



Intel[®] Technology Journal

Intel[®] Pentium[®] 4 Processor on 90nm Technology

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

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

[®] Intel 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

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.zip, 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

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

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

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.

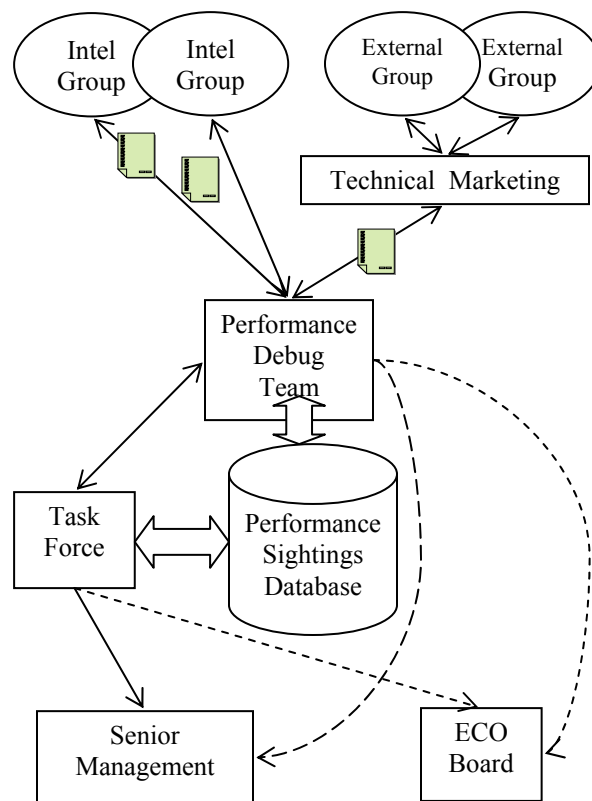


Figure 1: Performance sighting life cycle

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

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.

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

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.

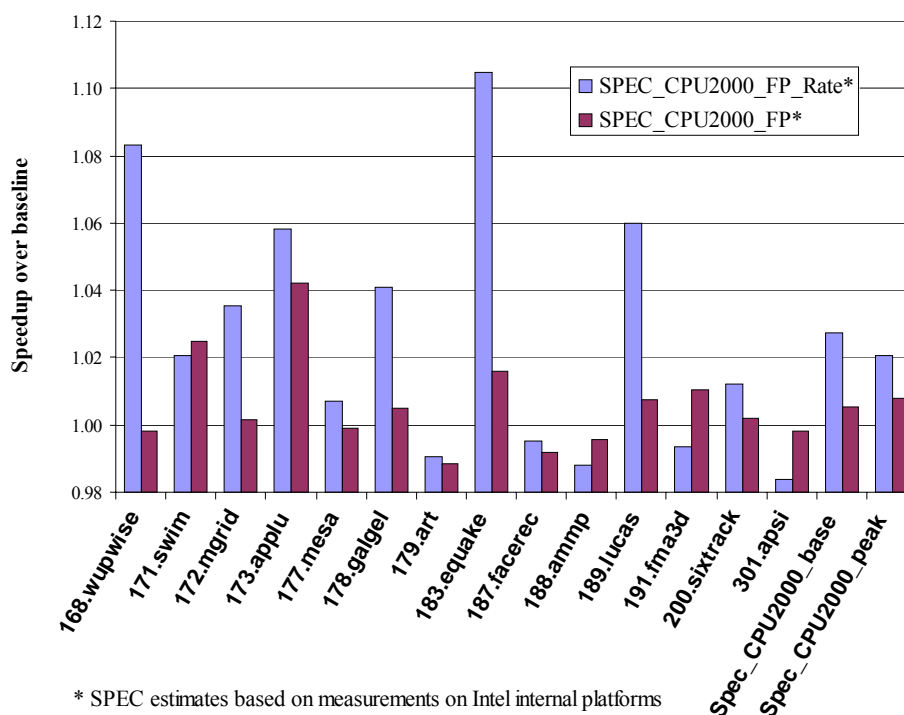


Figure 2: Results obtained via performance feature tuning experiments on SPEC*_CPU2000_FP and SPEC_CPU2000_FP_rate

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

The work described in this paper is due to the efforts of many members of the Intel Pentium 4 processor performance team over a multi-year period, all of whom deserve credit for the successful performance validation of the Pentium 4 processor. The team was co-managed by Deborah Marr, Michael Upton (Desktop Platforms Group) and Gustavo Espinosa (Technology and Manufacturing Group).

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@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.

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

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..

Copyright © Intel Corporation 2004. This publication was downloaded from <http://developer.intel.com/>.

Legal notices at <http://www.intel.com/sites/corporate/tradmarx.htm>.

[®] 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.

For further information visit:

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