {
  omp_set_lock(&lock0) ;
     x = x * foo(&x, i)  + …
  omp_unset_lock(&lock0) ;
}


… …
tid = __kmpc_get_global_thread_num()      ; get thread id outside the loop
… …
$B1$25:
     mov     eax, DWORD PTR ___intel_cpu_indicator
     cmp     eax, 1                          ; CPU dispatch
     je      $B1$27
$B1$26:
     mov     eax, DWORD PTR [ebp+16]
     push    DWORD PTR [eax]        ; passing lock0
     push    ebx                    ; passing thread id
     push    OFFSET FLAT: _kmpc_loc   ; location info
     call    ___kmpc_set_lock        ; dispatching fast lock routine
$B1$49:
     add     esp, 12
     jmp     $B1$28
     ALIGN   4
$B1$27:
     mov     eax, DWORD PTR [ebp+16]
     push    DWORD PTR [eax]        ; passing lock0
     call    _omp_set_lock          ; call  generatic lock routine
$B1$50:
     pop     ecx
$B1$28:
     ... ...
```

Figure 4: Example of dispatching fast lock routines

Originally, the query of thread *id* was done inside the lock routine. Each lock routine accesses a shared data structure that is maintained by the runtime library, which can cause heavy bus traffic and resource contention due to frequent access to the shared data structure. With the fast version of lock routines, we get the thread *id* outside the loop and re-use it for invoking the fast-version of lock routines (e.g., *__kmpc_set_lock/__kmpc_unset_lock*) at runtime based on Auto-CPU-dispatch, which reduces the memory access contention and bus traffic significantly for all lock routines, while minimizing the overhead of those frequent queries of thread *id*. This is important to get performance gain on Intel hyper-threaded processors. In summary, the Intel 8.0 compiler generates code to call two versions of the lock functions. A generic version of the lock function is invoked that will run on any x86 processor. Another fast version would be tuned for the Intel Pentium 4 processor family enabled with HT Technology.
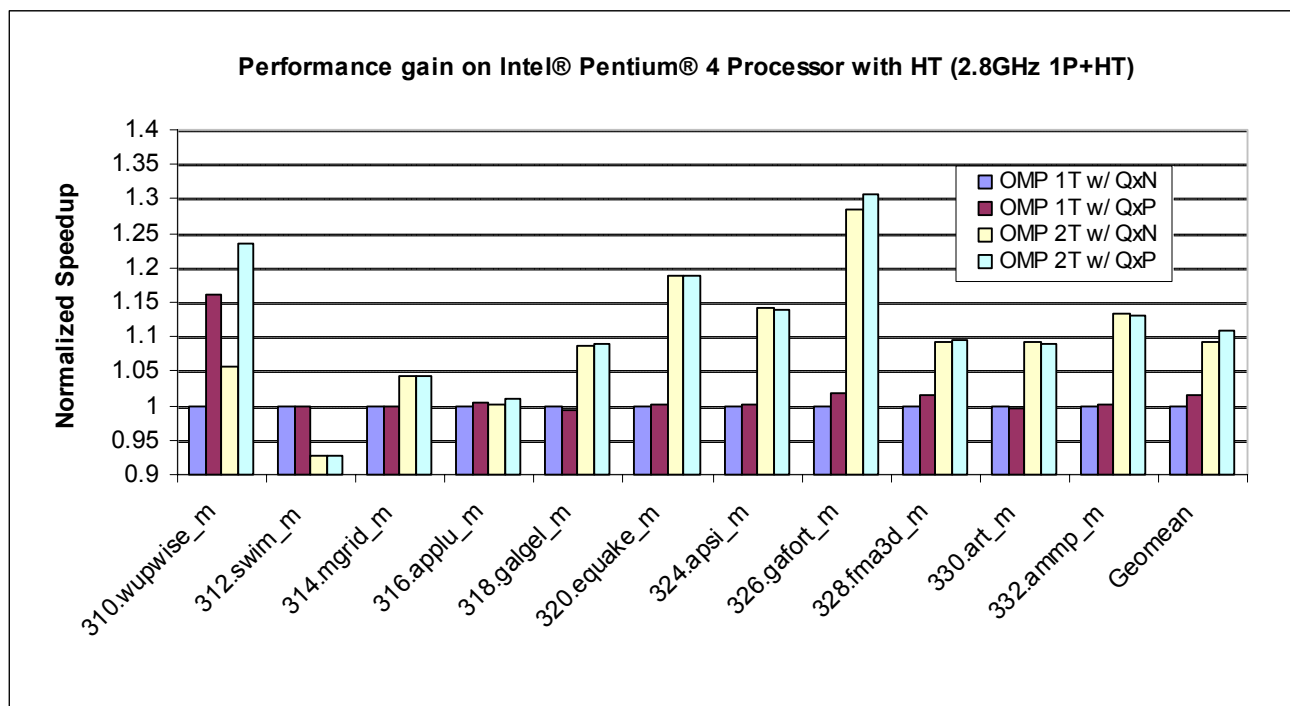
Figure 5: Speedup of SPEC* OMPM2001 estimates (based on measurements on Intel internal platforms)

SPEC* OMPM2001 Performance Results

The performance study of SPEC OMPM2001 benchmarks is conducted on a pre-production single physical processor system built with the latest Intel Pentium 4 processor with HT Technology, running at 2.8GHz, with 2GB memory, an 8K L1-Cache, and a 512K L2-Cache. The SPEC OMPM2001 is a benchmark suite that consists of 11 scientific applications.

Those SPEC OMPM2001 benchmarks target small- and medium-scale SMP multiprocessor systems, and the memory footprint reaches 2GB for several very large application programs. For our performance measurement, all SPEC OMPM2001 benchmarks are compiled by the Intel 8.0 C++/Fortran compilers with two sets of base options: -Qopenmp –Qipo –O3 –QxN (OpenMP w/ QxN) and –Qopenmp –Qipo –O3 –QxP (OpenMP w/ QxP). The normalized performance speed-up of the SPEC OMPM2001 benchmarks is shown in Figure 5. This demonstrates the performance gain attributed to Intel Hyper-Threading Technology and Intel 8.0 C++/Fortran compiler support by exploiting thread-level parallelism.

The performance scaling is derived from the baseline performance of single thread binary with OMP 1T w/ QxN, single thread execution under OMP 1T w/ QxP, and two threads execution under OMP 2T w/ QxN and OMP 2T w/ QxP, respectively.

As we can see, the multi-threaded code generated by the Intel compiler on a Intel Pentium 4 processor 1-CPU system enabled with Hyper-Threading Technology achieved a performance improvement of 4.2% to 28.6% (OMP 2T w/ QxN) on 9 out of 11 benchmarks except 316.applu_m (0.0%) and 312.swim_m (-7.3%). The 312.swim_m slowdown under the two-thread execution mode was well known: it is due to the fact that the 312.swim_m is a memory bandwidth bounded application. Overall, the geometric mean improvement with OMP 2T w/ QxN is 9.2% by running the second thread on the second logical processor. Furthermore, the geometric mean improvement with OMP 1T w/ QxP is 1.6%, due to the optimizations presented earlier. Under OMP 2T w/ QxP, we achieved a performance gain range from 4.2% to 30.6% for 9 out 11 benchmarks except the benchmarks 312.swim_m (-7.4%) and 316.applu_m (0.0%), and the geometric mean improvement was 10.9%. These performance results demonstrated that the multi-threaded codes generated and optimized by the Intel 8.0 compilers are very efficient when used with the support of the well-tuned Intel OpenMP runtime library.

CONCLUSION

Due to the complexity of modern microprocessors, compiler support has become an important part of obtaining high performance. The Intel 8.0 compilers provide full support for the rich features of the latest Intel Pentium 4 processor with Hyper-Threading Technology. The compilers perform automatic optimization of programs using vectorization and advanced instruction selection techniques that together can yield up to a 2x performance improvement compared to the default level of optimization. Additionally, the Intel compilers provide support for programmer-guided performance improvements by supporting the Steaming-SIMD-Extensions 3 (SSE3) directly using compiler intrinsics and inline-assembly, and by improving OpenMP directive support, enabling performance improvements of up to 30% for software using OpenMP directives. These capabilities allow software developers to use the Intel compilers to optimize the performance of their software for the latest Intel Pentium 4 processor.

More information on the Intel 8.0 compilers can be found at http://www.intel.com/software/products/compilers.

ACKNOWLEDGMENTS

REFERENCES

[1] Vishal Aslot et. al., "SPEComp: "A New Benchmark Suite for Measuring Parallel Computer Performance," in *Proceedings of WOMPAT 2001*, *Workshop on OpenMP Applications and Tools, Lecture Notes in Computer Science*, 2104, pp. 1-10, July 2001.

[2] Aart J.C. Bik, Milind Girkar, Paul M Grey, and Xinmin Tian, "Automatic Intra-Register Vectorization for the Intel Architecture," *International Journal on Parallel Processing*, 2001.

[3] Aart J.C. Bik, *The Software Vectorization Handbook: Applying Intel® Multimedia Extensions for Maximum Performance*, Intel Press, April 2004, http://www.intel.com/intelpress/sum_vmmx.htm.

[4] D. Callahan, K.D. Cooper, K. Kennedy, and L.M. Torczon, "Interprocedural Constant Propagation," in *Proceedings of the SIGPLAN Symposium on Compiler Construction*, New York, 1986.

[5] International Organization for Standardization, ISO/IEC 9899:1999. The standard can be obtained at http://www.iso.org/.

[6] Samuel Larsen, Emmett Witchel, and Saman Amarasinghe, "Increasing and detecting memory address congruence," in *Proceedings of the 11th International Conference on Parallel Architectures and Compilation Techniques*, 2002.

[7] OpenMP Architecture Review Board, "OpenMP C and C++ Application Program Interface," Version 1.0, October 1998, http://www.openmp.org.

[8] OpenMP Architecture Review Board, "OpenMP Fortran Application Program Interface," Version 2.0, November 2000, http://www.openmp.org.

[9] Standard Performance Evaluation Corporation. SPEC CPU2000, http://www.spec.org/.

[10] Xinmin Tian, Aart J.C. Bik, Milind Girkar, Paul M. Grey, Hideki Saito, and Ernesto Su, "Intel OpenMP C++/Fortran Compiler for Hyper-Threading Technology: Implementation and Performance," *Intel Technology Journal*, Vol. 6, Q1 2002.

[11] Intel® C++ Compiler for Windows Systems User Guide, http://www.intel.com/software/products/compilers/cwin/docs/ccug.htm

[12] Intel® C++ Compiler for Linux Systems User Guide, http://www.intel.com/software/products/compilers/clin/docs/ug/index.htm

[13] Intel Corporation. IA32 Intel Architecture Software Developer's Manual, Volume 1: Basic Architecture, 2003. Manual available at http://developer.intel.com/.

AUTHORS' BIOGRAPHIES

Kevin B. Smith received his B.Sc. degree in Computer Science from Iowa State University in 1981. From 1981 to 1990 he worked at Tektronix, Inc on C and Pascal compilers and debuggers targeting 8086, Z8000, 68000, 68030, PDP-11, and VAX microprocessors. He joined Intel Corporation in 1990 working on compilers for the Intel i960® microprocessor, and in 1996 began working on compilers for the Intel Pentium® II, Pentium® III, and Pentium 4 processors. He is currently working in the Intel Compiler Lab as team leader for the IA32 code generator. His e-mail is kevin.b.smith at intel.com

---

® i960 and Pentium are registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Aart J.C. Bik received his M.Sc. degree in Computer Science from Utrecht University, The Netherlands, in 1992, and his Ph.D. degree from Leiden University, The Netherlands, in 1996. In 1997, he was a post-doctoral researcher at Indiana University, Bloomington, Indiana, where he conducted research in high-performance compilers for Java[*]. In 1998, he joined Intel Corporation where he is currently working on automatically exploiting multimedia extensions in the parallelization and vectorization group. His e-mail is aart.bik at intel.com

Xinmin Tian is currently working in the parallelization and vectorization group at Intel Corporation, where he works on compiler code generation and optimization for exploiting thread-level parallelism. He leads the OpenMP parallelization team. He holds B.Sc., M.Sc., and Ph.D. degrees in Computer Science from Tsinghua University. He was a postdoctoral researcher in the School of Computer Science at McGill University, Montreal. Before joining Intel Corporation he worked on a parallelizing compiler, code generation, and performance optimization at IBM. His e-mail is xinmin.tian at intel.com

---

[*]Other brands and names are the property of their respective owners.

THIS PAGE INTENTIONALLY LEFT BLANK

Performance Analysis and Validation of the Intel® Pentium® 4 Processor on 90nm Technology

Ronak Singhal, Desktop Platforms Group, Intel Corporation
K. S. Venkatraman, Desktop Platforms Group, Intel Corporation.
Evan R. Cohn, Technology and Manufacturing Group, Intel Corporation
John G. Holm, Desktop Platforms Group, Intel Corporation
David A. Koufaty, Desktop Platforms Group, Intel Corporation
Meng-Jang Lin, Desktop Platforms Group, Intel Corporation
Mahesh J. Madhav, Desktop Platforms Group, Intel Corporation
Markus Mattwandel, Desktop Platforms Group, Intel Corporation
Nidhi Nidhi, Desktop Platforms Group, Intel Corporation
Jonathan D. Pearce, Technology and Manufacturing Group, Intel Corporation
Madhusudanan Seshadri, Desktop Platforms Group, Intel Corporation

Index words: performance, validation, RTL, tracing, analysis, benchmarking, Hyper-Threading Technology

ABSTRACT

In addition to the considerable effort spent on functional validation of Intel® processors, a separate parallel activity is conducted to verify that processor performance meets or exceeds specifications. In this paper, we discuss both the pre-silicon and post-silicon performance validation processes carried out on the 90nm version of the Intel® Pentium® 4 processor.

For pre-silicon performance work, we describe how a detailed performance simulator is used to ensure that the processor specification meets the product's performance targets and also that the implementation matches the defined specification. Additionally, we describe how we project the performance of the Pentium 4 processor on key applications and benchmarks.

Once silicon arrives, the second phase of performance verification work starts. We describe the process for detecting post-silicon performance issues, developing associated optimizations, and communicating these to

application engineers, compiler teams, and microprocessor architects for appropriate action. We also discuss the tools used to gather performance metrics and characterize application performance.

INTRODUCTION

Performance validation is a crucial counterpart to functional validation. Delivering a functional processor is not sufficient in today's competitive marketplace; we must also deliver compelling performance to the end user. Performance analysis and validation for modern microprocessor development projects such as the Intel Pentium 4 processor require complex modeling at different levels of abstraction. These levels range from high-level microarchitecture-specific performance simulators to detailed Register-Transfer-Level (RTL) models that perfectly capture the logical behavior of the CPU. A set of microarchitectural features that define and meet the performance goals of the Pentium 4 processor are simulated in all of these models. A close working relationship among the architecture, design, and performance validation teams ensures that performance does not degrade as the design progresses and that appropriate trade-off decisions are made at every stage of the project, even after silicon is delivered.

In subsequent sections of this paper, we describe the various tools used and methodologies followed for

analyzing and validating the performance of the Pentium 4 processor, during both the pre-silicon and post-silicon phases.

PRE-SILICON PERFORMANCE VALIDATION

Pre-silicon performance work involves two broad areas: feature specification and feature validation. During the initial stages of the project, performance work is focused on determining the set of features needed to reach target performance levels. The second stage of pre-silicon performance work involves ensuring that the implementation achieves the desired level of performance and that performance does not degrade during the design convergence effort. Specific validation must be done for performance since functional validation will not catch many performance issues. For instance, consider a case where the branch predictor of a processor is always incorrect, resulting in every branch being mispredicted. If the branch recovery logic was implemented correctly, no functional failure would ever occur but performance could be severely impacted. This is an extreme case but it typifies the scenario that a performance validation team tries to address.

Tracing and Workload Collection

In order to start pre-silicon performance validation, it is necessary to have inputs to the simulator that reflect the most important applications and benchmarks for the processor. It is also important that these can be run on the various simulators.

First, consider what makes up a "trace" that can be run on the performance simulator. These are not traces in the classical sense, where a trace simply defines the control flow of a program and provides memory access information. Our "traces" consist of a preliminary architectural state and the memory image of the corresponding process. Our performance simulators mostly use hardware Long Instruction Traces (LIT) of real-world applications. These traces are a tuple of processor architectural state, system memory, and "LIT injections" that are external events like Direct Memory Access (DMA) and interrupts. They are needed to ensure that the simulator follows the exact execution path that the application took on the machine where the trace was collected.

The traces collected are carefully chosen to provide wide coverage of applications and benchmarks that cover all market segments (server, mobile, desktop). Each application is typically represented by 25 traces. A trace captures about 30 million instructions. The traces are regularly updated as new applications or newer

versions of existing applications appear. This ensures that the microarchitecture development constantly tracks the changing needs of the marketplace. Table 1 lists the different categories covered by our traces and examples of applications in each category. The majority of our traces are collected using special trace capture hardware that employs a logic analyzer interface (LAI) inserted between the processor core and the main memory subsystem. The main benefit of hardware trace collection is the ability to record and replay events external to the processor that affect overall performance.

Table 1: Trace categories and sample applications

Category	Sample Applications
SPECint* 2000	164.gzip, 181.mcf
SPECfp* 2000	179.art, 200.sixtrack
Kernels	Fourier transforms, saxpy, daxpy
Multi-media	Unreal* Tournament, 3dMark*
Productivity	Sysmark*, Business Winstone*
Workstation	Autocad*, Ansys*, Nastran*
Internet	SPECjbb*, Netscape*, WebMark*
Server	TPC-C*, TPC-W*
Threaded	Adobe Photoshop*, 3dStudio Max*

In addition to the traditional hardware trace collection methodology, a tool called UserLIT has been developed that allows for trace collection from an executable by trapping system calls without any specialized hardware. (The name UserLIT refers to a program running in user space and capable of collecting LITs.)

A UserLIT trace must also include a minimal "operating system" that sets up descriptor tables, page tables, and system call handlers. The instruction stream, of course, is stored in the memory image. The simulator does not execute system calls since their behavior is non-deterministic and reproducibility is a requirement for simulation. Thus, any system calls encountered during trace collection will need to be handled to preserve their effect on the memory image and keep the execution of the program on the correct path. We accomplish this by keeping a system call log that is replayed during simulation.

---

* Other brands and names are the property of their respective owners.

The Linux[*] operating system allows users to intercept calls into the kernel using the *ptrace*() system call. This utility allows the ptracing parent process to monitor, pause execution, and peek/poke the state of a running child process. Many Linux users are already familiar with a tool called *strace*. Strace is an example of how ptrace can be used to extract information that crosses the user-kernel boundary. [2]

The UserLIT invocation sets up the ptrace traps, forks off the child process being traced, and waits for ptrace to snare a system call. Upon encountering a system call, the child process pauses execution and allows UserLIT to record relevant information about the system call, including arguments and a return value. All this information is stored in the system call log. When system calls are encountered during simulation, the execution goes to a special system call handler that verifies the arguments and writes the output in the proper location. Replaying of the log not only ensures accurate simulation, but also acts as a sanity check; for example, if a call is missed, the simulation breaks out with an error.

The other information UserLIT grabs when creating traces is the memory image. The Linux operating system gives privileged users the ability to peek at process memory. This information is copied and used as the basis of our trace.

Yet another method of generating an input for the simulator is another software-based tracing tool called SoftLIT. With this methodology, traces are generated by running the workload on top of SoftSDV [3], a processor and platform emulator used to enable prototyping of system software. The memory dumps and trace collection can be done at a lower level than with UserLIT, allowing traces of privileged (Ring 0) code to be collected. SoftLIT also has full knowledge of all system memory I/O and can create LIT files containing this I/O for realistic playback during simulation. While UserLIT is the choice for quick turnaround application tracing, SoftSDV is used to collect system-level routines such as operating system boot and interprocess communication.

---

[*] Other names and brands are the property of their respective owners.

Performance Simulator Development

For all generations of the Intel NetBurst[®] microarchitecture, a highly detailed performance simulator was developed to be used as the primary vehicle of pre-silicon performance analysis. This model is used to both drive the definition of the microarchitecture, as well as to serve as a specification for performance validation against an RTL model. The simulator uses a performance model-driven functional execution paradigm, meaning that the performance model of the simulator drives the control flow of the simulation. As a result, it is imperative that the performance simulator also produce functionally correct answers.

To ensure functional correctness, the architectural state of the processor is checked against a "golden" functional model throughout the simulation. The benefit of forcing the performance model to be functionally correct is that it provides some level of confidence in the features implemented in the simulator. Without the need for functional correctness, it is possible to overstate the performance impact of a microarchitectural feature. For instance, consider a simulator where a bug is introduced such that dirty data in the caches are never evicted but are merely overwritten. Such a bug would only be caught by an analysis of results but could greatly skew performance results until the bug is found. With our performance simulator, a functional failure would occur indicating a problem with the performance model and invalidating any conclusions that could be generated from these flawed simulations. Obviously, this serves as only one means of validation and it is still necessary to conduct other steps to ensure the accuracy of the simulator.

For the Intel Pentium 4 processor, the performance simulator used for the prior generations of the processor was the starting point. On top of the existing model, modifications were made to reflect changes in the pipeline, feature set, and microcode to emulate the new processor under development. Given this model, comparisons of the performance of this processor to its previous implementations were generated and used to drive a performance analysis effort. The simulator is also capable of simulating threaded workloads in both single processor and multi-processor configurations. This is crucial in understanding and analyzing the performance of Hyper-Threading Technology performance, which is a key feature of this processor.

[®] NetBurst is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

RTL Correlation

Once the feature set of the processor is determined through studies conducted in the performance simulator, the next step in performance verification is to ensure that the implementation of these features delivers the expected performance. The actual implementation is represented in the RTL model that is a gate-level accurate representation of the processor. A set of tests is run on both the performance simulator and the RTL model and the number of clock cycles needed to complete the simulation on each model are compared. If there is a large miscorrelation, it is investigated with the analysis likely pointing to either a modeling inaccuracy in the performance simulator or a bug in the RTL.

An RTL simulation takes around 500 times longer than a run of the same trace through the performance simulator. Due to its slow simulation speed, it is practical to run only very short tests through the RTL. Since this length is clearly not long enough to warm up all of the structures on the processor, state is injected at the beginning of the simulation into the largest structures, such as the caches, Translation Lookaside Buffers (TLBs), and branch predictors.

Every week thousands of tests are created and run on RTL and the simulator, and if miscorrelations are uncovered, the simulation results are saved for further investigation. The tests run are subsets of the longer traces used for performance studies. This process is called the Billion Cycle Search (BCS). It is aptly named, since each month around one billion cycles of RTL are run through our distributed computing environment.

Once a miscorrelation between the performance simulator and the RTL is detected, the offending test is investigated to understand the source of the discrepancy. It is often the case that the performance simulator did not model certain boundary conditions correctly. In these instances, the performance model is updated to reflect the true behavior of the processor. The opposite case, where the simulator is operating correctly and the RTL is not, is more interesting. This can occur because new features were not correctly coded in the RTL. In cases such as this, a performance bug is filed against the RTL. The resolution of the bug follows the same process as any functional bug. On the Intel Pentium 4 processor, performance bugs mostly came in three basic forms:

1) *New features were not implemented correctly.* For instance, several new predictors were added or enhanced on this processor, but they sometimes required multiple iterations of design changes in order to get them to match performance expectations.

2) *An existing protocol was broken.* This can happen if logic is partitioned across pipeline stages differently from how it was partitioned on previous processor implementations.

3) *An optimization is discovered.* There are cases where a better algorithm or implementation method is found through performance analysis. An example of this is microcode implementation where opportunities for performance enhancement often exist at little to no hardware cost.

As a result of the correlation efforts, the simulator correlates on average within 3% of the RTL on the tests that are run as part of BCS. By creating a simulator to this degree of accuracy, we have high confidence in using it to predict which features make sense to implement in the processor, as well as having a tool that is useful for predicting and analyzing the performance of software applications and benchmarks.

PERFORMANCE PROJECTION METHODOLOGY

A "projection" process is used to accurately estimate and track performance during the lifetime of the project on the highest profile benchmarks, such as SPEC* CPU2000 [1]. The projection methodology is different from a routine performance study in that it utilizes all the traces available for a given application and tries to correct for simulator miscorrelation with actual silicon. A good projection methodology should provide accurate estimates (>95%) with quick turnaround times while remaining easy to use. As a result of these goals, the methodology itself is under continuous development.

The Intel Pentium 4 processor SPEC CPU2000 projection process started with the evaluation of benchmark performance on several existing baseline hardware configurations. The configurations were chosen to provide variation in microarchitecture, cache size, core frequency, and main memory performance. These variables are critical to expose simulator modeling inaccuracies and trace inadequacies. Specifically, Pentium 4 processors with different cache sizes running at different frequencies were used with different memory technologies (RDRAM, DDR memory, etc.)

For the simulations, 550 traces covering the 26 applications of the SPEC CPU2000 benchmark suite were used. The traces were fed through the simulator for each of the baseline configurations and the target

---

* Other names and brands are the property of their respective owners.

configuration. The simulator starts collecting performance metrics after an initial warm-up phase of 500,000 instructions. The simulated application execution time is calculated by simply summing up the reported execution times for all the traces, which is no different from assuming an equal weighting for every trace. We are investigating this area for future enhancements to the projection methodology.

A linear regression equation using the method of least squares is then derived using the simulated and measured silicon times for the baseline configurations: it is done on a per-application basis. This equation is applied to the simulated time of the target configuration to directly calculate the silicon execution time of that application. The Root Mean Square (RMS) value of the regression equation is indicative of the projection error. This process is repeated for all the 26 applications of the SPEC CPU2000 suite. The estimated times are converted to SPEC CPU2000 scores in the standard manner.

Using this methodology, the estimated SPECint*_base and SPECfp*_base geometric mean scores were within 2% of actual silicon measurements for the Intel Pentium 4 processor.

POST-SILICON PERFORMANCE VALIDATION

Between completion of the pre-silicon design and arrival of first silicon, the post-silicon performance team consisting of various groups (benchmarking, architecture, compiler, software, and platform teams) develop plans for ensuring that the processor provides the expected level of performance when placed in a system environment. We outline the various activities involved with post-silicon performance validation at different stages, starting with first silicon; we then move onto performance characterization, detection and resolution of performance anomalies along with optimization of specific processor features.

First Boot and Bring-up Activities

Approximately a month before silicon arrives, target systems that will accommodate the new processor are built and tested. These also serve as reference systems for performance comparisons with previous versions of the Intel Pentium 4 processor family, enabling baseline performance data to be generated. The systems are configured with high-end peripherals so as to not cause a performance bottleneck. The initial set of benchmarks

---

*Other names and brands are the property of their respective owners.

are chosen across several operating systems to establish a representative sample of benchmarking classes, including micro-benchmarks, games, and important desktop applications. At least one long-running performance test is also included to accommodate overnight stress testing in addition to measuring processor performance.

When processor silicon arrives, pre-built hard drive images are used to establish initial processor health. Pre-installed operating systems are booted at this time. Once an operating system boots successfully, basic functionality is established by means of short stress tests. If the system passes these tests, the first benchmarks are run, starting with "canary" tests that stress key system components, such as graphics, memory, and raw processor speed. Front-side bus, core frequency, and cache size scaling are also measured at this time. This gives an indication of preliminary performance health, as well as how measured performance correlates back to performance projections.

Performance Parameter Characterization

Once the preliminary performance analysis is completed, basic microarchitecture characteristics are measured and compared to expectations and to previous processor family implementations. This activity is carried out in parallel with the collection of application performance results.

Measurements are taken for key latencies, such as store-to-load forwarding, lock execution, cache access, SSE3 instructions, system calls, mispredicts, and microcode assists. Throughput data for certain key operations, such as strings and fast integer operations, are also measured. These measurements are primarily taken through micro-benchmarks that are developed internally. For certain measurements, benchmarks or special tests are used instead of targeted micro-benchmarks. For example, cache latencies are measured by running a standard latency test such as *LMbench* [4]. Programs that perform *memcpy* are used to understand string performance. For the new SSE3 instructions, a program that implements a motion estimation algorithm is used to confirm that the algorithm shows the expected performance benefit.

In addition to running tests, we produced an instruction latency document to aid the characterization work and analysis of performance sightings. This document gives the static execution latency for most of the Pentium 4 family of processors. For each instruction, the latency is determined by obtaining a list of micro-ops that comprise the instruction, finding the critical path to the destination for this instruction, and calculating the number of cycles it takes to execute the serially

dependent sequence of micro-ops that are on the critical path.

Life Cycle of a Performance Sighting

Performance issues, anomalies, and unexpected behaviors are reported, or "sighted," to the Performance Debug Team. Issues internal to Intel are reported directly to this team, while issues elevated by groups external to Intel are communicated via Technical Marketing groups. Communication between the Performance Debug Team and the sighting reporting groups is bi-directional in order to guarantee complete closure for all parties associated with a performance issue.

Once a performance issue has been sighted, it is entered into the Performance Sightings Database, and its progress is tracked by the Performance Debug Team. The team meets on a regular basis to coordinate performance debug activities and share results with the goal of achieving optimal progress on all open performance sightings. Often other experts within the company, ranging from processor and chipset designers to operating systems, compilers and application software experts, are consulted to provide feedback. Sightings that require focused attention or expediency result in the formation of a specialized task force. Such task forces typically meet on a daily basis in order to fully understand and provide a solution to performance issues.

The Performance Sightings Database is continually updated by both the Performance Debug Team and performance task forces with results and conclusions. Multiple teams have access to these data so that known issues can be correlated quickly to new sightings. An illustration of this process is shown in Figure 1.

Both the Performance Debug Team and the task forces periodically report a summary of their findings to senior management. Additionally, if a change is required within the processor to resolve the issue, it is driven through the Engineering Change Order (ECO) process for formal approval. This guarantees that any performance-related design changes are propagated consistently to all current and future Intel designs.
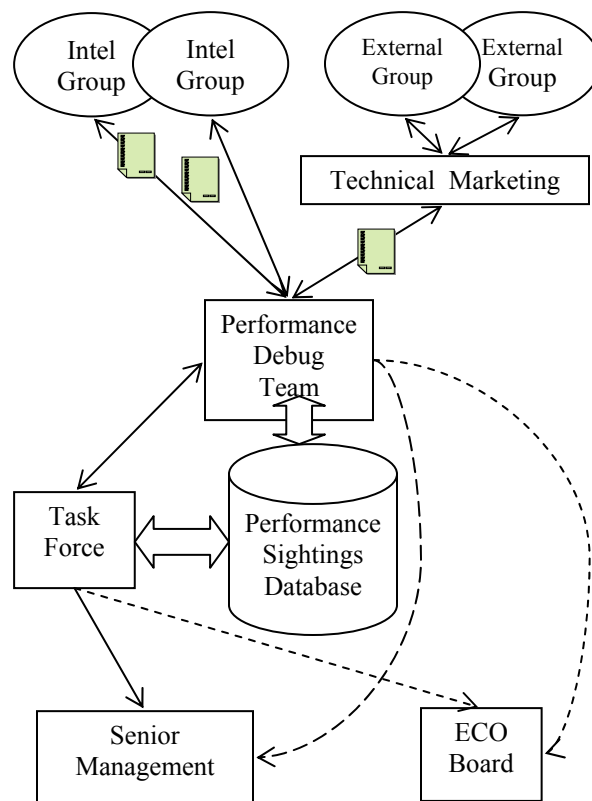
Figure 1: Performance sighting life cycle

Interaction with Other Teams

As soon as silicon is functionally healthy and performance measurements begin, many questions are raised that frequently require the engagement of other teams. The main interactions are with the chipset, benchmarking, and compiler teams, but also with the software teams, and often in that order. A typical benchmarking sequence proceeds as follows:

- Measuring the latency and throughput of instructions on the integer and FP side.

- Measuring cache latencies at all levels.

- Targeted measurements of other features such as the hardware prefetcher.

- Memory bandwidth and platform tests.

- Off-the-shelf benchmarks for which the source code is not available (such as Sysmark 2002).

- Compiled benchmarks (such as SPEC CPU2000).

While these performance-testing activities often start simultaneously, they typically finish in the order listed above due to the complexity and runtime of the tests. The testing activities also have the property of first

targeting the core, then the platform, and finally the software.

The first programs run for performance are typically small kernels or micro-benchmarks that are internally generated or benchmarks written externally that are targeted towards specific processor features. Such targeted tests usually lead to the first performance sightings on the processor. Once the tests that target the CPU core have run, memory and platform tests are completed. These memory and platform tests typically have an expected result. For example, two channels of DDR400 should have a peak bandwidth of 6.4GB/s when the front-side bus is running at 200MHz on a streaming memory test. If memory bandwidth tests were to show lower-than-expected performance, a performance sighting is filed and work would begin to resolve it. If the test has already proven itself on previous processors and platforms, the performance team will start looking at possible problems within the CPU or chipset.

After running targeted tests, higher-level benchmarks are run. These include both compiled benchmarks and benchmarks that come in binary-only form. The results are compared against projections and real results from previous systems. If performance is different from what is expected, a performance sighting is filed and diagnosis begins. Initial data are usually obtained from the processor performance counters to narrow down the problem type. Higher-level performance tools, such as the Intel VTune™ Performance Analyzer [5], are then used to find regions of code in the benchmark that may be responsible for the performance delta. If it is determined that the software can be changed to alleviate the problem, the appropriate software teams are engaged to look at possible fixes. In the case of compiled code, the compiler team is involved to produce more optimized code generation.

Since compiler performance affects all applications, the performance team periodically meets with the compiler team. Experts from the architecture and compiler team work side-by-side looking at generated code, taking into account the subtleties of the microarchitecture and finding ways to improve the code.

Once the compiler is optimized, benchmark performance has been analyzed, and all the outstanding issues have been closed, the performance team transitions from proactive mode to reactive mode. Application engineers, Original Equipment Manufacturers, and independent software vendors will

---

™ VTune is a trademark of Intel Corporation or its subsidiaries in the United States and other countries.

then file performance sightings on applications that they run. At this point, the performance team attempts to reproduce the problem and determine solutions using the methods outlined previously.

In addition to the CPU architecture team, the chipset architects, compiler team, and software developers play an important role in overall system performance. The performance validation effort actively engages these teams to diagnose and solve performance issues.

Tools Used for Post-Silicon Performance Analysis

Once a comparison is made between a new processor and a baseline, relative performance anomalies are analyzed using the Intel VTune Performance Analyzer. Time- and/or event-based profiles are taken on both systems and compared to understand processor behavior.

In this section, we discuss the two tools most commonly used for performance debug: the Intel® EMON performance monitoring tool and the Intel VTune Performance Analyzer. These tools are used in two steps mainly involved in performance debug: pinpointing the location of the performance anomaly and identifying why it is occurring.

The EMON Performance Monitoring Tool

The EMON tool allows collection of data from the hardware performance monitoring counters. We used this tool extensively to get a high-level view of workload characteristics. EMON is fed with an extensive list of performance events, including, but not limited to, clock ticks, instructions retired, mispredicted branches, cache misses, and bus traffic. The typical usage model is to collect the events during the whole workload execution by passing the original command line to EMON and allowing it to execute the workload as many times as it needs until all requested events are fulfilled. Due to hardware restrictions, the number of events that can be collected simultaneously is limited. Therefore, the event list is typically hand optimized to maximize the number of events collected on each run of the workload and minimize the total number of runs needed.

The result of the EMON run is imported into a custom spreadsheet. The resulting data can be used to do high-level analysis of a workload. For example, by comparing it to a second configuration with a different processor, it is possible to determine what microarchitectural features might be responsible for the performance difference.

The Intel VTune™ Performance Analyzer

The Intel VTune Performance Analyzer usage within the CPU performance team is typically three-fold: time-based sampling, Pdiff, and event-based sampling.

Time-based sampling is most commonly used to identify the location of performance anomalies during performance debug. Additionally, the user can inspect the source code or disassembly at the identified locations to help understand the issue.

Pdiff refers to the table-based output that the Intel VTune Performance Analyzer can generate when comparing two configurations. Time-based samples are bucketed based on a set of consecutive address bytes as specified by the user. The output consists of a table in a spreadsheet format, where each row is the delta between the number of samples in each configuration for each address range. Since the samples are time based, the delta effectively corresponds to the performance difference observed between the two configurations for a particular address range. This allows performance engineers to focus on the sections of code that contribute the most to the total slowdown.

Event-based sampling is most commonly used in conjunction with the results from EMON. Whenever an event of interest is identified by the EMON tool as a potential reason for a performance delta, event-based sampling provides a powerful way to find the exact location in the code where the event is triggered. Once identified, the engineers can proceed to analyze and investigate further.

Optimal CPU Performance Feature Tuning

On the Intel Pentium 4 processor, a large number of performance features can be tuned in silicon, changing the way they work and the benefit they provide. On Pentium 4 processors, many features are designed with both the ability to be completely disabled and to have their characteristics modified after silicon is available. The ability to disable a feature is crucial for functionality, as it is possible that a new feature introduces a new functional bug, as well as for performance, so that the benefit of the new features can be isolated and checked against expectations. Additionally, for features that are defined by a large set of variables, it is often prohibitive to try and simulate the entire possible search space doing pre-silicon simulations because of the high cost of these simulations. Furthermore, with post-silicon tuning, a large number of workloads can be tested to find optimal settings than can be accomplished via pre-silicon simulations.

A small number of tests was conducted manually by turning a set of features off, including performance features not found on previous implementations of the Pentium 4 processor, to confirm that they actually provided a benefit. Settings for some features such as the aggressiveness of the hardware prefetcher and dynamic thread partitioning algorithms in the uop schedulers have a large search space. For features such as these, an automated approach, driven by a genetic optimization algorithm, was used to find an optimal operation point. To provide the genetic-based approach with enough sample points to make a valid judgment, two or three tests from SPEC CPU2000 were selected that show sensitivity to the features being tuned to act as a proxy for each study.

The performance improvements produced by the automated approach for three out of six features chosen for tuning were negligible or were outweighed by the negative outliers. However, the improvements for the other three features produced worthwhile speedups. The first two features were simply additive in their effects (a +1.4% improvement on estimated SPECfp_rate2000), but adding the third tuned feature on top of that degraded the overall SPECfp_rate2000 speed-up to only 0.7%. In order to limit the search space, some settings that were already at their best possible value were "frozen" and removed from the search space. This approach was successful in achieving a speed-up of +2.7% on estimated SPECfp_rate2000 as shown in Figure 2, which is greater than the sum of the improvements from each of the individual features. The performance improvements represent a speed-up over the initial default settings that the Pentium 4 processor was taped out with.

* SPEC estimates based on measurements on Intel internal platforms
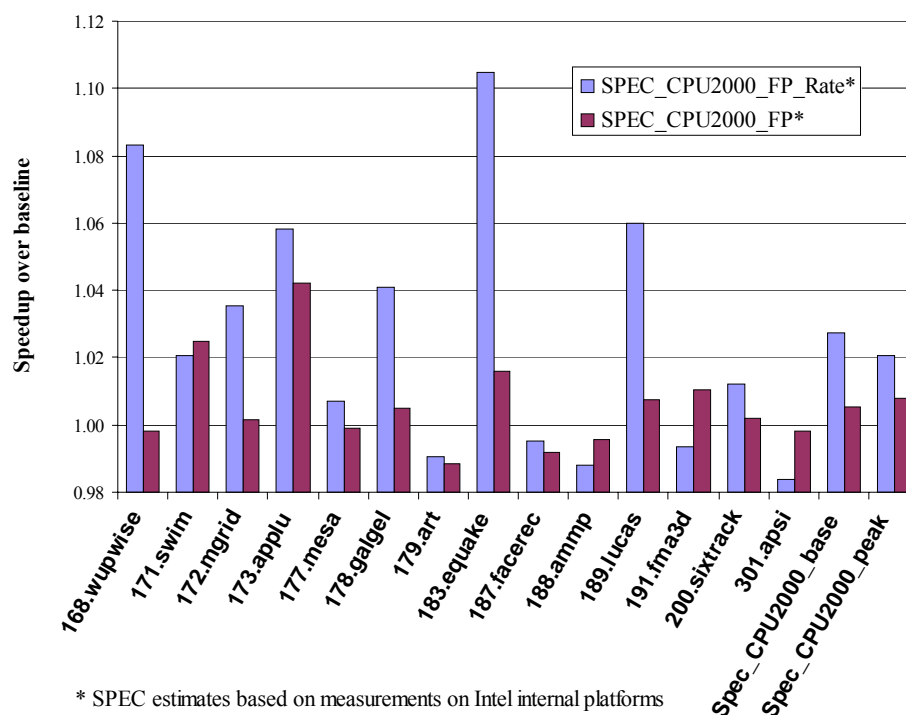
Figure 2: Results obtained via performance feature tuning experiments on SPEC*_CPU2000_FP and SPEC_CPU2000_FP_rate

CONCLUSION

Performance validation has become an integral part of the complex design process. Detailed and regular performance analysis coupled with timely bug fixes enable the Intel Pentium 4 processor to not only meet and exceed its performance targets but also to deliver a high level of performance to the end user on key applications.

ACKNOWLEDGMENTS

REFERENCES

[1] Standard Performance Evaluation Corporation. http://www.spec.org

[2] http://sourceforge.net/projects/strace

[3] Uhlig, R.; Fishtein, R.; Gershon, O.; Hirsh, I.; Wang, H., "SoftSDV: A Pre-Silicon Software Development Environment for the IA-64 Architecture," December 1999. http://www.intel.com/technology/itj/q41999/articles/art_2.htm

[4] http://sourceforge.net/projects/lmbench

[5] Intel VTune™ Performance Analyzer http://www.intel.com/software/products/vtune/index.htm

AUTHORS' BIOGRAPHIES

Ronak Singhal received B.S. and M.S. degrees in Electrical and Computer Engineering from Carnegie Mellon University. He joined Intel in 1997 and has spent the majority of his time focused on microarchitecture performance analysis and validation for the Pentium 4 family of processors. His e-mail address is ronak.singhal at intel.com.

K. S. Venkatraman received his B.S. degree from Birla Institute of Technology and his M.S. degree from Villanova University. He joined Intel in 1997 and has focused on microarchitecture as well as post-silicon performance analysis for the Pentium 4 family of

processors. In his spare time, he enjoys amateur radio and riding his motorcycle. His e-mail address is k.s. venkatraman at intel.com.

Evan R. Cohn received a B.A. degree in Applied Mathematics from Harvard in 1982 and a Ph.D. in Computer Science from Stanford in 1988. He initially joined Intel in the supercomputer division and eventually migrated to the Design Automation group in LTD. His e-mail address is evan.cohn at intel.com.

John G. Holm received a B.S. degree in Computer Engineering from the University of Michigan, Ann Arbor in 1989 after which he attended the University of Illinois, Urbana-Champaign where he received an M.S. degree in 1993 and a Ph.D. degree in 1997. He joined the Itanium® Architecture team in 1997 where he worked on TPC-C analysis and simulator development. He joined the Pentium 4 Architecture team in 2001. His e-mail address is john.g.holm at intel.com.

David A. Koufaty has been a part of the microarchitecture and performance team for the Intel Pentium 4 processor and a key developer of Hyper-Threading Technology. David has also been involved with post-silicon performance debug of various x86 server and desktop processors. He received B.S. and M.S. degrees from the Simón Bolívar University, Venezuela in 1988 and 1991, respectively, and a Ph.D. degree in Computer Science from the University of Illinois at Urbana-Champaign in 1997. His main interests are processor microarchitecture and performance. His e-mail address is david.a.koufaty at intel.com.

Meng-Jang Lin received a B.S. degree in Electronics Engineering from National Chiao-Tung University in Taiwan, an M.S. degree from the University of Oxford, and a Ph.D. degree in Computer Engineering from the University of Texas at Austin. She joined the Intel Pentium 4 Architecture team in 2002. Her e-mail address is meng-jang.lin at intel.com.

Mahesh J. Madhav received an Sc.B. degree in Engineering from Brown University in 1999. He then helped research and design an x86-based hand-held computer jointly at TIQIT Computers and Stanford University, where he received an M.S. degree in Computer Science in 2002. Hired into the Pentium 4 microarchitecture team in 2002, he spends his time exploring new simulation technologies, post-silicon performance debug, and hacking code. His e-mail address is mahesh.j.madhav at intel.com.

Markus Mattwandel received an M.S. degree in Computer Engineering with an emphasis on High Performance Computing from Oregon Graduate Institute in Beaverton, Oregon in 1991. He joined the Intel Desktop Products Group Compatibility Validation team in 1994 where his primary responsibility is to direct performance validation of Intel processors. He has worked on Pentium® Pro, Pentium® II, Pentium® III, Celeron® and Pentium 4 processors. His e-mail address is markkus.mattwandel at intel.com.

Nidhi Nidhi received a B.S. degree in computer engineering from Delhi College of Engineering in 2000 and an M.S. degree from the University of Wisconsin, Madison in 2002. She joined the Pentium 4 performance team shortly after. Nidhi holds a patent on branch prediction schemes and likes to follow the financial markets. Her e-mail address is nidhi.nidhi at intel.com.

Jonathan D. Pearce received B.S. and M.S. degrees in Computer Engineering from Carnegie Mellon University in 2002. He joined the Pentium 4 processor performance team after graduation where he worked on processor performance optimizations. He is currently a member of the Logic Technology Development architecture team working on the lead processor for Intel's 65nm process technology. His e-mail address is jonathan.d.pearce at intel.com.

Madhusudanan Seshadri has worked on Pentium 4 performance validation since joining Intel in 2002. Madhu holds an M.S. degree in Electrical Engineering from the University of Wisconsin-Madison and a B.E degree from Anna University, India. His interests include processor microarchitecture and VLSI design. His e-mail address is madhusudanan.seshadri at intel.com..

® Itanium is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

® Pentium is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

® Celeron is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

LVS Technology for the Intel® Pentium® 4 Processor on 90nm Technology

Dan J. Deleganes, Desktop Platforms Group, Intel Corporation
Micah Barany, Desktop Platforms Group, Intel Corporation
Daniel Chow, Technology and Manufacturing Group, Intel Corporation
Tom D. Fletcher, Desktop Platforms Group, Intel Corporation
George L. Geannopoulos, Technology and Manufacturing Group, Intel Corporation
Kurt Kreitzer, Desktop Platforms Group, Intel Corporation
Anant P. Singh, Desktop Platforms Group, Intel Corporation
Sapumal B. Wijeratne, Technology and Manufacturing Group, Intel Corporation

ABSTRACT

To meet the demands of low-latency integer operations, the Intel® Pentium® 4 processor architecture implements fast integer operations using a 2x frequency core clock. The frequency advances enabled by Intel's new 90nm technology when paired with a 2x frequency multiplier require novel circuit topologies if latency is to be optimized. The discussed solution uses unprecedented levels of small signal random logic to implement a double frequency X86 integer core. This circuit technology, termed "Low-Voltage Swing" (LVS) enables the Pentium 4 processor [1] to take full advantage of Intel's new 90nm technology [2].

INTRODUCTION

Microprocessor performance can be defined as the product of latency and parallelism. Since parallelism has been well exploited in previous microprocessor generations, the integer performance in the Intel Pentium 4 processor architecture is achieved using ultra-low-latency integer operands. The reduced latency when then paired with Hyper-Threading Technology (parallelism) empowers a one-generation-ahead design. Like the preceding Pentium 4 processor designs, the newest member of the family on Intel's 90nm technology enables ultra low-latency integer ops by running the integer core

at twice the core frequency of the microprocessor. At today's clock rates, this operating frequency is in and of itself notable. For example, a 3.4 GHz processor would have the integer logic functioning at 6.8 GHz. Such a frequency target is on the low end of 90nm technology capabilities–that is, at the beginning of process life. End-of-life process technology frequency expectations are far higher. In this paper, we describe the implementation of the newest Pentium 4 processor integer logic core using Low-Voltage Swing (i.e., differential small signal) logic. This circuit topology, referred to most frequently as "LVS," is designed explicitly to take advantage of the frequency headroom enabled by Intel's new 90nm technology. In this paper we explain the overall circuit topology, and take you on a walk-through of three core blocks: the Alignment Mux, Adder, and Rotator. A section describing the tools/methodologies for pre-silicon verification necessary for high-volume manufacturing (HVM) is outlined, which includes small signal path tracing, merging dynamic and static timing, and matched layout. Finally, you will see up-to-date post-silicon data demonstrating the integer core running at higher frequencies than any other published X86 integer cores.

® Intel and Pentium are registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
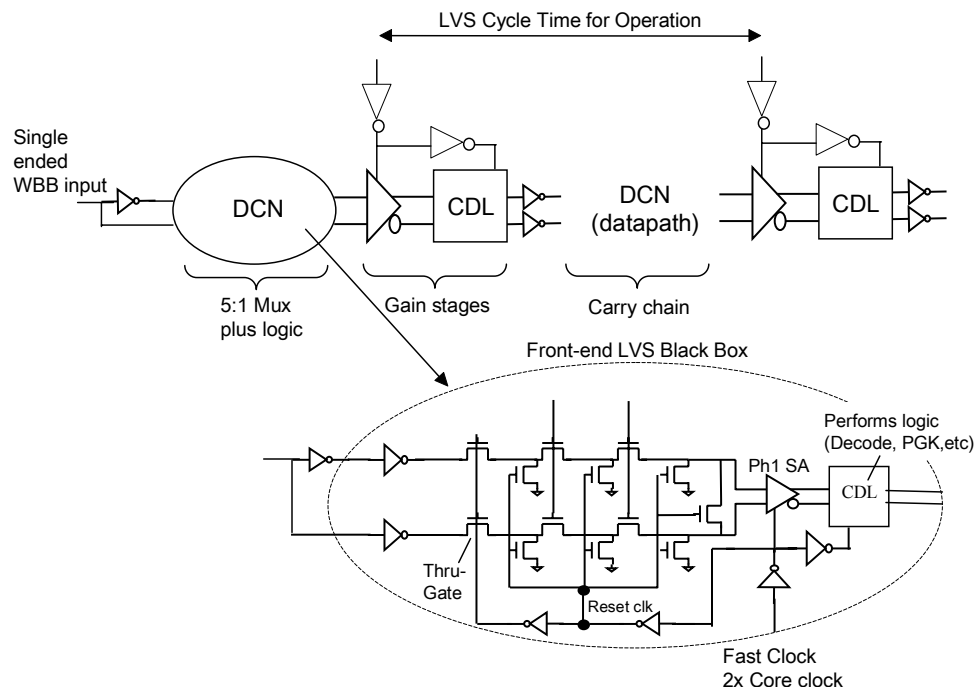
Figure 1: LVS circuit block diagram

LOW-VOLTAGE SWING LOGIC AT INTEL

In 1997 Intel researchers began investigating ways to continue designing the Intel Pentium 4 architecture's 2x frequency integer core on process technologies many years in the future. Looking several generations ahead, they were concerned that the self-resetting domino topologies used so effectively in the original Pentium 4 design would need to be replaced with even faster circuit topologies, if the integer core was to keep pace with the capabilities of future manufacturing technologies. These researchers, led by our co-author Tom Fletcher, determined that large Diffusion Connected Networks (DCN) with multiple inputs and outputs could be used to implement significant logic functions in a single stage. Although such structures are excruciatingly slow at creating standard CMOS voltage levels, it was recognized that by using differential (true and complement) functions, the resulting "small signal" voltages could be differentially sensed and amplified into a "large signal." This circuitry operated faster than even our fastest domino circuits. The delays through two stages of sense and gain were costly, but since the diffusion connected

network was capable of doing six to eight stages of logic in a single stage, the overall time to implement a complex logic function was a net performance win over other topologies. Furthermore, it was determined that such networks could readily take advantage of straightforward pass-gate algorithms, such as carry skip addition, to minimize the number of series devices. The resulting differential Low-Voltage Swing (LVS) topology used fewer transistors to implement a given logic function, which lead to an area advantage over traditional static or domino circuits. The topology also promised low-power opportunities at equal frequencies due to reduced voltage transitions. The performance of speed, area, and power wins led to the technology being selected for the next-generation design. During technology development, the LVS circuit topology that delivered the best performance operated like domino logic, with evaluate and reset phases. The higher linear region currents of N-transistors make them the device of choice for DCN pre-conditioned to ground being selected. A P-type sense amplifier senses the differential output of these pass-gate DCNs.

The potential gains of this new topology promised to be significant. However, the design complexity was identified as a major concern. The sheer amount of small

signal logic that would be needed to implement an entire X86 logic core was unprecedented. Consider this: the transistor count of this execution core exceeds that of the entire Pentium Pro design! There were no tools for timing analysis, noise analysis, or logic verification. To minimize the differential and common-mode noise, new layout checks were needed to ensure that custom devices in random logic met analog layout requirements. Pulsed clocks required careful crafting. Clearly, the challenges of implementing an entire Pentium 4 integer core using small signal circuits to implement logic functions would be an extreme challenge. The work began!

LOW-VOLTAGE SWING CIRCUIT ARCHITECTURE

Figure 1 shows the basic topology of the LVS circuits used. An LVS circuit, called the Front End (FE), implements an LVS multiplexer to select among the Write Back Buses and the Source Buses. For simplicity, the DCN diagram shows only two N-pass-gates connected to each node, when in reality, each node would have multiple inputs. The Complementary Domino Logic (CDL) gain stage restores the sense-amplifier output's ratioed voltage levels. The CDL in most cases also implements logic; for example, in the Adder this would be a Propagate, Generate, Kill (PGK) function generator. Also shown is the thru-gate, which acts as a min-delay blocker by gating static data entering the DCN. The thru-gate is controlled by a clock that turns on one inversion after the de-assert of the reset clock. By ANDing the thru-gate clock with a logic signal, one series device can be removed, further improving speed. In Figure 1, the carry chain is collected into a second LVS blackbox that produces the result of a 16-bit add.
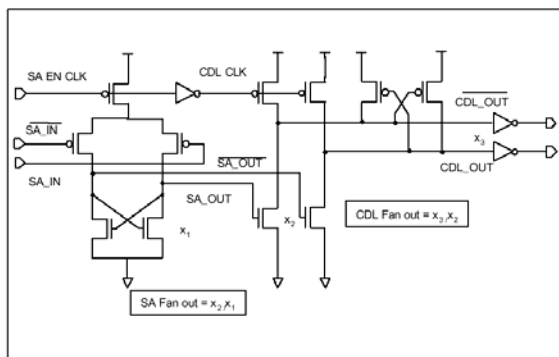
Figure 2: Sense-amplifier and CDL inverter followed by a CMOS inverter

Figure 2 illustrates the ratioed P-type sense-amplifier driving a simple CDL inverter. The reset devices are removed to simplify the diagram. The outputs of the LVS carry-chain DCN connect to "SA IN" and its complementary pin. The timing relationships between the

LVS DCN, the sense-amplifier, and the CDL are shown in Figure 3. The sense-amplifier and the CDL are in phase and are controlled by the clocks named "SA EN" clock and "CDL CLK," respectively. The rising edge of "SA EN" clock initiates the reset of the sense-amplifier outputs, and it is immediately followed by the precharge of the CDL outputs. During this time the LVS DCN (carry-chain) is in evaluation and generates a differential voltage at the inputs of the 17 receiving sense-amplifiers of the 16-bit adder. The falling edge of the "SA EN" clock triggers evaluation of these sense-amplifiers. This event is depicted in Figure 3 with a vertical line that intersects the 50% transition point of the falling "SA EN" clock. In this example, it can be seen that at the sense-amplifier trigger point, the input differential is approximately 344 mV with 49 mV of common mode. The lower plot in Figure 3 illustrates the sense-amplifier outputs resolving this input differential. The non-zero offset level on the sense-amplifier '0 output can be seen to induce a glitch on the non-switching terminal of the CDL (middle plot) that is mitigated by the cross-coupled P-keepers on the CDL. The magnitude of this glitch is a decreasing function of the sense-amplifier input differential. Below a certain minimum input differential voltage, the CDL glitch could potentially induce a domino false-discharge failure mechanism on the CDL output, resulting in a speedpath or logic failure.

It can be seen in the bottom plot of Figure 3 that the DCN reset clock resets the inputs of the sense-amplifier shortly after the sense-amplifier trigger point. In fact the DCN reset is initiated only one inversion after the sense-amplifier. If the inputs to the sense-amplifier are reset before the sense-amplifier resolves, then a functional failure will occur. The part will then not operate at any frequency. This race is known as the "sense versus reset" race, and it is the only functional race in this LVS design.

A typical LVS circuit is allocated only about two inversions to develop differential!

Full details on the adder circuitry are detailed in the Adder Circuit section below.
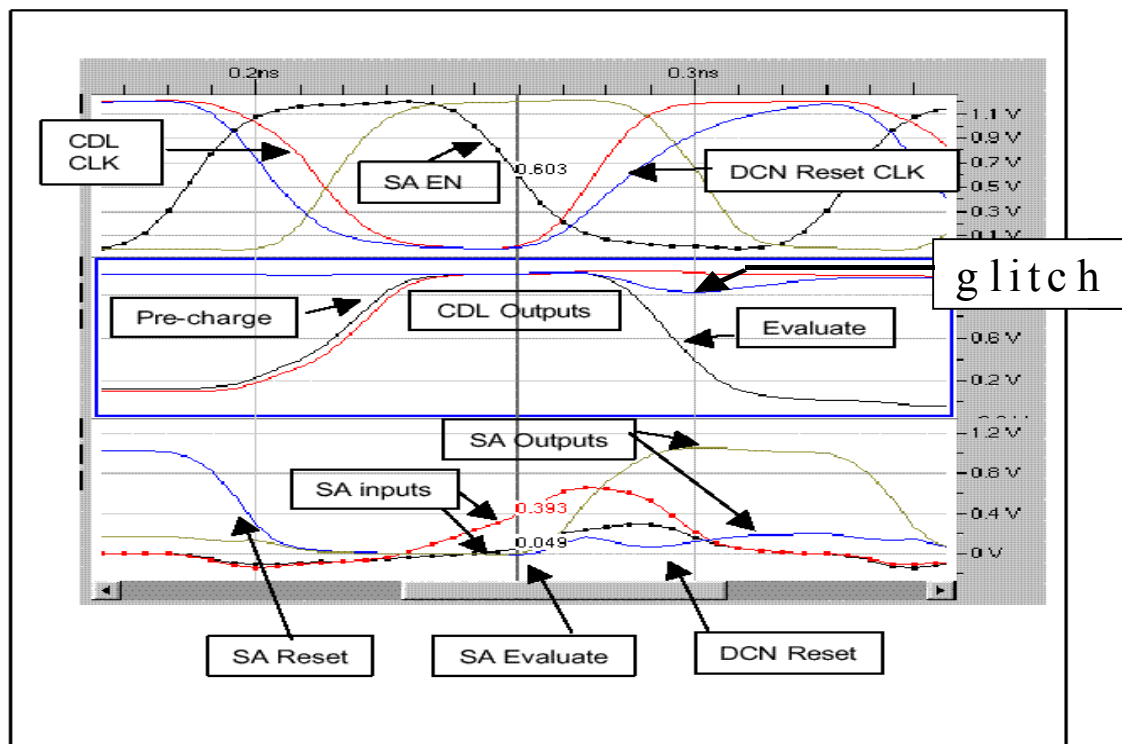
Figure 3: LVS waveforms

CLOCKING

The core logic of the CPU is running at the specified processor frequency. For a 3.5 GHz core clock frequency, the Main Core Clock (MCLK) period is about 285 ps. The pulsed Fast Clock (FCLK) doubler circuit doubles the MCLK frequency, in this case to 7 GHz. One FCLK phase is allocated for LVS DCN signal development, and the other FCLK phase is allocated for DCN precharge. The pulsed clocks used on the previous 2x Intel Pentium 4 integer cores were ideal for clocking LVS circuits because they would only stop in the reset phase. This meant that when the clock was stopped there would be zero source drain leakage for the pass-gates because all nodes would be reset to ground. The pulsed FCLK is generated by combining two tunable NAND chopper delay circuits into a pseudo wired-or. One chopper is sourced from the MCLK, the other, one inversion later. In order to provide symmetric FCLK pulses for both phases of the MCLK, the low phase of the MCLK is one inversion delay longer than the high phase to account for the additional inversion to the second chopper. This non-fifty percent duty cycle of the MCLK allows the core circuitry to do more work in the low phase of the MCLK for the non-LVS MCLK circuitry, but limits the bandwidth of the global clock distribution more than if it

was a true fifty percent duty cycle clock. Figure 4 shows the clocking edge relationships.
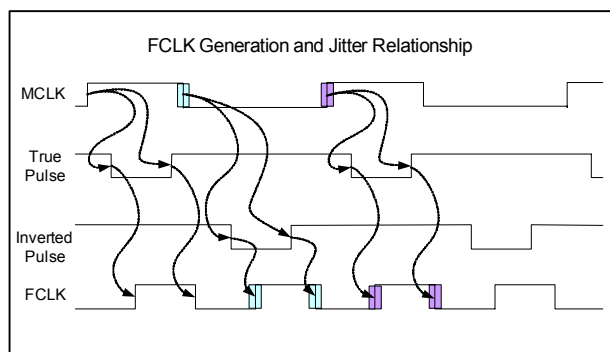
Figure 4: Fast Clock (FCLK) timing edge relationships

Clock skew and jitter posed significant challenges to LVS design, especially as these would degrade an FCLK phase speedpath four times as much as an MCLK cycle path. Exact control of the MCLK high and low phases have a direct impact on the allowable time for the low phase of FCLK; if either phase of MCLK gets smaller, the FCLK low phase will also get smaller by the same amount. A key advantage of pulsed clocking is that the FCLK high phase is not affected by MCLK jitter and skew, since both FCLK edges are generated from the same MCLK edge.

With cycle-to-cycle jitter, just the low phase of the FCLK will be impacted. The design of the LVS blocks took advantage of these effects and accounted for them directly during timing analysis.

LOW-VOLTAGE SWING USAGE WITHIN THE INTEGER CORE

Figure 5 provides an architectural block diagram, showing the critical integer core components. LVS circuitry enabled the Intel Pentium 4's low latency, used in the critical L0 load pipeline's alignment mux, adders, logic unit, rotator, and the address generation unit. Details of LVS used for the adder, rotator, and alignment mux circuits are given below.
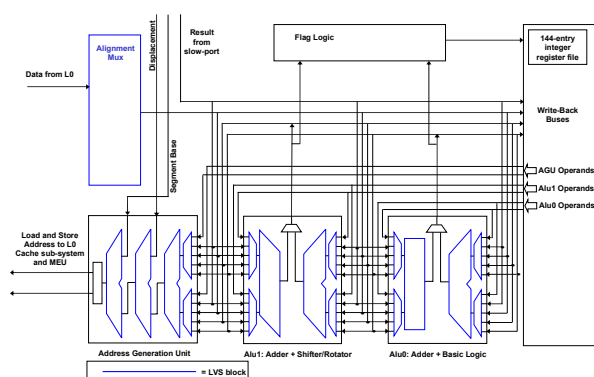
Figure 5: Integer core architectural diagram showing LVS usage

Adder Circuit

The LVS carry chain for a 16-bit adder is shown in Figure 6. It is built upon 16 cascaded LVS PGK cells, named "0-F," with carry-skip pass-gates "s0-s9" placed between them such that the span of any carry propagation path is limited to no more than 6 series devices. The LVS cells that make up the LVS carry-chain are shown in Figure 7. The LVS XOR gates in Figure 7 hook up to each polarity of the carry [n] nodes along the carry-chain to produce the sum [n+1] result. The typical critical path begins with the turning ON of the "s0" skip pass-gate that allows the "Cin" to charge up the precharged-low carry-chain and develop differential at the inputs of 17 PMOS sense-amplifiers that sense the 16 bit sum and the carry out. This 16 bit adder spans half the datapath height. It's length represents the total interconnect distance that has to be traversed for a carry-chain to propagate from "Cin" to "Carry<15>." A bit slice of the LVS adder, including adder PGK controls and clocking, is shown in Figure 8. The LVS front-end evaluates in phase 2 of the FCLK and presents source data to the first sense-amplifier "P-SA-1" that is triggered on the fall of the "ckxf1pb6_b" clock. The next stage "CDL-1" is an inverting level-restoring stage that has integrated PGK logic. This circuit is a complex CDL gate that begins evaluation on the rise of the "ckxf1p7c" clock. Exactly one FCLK phase later the fall of the "ckxf1p7c" clock triggers the 17 sense-amplifiers commonly titled "P-SA-2" (see Figure 8) that capture the sum and carry results of the 16 bit LVS adder.

Typically, the critical path goes through the "gp [n]" group-propagate signals. The wide NOR gates that generate "gp [n]" group-propagate functions are allocated only 16 ps. A conventional design of a fast ratioed-NOR, similar to the one illustrated in Figure 9, could not be used to implement wide, single-stage, precharged-low NOR functions required by the LVS adder. For certain input combinations, the gates' pull-up and pull-down networks are on simultaneously, and for these cases the ratioed-NOR gate's output can produce a steady-state noise source that is typically in the ~200-400 mV (Vcc = 1.2 V, 1262) range. A novel p-interruptible ratioed-NOR gate (RP-NOR), illustrated in Figure 10, was designed in place of a ratioed-NOR (RNOR) gate. RP-NOR gates can implement fast NOR2-NOR5 gates while limiting the contention-induced noise to a small and narrow glitch. The signals "pc," "pa," "pb" are the inverse of "pcn," "pan," and "pbn," respectively. All inputs are precharged high. During precharge, the top P-device is on, enabling it to precharge the internal node n1 thus facilitating a fast rising transition. If "pdn" falls, then the P-stack is enabled and the NOR will begin to rise. This transition is allowed to continue only if "pan," "pbn," and "pcn" also fall, because this is the only condition for which the top P-device will remain enabled. For all other combinations, the P-device is disabled within one gate delay, limiting the contention to a narrow pulse that is approximately one gate delay wide. Figure 11 shows the comparison of the RP-NOR contention noise pulse to the DC contention waveform produced by a RNOR within the context of the LVS adder's evaluation window. A RNOR produces a contention waveform that exists for the entire duration of the LVS evaluate window, allowing the off skip device driven by it to leak opposite charge onto the carry-chain, degrading or even destroying the DCN signal development. The new RP-NOR, however, produces its contention glitch only at the onset of LVS evaluation leaving the carry-chain a significant amount of time to recover and develop positive differential unimpeded by any further contention-induced differential noise.
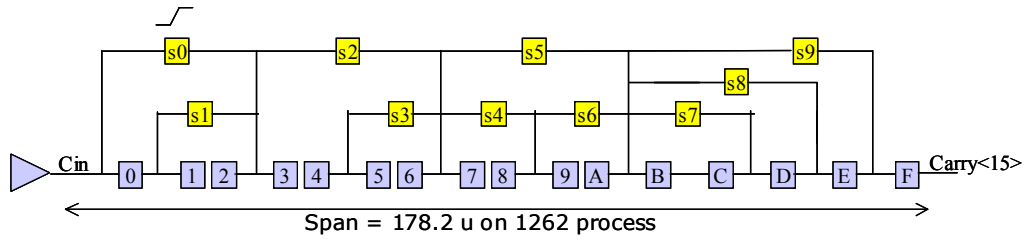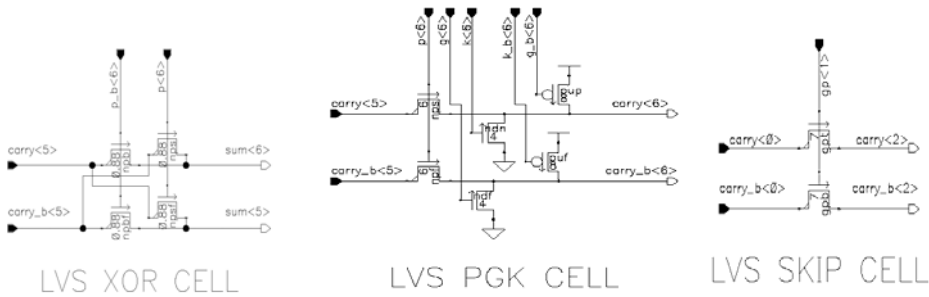
Figure 6: 16-bit adder LVS carry-chain structure

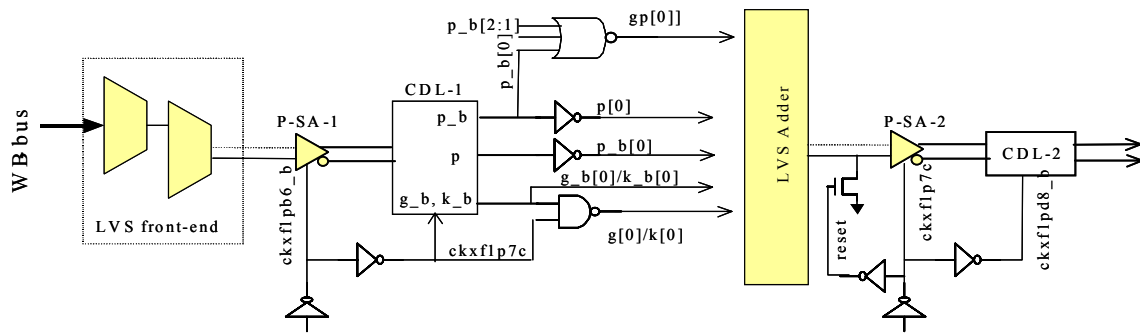**Figure 7: LVS PGK and XOR cells used in the LVS carry-chain**
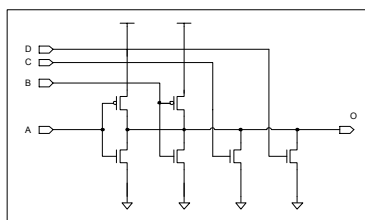

Figure 8: Logic for adder bitslice

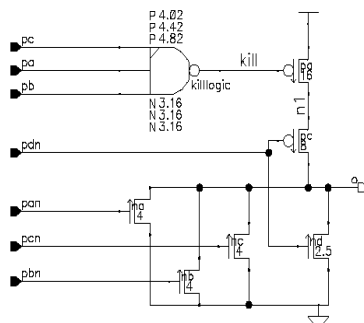**Figure 9: A 4-input ratioed NOR (R-NOR) gate**

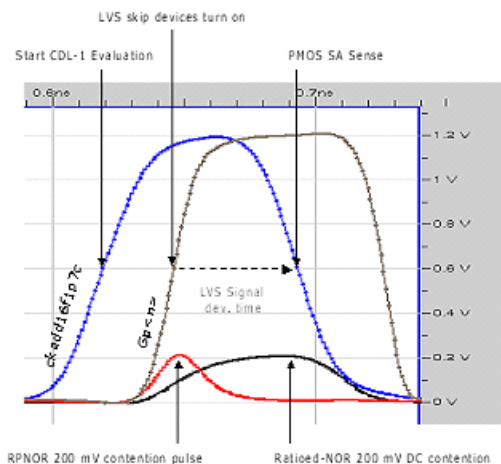**Figure 10: A 4-input ratioed P NOR (RP-NOR) gate**

Figure 11: Contention behavior of ratioed-NOR gate (RNOR) versus new RP-NOR gate

Alignment Mux Circuit

LVS makes possible the implementation of the Alignment Mux function in an MCLK phase (144ps), reducing the critical "load pipeline" latency in the integer core. It provides a 2x speed improvement over the traditional multi-stage domino design. The Alignment Mux datapath function consists of 128 individual 32:1 dual rail muxes distributed across the datapath width of the entire L0 cache. Muxing is performed with a single-stage DCN pair followed by the large distributed mux node connected to the sense-amplifiers. This RC requires designing the

Alignment Mux at MCLK frequencies, whereas all other LVS blocks operate at FCLK. Source inputs to the DCN are full-swing dual rail signals from the L0 cache. The clock gated control logic generates the full-swing DCN selects, which are replicated across the entire mux height to reduce loading and RC. Figure 12 shows the circuit topology of the Alignment Mux.
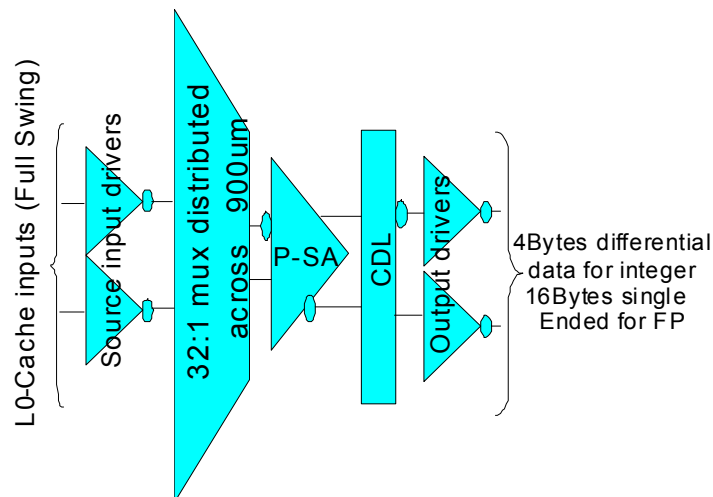
Figure 12: Alignment Mux circuit block diagram

LVS was the ideal technology for the Alignment Mux, with its muxing and distributed RC. LVS technology proved crucial in reducing the L0 cache latency and has enabled aggressive frequency headroom for subsequent Pentium 4 steppings.

LVS Rotator and Shifter

The LVS rotator/shifter performs these operations (ops): Rotate Left (ROL), Rotate Right (ROR), Shift Left (SHL), Shift Right (SHR), Shift Arithmetic Right (SAR), Byte Swap, and High-Low Swap. The only 8-bit operations supported are "8L," performed on bits [7:0] of the operand. "8H" rotates and shifts are done in the Intel Pentium 4 processor slow port datapath and are longer latency operations. For 8-bit and 16-bit rotates and shifts, the remaining bits of the operand are passed through unchanged to the result. For SHL and SHR ops, the value of '0 is padded in from the least significant or most significant position, respectively. For SAR ops, the value of the most-significant bit is padded in from the most significant position.

Rotate and shift operations are done by first rotating the operand according to the shift count, and then selecting either the rotated value or the "kill value" to produce the final result. The kill value is always '0 for SHL and SHR ops, and it is the size-appropriate, most-significant bit for SAR ops.

As in reference [3], the block algorithm takes advantage of symmetry to streamline the rotation portion of the datapath.

Right rotates with a shift count of *r* are done by rotating the operand to the left by *r#* + 1 places. For right ops, the shift count is complemented prior to entering the shift count decode logic, and the extra "+1" place is taken care of in the datapath.

Rotator circuits are traditionally a series of muxes wired up with long interconnects to steer the operand over the length and breadth of the datapath. This makes the rotator particularly suited to LVS technology. The datapath muxes are implemented with a wide DCN that ends up at 32 sense-amplifiers for the "prop value" and an additional 32 sense-amplifiers for the "kill value." A final muxing stage selects between the outputs of these two sets of sense-amplifiers. The selects for this muxing stage are bitwise. The complex shift logic is implemented using LVS circuits. The decode of the shift and rotate count is done in the Front-End by embedding logic into the CDLs and the subsequent static logic stage. LVS technology has enabled us to implement Fast Rotate and Shift ops in Pentium 4 processors, which provides a significant and measurable performance gain.

TOOLS AND METHODOLOGY

A large challenge to enabling the design of LVS circuitry was to provide the innovative tools and methodologies that enabled us to successfully apply this technology to a HVM environment. We created LVSTNT (LVS Timing and Noise Tool), a custom dynamic simulator and interface, that calculates required and valid times for groups of data and pass-gate signals interfacing with the LVS circuitry. These results were merged with our project standard static timing tools and flows. A custom LVS Layout Rule Checking (LRC) tool was developed to eliminate non-common mode noise and to ensure layout matching that can tolerate process variation.

Dynamic Simulation and Timing Issues

LVS timing performance depends upon the relative arrival of dozens of signals, followed by the generation of a small differential signal, which requires a uniquely complicated timing analysis. In contrast, traditional static timing analysis simply assumes that a single signal generates a timing path. And unlike a typical dynamic simulation, which simply verifies that a circuit operates at a target frequency, our LVSTNT dynamic simulator provides a key advantage by calculating the worst-required times for the input signals. By having the required times, we know how much timing margin a given input has, enabling us to make valuable area/power/delay trade-offs. To avoid exponential growth in the number of timing paths when calculating the required input timings of combinations of multiple signals, we made creative, specific assumptions to maintain a linear number of simulations. And even after pruning the number of timing paths using patented algorithms [4], the rotator alone required simulations on

more than 60,000 paths to characterize the circuit. To address the associated huge runtime and database size, LVSTNT partitions the LVS circuitry into the minimum database needed to dynamically simulate each unique path. We can quickly, interactively, simulate a single path of interest. As even a single path requires 3-12 simulations to find the passing conditions and input required times, understanding so many simulation results proves daunting, so LVSTNT automatically merges the worst-case results from all simulations for dozens of timing constraints. While analyzing the complete circuit, we batch and send all simulations to our compute farms, utilizing hundreds to thousands of machines worldwide.

Transparently clocked designs provide greater tolerance to clock skew on silicon, which is a significant portion (15-20%) of the FCLK cycle time. The LVSTNT required times are merged into our normal static timing tools, enabling us to take full advantage of transparency through latches and domino state elements. To provide a transparent timing interface, DCN selects and dual rail data inputs were ideally designed to be precharged. During evaluate, the DCN select gate inputs and data and data# inputs would transition monotonically; if this occurs after the thru-gate opens, then we have a nicely transparent timing path. This elegant interface is not feasible when just single rail data inputs are available because both data and data# cannot be precharged to ground. Generating data# locally, results in either data or data# starting out high before evaluate, and as the first pass-gate opens, the DCN would start developing the opposite logic value before developing the correct small signal waveform. This posed significant simulation modeling challenges. Aiming for a robust design, we decided to prevent generating this wrong differential by adding an extra clocked n device, the thru-gate, and then requiring that data be set up to the thru-gate opening. While the thru-gate intrinsically slows the circuit, due to the additional n device in series, this greatly simplified the timing complexities of both dynamic and static analysis, enabling robust tools.

Determining whether the circuit operated or failed raised many questions in our challenge to enable HVM. Sense-amplifiers in the ideal world of a dynamic simulator resolve with just a few electrons. On silicon, the coupling noise to signal waveforms and power supply alone contribute significant complexity to an ideal model. A few failure criteria are described below.

An initial failure criterion used during pre-silicon verification was the magnitude of the CDL output glitch. If this noise glitch propagated to a subsequent domino stage or state element, a logic error or severe speedpath could occur. This provided an easily implementable pass/fail criterion that was based upon an observable

circuit failure. In standard CMOS designs, noise tools verify that the circuit will not falsely discharge a domino node. In LVS designs, we not only guarantee this will not happen, but this failure point can directly dictate the required set-up times to the sense-amplifier. Traditional static and dynamic circuits never attempt such closely intertwined timing and noise requirements.

A second failure criterion mandated a minimum differential voltage. The total requirements ranged from 50-100 mV, or roughly 5-10% of VCC. Device variations due to process (Le, Vt, dual-Vt, etc.) was analyzed for our library of sense-amplifiers, and accounted for nearly half of the signal voltage requirement, with the remaining attributable to incomplete precharge, noise, and the inherent differential needed to sense correct data. This requirement enhances confidence in the design, and it covers several corner cases and simulation artifacts not caught by the CDL failure criterion.

A third criterion avoids designing non-full-rail static signals, which static timing tools ignore, but are very easy to create at such extremely high frequencies. While a dynamic simulator shows circuit robustness with non-full-rail input signals, real silicon in HVM would not be nearly as forgiving. An additional motivation in avoiding non-full-rail signals is the inability to define them when we translate waveforms back to the static timing tool environment.

Functional race analysis across process corners poses additional failure, modeling, and runtime concerns. Our implementation contains just one functional race, the sense-amplifier enable assertion to the DCN reset clock. Minimizing functional races (mindelays) is important, because if they fail to make speed, the part will not function at any frequency. If mindelays had occurred, they would have required significant timing guard band (impractical at these frequencies), and/or significant effort to simulate the circuit (with additional timing paths) across process variation. In light of design for debug and testability, we added software controllable circuitry to vary our functional race margins.

Merging Static and Dynamic Timing Tools

Creative solutions enabled interfacing our dynamic timing tools with standard project tools into a seamless tool flow that could be batched. Specific attention was paid to drawing a precise boundary of what was dynamically analyzed: for dynamic simulation. We automated netlist extraction to form a black box containing just the data inverters, DCN, sense-amplifiers, and CDL. LVSTNT provided the minimum number of timing edges, leaving the project static timing tools to analyze all but the small signal development and failure criteria. To enable the static tools to analyze the black box for the remaining

edges, we fully automated the generation of timing tool assertions. For example, the set-up of static data falling would be checked against the thru-gate clock rising. Outside the black box, static timing tools analyze the select and data timings and convert their timings into waveform inputs to the dynamic simulation. Black box interface timings come from a combination of the dynamic simulator, the static simulator, or a worst-case merging of max and min timing. (Providing details on over 30 classes of signal types, domino, static, etc., and edges, rise, fall, lead, trail, is beyond the scope of this paper.) The fully automated performance verification tool suite provides a huge return on investment, given the design size, complexity, and multi-year life cycle of a production processor. This automation greatly increased our productivity, providing consistency of assumptions among our designs, and it enabled high-quality audits of the correctness and thoroughness of our checks.

Dynamic Noise Analysis of LVS

Noise easily overwhelms the tens of millivolts of signals inspiring us to support dynamic noise simulation directly within our LVSTNT timing tool. Project noise analysis tools provided the input waveforms, based on the circuits (static or LVS) driving into the LVS block. LVSTNT super-imposes DC and pulse-wise-linear noise waveforms onto the DCN data and select gates. Logically off devices will result in their gates being slightly turned on, with the device's source tied to the rail that provides worst-case leakage, with respect to the signal being developed. The methodology and algorithms are fairly complicated and posed several logic and circuit challenges. A few early designs proved highly susceptible to leakage constraints, so we formed several guidelines on circuit topologies and sizing to deal with these problems.

Layout Rules and Matching

High-quality layout was a key enabler to not just functional first silicon, but to HVM. Sensing signals on the order of 50-100 mV (~5% of vcc) demanded diligent elimination of all noise, be it from residual precharge, leakage, gate to drain coupling, wire coupling, or non-common mode noise caused by mismatched layout or process variation. Careful, up-front attention to layout enabled early identification, elimination, or mitigation of noise sources.

The creation of a custom LVS Layout Rule Checking tool enabled us to create correct-by-construction layout by highlighting non-common mode geometries. The LRC tool helped guarantee that all differential paths were matched in terms of layout geometries. It analyzed all pertinent device and metal layers, handling process patterning and variation issues, enforcing consistent shielding, and ensuring all signal attackers (cross

capacitance) were common mode. This correct-by-construction layout was absolutely necessary for high-volume production and for the creation of a database of this size. SRAM and analog designs deal with similar layout matching issues, but on a much smaller scale; e.g., SRAMs involve just one arrayed memory cell. LVS circuitry contained hundreds of thousands of unique layout geometries and timing paths, and ensuring matched nlayout alone was a feat never before accomplished on this scale.

To date, no tool or methodology holes have led to functional, yield, or speed issues on silicon.

CONCLUSION

Low-Voltage Swing circuit technology utilizes custom tools and methodologies to implement random small signal logic at an unprecedented scale. Our 2x frequency integer core implementation on Intel's 90nm process meets the present Pentium® 4 processor product demands. With process and post silicon optimizations the design will support increasingly higher frequency processors.

ACKNOWLEDGMENTS

REFERENCES

[1] Dan Deleganes, et. al., "Low-voltage-swing logic circuits for a 7Ghz x86 integer core," *IEEE ISSCC*, February 2004.

[2] S. Thompson, et. al., "A 90nm technology featuring 50nm strained silicon channel transistors, 7 layers of Cu interconnects, low k ILD, and 1 um2 SRAM cell," *2002 IEDM Digest*, Dec. 2002, pp. 21-64.

[3] Pereira, R., and Mitchell, J.A., and Solana, J.M., "Fully pipelined TSPC barrel shifter for high speed applications," *IEEE Journal of Solid-State Circuits*, Vol. 30, No. 6, June 1995.

[4] Stevens, K. and Morrise, M., "Algorithm for finding vectors to stimulate all paths and arcs through an LVS gate," U.S. Patent 6557149.

AUTHORS' BIOGRAPHIES

Daniel J. Deleganes is the project manager of the Intel Pentium 4 processor integer execution cluster described. Daniel received a B.S.E.E. degree from the University of Washington in 1988 and an M.S.E.E. degree from Cornell University in 1989. At Intel he contributed to high-speed Intel i486™ and Intel Pentium processor 2nd level cache SRAM families, several Pentium processor generations, and all Pentium 4 processor generations. His e-mail is daniel.j.deleganes at intel.com.

Micah Barany is the microarchitecture manager of the Pentium 4 processor integer execution cluster described. Micah received a B.S. degree in Engineering Physics from the University of California, San Diego in 1989, and an M.S.E.E. degree from Stanford University in 1993. At Intel he contributed to Intel i386™ SX and Intel i486 SL microprocessor families, and several Pentium and Pentium 4 processor generations. His e-mail is micah.barany at intel.com.

Daniel Chow is a memory design lead and integrator in the Execution Cluster. He joined Intel in 1996 and has been involved with the methodology definition and implementation of high speed circuits in all Pentium 4 processor generations. Prior to Intel, he worked at Motorola as the primary memory designer for the MC68060. He received his MSEE and BSEE from Oregon State University. His e-mail is daniel.c.chow at intel.com.

Thomas D. Fletcher directs circuit methodology and research for new microprocessors in DPG. He has worked at Intel since 1991 and is a senior principal engineer. He was the clock unit owner for the Pentium Pro processor and has steered early circuit design and research for several generations of the Intel Pentium 4 processor. He is listed as inventor or co-inventor for 45 Intel patents and has 5 IEEE publications. His e-mail is tom.fletcher at intel.com.

George L. Geannopoulos co-managed the LVS design team and is currently managing the PLL team in LTD. He joined Intel in 1994. His interests include high-speed circuits, PLL, clock generation and analog design. Prior to joining Intel, he worked at Bipolar Integrated Technologies as a design manager designing VLSI ECL RISC FPUs FPCs and register files. He also worked at MMI/AMD designing programmable array logic (PALs). He received a B.S.E.E. degree from the University of

---

™ i486 and i386 are trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Illinois, Champaign Urbana. His e-mail is george.geannopoulos at intel.com.

Kurt Kreitzer currently manages the LVS design team. He recently worked to spec and develop quality LVS design tools and methodologies. Kurt received a B.S.C.E. degree from Oregon State University in 1994. After working on the Pentium® Pro, he created the modular SRAM array used throughout the Pentium 4 and other projects and implemented the Pentium 4 Trace Cache. His e-mail is kurt.kreitzer at intel.com.

Anant P. Singh is a senior designer on the LVS team. His recent focus is in mixed signal design where he has worked to define, implement and productize LVS circuits in the dual pumped execution core of the next-generation Pentium 4 processor. He has also designed circuits to enable the backside bus logic on Pentium® III processors. Anant has an M.S.E.E. degree. from the University of Washington and a B.S.E.E. degree from Delhi University. Prior to Intel, Anant worked in the field of control systems and automation. His e-mail is anant.p.singh at intel.com.

Sapumal B. Wijeratne co-designed the LVS AGU/ALUs. Prior to this work Sapumal held numerous technical leadership positions including methodology lead for domino circuits and register files. He is currently co-managing the next lead processor's integer execution core design team in LTD. Sapumal received a B.S.E.E. from Lafayette College in 1984 and a M.S.E.E. from Purdue University in 1986. His e-mail is sapumal.wijeratne at intel.com.

® Pentium is a trademark of Intel Corporation or its subsidiaries in the United States and other countries.

THIS PAGE INTENTIONALLY LEFT BLANK

Library Architecture Challenges for Cell-Based Design

Barbara Chappell, Technology and Manufacturing Group, Intel Corporation
Amanda Duncan, Technology and Manufacturing Group, Intel Corporation
Kiran Ganesh, Mobile Platforms Group, Intel Corporation
Manoj Gunwani, Mobile Platforms Group, Intel Corporation
Abhinav Sharma, Mobile Platforms Group, Intel Corporation
Madhu Swarna, Desktop Platforms Group, Intel Corporation

Index words: Cell-Based Design, standard cell library

ABSTRACT

The Intel® Pentium® 4 processor on 90nm technology is the first Intel microprocessor whose significant portion (~50% of the non-cache devices) was designed using a Cell-Based Design (CBD) methodology. In the CBD methodology, Electronic Design Automation (EDA) tools are used with a library of standard cells to build up a large and complex design. This paper describes the challenges involved in designing a standard cell library to enable the CBD methodology to be applied on a large scale on a chip with an aggressive performance target. Factors critical in enabling CBD on the Intel Pentium 4 processor included the breadth of library content, the physical architecture and design guidelines of the cells, the circuit optimization methodologies, and the functional validation of the cells. In addition to these design concerns, careful modeling for timing, noise, reliability, formal verification, and place and route were required. In this paper, we present an overview of the CBD flow, and we discuss these cell library design and modeling issues.

INTRODUCTION

Cell-Based Design (CBD) refers to a design approach that uses a library of basic building blocks called *cells*. Using cells from the library, larger, more complex functions are realized. In contrast to transistor-level *in situ* customization of cell designs [1], the cells are treated as black box entities by the design and verification tools and are fully characterized for timing, noise, reliability, etc.

® Intel and Pentium are registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

The use of CBD in the Intel Pentium 4 processor on 90nm technology enabled large "sea-of-cells" designs for improved global optimization and more rapid design convergence. Pre-qualification of the cells and a reduced amount of unique layout contributed to the quality control for the product. The CBD library also aided estimation of chip floor-plan and architectural trade-offs. Early explorations of library architecture in conjunction with 90nm technology pathfinding helped us evaluate the impact of the technology on circuits.

The quality of the library plays a key role in producing a competitive design with the CBD methodology. The goal of this paper is to present some of the technical challenges in cell library design and modeling faced by the designers.

CELL-BASED DESIGN FLOW

The Cell-Based Design (CBD) flow for the Intel Pentium 4 processor consisted of four basic steps:

1. Netlist generation

2. Cell placement

3. Routing

4. Design verification

Due to the very aggressive performance targets of the Intel Pentium 4 processor and other constraints specific to portions of the design, varying degrees of automation were used.

Gate-level netlists were created both by directly synthesizing Register Transfer Level (RTL) code and by manually drawing schematics. Cell placement and interconnect routing were generated using both automatic techniques and manual specification. Design verification

included domains such as logic, timing, noise, and reliability.

One of the main challenges in the CBD flow was design convergence. Standard circuit and layout techniques were applied to solve timing problems including max-delay, min-delay, and max slope. Wire-spacing, shielding, buffer sizing, and other solutions were used to address noise issues. Reliability convergence for electromigration and self heat was achieved mainly through slope fixing, wire sizing, and thermal simulation. In addition, the CBD flow had to automatically perform design completion tasks such as scan insertion, scan chain hook-up, clock tree synthesis, and sizing of clock buffers.

The CBD flow was used across the board on the Intel Pentium 4 processor to implement a variety of design blocks. Some designs such as cache, register files, domino and analog circuits were implemented using custom techniques. Table 1 shows the percentage of area, transistor count, and cell count in the CBD and non-CBD sections of the chip, where cell count in the non-CBD areas refers to the number of custom cells.

Table 1: CBD usage in the Pentium® 4 processor

	CBD	Non-CBD
Area	52%	48%
Cell count	44%	56%
Transistor count	50%	50%

CELL LIBRARY DESIGN

Library Content

The library consisted of over 1600 cells, covering over 130 unique logic functions, and associated collateral included over 20 file types. Most cells had variants implemented with different threshold voltage transistors to enable trade-offs between delay and leakage power. The percentage of library content, usage, and effort for different cell types is shown in Figure 1. The library content included complex cells that were aimed at device-dominated, high-speed datapath and control blocks. Buffers and sequentials for use as repeaters and drivers for global nets were also included in the library. Sequentials included latches and flip-flops with a wide variety of functions, and scan and non-scan variants for almost every type. As shown in Figure 1, although the sequentials made up less than 45% of the library, they accounted for more than 65% of the library effort, due to their design and characterization complexity.

A wide range of drive strengths was designed for each logic family. Drive strengths were selected to maximize transistor density within the cells as much as possible,

while providing adequate granularity of drive strengths for optimal tuning of paths.
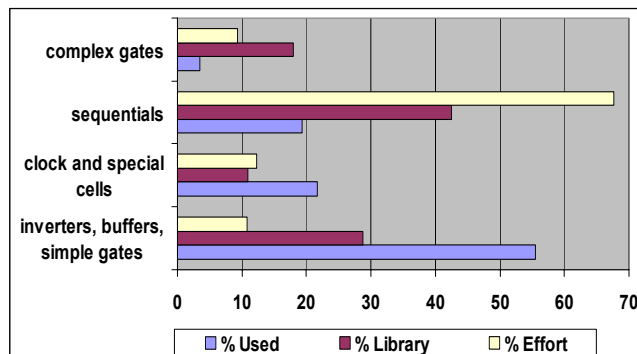
**Figure 1: Library percentage content, effort, and usage**

Library Architecture

The architectural specification is what distinguishes a CBD library from a random collection of cells. The scope of this specification is broad–including naming convention, physical design template, drive-strength definitions, electrical and physical design guidelines, and methodology for verification and for production of collateral for chip design. For the Intel Pentium 4 processor project, the fundamentals of the library architecture were defined very early in conjunction with the device, design rule, and fabrication technology definition for the 90nm generation [2,3]. It was revised multiple times before production versions of the library were used in the final design convergence of the Intel Pentium 4 processor.

The physical architecture of the library, Figure 2, is row-based. All cells are 15 metal 2 (M2) tracks tall and a variable, but integer, number of metal 3 (M3) tracks wide. It features wide M2 power busing over the devices and 11 M2 tracks for signals. The outer tracks were useful for routing on metal 1 (M1) and poly, thereby minimizing the need for M2 for intra-cell routing, even in complex cells. No M3 was used inside the cells. Pins in the cell were in M1 in nearly all cases, and they conformed to a specification that balanced cell area against block place-and-route efficiency.

The cell template and rules for internal cell routing were carefully designed to restrict the delay due to resistance*capacitance (RC) to an acceptable level, while minimizing cell area and the use of upper levels of metal. For example, RC delay in the polysilicon layer was acceptably low even when **p** or **n** device widths filled the cell height, as illustrated in Figure 2, but would have precluded the use of a wider gate, even if the cell height was taller. Poly routing was used to cross under M1 routing in the cells (as an alternative to M2 routing) only

if the resulting RC delay was acceptable. All stages with the fastest delay targets use full metal strapping of source and drain diffusions, but limited use of diffusion without metal strapping was allowed on series-connected devices or other devices with longer delays. Netlists, including resistances, were extracted from the cell layout to ensure performance and margin verification accuracy.
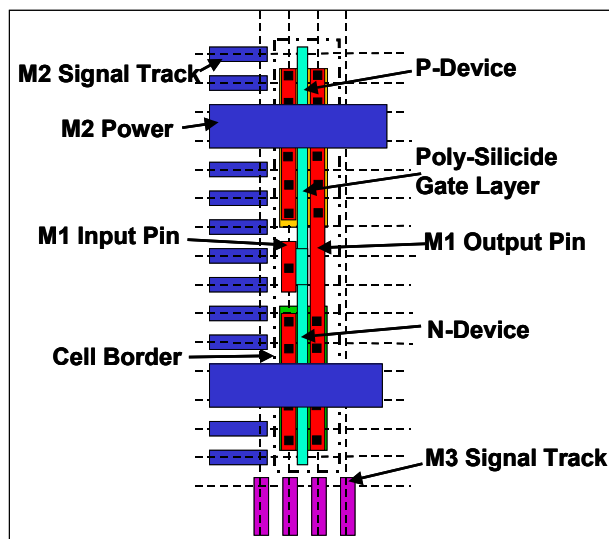
Figure 2: Illustration of standard cell architecture

Cell Design Methodology

P/N ratios for single-stage gates (such as inverter, nand, nor, and-or-invert and or-and-invert) were determined by path-based delay optimizations. Average delay was minimized for a given total gate area, under nominal loading conditions. These ratios were typically determined once per logical family and applied across all of its drive strengths.

Multi-stage gates (such as muxes, xor/xnor gates, etc.) were individually optimized across all drive strengths to minimize objective functions specified by the designers. Typically, the objective was to minimize a power-delay product under nominal loading conditions, while meeting any specified constraints for noise margins, transition times, etc.

Sequential cells are special cases of multi-stage gates. The objective functions for optimization included power-delay trade-offs, noise, charge-sharing, stability, and side-pin (reset/preset/enable) constraints. We modeled each of these constraints under the appropriate skew and voltage conditions. For master-slave flipflops, our key objective was the minimization of its black-hole time (clock-to-out + setup). Prior to the design execution, we performed

several experiments to select the optimal activity factors, driver sizes, waveform shapes, noise criteria, length of clock chains, P/N ratio of output driver, and pass-gate size restrictions. We then leveraged these results during the design of all sequential families.

Individual sizing of each drive strength helped achieve optimal transistor sizes. For example, smaller sized latches were more susceptible to noise and less prone to restore time failures, and we sized the transistors appropriately to reflect this.

The library included phase-1 and phase-2 scan versions of all sequential elements. In phase-1 scan sequentials, the clock is in phase with the scan clock, and in phase-2 versions, the clock is out of phase with the scan clock. Min-delay and max-delay constraints from the scan-load/scan-store operations primarily drove the sizing of scan circuitry. Leakage power considerations drove the choice of non-minimum transistor lengths. Scan cells were built by taking the regular sequential elements and adding a scan gadget on top of them. Using the same gadget sizes for all sequential cells helped minimize design work. The combined scan cell was validated for noise and delay constraints. (See "Full Hold-Scan Systems in Microprocessors: Cost/Benefit Analysis" in this issue of the *Intel Technology Journal* for additional details on scan.)

We targeted some family types, such as clock buffers and min-delay cells, for specific operating conditions. They had their unique optimization methods under restricted delay and transition time domains. Cells compatible with Focused Ion Beam editing, de-coupling capacitors, and other layout completion cells were driven by layout and process requirements.

The 90nm process used in the Pentium 4 processor design featured a choice of dual threshold-voltage transistors. These can be used for power-performance trade-offs. In the library, low Vt transistors were primarily used to gain speed-up for the same layout footprint. These cells were generated from the nominal versions by converting selected devices to low Vt, without transistor size changes.

A host of internally developed automation tools helped ensure high productivity even when cells were individually optimized. These included tools for circuit optimization, parasitic and reliability estimation, low Vt variant generation, and layout automation. Once a design was set up for optimization, it propagated through process file revisions, design target changes, creation of design variants, and the addition of new drive strengths with minimal effort.

Library Qualification

Once a library cell is designed, it must be qualified to meet its logic, circuit, and layout design specifications. Logic equivalence tools were used to validate that each cell implemented the logic function for which it was designed. A host of validation checks was performed to guarantee circuit functionality and performance across wide ranges of temperature, power supply, signal slopes, and process skews. Functional checks for circuits covered noise margins, writability, and node recovery times. Performance checks included delay, setup/hold times, and maximum signal slopes.

In addition to manual reviews, cell layout was checked by a cell architecture verification tool to ensure that it was compatible with the place and route tool. The reliability checks on each cell included those for electromigration, self-heat, and IR drop on the power rails.

Multiple library releases were made during the course of the project. Regression tests were used to ensure that a new library release did not significantly perturb the existing state of the design. Pilot blocks were re-created using the new library to gauge the impact on the design before the library release.

CELL LIBRARY MODELING

Modeling for Timing

To ensure accurate modeling of cell timing, the mathematical timing models used by the static timing analysis tools (typically of the form $timing=f(C_L, TT_{in})$ where C_L is the external load of the cell and TT_{in} is the transition time of the input) were compared against dynamic simulations at each characterized value of the input parameters. This was done for every cell, and problematic cells were studied for further action, which often simply involved recognizing that the models showed large errors for the input parameter ranges at which the cells were rarely used.

Setup time modeling for latches and flip-flops was done based on a set of criteria that involved constraints on storage node transitions at the clock arrival time and acceptable errors in cell delay timings at setup, with respect to delay timings at infinite separation between data and clock. Ideally, in order to reduce modeling errors we would want zero errors in cell delay timings. However, this approach is impractical as a flip-flop's black hole time (clock-to-out delay plus setup time) may not be optimal at this point. Therefore, a point was chosen for the setup time modeling that gave optimal timing at the cost of an acceptable error between cell delays at setup and infinite setup (see Figure 3).
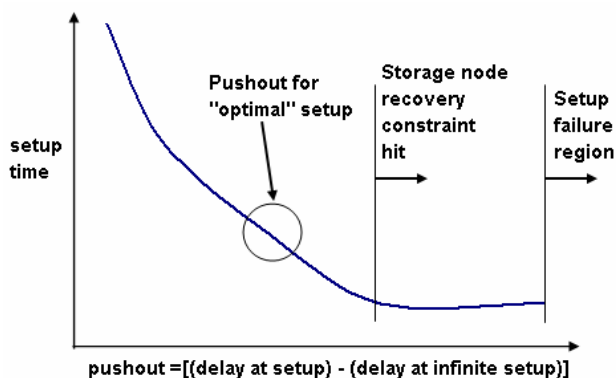

Figure 3: Definition of setup time

Some cells are very sensitive to the environment in which they are placed, which creates timing modeling challenges. One example is a cell that has input pins directly connected to pass-gates. For such cells, large differences in timing due to charge sharing can be seen for timing arcs that involve the switching on of the pass-gates. The charge sharing problem is worse for large pass-gates with weak external drivers. To mitigate this problem, cells with inputs tied directly to pass-gates were designed with restricted pass-gate transistor widths, and they were characterized with the assumption of the weakest possible driver allowed.

Typical cell timing characterization involves a single input transition causing an output transition, but in some instances it is possible that more than one pin transitions simultaneously. Such scenarios were handled for combinational cells by modeling possible increases or decreases in delay when more than one pin transitions at the same time. There are generally a large number of input parameter combinations that could be considered. To constrain the characterization effort, only the most important input parameter combinations were considered, while eliminating those combinations that were found to have a smaller impact on the timing of the cell.

Another interesting case of modeling timing was encountered in the fully decoded multiplexer cells, an example of which is shown in Figure 5. In such cells, only one select pin (sa or sb in the figure) can be on at any given time. Also, select-to-output timing is degraded if one of the select pins transitions high and the other one transitions low simultaneously, relative to the case where only one select pin transitions high with the other one held at a constant low. To model this timing difference, worst-case select-to-output delays were modeled by simultaneous switching of two select pins with one rising and the other falling, and the best case delays were modeled by only one select pin rising.

Modeling for Noise

In the CBD flow, any noise failure on a cell instance can be fixed only by changing the environment around the cell. Therefore, it is important to design cells with good noise immunity while maintaining acceptable timings.
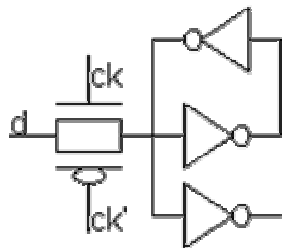
Figure 4: Noise-sensitive latch

An example of a cell in which the noise versus timing trade-off was critical is shown in Figure 4. The latch in the figure is sensitive to noise on both the d and ck input pins. An initial design with a very tight noise constraint turned out to have the same timing as a cell with an extra inverter in the d to output path. A revised noise spec based on a more reasonable assumption of noise attackers gave a design with much better timing. This made the design less robust to noise, but the number of noise failure cases at the CBD level was still manageable. Such cases were used as a reference to come up with suitable noise versus timing design trade-offs for other cells with noise-sensitive architectures. Noise specs were revised for some architectures to give timings that met expectations with the understanding that usage of such cells in CBD would be limited to cases where the noise levels are manageable.

Interconnect parasitics have a big impact on the timing and noise behavior of standard cells, especially when it comes to cross capacitance either between nets within the cell or between nets inside the cell and the ones surrounding the cell. For the library cells, the extraction tool modeled the cross capacitance between nets internal to the cell by looking at the actual geometry and routing of the nets. For the internal net to external net cross capacitance estimations, assumptions were made about what the external routing would look like and the capacitances were extracted based on those assumptions. For timing purposes, the external ends of these capacitances were grounded to prevent unrealistically pessimistic modeling. For noise purposes and for reporting noise behavior of cells to the CBD tools, actual attackers were assumed on the external ends of these capacitors to come up with the worst-case attacker switching scenario. A realistic "derating" of the attacker strength based on the type of layer of the cross capacitance was done to avoid making the results too pessimistic.

Modeling for Reliability

The primary reliability concerns addressed here are Electromigration (EM) and Self-Heating (SH) problems. This was a bigger concern in the 90nm process used for the Intel Pentium 4 processor because of the low thermal conductivity of the low-K inter-layer dielectric.

The analysis at the block level fell under a couple of different categories: (a) validating the cell internals for a given external loading and (b) validating the interconnect between the cells for a certain routing topology. Power grid validation for EM, SH, and voltage drop was a special case of the latter.

At the cell level, the characterization data were analyzed for the product's end-of-life operating condition. The EM/SH characterization data for cells depended on whether a net is a power or non-power (signal) net. For signal nets, the cell characterization data included the maximum average and root mean square (RMS) currents a cell pin could support. For power nets, the characterization produced a model of the cell's power network including resistors and current sinks.

At the block level, the data from cell-level characterization were used to do the roll-ups. The power grid was simulated to test the accuracy of the heat produced by the power nets. The temperature simulator generated a temperature map for the whole die, from which local temperatures for coarse pixels are determined. An EM violation could be waived by reducing the SH current in a neighboring wire that affects the temperature.

Modeling for Formal Verification

The CBD flow required a library rich in content, including cells whose logic did not fit the norm of standard cells in previous libraries.

Cells with constrained inputs needed adequate modeling to ensure that they are handled appropriately. As an example, consider the 2-to-1 multiplexer circuit shown in Figure 5.
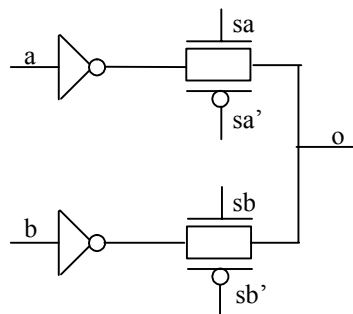
Figure 5: 2-to-1 multiplexer with decoded selects

The logical behavior for this circuit in HDL syntax is expressed as "if sa then a else if sb then b." However, this description has two pitfalls: (1) it implies priority of the sa select over sb when both are active and (2) when both sa and sb are inactive it implies that the circuit holds its values, i.e., behaves as a latch. Moreover, the circuit implementation is such that both selects cannot be made active simultaneously since that can create a short-circuit path. For correct usage of this cell, the constraint that one and only one of the selects (sa or sb) must be active at all times must be met when using this cell at the block level.

Another example of a cell with constrained inputs is a fast XNOR gate with logic function "a*b + ab*bb" needed for high-speed applications. The a & ab inputs, as well as the b & bb inputs are assumed to be complementary signals. This assumption needed to be propagated correctly to the formal verification flow.

Some circuits such as the scan cells were too large for analysis at the transistor level. This issue was overcome by carefully decomposing the circuit into a hierarchy of logic functions, each of which could then be successfully verified.

Formal verification tools have used such constraints but at higher levels in the design hierarchy. The CBD flow necessitated their use even at the library level so that the cells could be black-boxed for logic validation.

Modeling for Place and Route

The advanced 90nm process technology used in the Intel Pentium 4 processor included complex layout design rules that are not adequately comprehended by current place and route tools.

Traditionally, placement tools assume that the spacing between cell layout polygons and the cell border is greater than or equal to half of the design rule spacing for each layer. Thus, the spacing constraints are satisfied even when cells are abutted against each other. The cell layouts had to be carefully designed and modeled to ensure this. Extra checks were added to the layout verification flow to guard against violations.

During routing, cells are traditionally modeled as abstracts consisting of *terminals* (target connection points for routing) and *obstacles* (areas where routes cannot be placed) This posed a couple of problems in the Intel Pentium 4 processor design, including the following:

1. False design-rule violations on obstacles: The obstacles within cells are assumed to originate from physical wiring within the layout and were expected to satisfy the layout design rules for the layer they were on. However, sometimes only part of the wiring on a net can be marked as a terminal for process or circuit performance reasons. This led to router issues

because the unmarked segments created design-rule violations.

As an example, consider the layout in Figure 6 where the via cover on output net O cannot be marked as terminal because it is too narrow to pass the reliability checks. In this case, routing connections must be made only to the wider segment. The existence of other wiring (Net X) prevents the via cover from being widened to pass the reliability check. The part of the via cover outside the fat segment must therefore be marked as obstacle to enforce this–and that part is not wide enough to meet design rules on its own.
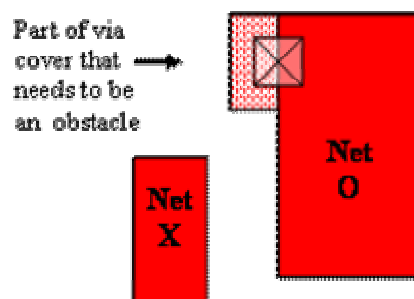
Figure 6: Modeling issue with obstacle

2. Fixed vs. variable spacing constraints: Adequate support for obeying spacing constraints based on local polygon density was not available (similar to the placement issue described above)

The first issue led to the loss of terminal area in some cell abstracts since obstacles had to be expanded to meet minimum width requirements. In some cells, layout rework was needed to satisfy the requirements of the routing tool. For the second issue, a workaround was developed whereby the shape of the obstacle generated was based on the local polygon density of the layer, and the routing tool used a spacing constraint value that ensured no violations were created by it. This workaround required special care during the layout design so that the generated obstacle shape satisfied all design rules as required by the previous limitation mentioned.

CONCLUSION

For CBD to be an effective methodology for a high-performance product like the Intel Pentium 4 processor, many considerations must be addressed during library design in order to use CBD widely without compromising the design. Library richness, in terms of logic functions, drive strengths, and collateral types, as well as an optimized architectural specification, including the physical design template and guidelines, play a key role. Both power and delay must be considered during cell

optimization as well as other constraints for specific cell types. Because the cells are treated as black boxes in the CBD flow, it is critical that they are well-tested for functionality and performance and meet the requirements of the place-and-route tool and all process design rules. There are many modeling issues that must be addressed during the design of a library for CBD. Challenges include the treatment of pass-gate inputs, multiple-input switching, trade-offs between timing and noise robustness, cells with problematic logic descriptions, cells too large for formal verification at the transistor level, and complex layout design rules that may not be completely comprehended by the place and route tool. Careful consideration of these and other issues during the construction of the cell library helped enable the CBD methodology to be used to an unprecedented extent in the Pentium 4 processor.

ACKNOWLEDGMENTS

REFERENCES

[1] Northrop, G.A. and Lu, Pong-Fei, "A semi-custom design flow in high-performance microprocessor design," in *Proceedings of the 2001 Design Automation Conference*, pp. 426-431.

[2] Jan, C.H. et. al., "90nm generation, 300mm wafer low k ILD/Cu interconnect technology," in *Proceedings of the IEEE 2003 International Interconnect Technology Conference*, pp. 15-17.

[3] Thompson, S. et. al., "A 90nm Logic Technology Featuring 50nm Strained Silicon Channel Transistors, 7 layers of Cu Interconnects, Low k ILD, and 1 um2 SRAM Cell," in *Proceedings of the 2002 International Electron Devices Meeting*, pp. 61-64.

AUTHORS' BIOGRAPHIES

Barbara Chappell is an Intel principal engineer in an R&D design organization within the Technology Manufacturing Group. During her eight years with Intel, her contributions have been in the fields of synthesis and library methodologies, in evaluating the impact of process technology on design, and in circuit techniques for high-speed logic. Barbara holds an M.S.E.E. degree from the University of California at Berkeley. Her e-mail address is barbara.a.chappell at intel.com.

Amanda Duncan manages the Standard Logic Implementation group in Intel's Logic Technology Development organization. She has been with Intel for seven years. Her interests include library design and modeling. She holds B.S., M.S., and Ph.D. degrees in Electrical Engineering from the University of Illinois at Urbana-Champaign. Her e-mail address is amanda.duncan at intel.com.

Kiran Ganesh manages the Cell Libraries Development group in the Design Technology organization. He has been at Intel for seven years. His primary interests include design and modeling of cell libraries and physical design. He holds a Bachelors degree in Electrical Engineering from the Indian Institute of Technology, Madras and a Masters degree in Computer Engineering from Syracuse University. His e-mail address is kiran.ganesh at intel.com.

Manoj Gunwani is a project leader in the Cell Libraries Development group and has been with Intel for seven years. His primary interests are in VLSI library and physical design. He holds Masters degrees in Computer Science and Neuroscience from Syracuse University, a Bachelors degree in Electronics Engineering from IIT, Chennai and also pursued doctoral studies in Electrical Engineering at Stanford University. His e-mail address is manoj.g.gunwani at intel.com.

Abhinav Sharma works as a library design engineer in the Cell Libraries Development group in Design Technology, Intel. His interests are standard cell libraries and their usage by automated synthesis and P&R. He received his MSE degree in Electrical Engineering from Arizona State University and his B. Tech degree in Electronics & Instrumentation from Nagarjuna University, India. His e-mail address is abhinav.sharma at intel.com.

Madhu Swarna is a design automation engineer in the Desktop Products Group. He has been at Intel for seven years. His primary interests are in design for low power, timing analysis and library modeling. He earned his Master's degree in Computer Science from Texas A&M University in 1997. His e-mail address is madhu.swarna at intel.com.

THIS PAGE INTENTIONALLY LEFT BLANK

Full Hold-Scan Systems in Microprocessors: Cost/Benefit Analysis

Ravishankar Kuppuswamy, Technology and Manufacturing Group, Intel Corporation
Peter DesRosier, Technology and Manufacturing Group, Intel Corporation
Derek Feltham, Desktop Platforms Group, Intel Corporation
Rehan Sheikh, Desktop Platforms Group, Intel Corporation
Paul Thadikaran, Desktop Platforms Group, Intel Corporation

Index words: Microprocessor, Scan, Test, DFT, DFM, ATPG, DPM, Fault Grading

ABSTRACT

Ever-shrinking microprocessor product development times require enhanced High-Volume Manufacturing (HVM) techniques. This paper describes the full hold-scan testing system implemented in the 90nm Intel® Pentium® 4 processor. Benefits of this scan system include significantly reduced functional test-writing and fault-grade effort, extensive initialization of the design for test and debug, massive visibility into the design for post-silicon debug and fault isolation, and ultimately, a significantly accelerated ramp to production test quality. Any full hold-scan system such as this impacts timing, power, area, and schedule. In a high-performance microprocessor, in particular, this significantly impacts product viability and must be closely managed. In this paper, the Intel full hold-scan system is described, particularly the design challenges, cost optimizations, and test benefits, and we also discuss the costs and benefits of having implemented this successful testing system.

INTRODUCTION

Our full hold-scan system (Figure 1) comprises a full-chip scan bus that acts as a communication channel between the Test Access Port (TAP) and all units. The TAP controller converts data between the TAP clock domain at the pins and core clock domains at the full-chip scan bus interface. This controller also supports serial (TDI/TDO) and parallel modes (36 in/36 out data & address bus pins). There are 29 Scan Functional Units (SFUs) distributed across the chip that interface between the full-chip scan bus and the intra-unit scan chains. Each SFU supports 18

logic chains, 5 reserved chains and up to 3 custom chains. The logic chains provide access to ~200k scan sites in the design. The scan sites are implemented with hold-scan cells, which provide a full shadow of the machine state and enable non-intrusive operation while the system is running or while system clocks are frozen.

Full hold-scan systems have an inherent die area cost and have a critical impact on the timing performance and power metrics of the microprocessor. The transistor scaling for the development of high-speed circuits further exacerbates the power problem. Additionally, the scan circuits contribute to local device and wire congestion leading to design convergence issues. We used a variety of architectural and circuit design solutions to build a commercially viable low-power full-scan system. We describe the full-scan circuits in the latest 90nm Intel Pentium 4 microprocessor. We also discuss the various design trade-offs (cost vs. benefit) and the evolution of Intel's first full hold-scan system design in a microprocessor.
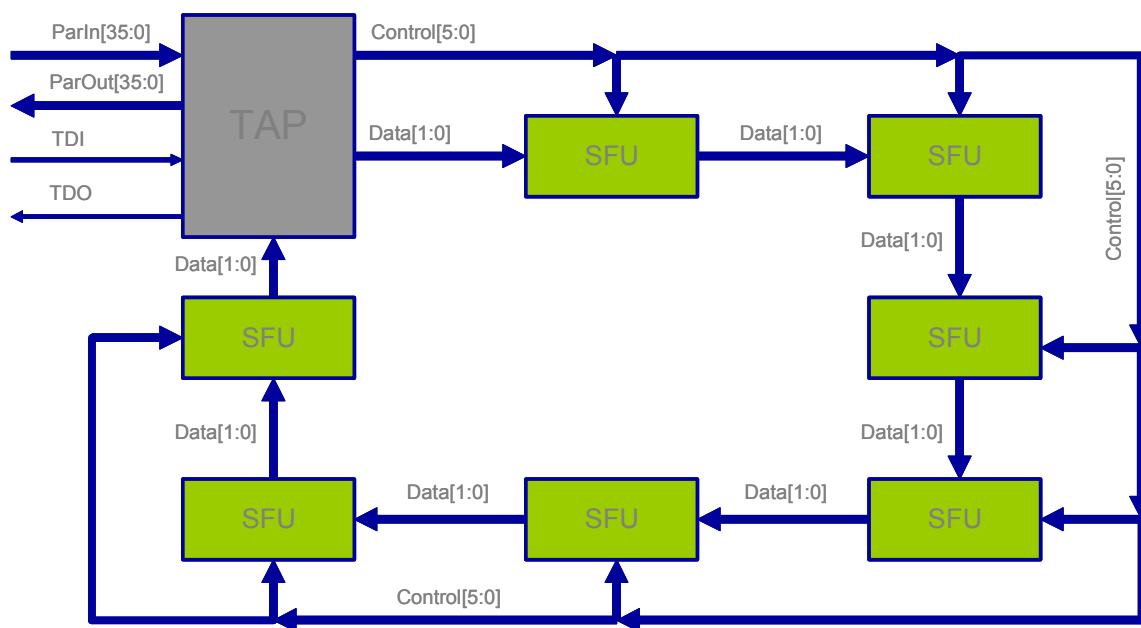

Figure 1: Full-chip scan system

DESIGN AND TEST CHALLENGES

Intel's move to full hold-scan in this design generation was motivated fundamentally by the need to continue accelerating new product development. Historically, Intel has used custom Design For Test (DFT) features and post-silicon test-writing to create high-quality production test suites for lead microprocessor designs [1]. Past Intel® processor design teams have consumed upwards of 50 person-years to develop the production test suite for a new design. The significant post-silicon effort of this approach was justified by the product cost savings that could be realized through saving the die area and timing margin that more systematic DFT would have cost. This cost/benefit trade-off has remained true for many generations of Intel Pentium® processor design, as driven primarily by the economics of extremely HVM.

With every design generation, however, the challenge of creating a high-quality production test suite [2] in a timely fashion with reasonable resources is increased. And in every design generation, the business environment changes to even further increase the economic emphasis on time-to-market and on the need to move critical design resources to the next project as soon as possible. In the latest microprocessor design generation, all these factors

finally led us to doing extensive systematic DFT in our lead processor.

The benefits that would have to be realized to justify this move to full hold-scan were as follows:

1. Reduced test generation and fault grade effort.

2. Increased visibility into internal state, for post-silicon debug and fault isolation productivity.

3. Improved initialization for maximally effective functional signature testing.

4. High burn-in toggle capability in a limited pin-count test environment.

5. Significantly accelerated ramp to HVM, enabling production test quality.

Automated DFT provides the promise of quick, reliable test generation for a new product and all of its design proliferations, quick stabilization in factory test programs during the critical debug and ramp phase, and ultimately significant factory cost savings because of a much earlier reduction in system-testing to ensure outgoing quality levels.

Reduced debug time is a fundamental goal in itself. It reduces the overall time-to-market of a product and also accelerates the pace with which design modifications can be turned for debug onion-peeling. The standard interface and massive observability of a scan system is understood industry-wide for its many benefits to debug.

Initialization is also a key engineering challenge to rapidly achieving high test quality, particularly for a signature-based functional test of a design. The inclusion of a full hold-scan system and minimal extra DFT provides a much easier path to massive system initialization in advance of a deterministic test–both for functional signature testing and for Automatic Test Pattern Generation (ATPG).

Finally, maximizing toggle coverage (circuit excitation) during burn-in on a low pin-count test board can be achieved through the use of full hold-scan.

All of the above led to a capability to accelerate the ramp from initial tape-out of a new product to high-volume, high-quality stable manufacturing, and also to maximizing the capability of the manufactured product.

Even given the expected test benefits, it was still critical to pay attention to the historical design challenges that had worked against choosing full hold-scan in the past. To ensure that scan achieved the right economic trade-off for the product, the following design goals also had to be met:

1. Integrate debug and test capabilities into a single cost-optimized scan system.

2. Minimize scan impact to die area, power, timing, and design schedule.

3. Minimize reliability impact due to contention under test.

4. Keep the overall system, library requirements, and design rules as simple as possible, so as to minimize validation complexity and maximize achievable coverage benefit.

These design requirements led to the scan system described in this paper. In particular, requirements 3 and 4 led to the specific full hold-scan system that was adopted. Specifically, we tried to minimize variation in our scan system design, so as to streamline the effort for all peripheral teams involved in getting full hold-scan into the Intel IA-32 microprocessor design environment for the first time, and to maximize the chances of taping out a fully bug-free scan system into first silicon. Other technical requirements that shaped the system we adopted included the need for compatibility with the structural testers [3][4] in use in our high-volume test floors, enabling efficient parallel access for HVM test time minimization, providing serial access for burn-in and system debug, providing hooks for making easy enabling/disabling changes to manufacturing test programs, and providing extensive signature capability for maximum test quality from our extensive legacy functional test base used for at-speed testing.

SCAN SYSTEM ARCHITECTURE

Scan Functional Unit Features

The full hold-scan system comprises 29 SFUs distributed across the different functional blocks of the microprocessor. Each SFU is an intermediate control station in the scan system that can be configured to access different scan chains in the corresponding partition. SFUs are configured using address scan registers that exist in each SFU and get connected as a serial shift chain in scan address mode (Figure 2). Each SFU provides access to 5 reserve and 3 custom chains as shown in Figure. 3. Custom chains are available for additional future visibility. Reserve chains are used for various scan usage modes (which are mentioned later in this paper). We do not, however, present details of reserve chain usages in this paper.
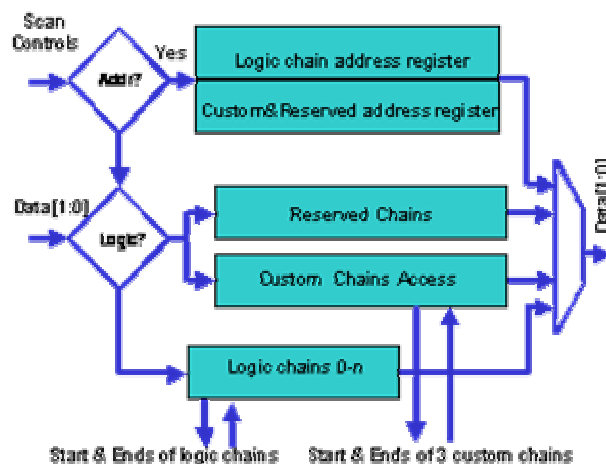

Figure 2: Scan functional unit block diagram

Logic Scan Chains

There are ~200k scanned states on the die. The scan system for 15% of these states operates at full-core frequency. These full-core frequency scanned states are consolidated onto 2 of the sub-chains within each SFU. These chains are referred to as "fast" chains (total fast chains = 70). The remaining 16 sub-chains in each SFU operate at (up to) one-half of the core frequency. These SFU sub-chains are referred to as "slow" chains. All 18 chains in each SFU share the same clocking network. Originally all chains were designed to run at the full core frequency. This approach provided additional benefits such as running "transition fault" tests at the core frequency. However the scan logic on 16 of the chains in each SFU was re-designed specifically to save power. The two "fast" chains in each SFU were preserved for

usage in the full frequency modes (snapshot and signature modes).

TAP Controller and Full-Chip Scan Bus

The Test Access Port (TAP) controller converts data between the TAP clock frequency domain at the pins and internal core clock domain at the full-chip scan bus interface. The scan system in the TAP operates in either serial or parallel mode. In parallel mode, the TAP communicates with an Automated Test Equipment (ATE) tester using 36 input and 36 output pins. In serial mode, the TAP communicates with the ATE tester using TDI and TDO. The TAP always shifts data into the two scan bus data bits simultaneously. The full-chip scan bus is comprised of two bits of data and five bits of control signals. The two bits of data serially connect each of the SFUs while the 5 bits of control signals are routed to each of the SFUs as shown in Figure 4 in a balanced tree with equal latency. The scan bus provides a non-intrusive operation while the system is running or while the system clocks are frozen. In parallel mode, the tester sends out data (36 bits every TAP clock cycle) on the input bus. Eighteen of the inputs received by the TAP are shifted into one of the data bits in the scan system bus while the other 18 inputs are shifted into the other data bit of the scan system bus.
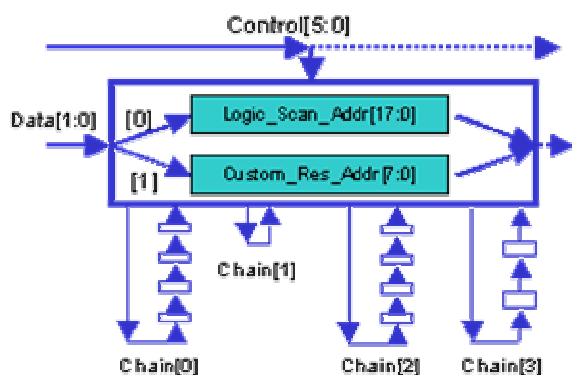

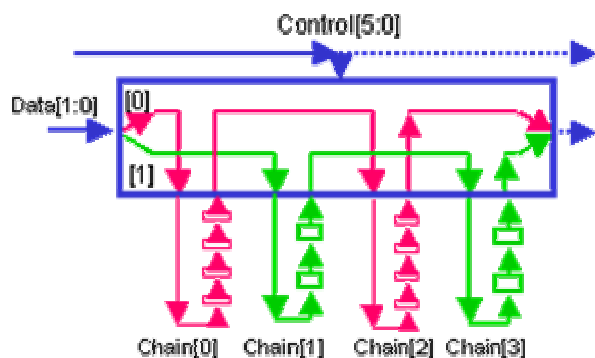**Figure 3: Addressing inside the SFU**

Figure 4: Logic chain connections inside the SFU

HIGH-LEVEL OPERATIONS AND MODES

There are two primary arenas in which the scan system is utilized. HVM, and Silicon Debug (1st Silicon Debug, System Debug, Low Yield Analysis Debug (LYA)).

Snapshot for 1st Si System and LYA Debug

In both the tester and the system debug environments, system data are captured in the scan states "on the fly" while the system clock is running; this is called a "*scan snapshot.*" These data are then shifted along the scan chains and out of the chip in an operation called a "*scan dump.*" At the chip interface, the captured states can be serially shifted out on TDO pins or converted into parallel data and sent out on external data and address pins.

During system debug, the architectural state of the machine is dumped periodically using the Periodic System Management Interrupt mechanism (details are not included in this paper). When the system debugger would like to know the "scannable" system state, he or she can also perform periodic scan dumps. Consequently, the scan dump operation must be non-destructive (not change the state of the machine), and it must be possible to continue normal system operation during and following the scan dump.

A common use of scan snapshot on an Automated Test Equipment (ATE) tester is to run a given test, perform a scan snapshot at cycle N and dump the scan data, re-start the test, do a scan snapshot at cycle N+1 and dump the scan data. Continue in this fashion until K cycles of snapshot data have been collected.

Snapshot Compression for 1st Si Debug

For repeatable errors, data are dumped and compared several times to find when data goes from good to bad. To reduce the amount of data that need to be collected, scan chain data are compressed on-chip and only a "signature" is dumped out. The scan chain ("fast" chains only) outputs are fed through Linear Feedback Shift Registers (LFSR) in each SFU and finally compressed further by another LFSR in the TAP to produce the signature. About 35k bits of snapshot data are compressed down to 32 bits.

On-Die Snapshot Diff for 1st Si Debug

Snapshot compression is more useful as part of a technique called "*on-die snapshot diff.*" Often the debugger is only looking for differences in the data between a given test run and a known good run, thus the snapshot data are compressed on-chip. The known good signature can also be stored on-chip. The signatures produced by subsequent test runs can be compared with

the golden signature on-chip, with only a pass-fail bit shifted out of the chip.

Signature Mode for Functional Tests in HVM

The shortcoming of the previous usage models is that, to detect an error, the debugger must first find a point in time during the test run where the error appears when a snapshot was taken. "*Signature mode*" solves this problem by compressing scan snapshot data taken every clock cycle over a period of time. In addition to using the LFSRs mentioned above to compress the scan data spatially, the individual scan cells can compress the scan data temporally. For example, instead of overwriting an old bit of snapshot data, scan cells can XOR the old data with new bits of snapshot data.

Signature mode is primarily useful in the production test environment; especially while running hand-written architectural tests. Instead of having to understand the full-chip architecture to propagate a fault effect all the way to a chip pin; the test writer's job is greatly simplified: he or she only needs to figure out how to propagate the effect to a signature scan node.

Automatic Test Pattern Generation for Various Faults Types in High-Volume Manufacturing

The previous usage models only require the scan cell to be able to observe a system state, which is reached by running an architectural test. Scan is a much more powerful feature when it is used to control, or modify, the system state. Then, a stimulus pattern can be shifted into the scan chain and applied to the system logic. The response pattern can then be shifted out of the scan chain and compared with an expected response obtained through simulation. An ATPG tool determines the stimulus and response patterns.

At the time of this design a microprocessor-sized netlist was too large for an ATPG tool to handle all at once, thus the design was partitioned into multiple clock domains. Scan tests were generated and applied to one domain at a time. The simplest type of fault for which to test using ATPG is the stuck-at fault.

Toggle Coverage in Burn-in Mode in High-Volume Manufacturing

During burn-in testing, the scan system is used similarly to how it is used in ATPG mode, except that the serial interface to the scan system (via only the TAP port) is used instead. Content loaded into the design for burn-in testing can be either pseudo-random or generated by the ATPG tools.

Device Initialization in Test Modes in High-Volume Manufacturing

The scan chains are also used for device initialization in test modes. In this usage case, the scan system is loaded with all 0s during reset pin assertion, and then a scan store operation is executed to initialize the state of 200k non-array sequential elements in the design to 0. This is done early in the microprocessor reset sequence.

FULL SCAN DESIGN EXECUTION

Floor-Planning

The full hold-scan execution effort started with a detailed floor-planning exercise to accommodate the 200K scanned states on the chip. A detailed scan device area estimation helped partition the full-chip functional block boundaries as shown in Figure 5. We identified key congestion areas to determine the critical cut of the die and help shape the boundaries. Regular audits of bottom-up area estimates were used to alter floor-plan assumptions. The clock and control signal distribution networks account for ~35% of the scan device area and hence required congestion studies. The control signals were routed in a balanced tree network to the different SFUs and scan elements to optimize the signal tracks.


**Figure 5: Full-chip floor plan**

Library Development

The library cells are the fundamental building blocks of the full hold-scan system. We designed independent scan libraries to cater to the two distinct design methodologies on the die: Cell-Based Design (CBD) and Embedded Building Blocks (EBB). The library cells are designed independently and pre-characterized to describe their

behavior for timing, logic, noise, reliability, etc. The microprocessor uses a wide variety of latches and flip-flop circuits to implement the logic functionality. The scan system now requires scanned variants of every latch and flip-flop on the die. A scanned latch shown in Figure 6 comprises two distinct circuits: a system latch and a scan gadget. The system latch is the pristine storage element catering to the system functionality needs, while the scan gadget is comprised of a fixed size storage element and interface circuits to meet communication needs (shift, capture, store) as dictated by the scan architecture. The interface circuits are sized in relation to the system latch.
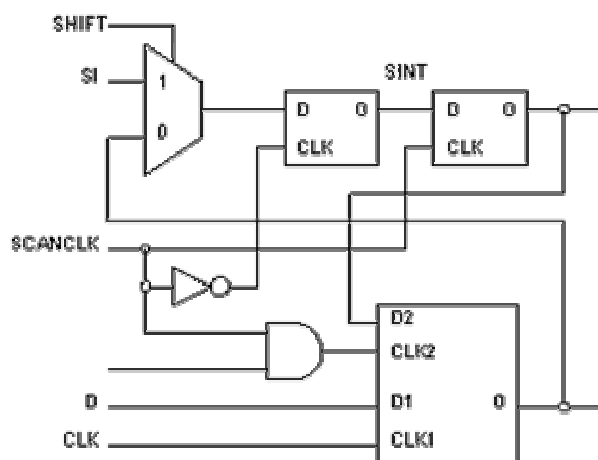
Figure 6: System latch with scan gadget cell
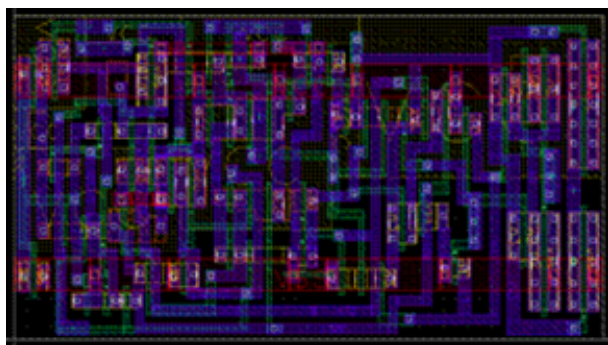
Figure 7: System latch with fast scan gadget layout

In a full hold-scan design, the large usage numbers of scanned state elements (~200K) impose a high standard on the scan gadget design. Individual scan gadgets have been designed for use in fast and slow chains. The fast gadget shown in Figure 7 needs to support shift, snapshot and store operations at full-speed, while the slow one needs to operate the same functions at half core-speed. This not only provides an opportunity to size down the devices but also allows 100% use of low-leakage devices in slow gadgets. Special care was taken to reduce the amount of clock switching capacitance, a source of dynamic power. Library cells were characterized first at

the cell level to meet all scan timing requirements. Scan cell timings were designed with an additional 5-10% margin to guarantee that process variation does not push the scan circuits to be the speed-limiting paths on the die.

Scan Design Flows and Methodologies

There are two unique types of design methodologies used on the microprocessor: EBB and CBD [5]. The physical break-up of the two types of blocks is approximately equal on the chip. While EBBs are hand-crafted by individual designers, the CBD blocks use automated techniques to implement the design. Although the implementation flows of the two types of blocks are vastly different, there is little difference in the logic model. The design has a small fraction of blocks that are deviant from the CBD methodology called Structured Data Path (SDP) blocks. SDP blocks need hand-drawn schematics and some manual intervention to help in the automation of the convergence of the design.

The logic representation of a microprocessor design undergoes significant early changes during the functionality definition and logic convergence phases of the design. To enable validation of the logic and downstream steps of a microprocessor design flow, snapshots of the logic model are released on a regular basis. To enable such rapid changes in succession, it is important to keep the scan-related steps and collateral minimal while still enabling validation of scan features in the model. This requirement for a light-weight process drove the scan flow strategy. The key steps involved are scan selection, insertion, ordering and stitching, clock tree synthesis, and rules and checkers.

Scan Selection

The scan selection step involves marking the correct states to be scanned. This step is required even in a full hold-scan methodology to enable exceptions and to avoid scanning transparent latches, states supporting the scan system, and other test or debug features.

Scan sites were selected in the logic using a scan selection tool *scansel*. The *scansel* needs to not only identify transparency but also recognize the clock phase of the different states when the global clock of the microprocessor is shut off. This is a key requirement to guarantee functionality of the different modes of the scan system. To enable appropriate accounting, all states in the logic microprocessor design were divided into three classes: *scan, do_not_scan & scan_exception*. Examples of states that fall into *do_not_scan* are transparent latches, states supporting other test/debug features, etc. Examples of states that are classified as *scan_exception* are states in blocks that are exempt from full scan requirements. All blocks in the design fall into two full scan categories: a) full scan targeted or b) full scan exempt.

Scan Insertion

The scan insertion step involves incorporating scan by replacing the non-scan state elements with scan equivalent state elements and connecting the scan control ports of the scan cell to the scan control signal distribution network. In EBBs this was a manual operation while in CBD blocks, automation manipulated the netlist representation of the design. For semi-custom (SDP) and synthesized (CBD) blocks, scan insertion was automated through the use of API functions available within the logic synthesis tool *Design Compiler*[*].

Scan Ordering and Stitching

The scan ordering step involves connecting the scan-out port of a scan cell to the scan-in port of the next scan cell to form a serial chain. This enables a scan ordering that is optimal in terms of usage of chip area used to route the scan chain. This functionality was implemented using features available with the place and route tool *Apollo*[*]. The scan chain ordering flow was constrained to meet slope rules for all scan signals and full functional timing goals on select scan chain paths. To meet these requirements, large repeater networks were designed to enable distribution of scan signals. Additionally, these requirements had to be met within tight block area constraints.

Scan Clock Tree Synthesis

The scan states on the die use generic core clocks to avoid additional global clock distribution costs. Although the global clock grid is shared between system and scan, the scan local clock buffers have been specifically optimized for power and area constraints. There exists a special local scan clock network delay and slope tuned to meet scan functionality while limiting degradation to system performance. The scan clock network below the global clock grid consists of a Regional Clock Buffer (RCB) and a Local Clock Buffer (LCB). The RCB is a scan-function-enabled driver that services a small number of LCBs while meeting fan-out guidelines. The scan LCBs are un-gated buffers driving ~10-15 scan states.

Scan Rules and Checkers

The scan states are present across the die in and around different types of circuit configurations. A set of rules was developed to ensure functionality and maintain consistency across the design. They primarily belong to two classes: Scan Friendly Array (SFA) and Scan Friendly Mega Block (SFMB). The former deals with scan logic interaction with memory arrays, and the latter focuses on interactions between scan elements of one logic block with another neighboring block.

---

[*] Other brands and names are the property of their respective owners.

Design Rule Checkers (DRC) are deployed to check for compliance of the design to scan rules. These scan rules support the scan ATPG strategy adopted, and compliance to these rules is critical to the success of obtaining the full scope of the test coverage benefits. The result from the DRCs was a pass/fail for a scan-targeted block.

COST AND BENEFIT ANALYSIS

Timing/Performance

System performance is critically dependent on the scan gadget architecture and operating modes. The scan gadget circuits add capacitance to the system nodes thereby causing degradation to the setup and clock-to-out times of the system. The scan gadget design is tuned to work well within its operating modes to limit this timing degradation. The scan contribution of the setup and clock-to-out penalties are summed into a new metric called "Black-hole time." A select number of families were simulated and the black-hole times are reported in Figure 8. We used a variety of circuit design techniques to reduce black-hole time penalties. Furthermore, the system timing degradation will translate into actual maximum frequency (Fmax) loss if and only if the impacted system paths are the speed-limiting paths on the manufactured chip.
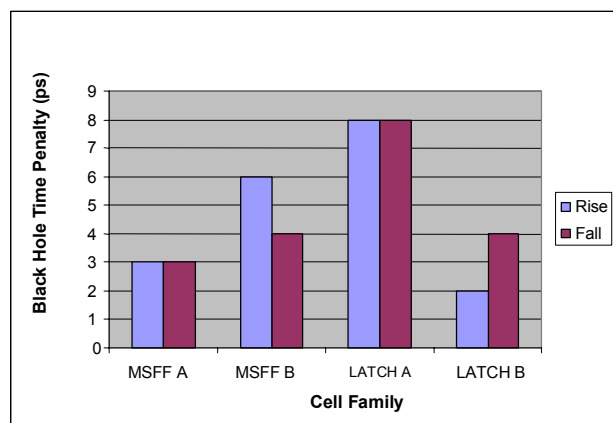
Figure 8: Black-hole time performance penalty

Die Area

The full scan system area contributions are primarily shared by the scan gadget, scan clocks, and control distribution circuits. These critical components of the scan system tend to affect the critical cut of the die. This issue can be mitigated with early floor planning and with the use of optimized place and route tools. Our scan methodology allowed for full scan exception cases in select areas to ease local device and wire congestion.

Power

Although the dynamic power is important from the standpoint of power delivery capability in test systems, the critical contribution from scan is the leakage power during normal system operation. The scan system was partitioned into fast and slow scan chains with ~15% cells in fast chains. This inherently allowed us to reduce leakage power to a minimum in slow chains. Even in the fast chains, the scan operating modes were optimized; i.e., all non-essential functions were slowed down to optimize for power. The scan gadget nodes are parked in non-toggling state, and furthermore, scan clocks do not toggle during normal operation. The control signal distributions were architected to operate at low frequency thereby saving power. Scan power has been reduced to ~2.5% of the total power for the processor.

Time to Quality and Factory Savings

The primary goal of adding full hold-scan into the design was to reduce HVM test development effort for the product, without compromising test quality. The final fault coverage for this product was achieved through a combination of scan ATPG testing, other DFT-based tests (such as directed array testing), and full-speed functional tests. The full-speed functional test base consists of significant existing "legacy" tests and many newly-written tests targeted at fault coverage enhancement. All classes of tests are graded against the current design, to

assess coverage and ensure quality. Against an overall graded single-stuck-at fault coverage goal of ~96% for this product, scan provided unique coverage of 6 percentage points above all other content in the test base, as shown in Figure 9.

This unique coverage translated directly to saving significant manual test-writing effort in the "last mile" (the most labor-intensive portion) of test development, that would have otherwise had to be done. Recent Intel test development experience (from both the current high-volume 130nm design and the new 90nm Intel® Pentium® 4 microprocessor) has indicated that "last mile" manual test development costs on the order of 5 person-years effort per percentage point of full-chip coverage. Hence, full hold-scan in this design directly saved ~30 person-years of test development for HVM. Practically speaking, this can alternately be stated as a 6-month pull-in from test development schedule to quality for the product, which is a significant fraction of our typical timeline from 1st silicon through debug to product introduction.

Apart from direct engineering cost savings, the reduction in effort has an even more significant benefit in the factory: reduction in time-to-quality leads to earlier detection of real testing issues (as opposed to start-up issues), much better predictability of the test program ramp to stability, and much quicker cost-reduction of expensive screening tests, which are always employed in early testing of a new product. This alone has been estimated to represent large savings in factory test cost, in the early testing ramp for this 90nm-generation microprocessor.
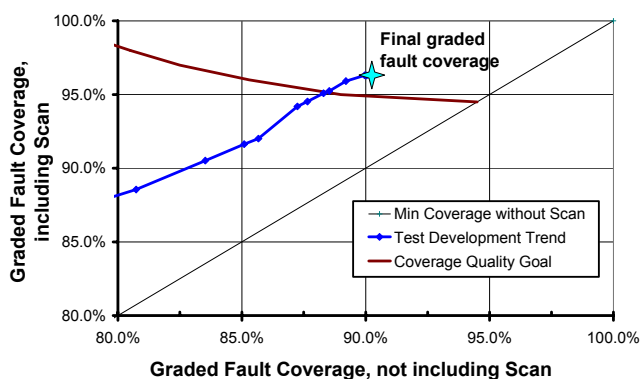

Figure 9: Fault coverage goals

Yield Learning and Quality Improvement

The extent of scan in this design also provides a powerful diagnostic technique for failure analysis of bad parts in the early days of product ramp. This has benefits both for yield learning on a new process, and for test program

improvement through early and quick identification of test holes in the random logic of the die. For many product generations, Intel has had industry-leading array DFT and techniques for failure analysis and fault isolation in array logic, but the same process for random logic has been highly customized and laborious. With the advent of full hold-scan, we bring the fault analysis time and cost in random logic down from weeks to hours.

Historically for Intel, the potential test-writing effort savings possible through full hold-scan was swamped by the die area cost in the volumes of product that we run. The potential debug and manufacturing benefits of full hold-scan were well understood, but existing methods were mature, productive, and cost-effective enough to keep HVM test development well off the critical path for new product ramp. The pressure that has been mounting, and what changed specifically with this processor generation, is the need for even quicker design cycles, and historically unprecedented manufacturing volume shift to each next-generation product. These factors shifted the economic return on investment for full hold-scan from negative to positive for this generation of product. We have taken significant cost to move our design processes to full-scan, but we are in turn reaping the benefit in the many debug and manufacturing ramp capabilities that this provides.

Additional significant benefits of a full hold-scan system are widespread initialization during the production test reset sequence, significant internal visibility into the design for debug, and burn-in toggle coverage capability. One thing that is uniquely beneficial to Intel's full hold-scan design is the ability to shift state into the design without causing any contention in the design. This is used in production test for design initialization prior to functional signature mode or ATPG testing, and in our burn-in environment for pseudo-random toggle coverage maximization without risking chip damage. The second capability, which is unique to our full hold-scan design, is the debug ability to capture almost full internal state in either the tester- or system-based debug environments, while running tests at full speed. This opens a new door for observability that is now being leveraged as we continue to develop debug capabilities around this test feature.
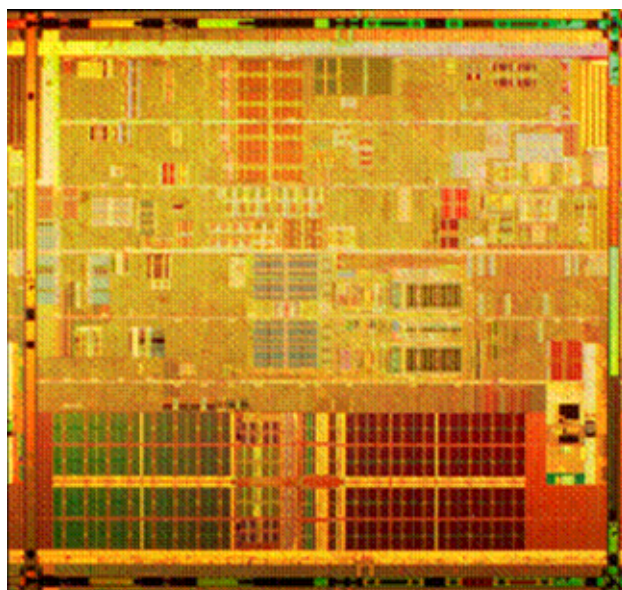
Figure 10: 90nm Intel® Pentium® 4 microprocessor die photo

CONCLUSION

Shorter product cycles are driving the need for a reduced test generation and fault-grading effort. There is an imminent need for a quick, reliable, test content and methodology that is capable of being proliferated to future generations of microprocessor design. Our full hold-scan system not only provides a solution to this problem but also helps reduce logic and speed-path debug time. One needs to pay close attention to the die area, power, and system performance impact to harness all the benefits of this new full hold-scan system. In summary, Intel's first low-power performance-friendly full hold-scan CPU has been designed and integrated into the Intel 90nm Pentium 4 microprocessor shown in Figure 10. This scan system will not only help decrease the time-to-market of this product but also set the standard for all our future generations of high-speed microprocessors.

ACKNOWLEDGMENTS

REFERENCES

[1] Carbine, A. and Feltham, D., "Pentium® Pro Processor Design for Test and Debug," in *Proceedings of the 1997 IEEE International Test Conference*, 1-6 Nov. 1997, pp. 294-303.

[2] Sengupta, S. et. al., "Defect-Based Test: A Key Enabler for Successful Migration to Structural Test," in *Intel Technology Journal*, Q1 1999.

[3] Mayberry, M.; Johnson, J.; Shahriari, N., and Tripp, M., "Realizing the benefits of structural test for Intel microprocessors," in *Proceedings of the 2002 IEEE International Test Conference*, 7-10 Oct. 2002, pp. 456-463.

[4] Tripp, M., "On-die DFT based solutions are sufficient for testing multi-GHz interfaces in manufacturing (and are also key to enabling lower cost ATE platforms)," in *Proceedings of the 2002 IEEE International Test Conference*, 7-10 Oct. 2002, p. 1232.

[5] Rich, S.E., Parker, M.J. and Schwartz, J., "Reducing the frequency gap between ASIC and custom designs: a custom perspective," in *Proceedings of 2001 Design Automation Conference*, 18-22 June 2001, pp. 432-437.

AUTHORS' BIOGRAPHIES

Ravishankar Kuppuswamy is a design manager in Intel's Technology and Manufacturing Group, specializing in mixed signal design and Design for Test (DFT) areas of high-speed microprocessors. Ravi received both a B.S. degree in Electrical Engineering and a M.S. degree in Chemistry from the Birla Institute of Technology and Science, Pilani, India in 1994 and a M.S. degree in Electrical Engineering from Arizona State University in 1996. He has worked on a large number of product areas including IO, clock generation, clock distribution, and features specifically targeted for High-Volume Manufacturing (HVM). Ravi's e-mail is ravishankar.kuppuswamy at intel.com.

Peter DesRosier: is a design manager in Intel's Technology and Manufacturing Group, specializing in DFT features and architectures and production test methods for high-speed microprocessors. Peter received his B.S. degree in Electrical Engineering from Montana State University in 1985. Peter's e-mail is peter.desrosier at intel.com.

Derek Feltham is a principal engineer in Intel's Desktop Platforms Group, specializing in DFT and production test methods. His technical interests include DFT features, test development productivity, and test quality prediction methods. Derek received his B.S. and M.S. degrees in Applied Science from the University of Toronto and a Ph.D. degree from Carnegie Mellon in 1992. Derek's e-mail is derek.feltham at intel.com.

Rehan Sheikh is a senior staff engineer in Intel's Desktop Platforms Group, specializing in DFT and HVM methods. His technical interests include DFT feature design, validation, initial Si debug, and advanced test methods. Rehan received his Masters of Electrical Engineering from Georgia Tech in 1994. Rehan's e-mail is rehan.sheikh at intel.com.

Paul Thadikaran currently leads a DFT Tools team in Intel's Desktop Platforms Group focused on developing DFT tools and solutions for IA-32 CPU designs. Paul received his Ph.D. in Computer Science from SUNY-Buffalo in 1997. His technical interests are CAD tools for DFT and Test generation and CAD algorithms for design optimizations. His e-mail is paul.thadikaran at intel.com

---

® Pentium is a registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

For further information visit:

developer.intel.com/technology/itj/index.htm