Desktop Performance and Optimization for Intel® Pentium® 4 Processor

Feb. 2001

Order number: 249438-01

About this Document

This paper describes the performance philosophy of the Intel® Pentium® 4 processor. It also describes software optimization techniques and tools to achieve leading-edge performance on current and future generations of the IA-32 high-performance processors. The information on performance results, tools and techniques for software optimization will enable managers, architects and engineers to deliver industry-leading software performance.

Table of Contents

1
Intel® Pentium® 4 Processor Performance Philosophy

1.0 Performance Goals of Pentium® 4 Processor

Throughout the history of Intel IA-32 processors, the early life cycle of each micro-architecture generation delivered a large performance gain over time. However, as the micro-architectural design matures, the performance gain starts to diminish, and a new micro-architecture is required to maintain the performance trajectory expected by the marketplace. The Intel® NetBurst™ micro-architecture is the latest micro-architecture from Intel that implements the IA-32 architecture. The Intel NetBurst micro-architecture, along with several extensions to the IA-32 architecture, allows the Pentium® 4 processor to deliver the next-generation performance needed to enhance the experience of PC and workstation users for multimedia and internet applications. This new micro-architecture enables performance to scale efficiently to high frequencies, and it is the foundation for future IA-32 processors to deliver industry leading performance for the next several years.

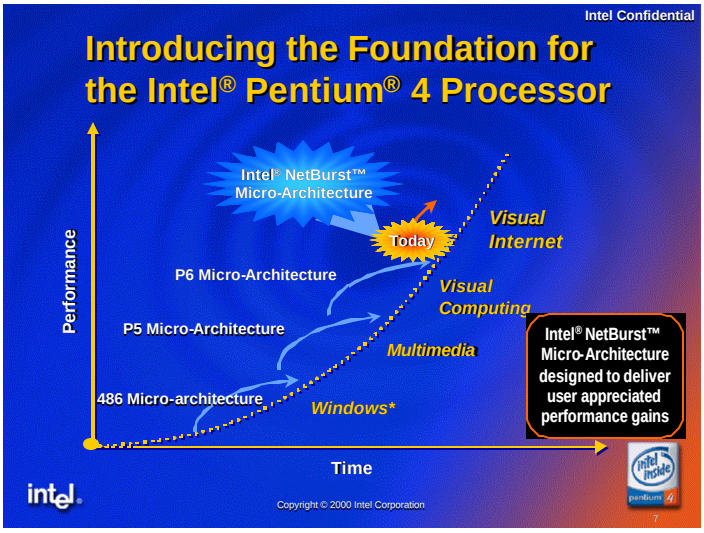

Figure 1. The evolution of micro-architectures of Intel® processors. Each micro-architecture delivers larger performance gain in early part of its life cycle, and matures at a significantly higher performance level than previous micro-architectures.

Designed for Performance

A focused architectural pre-design effort was undertaken to assess the benefits of many advanced processor technologies and to determine the best approach to improve the overall performance of the processor for many years to come. The result of the architectural effort was the implementation of a design that significantly increased frequency capabilities to well above 40% higher than that of the micro-architecture of the Pentium III processor, known as P6 micro-architecture, on the same manufacturing process. At the same time, this design effort focused on delivering an average instruction executed per clock (IPC) that was within approximately 10% to 20% of the P6 micro-architecture. The design effort focused on the following:

- Hyper-pipelining to enable higher processor frequencies
- Keeping the high-frequency execution units busy by increasing the system bandwidth and overlapping computations with memory accesses
- Enhanced out-of-order execution engine capable of finding more instructions to execute with deeper out-of-order resources (three times the number of instructions in-flight than with a Pentium III processor)
- Reducing the number of instructions needed to complete a task or program.

The result of this design effort is a brand new micro-architecture that delivers significantly higher levels of performance and frequency, and provides frequency head room for future IA-32 processor in the next several years. The design innovations of the Intel NetBurst micro-architecture is first realized in the Pentium 4 processor.

1.1 Intel® NetBurst™ Micro-Architecture

The Pentium 4 processor, utilizing the Intel NetBurst micro-architecture, is a complete processor redesign that delivers new technologies and capabilities while advancing many of the innovative features, such as "out-of-order speculative execution" and "super-scalar execution," introduced on prior Intel® micro-architecture generations. Many of these innovations and advances were made possible with the improvements in processor technology, transistor technology and circuit design, and they could not have been implemented previously in high-volume, manufacturable solutions. The new technologies and innovative features that are introduced in the Intel NetBurst micro-architecture are listed below:

Hyper-Pipelined Technology: The hyper-pipelined technology of the NetBurst micro-architecture doubles the pipeline depth, compared to the P6 micro-architecture, with a 20-stage pipeline. This technology significantly increases processor performance and frequency scalability of the base micro-architecture.

400-MHz System Bus: Through a physical signaling scheme of quad pumping the data transfers over a 100-MHz clocked system bus and a buffering scheme allowing for sustained 400-MHz data transfers, the Pentium 4 processor supports the industry's highest performance desktop system bus delivering a data rate of 3.2 Giga-Bytes per second (GB/s) in and out of the processor. This compares to 1.06 GB/s delivered on the Pentium III processor's 133-MHz system bus.

Advanced Dynamic Execution: The Advanced Dynamic Execution engine is a very deep, out-of-order speculative execution engine that keeps the execution units busy. It does so by providing a very large window of instructions from which the execution units can choose in order to get around stalls due to instructions that are not ready to execute based on some unmet dependency (such as waiting for data to be loaded from main memory). The NetBurst micro-architecture can have up to 126 instructions in this window (in flight) versus the P6 micro-architecture's much smaller window of 42 instructions.

The Advanced Dynamic Execution engine also delivers an enhanced branch prediction capability that allows the processor to be more accurate in predicting program branches and has the net effect of reducing the number of branch mispredictions by about 33% over the P6 micro-architecture's branch prediction capability. It does this by implementing a 4 Kilo Bytes (KB) branch target buffer in which to store more detail on the history of past branches as well as implementing a more advanced branch prediction algorithm. This enhanced branch prediction capability is one of the key design elements that helps to reduce the overall sensitivity to branch misprediction penalty of the NetBurst micro-architecture.

Rapid Execution Engine: Through a combination of architectural, physical and circuit designs, the Arithmetic Logic Units (ALUs) within the processor run at two times the frequency of the processor core. This allows the ALUs to execute certain instructions in ½ a core clock and results in higher execution throughput as well as reduced latency of execution.

Advanced Transfer Cache: The level 2 Advanced Transfer Cache is 256KB in size and delivers a much higher data throughput channel between the level 2 cache and the processor core. The Advanced Transfer Cache consists of a 256-bit (32-byte) interface that transfers data on each core clock. As a result, a 1.5-GHz Pentium 4 processor could deliver a data transfer rate of 48GB/s (32 bytes x 1 (data transfer per clock) x 1.5 GHz = 48GB/s). This compares to a transfer rate of 16GB/s on the Pentium III processor 1 GHz and contributes to the processor's ability to keep the high-frequency execution units busy executing instructions instead of sitting idle.

Execution Trace Cache: The Execution Trace Cache is an innovative way to implement a 1st level instruction cache. It caches decoded IA-32 instructions (or micro-ops), thus removing the latency associated with the instruction decoder from the main execution loops. In addition, the Execution Trace Cache stores these micro-ops in the path of program execution flow, where the results of branches in the code are integrated into the same cache line. This increases the instruction flow from the cache and makes better use of the overall cache storage space (12K micro-ops) since the cache no longer stores instructions that are branched over and never executed. The net result is a means to deliver a high volume of instructions to the processor's execution units and a reduction in the overall time required to recover from branches that have been mispredicted.

Streaming SIMD Extensions 2 (SSE2): With the introduction of the SSE2 extensions, the NetBurst micro-architecture now extends the SIMD capabilities of Intel® MMX™ technology and the SSE extensions by

adding 144 new instructions that perform 128-bit SIMD integer arithmetic operations and 128-bit SIMD double-precision floating-point (FP) operations. These new instructions provide programmers with new abilities to execute a particular program task on Pentium 4 processors with fewer instructions and in less time. As a result using SSE2 extension can contribute significantly to an overall performance increase.

In addition, the Pentium 4 processor has implemented a **Hardware Prefetcher**: The automatic hardware prefetcher operates transparently without requiring programmer's active intervention. It is triggered by regular access patterns and helps predict future accesses, thereby overlapping memory latency with computation. By enabling concurrency between memory accesses and computation, this maximizes the computational benefit of higher Pentium 4 processor frequencies

1.2 Desktop Performance Expectations

The scalability of application performance with higher processor frequencies vary greatly across applications. This is because different applications have different requirements and are coded differently. Application code can be divided into the following categories: integer and basic office productivity applications versus floating-point and multimedia applications. The instructions executed per clock achievable by these different application categories varies greatly, and this variance is strongly affected by the number of branches that application code typically takes and the predictability of these branches. The more branches taken with lower predictability, the more opportunity to incorrectly predict the result of the branches, and hence the possibility of performing nonproductive work.

Integer and basic office productivity applications, such as word and spreadsheet processing, tend to have many branches in the code, thus reducing overall IPC capabilities. As a result, the associated branch penalties and performance on these applications does not generally scale as well with frequency and are more resistant to improvements in micro-architectural means, such as deeper pipelines. However, significantly raising the performance level on these types of applications that run in basic, non-multitasking, environments does not necessarily increase the user's experience, because the processing power required by these types of basic applications and environments tends to be satisfied by today's higher end Pentium III processors.

Floating-point and multimedia applications tend to have branches that are very predictable, and thus naturally have a higher average IPC capability. As a result, these types of applications generally scale very well with frequency and are inclined to benefit greatly from deeper pipelines. In addition, the processing power required by these applications tend to be unbounded: the more performance that is available, the better the user's experience.

The Pentium 4 processor shows immediate performance improvements across most existing software available today, with performance levels varying depending on the application category type and the extent that an application is optimized for the new micro-architecture.

An increase in frequency with previous micro-architectural generation products, such as the Pentium III processor, generally did not yield performance increases equal to the frequency increases. The exact efficiency of performance increase versus frequency (comparing a Pentium 4 processor at 1.5 GHz and a Pentium III processor at 1GHz) depends on individual application (see Figure 2), but in general you should not expect to see a 50% increase in performance with a 50% increase in frequency (i.e. 100% efficiency of converting frequency increase into performance gain in Figure 2). With a 40-50% increase in frequency, the Pentium 4 processor was designed to yield in the range of a 20% gain on integer and a 20-70% gain on floating-point/multi-media performance. (In workloads that include system-level activities, such as disk and network accesses, the performance results depend less on processor performance. Therefore, the performance scaling tends to be lower, SYSmark* 2000 is one such case.) As seen in Figure 2, the Pentium 4 processor enables not only a large increase in frequency, but also demonstrates greater efficiency in translating this frequency into performance gains, when compared to the Pentium III processor.

Over time, as more applications are optimized, either by recompilation/linking using the latest NetBurst micro-architecture–optimized compilers and libraries, or via assembler-level optimizations specifically for the micro-architecture, we will continue to see even greater levels of performance scaling when the software runs on the Pentium 4 processor.

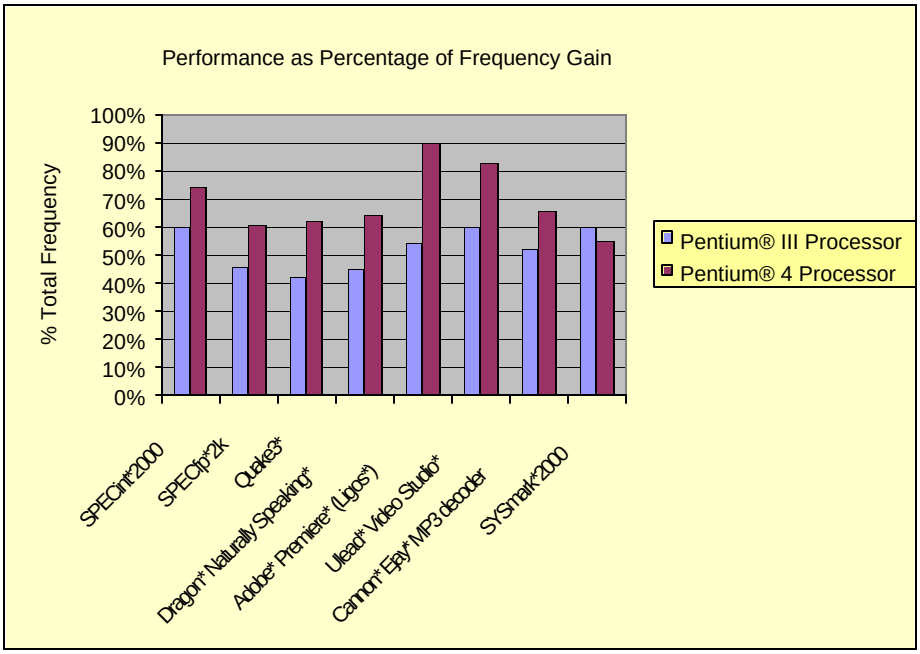

Figure 2. Scaling efficiency of application performance relative to processor frequency increase. Scaling efficiencies varies by applications, and are expected to be less than 100% due to factors such as types of code (integer versus floating-point/multimedia), degrees of braches, amount of system-level activities (disk, network, etc.). Pentium 4 processor has better scaling efficiencies than Pentium III processor across various benchmarks. Test configurations for benchmark results are listed in Section 6.

With Intel NetBurst micro-architecture as the foundation, the Pentium 4 processor delivers performance across applications and usages where end users can truly appreciate and experience the performance such as on Internet audio and streaming video, image processing, video content creation, speech, 3D, games, multi-media, and multi-tasking user environments. Section 4 of this paper shows both the performance results and the performance gains of the Pentium 4 processor on many of these applications. In addition, this paper presents multiple solutions to enable all applications to benefit from the performance capability and scalability of the Pentium 4 processor.

2 Performance Optimization Techniques for Pentium® 4 Processor

2.0 How Can My Application Benefit from Pentium® 4 Processor's Performance?

Many independent software vendors (ISVs) have software applications that deliver good performance with Pentium III processors. An important question for programmers and these ISVs is: how can my existing applications benefit from the performance potential of the Pentium 4 processor? The answer is that most applications will see immediate benefits from the higher processor clock rates and the many micro-architectural enhancements available in the Pentium 4 processor without any software optimizations. To obtain an even greater performance gain, an application may be recompiled using a compiler[1] that generates optimized code for Pentium 4 processor or linked with libraries optimized for the Pentium 4 processor. Finally, software vendors may achieve the highest performance gains by following the programming guidelines outlined in *Intel Pentium 4 Processor Optimization Reference Manual*[2] and using the SIMD integer and double-precision floating-point instructions included in the SSE2 extensions.

Another important question for ISVs is: how much performance gain can optimizing for Pentium 4 processor deliver for an ISV's application? Since Pentium 4 processors are designed to run at significantly higher frequency than Pentium III processors that are manufactured from the same process technology, a useful metric for performance gain is to compare the performance of a Pentium 4 processor running at a frequency that is 1.5 times that of a Pentium III processor. Two useful reference points for the performance gains of the Pentium 4 processor are: SPECint* 2000 and SPECfp* 2000 (collectively known as SPEC CPU 2000). The SPECint 2000 is a suite of workloads (including data compression utilities, a C compiler, a chess program, etc..) that are representative of many typical computational tasks implemented using integer code . The computational tasks selected in the SPECint 2000 workload are similar to those in a wide range of commercial applications. The SPECint 2000 workload provides a better measure of processor performance because it is not diluted by non-scaling code such as waiting for user input or input/output operations on peripheral devices. Using SPECint 2000 as a reference, integer code can expect about 1.2 times (1.2X) performance gain on a Pentium 4 processor relative to a Pentium III processor at the frequency rates described above. The SPECfp 2000 workload represents a wide range of floating-point-intensive computational tasks (including shallow water modeling, 3D graphics, neural network, computer vision, etc.). On a Pentium 4 processor, SPECfp 2000 can achieve 1.7X performance gain relative to a Pentium III processor. Other nominal floating-point and multimedia code may expect performance gains in the range of 1.3X to 1.7X, depending on the details of individual code constructs.

The SPEC CPU 2000 benchmark results illustrate the performance scaling of the Pentium 4 processor when the application code is generated by compilers that can produce optimized code for superscalar, out-of-order processors with some additional consideration for the Pentium 4 processor. Applications that have not been updated by such compilers can also benefit from re-compiling and/or linking with optimized libraries. The actual performance gain in applications will depend on many factors, ranging from the characteristics of the workload mix, degree of integer versus floating-point code, ability to identify and correct coding pitfalls, hardware and software configurations, test procedures, etc.

---

[1] Intel C/C++ and Fortran Compilers are available at http://developer.intel.com/software/products/compilers/ . Microsoft Visual Studio, Version 6, with Processor Pack is available at http://msdn.microsoft.com/vstudio/downloads/ppack/default.asp .

[2] The *Intel Pentium 4 Processor Optimization Reference Manual* is available at http://developer.intel.com/design/pentium4/manuals .

To realize the highest performance gains for a given application, this can be accomplished by applying a few Pentium 4 processor optimization techniques. Pentium 4 processor optimization techniques can be divided into groups of techniques in three broad areas: (1) the memory pipeline, (2) computational tasks, (3) efficient use of the system bus. Applying these optimization techniques to improve an application's performance is straightforward to implement. Because typically only a few specific coding techniques are needed in a given application, and re-coding is needed only over a localized extent of the source code. Furthermore, these Pentium 4 processor optimization techniques may benefit application performance on both Pentium III and Pentium 4 processors.

Sub-sections 2.1 through 2.3 summarize the key Pentium 4 processor optimization guidelines and techniques. For a comprehensive understanding of all of the Pentium 4 processor optimization recommendations, see the *Intel Pentium 4 Processor Optimization Reference Manual*. The potential performance gain of applying each of the coding techniques described in sub-sections 2.1 through 2.3 is summarized in table form in sub-section 2.4. These performance gains are approximate and meant to be used as hints for estimating potential performance improvement of un-optimized code in conjunction with the performance optimization methodology described in Section 3.

Section 3 will focus on a case study of tuning an MPEG 2 decoder application using the techniques discussed in Section 2 and applied to strategic sections of the application code. A methodology and several tools that can benefit software tuning are demonstrated in this case study. The case study consists of using performance analysis tools to identify critical code paths, and implementing localized code changes by applying key software optimization techniques.

2.1 Memory Pipeline Optimization Techniques

The Pentium 4 processor is designed to operate at frequencies well-above 1 GHz, and incorporates a high bandwidth memory sub-system (with a peak data rate of 3.2 GB/s). The Pentium 4 processor has multi-level caches that have low latency and very high data rates, so that data transfers efficiently at high speed in the memory pipeline (e.g. the Advanced Transfer Cache can support a data transfer rate of 48 GB/s at 1.5 GHz). One of the top priorities for software optimization is to prevent cases that stall the processor or the memory pipeline. The memory pipeline in Pentium 4 processor is improved over Pentium III processor by additional buffers (48 load buffers, 24 store buffers, 6 write-combine (WC) buffers, plus 4 read request buffers) to help reduce resource contention in the memory pipeline.

The most important memory pipeline optimization techniques are (in order of importance):

- Pay attention to store-to-load forwarding restrictions
- Avoid cache line splits
- Avoid alaising

The following paragraphs discuss these techniques.

Store-To-Load Forwarding Restrictions

Pentium 4 processor and P6 family processors employ a store-to-load forwarding (each load from the same address whose data had been modified by a preceding store operation) technique to enable certain memory load operations to complete without waiting for the data to be written to the cache. There are size and alignment restrictions for store-to-load forwarding cases to succeed. The store-to-load forwarding restrictions[3] are illustrated in Figure 3.

When a store-to-load forwarding restriction is not met between a store and the dependent load operation, the memory load operation is stalled. For example, when working with 32-bit packed RGBA color values the

[3] Store-to-load forwarding restrictions, code fragments and recommendations to prevent store-to-load forwarding pitfalls are discussed in more detail in Chapter 2 of the Intel Pentium 4 Processor Optimization Reference Manual, under the heading "Memory Accesses".

following coding situation is often encountered. An operation may generate a new value for the lower 8-bit component, store this value to a memory location, then read back the entire 32-bit value, and zero out the upper 3 components. This case of storing a small piece of data and loading back a larger operand that contains the stored data does not easily support store-to-load forwarding. This is because part of the data being loaded may reside anywhere in memory system (caches, DRAM). To prevent this type of stall, software could complete the processing of all four color values in a register, then write the full 32 bits to memory. To follow store-to-load forwarding restrictions, a dependent load must be loading data of the same size or less as the preceding store, and the starting address of the load and store must be the same.

Improving the memory access patterns of an application to observe store-to-load forwarding restrictions will deliver significant performance improvement on Pentium 4 processors as well as Pentium III processors.

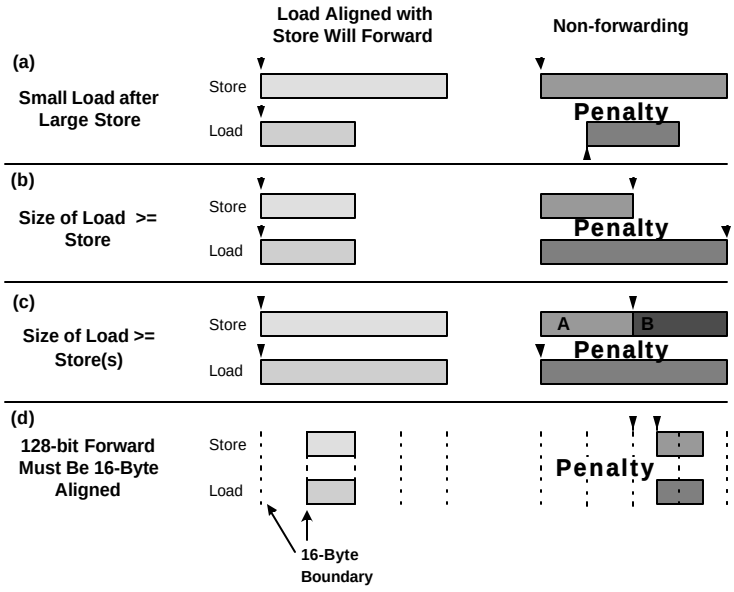

Figure 3. Alignment and data size restrictions on store-to-load forwarding cases. Four store-to-load forwarding situations on the left side meet both alignment and data size restrictions. Four store-to-load forwarding situations on the right side do not meet either alignment or data size restrictions.

Cache Line Splits

The cache line size is 64 bytes on a Pentium 4 processor, which is twice as large as the cache line size on Pentium III processor. To ensure that each cache line is used with maximum efficiency, each data access should not span across a 64 byte boundary on the Pentium 4 processor. This prevents a type of stall, known as a cache line split, from happening. When the address of a memory access does cross a 64-byte boundary, special processing is required and this can incur a performance penalty. A store operation crossing a 64-byte boundary incurs a larger penalty than a split load operation. The net result of each cache line split is inefficient use of high-bandwidth caches.

To prevent cache line splits and improve memory access performance, align data such that addresses lie on natural operand size boundary. This technique to prevent cache line splits can be applied to both Pentium 4 and Pentium III processors, e.g. data structures should be aligned to 16-byte or 32-byte boundaries.

Aliasing

Another coding pitfall occurs when the addresses of two memory accesses are aliased. If the linear addresses are aliased in the lower 16 bits (64K boundary), a stall can occur due to resource contention. Other aliasing situations (32K, 16K, or 2K boundary[4]) may also occur if the number of aliased memory accesses are greater than the number of ways available in the caches (4 ways in the 1st

```
If ((PtrA & 0xFFFF) == (PtrB & 0xFFFF)) {
PtrB += 0x1000;
}
```

Example 1. Pseudo code to offset two aliased pointers. Pointer addresses should respect alignment requirements.

[4] Aliasing conditions and recommendations to prevent them are discussed in more detail in Chapter 2 of the Intel Pentium 4 Processor Optimization Reference Manual, under the heading “Memory Accesses”.

level data cache, 8 ways in the Advanced Transfer Cache). One solution to prevent a 64K-aliasing situation is to allocate a larger data structure and then offset the buffer pointers. (See Example 1) Similar techniques can be used to prevent aliasing situations on 32K, 16K and 2K boundaries when a application needs to allocate many data structures. In practice, most applications are not expected to experience aliasing situations. In applications that do experience aliasing, eliminating these conditions result in performance gains.

Additional Considerations

The following techniques for effective use of the memory pipeline can also improve application performance:

- 3D applications use many parallel streams of data in the form of vertices, color coordinates, texture coordinates, etc. Arranging these parallel data streams to form a hybrid Structure of Array (SOA), similar to those shown in Example 2, can help SIMD performance by processing 4 parallel data elements at a time. Furthermore, the hybrid SOA approach improves the locality of the operands, allowing more efficient use of the memory pipeline. This is because only those operands used by a specific computation stage (e.g. x, y, z coordinates used by transformation) are included in a cache line. The traditional array-of-structure approach results in fetching unused color data when computing 3D transformations.

```
//AOS
struct {float x, y, z, r, g, b
}AoS_xyz_rgb[200];
-----------------------------------------------------
// SOA
struct {   float  x[200], y[200], z[200];
   float  r[200], g[200], b[200];
} SoA_xyz_rgb;
-----------------------------------------------------
// Hybrid SOA
struct {   float  xx[4], yy[4], zz[4];
} Hybrid_xyz[50];
struct {   float  rr[4], gg[4], bb[4];
} Hybrid_rgb[50];
-----------------------------------------------------
```

Example 2. An example of hybrid SOA data structure. Processing of vertices or color data can be implemented more efficiently using SIMD techniques. Memory accesses are also improved due to fewer streams and fewer DRAM page conflicts.

- When reading or writing to memory, consider using instructions to load/store 16 bytes at a time. Reading or writing 16 bytes to 16-byte aligned memory is the most efficient way to use the memory pipeline.
- Hiding data access latency and improving cache efficiency can be accomplished by using prefetch instructions. In general, prefetch instructions should be used only where the memory access pattern is predictable, and when the execution pipeline may stall if data is not available. In general, the instruction “PREFETCHNTA” may be the best choice when implementing software-based prefetch, since it minimizes evicting useful data from the caches. The best fetch-ahead distance to use should be determined based on the highest target processor frequency. Software prefetches should be used judiciously, i.e. one prefetch every 32 bytes is reasonable, but more frequent use may be inefficient.
- Because the size of the WC buffer is 64 byte, sparse data structures may require more bus transactions, resulting in performance penalties. Consider packing each sparse data structure into a more dense form.
- Spin-wait loops are used in software for synchronization. On Pentium 4 processors, software should ensure that spin-wait loops incorporate the “PAUSE” instruction, either directly or indirectly (through operating-system calls that implement the PAUSE instruction). This practice helps performance scaling.

2.2 Computational Optimization Techniques

The most important areas to focus on when implementing scalable, high-performance computational code are: improving branch predictability, avoiding x87 coding pitfalls, optimizing the performance of SSE/SSE2

code, and selecting optimal instructions in critical code paths. Techniques for optimizing computational code in these areas are summarized below:

Improve Branch Predictability

The Intel NetBurst micro-architecture significantly improves the branch prediction capabilities in the Pentium 4 processor to deliver scalable performance gains. However, when a branch is mispredicted, the misprediction penalty is typically the depth of the pipeline. Fewer mispredicted branches lead to better frequency scaling. The following techniques can greatly reduce the occurrence of misprediction penalties[5]:

- Place code and data on separate pages (avoid self-modifying code).
- Minimize the number of branches. (Conditional move instructions can be used to eliminate a branch.)
- Write code that is consistent with the static branch prediction algorithm.

Write Efficient X87 FPU Code

When implementing floating-point computations using x87 FPU instructions, there are several coding guidelines that can help x87 FPU code perform efficiently and prevent stalls.

- Minimize x87 Mode Changes

On the Pentium III processor, executing the FLDCW instruction to change the mode of operation of the x87 FPU is an expensive, serializing operation (i.e. the pipeline is flushed). On the Pentium 4 processor, the FLDCW instruction is improved for situations where an application alternates between two constant values that differ in bits 8 through 12 (i.e. precision control, rounding control, infinity control) of the x87 FPU control word (FCW), such as when performing conversions to integers.

In situations where an application cycles between three (or more) constant values of the FCW, the FLDCW optimization does not apply and performance will degrade for each FLDCW instruction. One solution to this problem is to structure the code such that the application can alternate between a pair of FCW values that differ only in bits 8 through 12 for an extended period before using a different pair of values (that also differ only in bits 8 through 12) for the next extended period. The performance degradation, in this case, only occurs when changing between different pairs of values, and not on each FLDCW instruction[6]. Other alternatives include using CVTTPS2PI and CVTTSS2SI instructions in SSE, which inherently use the "truncate" rounding mode, regardless of the rounding mode set in the MXCSR register.

- Keep Numerical Values in Range and Avoid Exceptions

When floating-point code encounters data that causes one or more masked floating-point exceptions, these situations cause performance to degrade because the execution pipeline needs additional assistance from slower microcode operations to handle these situations. Three of the most commonly encountered floating-point exceptions are: arithmetic overflow, arithmetic underflow, and denormal operand.

To reduce the impact of these masked floating-point exceptions on performance with x87 FPU code, applications should strive to keep data values within the numerical ranges to prevent overflow or underflow exceptions from occurring. Using double-precision computations can reduce the likelihood of encountering underflow or overflow conditions. Furthermore, denormalized floating-point constants should be eliminated in floating-point data to prevent denormal operand exceptions.

---

[5] Recommendation on improving branch predictability is discussed in more detail under the heading "Branch Prediction" in Chapter 2 of the Intel Pentium 4 Processor Optimization Reference Manual.

[6] Recommendation on working with x87 floating-point modes are discussed in detail under the heading "Improving the Performance of Floating-point Applications" in Chapter 2 of the Intel Pentium 4 Processor Optimization Reference Manual.

- Use Software-based Alternatives for Transcendental/Trigonometric Computations

If there is no critical need to evaluate the transcendental/trigonometric functions using the extended precision of 80 bits, applications should consider alternate, software-based approaches, such as a look-up-table-based algorithm using interpolation techniques. It is possible to significantly improve transcendental performance with these techniques by choosing the desired numeric precision, the size of the look-up table and taking advantage of the parallelism of the SSE and SSE2 instructions. Also, the Intel® Approximate Math Library[7] offers very fast solutions for scalar and packed processing of trigonometric, logarithmic and exponential functions with approximate results.

Writing Efficient SSE and SSE2 Code

The following techniques will improve the performance of SSE and SSE2 code when operating on packed floating-point operands:

- Flush-to-Zero (FTZ) Mode

The Pentium 4 processor handles masked overflow exceptions that occur while executing SSE or SSE2 code without any performance penalties. It can also handle underflow exceptions more efficiently when executing SSE or SSE2 code if the FTZ mode is enabled. The FTZ mode was introduced with the SSE extensions. When this mode is enabled, an underflow result is automatically converted to a zero with the correct sign. Enabling the FTZ mode improves the performance of both single-precision and double-precision floating-point SIMD code in applications where automatically converting underflow results to zero can be tolerated.

- Denormals-Are-Zeros (DAZ) Mode

The Pentium 4 processor can handle masked, denormal exceptions in SSE code more efficiently, if the "Denormals-Are-Zeros" (DAZ) mode is enabled. The DAZ mode is introduced with Pentium 4 processor to enhance the performance of SSE code when denormal input values are encountered occasionally. When the DAZ mode is enabled, denormal source operands are treated as zeros with the same sign. This architectural behavior applies to both single-precision and double-precision operands. When the DAZ mode is active, the denormal flag in the MXCSR register will not be set, irrespective of whether exceptions are masked or unmasked. The DAZ mode is available in the Pentium 4 processor in a subsequent stepping. Turning the DAZ mode on will improve significantly the performance of SSE applications that operate on denormal operands.

1a. Execute cpuid with input EAX = 1
1b. Verify the processor supports FXSAVE and FXRSTOR instructions by checking the FXSR bit in CPUID feature flags,
1c.Verify the processor supports either SSE or SSE2 instructions by checking the XMM bit and EMM bit,
2. Allocate a 16-byte aligned, 512- byte area of memory and initialize the 512-byte memory to zero,
3. Execute FXSAVE using the cleared memory
4. Examine Byte 28-31 of the FXSAVE image, these 4 bytes is the MXCSR_MASK field
5. If bit 6 of the MXCSR_MASK is set, then DAZ is supported

Example 3. Sequence of steps to detect the availability of DAZ mode.

The FTZ and DAZ modes should be used when speed is most important and a slight loss in precision is acceptable. The FTZ and DAZ modes are enabled by setting the FZ and DAZ bit in the MXCSR register.

[7] The Intel Approximate Math Library is available at http://developer.intel.com/design/Pentium4/devtools/ .

Select Short-Latency Instructions in Critical Paths

While the latencies of commonly-used instructions are comparable between Pentium 4 and Pentium III processors, the latency of a few instructions have increased in the Pentium 4 processor. Techniques for replacing long-latency instruction by several short-latency instructions include:

- Replace "IMUL/MUL" instruction with combinations of "ADD" and "LEA" instructions,
- Replace "SAL/SAR" instruction with several add instructions.

These optimization techniques based on instruction latency considerations should be implemented only in critical code paths that are frequently executed.

2.3 AGP and Graphics Optimization Techniques

Graphics applications typically generate frequent bus activities when exchanging data between the processor and the graphics card. The data rates and data paths that are available to graphics applications are illustrated in Figure 4. System memory and the graphics card are connected to the processor through the Intel® 850 Chipset. The system bus that runs between the processor and chipset supports a data rate of 3.2 GB/s. A matching dual-channel bus between the chipset and the system memory supports a maximum data rate of 3.2 GB/s. The Intel® 850 chipset supports the AGP4X specification for data transfer between the graphics card and the chipset. The maximum data rate of AGP4X is 1.06 GB/s.

When writing data from the processor to system memory, Pentium 4 processor supports data rate of 2GB/s. For 3D applications, this write-to-memory bandwidth is best suited for a Direct Memory Access (DMA) method, where the processor writes to system memory and the graphics cards reads the data from the same system memory. Alternatively, one can write data directly from the processor to the graphics card using a "Fast Write" method. The Pentium 4 processor supports a data rate of using the "Fast Write" method at up to 700 MegaBytes per second (MB/s). Traditionally, the Fast Write data path is used by 2D graphics applications, such as video codecs. Graphics drivers supporting 3D graphics frequently use the DMA data path.

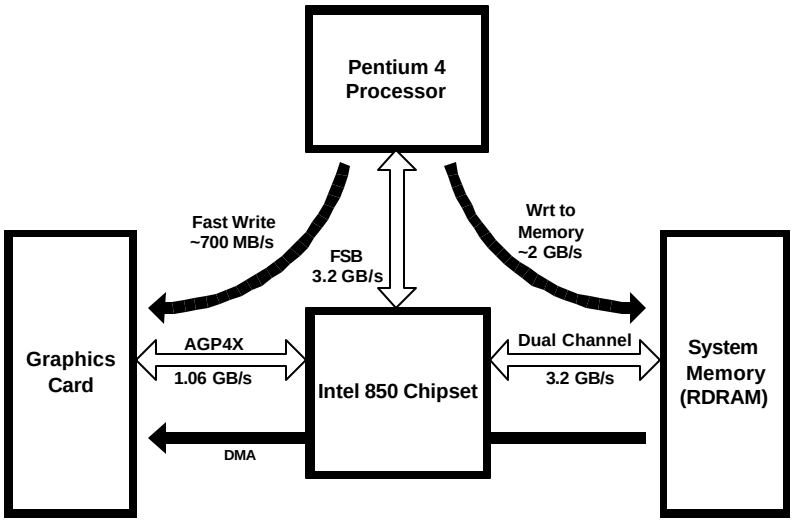

Figure 4. Data paths and data rates available to graphics applications on a Pentium 4 processor platform.

To ensure that graphics applications can take advantage of the high data rates available in a Pentium 4 processor platform, the following 3 recommendations should be adopted:

Ensure AGP and Fast Write are Enabled

To ensure that AGP4X is enabled on a Pentium 4 processor platform, proper driver files for Intel 850 chipset must be installed on the platform. If AGP4X is not enabled properly, the data rate for Fast Write will be reduced to about 180 MB/s, which is 1/3 of what is available with AGP4X enabled. Also, the DMA read rate is reduced by ½ , to ~500 MB/s

To query whether AGP4X and Fast Write are enabled on a platform, the AGP Command Register[8] in the Intel 850 chipset (located in PCI configuration space at address 0x800000a8) reports a 32-bit value that includes the following operational parameters : bit 8: AGP Enable, bit 4: Fast Write Enable, and bits 2-0: data rate for AGP. A hexadecimal value of 0x00000114 that is returned by the AGP Command Register indicates that AGP is enabled and the data rate is 4X, Fast Write is enabled.

If either the "AGP Enable" bit is off, or the data rate indicates that 4X is not operational, it is likely that an older driver is installed for the graphics card. Most graphics card vendors have newer drivers, available from their website, that recognize the Intel 850 chipset properly. Although not all graphics cards support Fast Write.

Avoid Partial Writes to Ensure Write Full Bandwidth

Typically, graphics memory (AGP buffers in either system memory or local frame buffer memory) is mapped as Write Combining (WC) memory. Writing to WC memory addresses involves writing to the processor's WC buffers, which can compete with other cacheable writes. The Pentium 4 processor improved the buffering situation for load and store operations by providing separate read request buffers for load operations and 6 separate WC buffers for store operations. The size of a WC buffer is 64 bytes on Pentium 4 processors. The WC buffers are used by both Writeback cached (WB) and WC store operations on the Pentium 4 Processor. (See Figure 5) The Pentium III processors have 6 buffers, 32 bytes each, which are used by WC stores, and all L1-missing load/store operations. When competing traffic closes a WC buffer before all writes to the buffer are finished, this results in a series of 8-Byte partial bus transactions rather than a single 32/64 byte write transaction. When partial-writes are transacted on the bus, the effective data rate to system memory is reduced to only 1/8 of the system bus bandwidth on Pentium 4 processors (1/4 on Pentium III processors). Therefore avoiding partial-write transactions is highly important to ensure full bus bandwidth utilization.

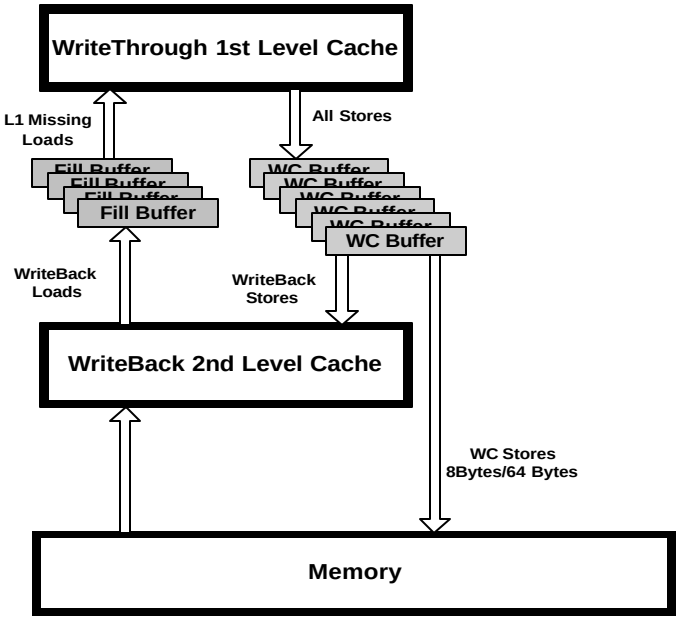

Figure 5. Buffers in Pentium 4 processor to handle load operations, Writeback stores, and Write-Combining store traffics.

Use Software Write-Combining to Utilize Full Bandwidth

To ensure that the number of bus transactions is minimized and 64 bytes are written to memory on each bus transaction, application or driver software can use a software write-combining technique to separate WC store operations from competing with WB store traffic. To implement software write-combining, uncacheable writes to memory with the WC attribute are instead written to a small, temporary buffer (WB type) that fits in the 1st level data cache. When the temporary buffer is full, the application/driver copies the content to the final WC destination. The copy loop does not cause resource contention on WC buffers. This

[8] AGP Command Register is documented in the Intel 850 Chipset: 82850 Memory Controller Hub (MCH) Datasheet, available at http://developer.intel.com/design/chipsets/datashts/ .

software write-combining technique is beneficial for both Pentium 4 processors and Pentium III processors, since it minimizes partial write requests and ensures full 64/32 byte transactions.

2.4 Potential Performance Gains of Code Optimization Techniques

The code optimization techniques described in section 2.1 through 2.3 are based on hands-on experiences of working with many existing software applications. Typically, an individual application has only a few top-level, coding issues that require applying these optimization techniques. Table 1 provides approximate ranges of likely performance gains, estimated at an application level, of applying individual optimization techniques. These are based on analysis of a broad range of applications. Actual performance results on target applications will be influenced by many factors, ranging from the workload characteristics of the application, implementation details of the re-coding effort, hardware and software configurations, etc. The approximate ranges of likely performance gains are based on a comparison of performance results between a typical Pentium 4 processor platform relative to a typical Pentium III processor platform, with similar hardware configurations and with the frequency of the Pentium 4 processor running at approximately 1.5X higher than that of the Pentium III processor.

Table 1. Approximate Ranges of Potential Application-Level Performance Gains of Several Code Optimization Techniques.

Item	Category	Coding Technique	Potential Relative Performance Gain
1	Memory	Pay Attention to Store-To-Load Forwarding Restrictions	~1.1 – 1.3X
2	Memory	Avoid Cache Line Splits, MOB Splits	~1.1 – 1.2X
3	Memory	Avoid Aliasing	~1.05 – 1.1X
4	Memory	Use 16 Byte Load/Store	~1.1X
5	Memory	Use Optimal Prefetch Instruction	~1.1 – 1.15X
6	Memory	Avoid Sparse Data Structures	~1.1 – 1.3X
7	Memory	Use Hybrid SOA Data Structure	~1.1X
8	Computation	Improve Branch Predictability	~1.05 – 1.1X
9	Computation	Minimize x87 Modes Changes	~1.1 – 1.3X
10	Computation	Eliminate x87 FP Exceptions	~1.1 – 1.3X
11	Computation	Enable FTZ/DAZ	~1.1 – 1.3X on SSE applications
12	Computation	Replace Long-latency Instructions	~1.1 – 1.2X
13	Graphics/Bus	Avoid Partial Writes/ Software Write-Combining	~1.1 – 1.2X
14	General	Integer work oads	~1.1 – 1.2X
15	General	Floating-point/SIMD workloads	~1.3 – 1.7X

3 Optimizing MPEG 2 Decoder for Pentium® 4 Processor

3.0 Overview

This Section describes a software optimization case study in which the optimization issues described in Section 2 were identified in an MPEG 2 decoder application, and several optimization techniques are applied in conjunction with a performance tuning methodology[9] to optimize the critical code paths in the application. The performance gains achieved with these techniques on the MPEG 2 Decoder application are presented.

3.1 Performance Optimization Methodology

Two important questions to ask when tuning software are: (1) how to identify what code to focus on, and (2) how to estimate the benefit of recoding, and/or re-compiling with an optimized compiler? A beneficial approach for getting answers to these questions is to sort the execution times of a given workload into sections according to the amount of time spent in each section of the executed code. By focusing on small sections of code that consume greater proportion of execution time and using an accurate tool for measuring performance improvement, the challenge of estimating the reward of optimizing an application becomes easier. Combined with an accurate tool for estimating likely application performance gain for each coding situations, this can ensure software tuning effort is focused on the primary coding issues.

In the following sub-sections, the answers to the two questions above are demonstrated by first describing the general performance optimization methodology and tools available to use in software tuning. Then, an explanation of how to combine the methodology and tools with micro-architectural and coding knowledge to the MPEG 2 decoder application is given.

Use Closed-Loop Cycle to Test and Evaluate Performance

Software tuning effort should employ a closed-loop process that includes the following steps:

- Establish a baseline reference of performance results.
- Collect useful data to characterize and correlate critical performance behavior with critical code paths.
- Generate alternative remedies for new implementation.
- Track and evaluate performance results.

Use the Right Tools

Using the right set of tools targeted for specific tasks in the closed-loop process makes code tuning more productive and efficient. The following tools are readily available when tuning code to deliver high performance levels on the Pentium 4 processor:

Documentation: The three volume *IA-32 Intel Architecture Software Developer's Manual*[10] provides programmers with basic information about the IA-32 architecture and the IA-32 instruction set. The *Intel Pentium 4 Processor Optimization Reference Manual* provides coding recommendations to help

---

[9] A tutorial on Intel® Performance Methodology is available at http://developer.intel.com/vtune/cbts/opttut/. Intel® Developer Service has additional papers and training material on performance optimization for Intel architecture processors, these materials are available at http://cedar.intel.com/cgi-bin/ids.dll/topic.jsp?catKey=Training under the "training" tab.

[10] The three-volume set of IA-32 Intel Architecture Software Developer's Manual, (Volume 1, Basic Architecture; Volume 2, Instruction Set Reference; Volume 3, System Programming;) is available at http://developer.intel.com/design/pentium4/manuals .

programmers take advantage of the features of the Intel NetBurst micro-architecture when optimizing software applications.

Performance Analyzers: There are a number of tools available for analyzing performance. Many of them are useful for performance tuning at a system level or application level, such as PerfMon*, APIMon*, Visual Quantify*. Intel VTune™ Performance Analyzer[11] provides broad capabilities, including time-based sampling (TBS), and event-based sampling (EBS) which can facilitate one's understanding of the interaction between coding pitfalls and processor performance. "PDiff" [12] is a utility tool available from Intel that can be used with VTune analyzer to identify performance bottlenecks and estimate performance gains delivered by various software optimizations.

Compilers: Microsoft* Visual C++* Version 6 (Service pack 4 with processor pack) includes new keywords, data types and C Run-Time functions to facilitate optimal alignment of code and data. The processor pack also supports IA-32 architecture's MMX Technology, SSE and SSE2 extensions through intrinsics and inline assembly. The upcoming version 7 of the compiler includes some of the important, new optimization techniques.

The Intel® C++ and Fortran Compilers, versions 5.0 and later, have automatic processor dispatch support which allows the generation of code for specific targeted processors within one executable file, and support for MMX technology, SSE, and SSE2 instructions via automatic vectorization (automatic generation of SIMD and prefetch instructions, intrinsics, vector classes, and inlined-assembly. It also implements a number of optimizations aimed at handling the performance issues outlined in this document such as store-to-load forwarding, branch prediction, cache-line splits, floating-point denormals, etc.. In addition, it generates highly optimized floating-point code and provides unique features such as profile-guided optimizations to further enhance application performance..

Optimized Libraries: The Intel Approximate Math (AM) library is a set of fast routines to calculate approximate results of trigonometric, reverse trigonometric, logarithmic, and exponential functions using SSE instructions. The processing speed of AM library is 5 to 8 times faster than that of the x87 FPU instructions. The Intel Performance Library Suite[13] (PLS) contains a variety of specialized libraries which has been optimized for performance on Intel processors. The PLS includes the Intel Math Kernel Library, the Intel Signal Processing Library, the Intel Image Processing Library, the Intel Integrated Performance Primitives, the Intel Speech Developer Toolkit, and the Intel Recognition Primitives Library.

Identify and Focus on Critical Code

The VTune analyzer is useful for identifying critical code paths and performance bottlenecks. For example, it can be used to sample and compare performance data when the application to be optimized is run on two different target processors; for example, a Pentium 4 processor running at 1.5 GHz and a Pentium III processor running at 1 GHz. This performance data from the two targets can be sorted and displayed at different scopes ranging from modules (as in the example "hot-spot" displays in Figure 6), to functions, to assembly code. This capability allows engineers to identify individual modules, and individual functions as "hot spots".

To determine whether a prominent hot-spot module or function is a cause of poor performance, the sampled data from VTune analyzer can be further processed for comparison based on a relative performance scaling factor between the two target processors. Typically, those modules (or functions) that represent performance bottlenecks are identified by a relative scaling factor, that falls significantly below 1.0 or another known

---

[11] Visit http://developer.intel.com/software/products/vtune/ for more information on Intel VTune Performance Analyzer.

[12] The "pDiff" utility tool requires latest version of VTune analyzer. ISVs can contact their Intel representative to use "pDiff" with VTune analyzer.

[13] The Intel Performance Library Suite is available at http://developer.intel.com/software/products/itc/ .

characteristic of the workload. An example of sorted hot spots with relative scaling factors calculated at modules, functions, and assembly code levels are illustrated in Figure 7.

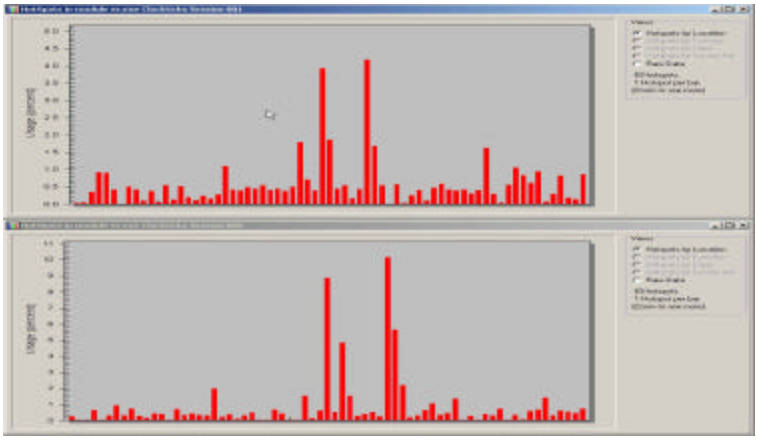

Figure 6. Two snapshots of "hot-spots" that are displayed after VTune analyzer captures sampling data from two sampling sessions. Snapshot of a Pentium III processor at 1GHZ is shown on the top half, Pentium 4 processor at 1.5 GHz is shown in the lower half. Hot spots can be identified at a module level, or function level.

Documentation such as the *Intel Pentium 4 Processor Optimization Reference Manual* helps engineers understand the interaction between the poor-performing code and micro-architectural considerations. The Pentium 4 processor provides a number of performance monitoring events[14], which are supported in VTune analyzer's event-based sampling feature. Using these performance monitoring events, engineers can narrow down the cause of a bottleneck to specific micro-architectural considerations and devise an appropriate remedy. When implementing alternate remedies to address coding pitfalls, it is very important to focus on one issue at a time and limit the scope of re-coding to the critical code paths.

Estimate Application-Level Gain for Re-coding

Estimating the performance gain in an application by re-coding and/or re-compiling a source module that contains a performance bottleneck can be calculated by substituting an estimated performance gain (One can use the approximate ranges of potential performance gains listed in Table 1 or the SPEC CPU 2000 performance gains as a starting-point.) into the measured performance scaling factor (which may be less than 1.0) of the module. The overall application-level performance gain can be estimated by averaging the performance scaling factors of all the modules in the application, weighted by the time spent in each module.

3.2 MPEG-2 Video Encoder – A Case Study

An MPEG 2 decoder is expected to have good performance scaling on Pentium 4 processors because it requires large amounts of data to pass through the memory pipeline and the computations required to process MPEG 2 data are conducive to SIMD coding techniques.

When a commercial implementation of an MPEG 2 decoder application exhibited poor performance scaling when running on the Pentium 4 processor, compared with Pentium III processor, an effort was undertaken to understand the root cause of the apparent poor performance, and to improve the application performance. The following paragraphs walk through the steps of applying the performance optimization methodology to the tuning of this application and presents the performance gains of this tuning effort.

Main Coding Issues Quickly Identified Using VTune™ Analyzer

The baseline performance data are collected by VTune analyzer using the time-based sampling (TBS) technique, and running a baseline executable on both a Pentium 4 processor and on a Pentium III processor. The baseline performance data from the two sampling sessions includes a measurement of the elapse time of each sampling session. A utility (pDiff) was used to process the two TBS samples and generate the spread sheet result shown in Figure 7. These results show the scaling of Pentium 4 processor versus Pentium III processor on a per module, per function, and per instruction-address-range granularity. Two modules

[14] Performance monitoring events for Pentium 4 processor are described in Chapter 14 and Appendix A of the *IA-32 Intel Architecture Software Developer's Manual*, Volume 3.

(decode.dll and grfx_drv.dll) contribute to approximately 90% of overall execution time, and are showing poor scaling (1.20X and 0.50X respectively).

By examining the scaling factors shown in Figure 7, performance bottlenecks are identified in a "synchr" function, and in a "memmove" function.

Using the "Instr. Bin EIP" address ranges, the assembly listing of these critical code paths can be examined in detail in VTune analyzer. Coding issues for these performance bottlenecks are covered in the optimization guidelines discussed in Section 2 and in the *Intel Pentium 4 Processor Optimization Reference Manual*. Collecting additional event-based profiles using the VTune analyzer can help confirm which issue is present.

With the MPEG 2 decoder, the root causes of poor scaling or non-scaling are identified to be:

- The frame rate was locked to the video refresh rate instead of using a double-buffering approach in the "synchr" function.
- The AGP4X was not enabled by graphics driver, which reduced the data rate of using Fast Write to

Module/DLL Name	*Time Spent (PIIIP)*	*Time Spent (P4P)*	*Actual Scaling*	***Function Name***	*Time Spent (PIIIP)*	*Time Spent (P4P)*	*Scaling*	***Instr. Bin EIP***	*Time Spent (PIIIP)*	*Time Spent (P4P)*	*Scaling*	*Est. Scaling*
decode.dll	60%	65%	1.20	synchr	5%	15%	1.1	0x0000-0x0040	5%	15%	1.1	1.50
grfx_drv.dll	25%	30%	0.50	memmove	25%	30%	0.5	0x4fc40-0x4fc80	25%	30%	0.5	1.50
app.exe	10%	4%	1.40	SplitStream	3%	1%	1.4	0x96540-0x96580	3%	1%	1.4	1.40
GID32.dll	5%	1%	1.20	SaveDC	1%	1%	1.2	0x21f40-0x21f80	1%	1%	1.2	1.20
Total	100%	100%	1.05									1.45

Figure 7. Relative performance scaling factor can be measured for each "hot spot", based on VTune analyzer's time sampling data. The decode.dll contains multimedia instructions, but the scaling is below that expected for multimedia applications. The grfx_drv.dll is scaling very poorly.

1/3 of the maximum FastWrite bandwidth and caused the "memmove" function to scale poorly,

Application Performance Gain Addressed Individually

To evaluate the return of investment for optimizing the performance bottlenecks that are identified above, the approximate ranges of performance gains listed in Table 1 can be used along with the scaling factors shown in Figure 7. The "decode.dll" module is expected to behave mostly like a multimedia workload. Once parallelism coding issues are addressed, the "decode.dll" module is likely to scale better than 1.2. Using a desired performance scaling factor that scale in-line with optimized multimedia workload, 1.5 for example, the estimated application scaling factor could improve from 1.05 to 1.23. Thus, re-coding the "synchr" function to implement a double-buffering approach is likely to result in substantial net performance gain for the application.

The work-load characteristics of grfx_drv.dll is largely bandwidth-limited, a reasonable choice for a desired scaling factor can be based on the approximate ranges of floating-point/multimedia workload in Table 1. Using a desired scaling factor of 1.5 to estimate addressing AGP4X/FastWrite and refresh issues, the estimated application scaling factor, due to fixing grfx_drv.dll alone, could improve from 1.05 to 1.30. This is another opportunity to deliver significant application performance gain.

This approach for estimating performance gains can be accurate, since the estimate is based on improving a very localized range of instructions. The remaining code retains its measured scaling. The combined impact of both changes is an overall scaling of 1.45X. This estimate is an almost 40% improvement over the performance of the original decoder. The measured performance gain after addressing these two issues is 1.45X over the original application code, this completes the closed-loop cycle and confirmed the accuracy of the tools and this methodology.

4
Pentium® 4 Processor Performance Results

4.0 Overview

The case study of optimizing the MPEG 2 decoder illustrates the substantial, application-level, performance gain that software can realize on a Pentium 4 processor by strategically identifying and addressing critical code paths in an application. Typically benchmarks deliver higher performance results on Pentium 4 processors due to higher processor frequency and the features of the Intel NetBurst micro-architecture. In some cases, newer benchmarks deliver performance results that are representative of the benefit of re-compilation using compilers that generate optimized code for Pentium 4 processor. This section presents the Pentium 4 processor performance results of several benchmarks and representative applications with comments on the workload characteristics of these results.

4.1 Pentium® 4 Processor SPEC CPU 2000 Performance

The SPEC CPU 2000 benchmark suite contains a wide variety of practical integer and floating-point tasks that are representative of the real-world software problems that modern computers try to solve. The suite contains two sets of benchmarks: the SPECint 2000 benchmark (which measures compute-intensive integer performance) and the SPECfp 2000 benchmark (which measures compute-intensive floating-point performance). The SPECint 2000 benchmark contains 12 applications in portable source code form, written in C and C++ languages. The SPECfp 2000 benchmark contains 14 applications in portable source code form, written in Fortran and C languages.

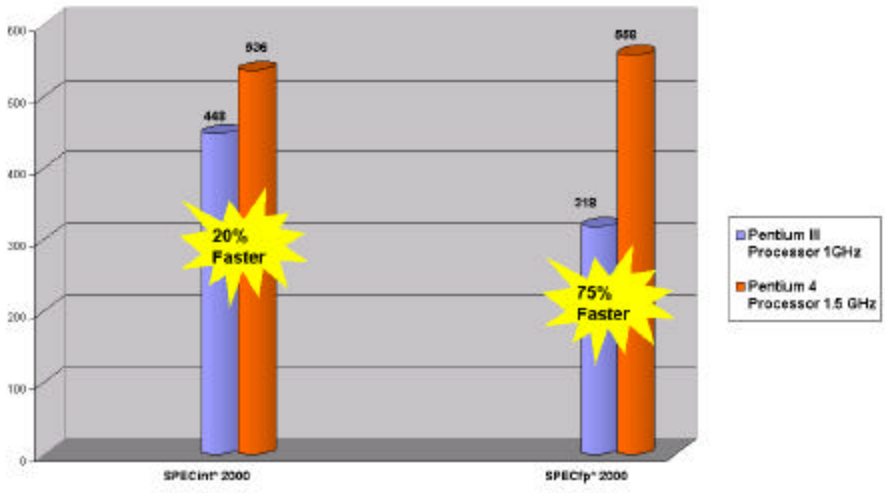

Figure 8. The Pentium 4 processor at 1.5 GHz delivers next-generation performance level than Pentium III processor at 1GHz on the same manufacturing process. The Pentium 4 processor is 20% faster on SPECint 2000 suite and 75% faster on SPECfp 2000 suite. Test configurations for benchmark results are listed in Section 6.

The executable binaries of the benchmark suite are generated from commercially available compilers using combination of Microsoft Visual C Compiler, Intel C++ and Fortran Compilers (See reference 1). Performance results of the SPEC CPU 2000 benchmark suite for Pentium 4 processor are consistent between 3rd party measurements submitted to SPEC and measured results published by Intel. The execution binaries generated by Intel compiler for the SPECfp 2000 suite contains very limited use of SSE/SSE2 instructions. The SPECfp 2000 workload executes ~28% of the x87 FPU instructions, and less than 3% of the SSE and SSE2 instructions combined. The performance scaling of the results of the SPEC CPU 2000 benchmark suite (See Figure 8) is a good indication of the appreciable performance gains an application may be able to achieve by re-compiling source code written in high-level languages with compilers that can generate optimized code for Intel NetBurst micro-architecture.

4.2 Pentium® 4 Processor Multimedia and 3D Performance

Multimedia applications (such as Ulead* VideoStudio* 4.0, Windows* Media Encoder) enable high quality video content to be easily created, edited and distributed to wide audiences. These video-editing applications contain varying degrees of mixtures of integer code, x87 FPU code, MMX technology code, as well as some non-scaling input/output tasks. The operation of these video-editing applications can also be influenced by the efficiency of the graphics hardware and its drivers and, in part, by 3rd party components, such as codecs.

The results of Ulead VideoStudio 4.0, Windows Media Encoder are indicative of the performance of workloads that contain code implemented with MMX technology which are compiled by Microsoft Visual C++ Compiler without the Processor Pack enhancement. These performance results demonstrate that multimedia workloads which have not been re-compiled for Pentium 4 processors, can realize appreciable benefits from the advanced micro-architectural features in Pentium 4 processor.

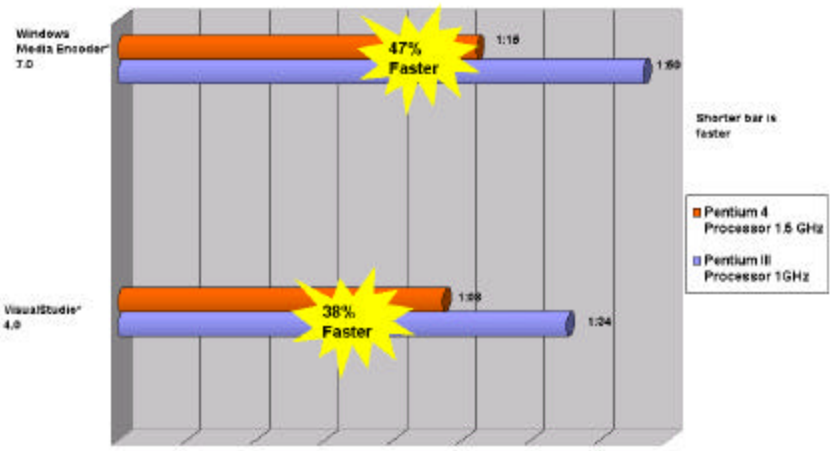

Figure 9. The Pentium 4 processor performs faster video editing, encoding on existing applications. Pentium 4 processor is 47% faster than Pentium III processor in Windows Media Encoder 7.0, and 38% faster in Ulead VisualStudio 4.0. Test configurations for video editing and encoding are listed in Section 6.

3D applications require huge processing power and bandwidth from the microprocessor to deliver a smooth, realistic user experience. Popular 3D games and benchmark suites (such as Quake* III Arena and 3D WinBench 2000 Processor Test) are commonly used to evaluate desktop PC's 3D performance. The workload in these applications contain varying degrees of mixtures of integer code, x87 FPU code, MMX technology code, and SSE code. The operation of these 3D applications is also influenced by the efficiency of the graphics hardware and by 3D APIs, such as OpenGL* or DirectX*. The results of Quake III Arena and 3D WinBench 2000 Processor Test are indicative of the performance of workloads that contain code implemented with x87 FPU code (Quake III) and SSE code (3D WinBench 2000) which are compiled by Microsoft Visual C++ Compiler.

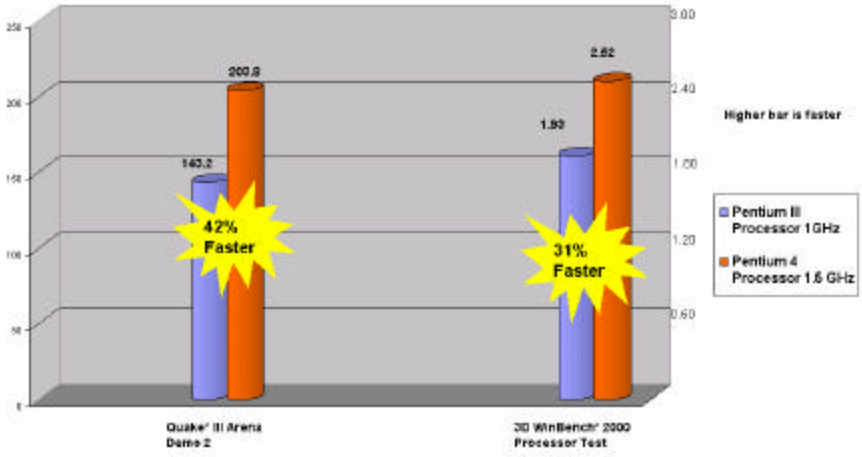

Figure 10. The Pentium 4 processor at 1.5 GHz delivers exceptional performance to enable smooth game and 3D graphics. The Pentium 4 processor is 42% faster than Pentium III processor in Quake III Arena and 31% faster in 3D WinBench 2000 Processor Test. Test configurations for 3D performance results are listed in Section 6.

4.3 Pentium® 4 Processor Performance on Emerging Applications

Speech-to-text translation and create-your-own-music programs are emerging as important productivity tool and popular entertainment applications, respectively. Speech recognition and MP3 encoding software, like Dragon* Naturally Speaking* Preferred 4.0 and Canon* eJay* MP3 Plus 1.3, contain varying degrees of mixtures of integer code, x87 floating-point code, MMX technology code, SSE code, as well as some non-scaling input/output tasks. The results of Dragon* Naturally Speaking* Preferred 4.0 and Canon* eJay* MP3 Plus 1.3 are indicative of the performance of workload that contain code implemented with MMX technology and SSE which are compiled by Microsoft Visual C++ Compiler. These performance results demonstrate that multimedia workloads that are implemented in MMX and SSE instructions without use of SSE2, can realize appreciable benefits from the advanced micro-architectural features in Pentium 4 processor.

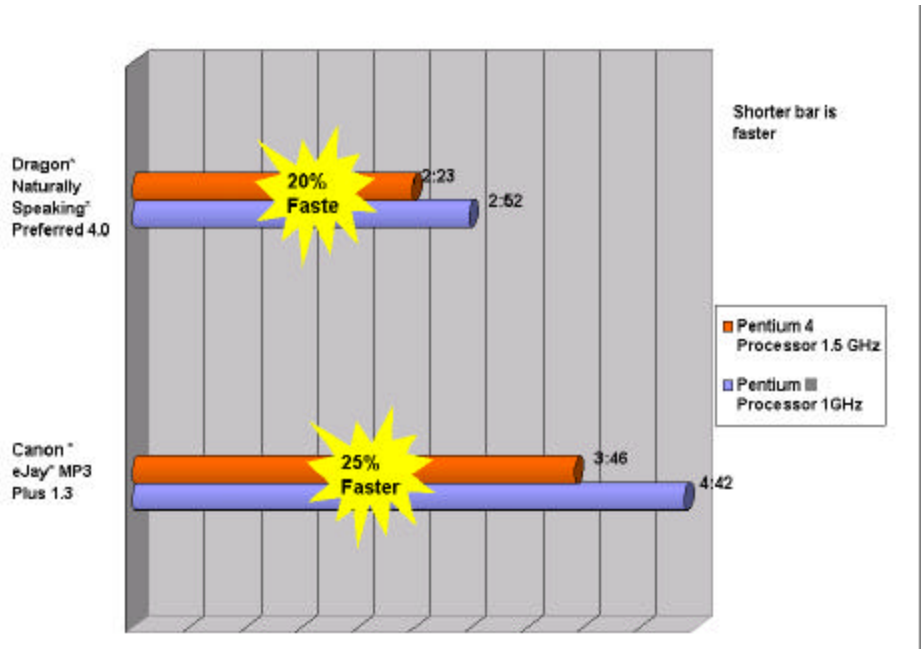

Figure 11. The Pentium 4 processor at 1.5 GHz empowers emerging applications to enhance productivity and enjoyment. The Pentium 4 processor is 20% faster than Pentium III processor in Dragon Naturally Speaking Preferred 4.0 and 25% faster in Canon eJay MP3 Plus 1.3. Test configurations for the performance results are listed in Section 6.

5 Summary

The Intel Pentium 4 processor is designed to deliver the next-generation performance for desktop and workstation clients. It is based on the Intel NetBurst micro-architecture, which enables significantly higher clock rates and better performance scaling efficiency. Many software applications deliver appreciable performance gains on the Pentium 4 processor by directly benefiting from higher clock rates and micro-architectural enhancements, such as Rapid Execution Engine and Execution Trace Cache while others can gain dramatic improvements by recompilation using the latest optimizing compilers and libraries, or via assembler-level optimizations specifically targeted for the micro-architecture and using the SSE2 instruction set. Using a closed-loop performance tuning methodology, compilers that can generate optimized Pentium 4 processor code, and the "pDiff" tool in conjunction with VTune analyzer, programmers can quickly identify critical code and opportunities to gain user-appreciable performance for Pentium III, Pentium 4 processors as well as future IA-32 processors.

6 Test Configurations

Processor	Pentium® 4 Processor 1.40, 1.50 GHz
Motherboard	Intel® Desktop Board D850GB
Motherboard BIOS	GB85010A.86A.0040.P03
Memory Size	128 MB PC800 RDRAM (except 256 MB for SPECint*2000 and SPECfp*2000) (Samsung KMMR16R88AC1-RKB 800-45)
Operating System	Windows* Millennium* (build 3000.2) w/ DirectX* 7.0b (except Windows* 2000 (build 2195) for SPECint*2000 and SPECfp*2000)
Hard Disk	IBM* 30GB ATA-100 DTLA-307030
Hard Disk Driver	Intel Ultra ATA Storage Driver Version 6.02 INF 2.50
Graphics Card	Creative* 3D Blaster* Annihilator 2 w/ nVidia* GeForce*2 GTS
Graphics Memory	32MB DDRRAM
Graphics Driver	NVidia* Detonator 3 v6.18
Graphics Settings	1024x768 resolution (except 640x480 for Quake* III Arena), 16-bit color
CD ROM Drive	Toshiba* 32X XM-6302B IDE
Sound Card	Creative Labs SoundBlaster* Live
Network Card	Intel Pro/100+ Management PCI LAN card
SPEC CPU2000 Compiler	Intel C/C++* and FORTRAN Compiler Plug-in 5.0 Microsoft Visual C/C++ 6.0 (for libraries)

Processor	Pentium® III Processor 800EB MHz, 1B GHz
Motherboard	Intel ® Desktop Board VC820
Motherboard BIOS	VC82010A.86A.0035.P15
Memory Size	128 MB PC800 RDRAM (except 256 MB for SPECint*2000 and SPECfp*2000) (Samsung KMMR16R88AC1-RKB 800-45)
Operating System	Windows* Millennium* (build 3000.2) w/ DirectX* 7.0b (except Windows* 2000 (build 2195) for SPECint*2000 and SPECfp*2000)
Hard Disk	IBM* 30GB ATA-100 DTLA-307030
Hard Disk Driver	Intel Ultra ATA Storage Driver Version 6.02 INF 2.50
Graphics Card	Creative* 3D Blaster* Annihilator 2 w/ nVidia* GeForce*2 GTS
Graphics Memory	32MB DDRRAM
Graphics Driver	NVidia* Detonator 3 v6.18
Graphics Settings	1024x768 resolution (except 640x480 for Quake* III Arena), 16-bit color
CD ROM Drive	Toshiba* 32X XM-6302B IDE
Sound Card	Creative Labs SoundBlaster* Live
Network Card	Intel Pro/100+ Management PCI LAN card
SPEC CPU2000 Compiler	Intel C/C++* and FORTRAN Compiler Plug-in 5.0 Microsoft Visual C/C++ 6.0 (for libraries)