Implementing Streaming SIMD Extensions on the Pentium III Processor

The SSE provides a rich set of instructions to meet the requirements of demanding multimedia and Internet applications. In implementing the SSE, the Pentium III developers made a number of design trade-offs to satisfy tight die size constraints and attain frequency goals.

Srinivas K. Raman
Vladimir Pentkovski
Jagannath Keshava
Intel Corporation

In the volume PC market, the demand is growing for general-purpose processors that facilitate visual and graphical computing on the Internet. Responding to this need, Intel developed the streaming SIMD extensions (SSE), a set of processor instructions designed to boost performance of multimedia and Internet applications, and implemented them on its Pentium III processor. The main challenge in implementing the Pentium III was to provide enhanced support for multimedia applications through the SSE while adhering to aggressive die size, frequency, and schedule goals.

At the heart of several visual and graphical computing algorithms are floating-point computations. The most cost-effective way to accelerate floating-point performance in general-purpose processors is to use the SIMD (single-instruction, multiple-data) parallel computation model. Therefore, a key SSE component is the SIMD-FP (floating-point) extensions. The processor attains high SIMD performance by balancing execution with increased memory bandwidth.

Several of the targeted multimedia applications have large working sets and are streaming applications—that is, applications in which data are usually read once and then discarded. For such applications, the Pentium III's instruction set architecture provides the programmer with primitives to manage data caching and minimize cache pollution. The cache control instructions enable the software to better tolerate increasing memory latencies. In addition to increased floating-point performance for 3D and speech recognition applications, we added new-media instructions to accelerate video encoding/decoding and 3D-software rendering.

SSE overview

The SSE is a rich set of 70 instructions in the categories shown in Table 1.

SIMD-FP instructions

The SIMD-FP instructions operate in parallel on all (packed mode) or the least significant pairs (scalar mode) of packed data operands. The instructions use an architecturally visible new state. Adding the new state reduced implementation complexity, eased

Table 1. Major types of SSE instructions.

Instruction type	Description
SIMD-FP, scalar arithmetic, logic	Addition, subtraction, multiplication, division, comparison, and other arithmetic functions of packed single floating-point numbers or scalar operands
Data movement, reorganization	Movement of packed and scalar operands and data reorganization of packed operands
Type conversion	Integer-to-floating-point and floating-point-to-integer conversion of packed and scalar operands
State save, restore	Save or restore SSE processor state to or from memory
Memory streaming	Data prefetch to specified level of cache hierarchy and streaming store
New media	Compute absolute difference, rounded average, min/max of packed MMX operands

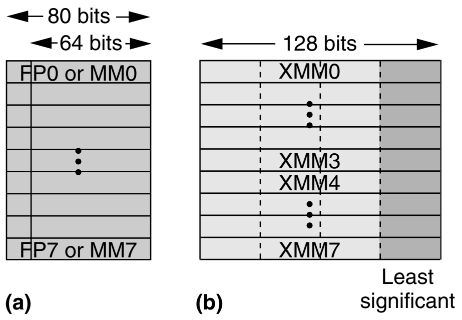


Figure 1. SSE registers in the Pentium III: eight 64-bit MMX technology/Intel Architecture X87-FP registers (a) and eight new 128-bit registers (b).

programming-model issues, allowed SIMD-FP and MMX technology or X87 instructions to be used concurrently, and addressed requests from software and operating-system vendors.

Figure 1 shows the MMX technology/X87-FP registers and the new 128-bit registers. The SSE state is completely separate from the X87-FP state, and a dedicated interrupt vector handles numeric exceptions. A new control/status register MXCSR masks or unmasks numerical exception handling, sets the rounding mode, sets flush-to-zero mode, and shows status flags. Because applications often require both scalar and packed operation modes, we defined explicit scalar instructions in the SIMD-FP mode. We provided support for two floating-point arithmetic modes: IEEE-compliant mode, for applications that need exact single-precision computation and portability, and flush-to-zero mode for high-performance real-time applications.

The single-precision floating-point comparison instruction CMP is similar to the MMX technology instruction variants and produces a mask for each 32-bit floating-point datum, depending on the results of the comparison. Programmers can use the mask with subsequent logic operations to perform conditional moves. We used an 8-bit immediate value to encode eight basic comparison predicates for the CMP instruction (programmers can obtain another four by using these predicates and swapping source operands).

The MOVMSKPS/PMOVMSKB instructions move four mask bits (each component's most significant bit) into an integer register and help simplify data-dependent branching.

The MINPS/PMIN instructions benefit the color clamping function in graphics applications. PMIN also improves the kernel performance in the evaluation of a hidden Markov model (HMM) by 33%. The HMM function is fundamental in many speech recognition engines and uses more than 80% of the execution time.

Data manipulation instructions

A processor can realize SIMD computation gains only if it can efficiently organize data into a SIMD format. To this end, the SSE supports the following data manipulation instructions:

- SHUFPS/PSHUFW instructions support reorganization of data from source operands and facilitate the realization of common functions such as rotate or swap

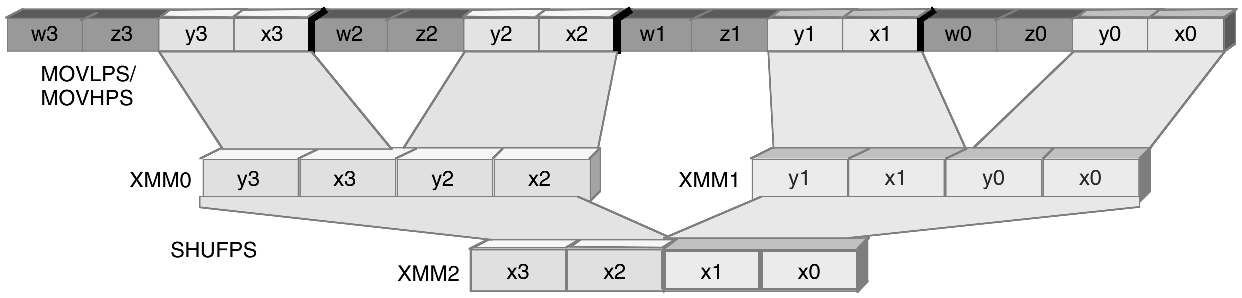


Figure 2. Using the MOVLPS/MOVHPS and SHUFPS instructions for 3D data reorganization. MOVLPS/MOVHPS moves the low part of the 128-bit source to the low or high part of the 128-bit destination. SHUFPS specifies which 32-bit suboperands of the two 128-bit sources should be gathered in the 128-bit destination.

of the data. In conjunction with SHUFPS, the 64-bit load/store instructions MOVLPS/MOVHPS enable efficient data organization of vertex components. Figure 2 shows 3D data reorganization using these instructions.

- UNPCKHPS/UNPCKLPS interleave floating-point data from the high/low part of a register or memory operand in a manner similar to the MMX technology unpack operations.
- PINSRW/PEXTRW (Figure 3) support scatter/gather operations for 16-bit lookup table processing. Hardware support to efficiently handle memory accesses that are not aligned to a 16-byte boundary is expensive, and independent software vendors prefer being alerted to misalignment through an explicit fault. Hence, all computation instructions using a memory operand should be 16-byte (128-bit) aligned. We provided unaligned load/store instructions for cases (video, for example) in which data alignment cannot be guaranteed.

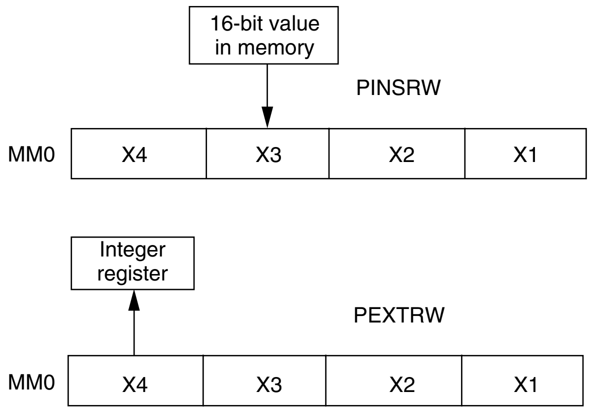


Figure 3. The PINSRW/PEXTRW instructions. PINSRW specifies which MMX technology operand to insert into; PEXTRW specifies which MMX technology operand to extract from.

Conversion operations

The SSE supports several conversion operations, including SIMD-FP to MMX technology for packed data, and scalar-FP to IA-32 integer (Intel Architecture 32-bit integer) for scalar data. To optimize the commonly used 3D geometry computations of division, square root, and 1/square root, which is used to calculate normalized values in 3D computations, the SSE introduces the two approximation instructions RCP and RSQRT. These instructions use hardware lookup tables and provide 12-bit precision. They are much faster than their IEEE full-precision counterparts (24 bits). When greater precision is needed, using these instructions with a single Newton-Raphson iteration achieves almost the same level of precision (~22 bits) at a lower latency and throughput.

Memory streaming

The Prefetch instruction allows the programmer to control data placement in the cache hierarchy and distinguish between temporal (frequently used) and nontemporal data (read and used once before being discarded). The SSE currently defines four possible prefetches, with room for future extensions. These instructions only provide a hint to the hardware, and they do not generate exceptions or faults. Besides multimedia applications, programmers can use the prefetch instructions

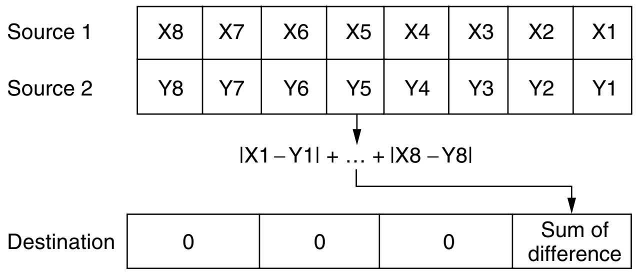


Figure 4. PSADBW (sum-of-absolute-difference) instruction sums the absolute differences of packed unsigned bytes to produce a word result.

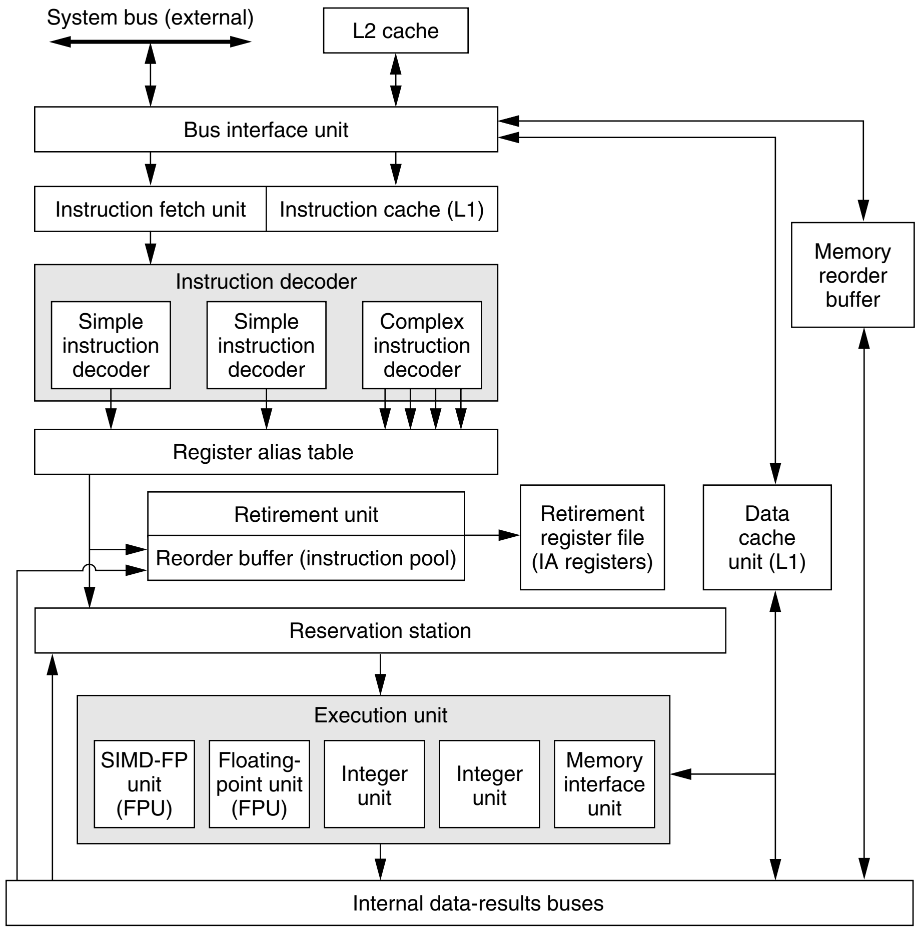


Figure 5. Functional block diagram of the Pentium Pro (P6) processor microarchitecture.

for general-purpose computing.

The complementary store instructions MOVNTPS (packed single-precision floating-point) and MOVNTQ (packed integer) enable the programmer to perform nonallocating (streaming) stores to prevent displacement of needed data in the cache. The lightweight SFENCE instruction facilitates efficient software-controlled coherency.

New media instructions

The PAVG instruction enables a 25% kernel-level speedup of the motion compensation portion of the MPEG-2 decode pipeline. The MPEG-2 specification requires rounding the results of pixel-averaging operations to the nearest integer. The rounding computation is equivalent to a 9-bit-precision operation. PAVG facilitates the use of existing 8-bit instructions by performing a 9-bit-accurate averaging operation.

Similarly, although the video-encoding pipeline involves many stages, the motion estimation function takes up the bulk of execution (40% to 70% at present). The sum of absolute differences (SAD) is a commonly used metric in motion estimation. The PSADBW instruction (Figure 4) retains byte-level execution parallelism working on 8 bytes at a time. This single instruction replaces about seven MMX instructions in the motion estimation inner loop, and consequently increases motion estimation performance by a factor of two.

Thacker et al.[1] discusses the SSE in more detail, and the Pentium III instruction set reference manual[4] contains a complete list of the instructions.

Implementing the SSE

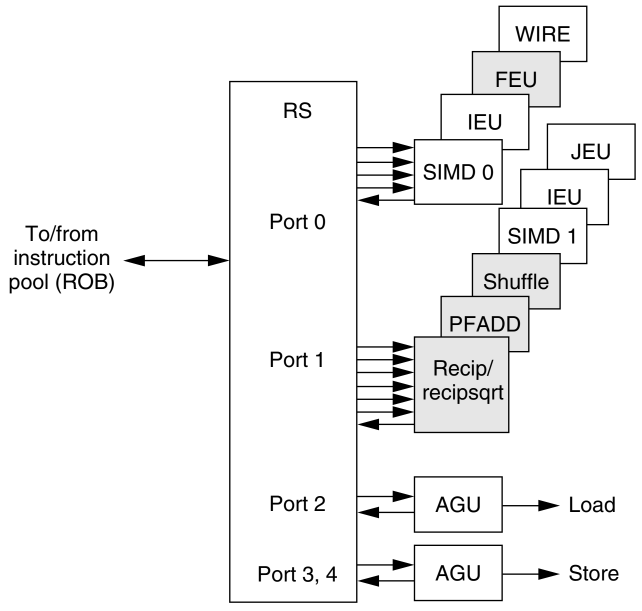
Several of the processor's target multimedia applications are inherently parallel. The SIMD feature of the extensions required us to implement support for parallel computation with wide and fast floating-point units. In addition, the streaming nature of the extensions (and the applications they support) necessitated high memory throughput.

Execution enhancements

Figure 5 shows the basic execution pipeline of the Pentium Pro processor series.[3] In the Pentium III, we enhanced floating-point performance by widening the Pentium II's floating-point hardware and putting the available resources to more effective use. Figure 6 shows the processor's dispatch/execute units with the Pentium III's new and modified units shaded.

The Pentium III implements each four-wide (128-bit) SSE computational macroinstruction as two, two-wide (64-bit) microinstructions. Although this implementation improves use of SIMD-FP hardware, it limits changes to the processor's front end. The instruction decoder transforms each 128-bit micro-operation into a pair of 64-bit internal micro-operations. Both 64-bit micro-operations are data independent with the exception of retirement control and are otherwise similar to the machine's existing 64-bit micro-operations. This approach avoids the massive and intrusive changes of adding 128-bit buses, 128-bit execution units, and register renaming in both the in-order and out-of-order portions of the machine. Such changes would have severely compromised our schedule, die size, and frequency goals.

Since the Pentium III is a superscalar implementation (that is, it has multiple execution ports), it can perform four floating-point operations every clock cycle. With this cost-effective approach, applications can theoretically achieve fourfold performance gains. Of course, a future 128-bit implementation will deliver a higher level of performance scaling.



AGU	Address generation unit
EU	Execution unit
FEU	Floating-point EU
IEU	Integer EU
JEU	Jump EU
PFADD	SSE adder
Recip/recipsqrt	SSE reciprocal
ROB	Reorder buffer
RS	Reservation station
Shuffle	SSE shuffle unit
SIMD 0/1	MMX EU
WIRE	Miscellaneous functions

Figure 6. Enhanced dispatch/execute units in the Pentium III. Compared to the Pentium II, shaded units are new or, in the case of the FEU, expanded.

Two-wide, single-precision floating-point units. The Pentium III floating-point units perform two-wide, single-precision operations, increasing their computation capability twofold over the Pentium II.

Parallel adds and multiplies. The Pentium III's multiplier and adder reside on separate ports. Hence, the processor supports parallel dispatch and execution of two-wide, packed, single-precision multiplies and two-wide, packed, single-precision adds. The multiplier resides on port 0 and is a modification of the existing floating-point multiplier. However, it performs single-precision multiplication with a throughput of two single-precision numbers each cycle and a latency of four cycles. (X87 single-precision processing has a throughput of one single-precision number every two cycles and a latency of five cycles.) The adder resides on port 1. It operates on two single-precision numbers with a throughput of one cycle and a latency of three cycles. (The adder also executes all compare, subtract, min/max, and convert instructions that are a part of the SSE.) This implementation provides a peak performance of 2.4 Gflops at 600 MHz.

Hardware support of data reorganization. Better support of data reorganization improves floating-point hardware use by speeding data manipulation. The new shuffle/logical unit on port 1 executes the unpack high and low, as well as the move and logical micro-operations. The unit performs the 128-bit shuffle operation through three micro-operations. It

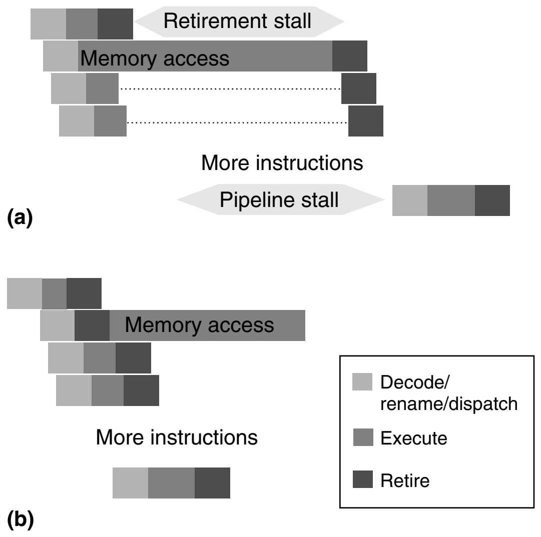


Figure 7. Difference in behavior of a load (a) and a prefetch (b).

also executes packed integer shuffle, PINSRW, and PEXTRW instructions.

Faster data copying. The IA-32 instruction set overwrites one of the operands of an instruction. For example, if the processor is adding operands *X* and *Y*, the result overwrites one of them (say, *X*). If subsequent computations require *X*, the software should copy *X* to a different register before the addition. This feature increases the need for move instructions in the code. By supporting move instructions on both ports of the Pentium III, we increased the move throughput and reduced the need for fine-grained scheduling of move instructions.

Exception handling. The cost-effective support of 128-bit processing with two 64-bit micro-operations makes an exception possible on either of the two independent micro-operations. For instance, if the first micro-operation retires while the second causes an exception, the architecturally visible 128-bit register will be updated only partially, resulting in an inconsistent architectural machine state. To continue supporting the IA-32 precise-exception-handling feature, we implemented the Check Next Micro-operation (CNM) mechanism. This mechanism, which ensures architectural consistency, works as follows: The first in a pair of two micro-operations, which must be treated as an atomic operation (and/or data type), is marked with the CNM flow marker. The CNM-tagged micro-operation's retirement is delayed until the second micro-operation is also guaranteed to retire without exception.

Since this mechanism throttles retirement, we also implemented the following optimization: In the case that all exceptions are masked, each micro-operation can retire individually. Since multimedia software usually masks exceptions to improve performance, there is no loss of computational throughput in most applications.

To further maintain high computational throughput, the Pentium III processes exceptions such as overflow, divide-by-zero, and flush-to-zero underflow in hardware rather than using microcode. The hardware handles these exceptions, which routinely occur during execution of multimedia algorithms, through modifications of the rounder/write-back multiplexers.

Memory and bus enhancements

The typical working-set size of multimedia applications such as 3D and video is fairly large. The cache(s) cannot contain all the data, which typically end up in memory. We developed the streaming instructions (such as Prefetch, MASKMOVQ) specifically to address these needs. We paid special attention to the problem of supporting prefetches, streaming stores, byte-masked writes, and store-fencing operations.

Prefetch support. Figure 7a shows the execution of a typical load instruction that misses the cache, causing a pipeline stall. Instructions subsequent to this load instruction can execute, but they cannot retire from the pipeline until the data return from memory for the load. These instructions accumulate in the pipeline until resource limitations, such as ROB (reorder buffer) entries, cause a stall.

Our prefetch implementation (Figure 7b) addresses this bottleneck, providing greater concurrency between execution and data prefetch. We accomplished this by allowing the prefetch instruction(s) to retire much earlier in the pipeline. Even in the case of a cache miss, the prefetch instruction retires almost immediately, and no stalls occur due to memory access latency.

Write throughput enhancements. Multimedia

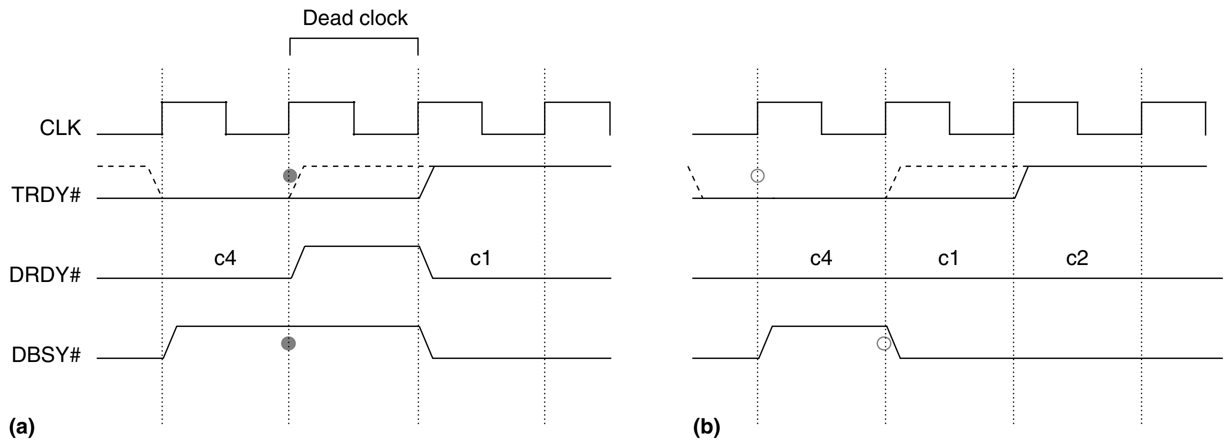


Figure 8. Write cycle dead clock removal: Pentium II (a); Pentium III (b). The Pentium III bus cluster supports a special optimization allowing TRDY# to be sampled early. Functional requirement: TRDY# must remain active in all cases after being asserted, until DBSY# is sampled as deasserted. Performance requirement: TRDY# must be asserted, at the latest, one clock before the DBSY# deassertion.

applications with large working sets demand as high a bandwidth for writes as for reads. On the Pentium III, we enhanced the write bandwidth and also improved the allocation and eviction policies for write-combining (WC) writes, which are most commonly used by multimedia applications. We improved the bus write bandwidth for these writes by 20%. The processor can saturate a 100-MHz bus with an 800-Mbyte/s write throughput. To make this improvement, we removed the dead cycle between back-to-back WC writes that had existed on the Pentium II (Figure 8).

The Pentium II's design aimed at the efficient execution of scalar applications. The average bandwidth requirements of these applications were comparatively small—approximately100 Mbytes/s. Buffers on the processor supported the high instantaneous throughput needed to support a burst of misses in such scalar applications. The SSE, on the other hand, enables the execution of vector algorithms, which demand a high average throughput. Through simulations, we found that to sustain the cumulative read and write throughput of SSE applications, we could reuse buffers that existed in the Pentium II. Therefore, we modified buffer allocation and eviction policies to improve their use. For instance, while the Pentium II allows only one buffer to act as a WC buffer, the Pentium III allows all four buffers to support WC writes. We also provided for fast draining (eviction) of these buffers to reduce the average occupancy time. Thus, in addition to the Pentium II's eviction conditions, the Pentium III supports several new ones. For example, the processor marks a buffer for eviction when all bytes are written (dirty), rather than waiting for a subsequent WC write to evict the buffer.

Cost-effective, scalable implementation

To remain within tight die size constraints and to attain frequency goals, we made several design trade-offs on the Pentium III. A full 128-bit instruction implementation would have had advantages—reduction of decoder bandwidth, less pressure on scheduling and retirement resources—and would have provided headroom for future implementations. However, we found that even with the 64-bit implementation, the computation bandwidth was balanced with the available memory bandwidth for most algorithms of interest. Providing more computation bandwidth without providing more memory bandwidth would not help in most cases.

For example, during the Pentium III processor's existence in the market, the code for the 3D-geometry kernel (a multimedia application with high computation and memory-bandwidth requirements) would process about 10 million vertices per second. Given a standard 32-byte vertex format, the instantaneous bandwidth required to sustain this rate is 640 Mbytes/s (one 32-byte read plus one

32-byte write per vertex). This requirement closely matches the peak bandwidth of an 800-Mbyte/s system bus (with some loss in efficiency due to bus and memory conflicts). We expect this balance to continue as processor frequency and system frequency scale.

Using the CNM approach described earlier, and making several other changes, we contained area growth and frequency impact while reaching a high and balanced computation throughput. One change was the widening of the port 1 write-back bus to 79 bits to accommodate the new floating-point units we were adding on the port.

Another change was the merger of the X87 multiplier with the packed-FP multiplier. This helped us attain significant die area savings while minimizing loading on the ports. Minimizing the load on the write-back buses was an important factor in achieving frequency goals; these buses had proven to be speed limiters in past implementations. We also evaluated merging the X87 adder with the FP adder, but did not follow through on this idea for schedule considerations.

Reusing the multiplier's Wallace tree to support the PSAD instruction was another creative way to achieve die efficiency. The PSAD instruction, which computes the absolute difference of packed integer values, uses three micro-operations: computation of difference, computation of absolute value, and sum of absolutes. To compute the sum of absolutes, we feed the bytes that must be added into the multiplier's Wallace tree, which normally sums the partial products of the multiplication. By reusing existing logic, we implemented the instruction with a very small die and frequency impact. Alternatives for executing this instruction with reasonable performance were significantly more expensive.

Implementation summary

The implementation of a 128-bit instruction set architecture based on a 64-bit data path makes a good trade-off between die size and performance. We implemented a four-wide ISA through two-wide execution units. The parallel two-wide adder and multiplier and improved methods of using peak floating-point performance allowed us to satisfy the computation throughput requirements of demanding and rich visualization applications.

Our motivation for implementing the streaming architecture was to meet the requirements of demanding multimedia applications by providing concurrent data-stream processing. From the hardware implementation standpoint, this means the processor should support concurrent execution of the computational stream and the memory access stream. We accomplished this by decoupling memory prefetch from the retirement of subsequent instructions. Decoupling removes the dependency between the two streams so that each stream's throughput can almost reach the theoretical maximum possible for a given task. Hence, the Pentium III's maximum achievable throughput for a given task equals the task's maximum memory throughput or its maximum computational throughput, whichever is lower.

Optimizing for the Pentium III

To ensure optimum use of the Pentium III's SSE and design features, we categorized multimedia applications into three classes and developed programming recommendations for each.

Computation-bound applications

The increased throughput of the Pentium III's SIMD floating-point units fully supports computation-bound applications such as AC3 audio, which require small memory bandwidth but large computation throughput. The optimization manual describes the techniques we developed to best utilize the Pentium III's computational power.[2]

Memory-bound applications

This category includes 3D processing and imaging applications and several enterprise-class applications. Their distinct feature is a large working set. Their data usually reside in memory, and the cache is not as effective as it is for computation-bound applications.

A typical 3D application consists of a stack of building blocks[5] (Figure 9, next page). The 3D-model database supplies the data for processing. The scene manager places objects in the scene, performs other manipulations of objects, and submits objects to the 3D library. The 3D-library dispatcher determines the type of objects and calls for transformation or lighting functions to perform further processing. The geometry engine performs the requested processing and generates commands to the graphics hardware. The graphics card (GC) driver sends these commands to the hardware.

An obvious SSE application is the acceleration of the geometry engine. Reprogramming the geometry engine to use the extensions can substantially boost performance. However, the application will benefit much more from a systemic software optimization approach. This does not mean reprogramming every building block in the stack. Rather, it means paying particular attention to the application's key input and output data streams. Programming these streams to make balanced use of the Pentium III's computational (SIMD-FP) and memory throughput (prefetch, streaming stores) capabilities enables the application to achieve significant performance gains.

For example, Figure 10a shows the traditional approach to programming the interaction of the geometry engine and GC driver. The geometry engine processes vertices and stores commands in an intermediate buffer. The driver then reads these commands and writes them to the graphics device—for example, to AGP (Advanced Graphics Port) memory. As a result of this sequential operation, the processor cannot execute the write stream (from the driver) concurrently with the data computation stream in the geometry engine. This approach leads to underuse of SIMD-FP and memory throughput and waste of memory throughput due to write-back of dirty data. Removing this bottleneck entails removing the intermediate buffer and writing the commands to AGP memory immediately after generation by the geometry engine, as in Figure 10b. This change significantly improves SIMD-FP and memory pipeline utilization and minimizes dirty write-backs, resulting in a speedup of about 20% at the application level.[5]

Multiple-working-set applications

Some applications, such as video encoding, work with multiple working sets of data, some of which fit in the cache and some of which do not. For these applications, it is important to separate frequently reused from infrequently reused data and to

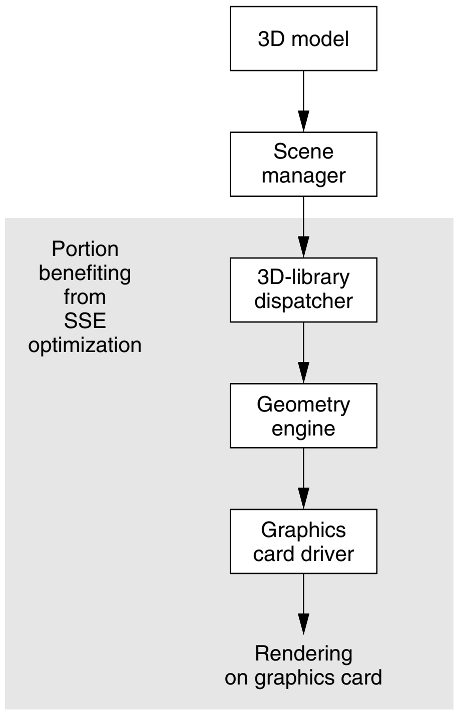


Figure 9. Stack of building blocks in a typical 3D application.

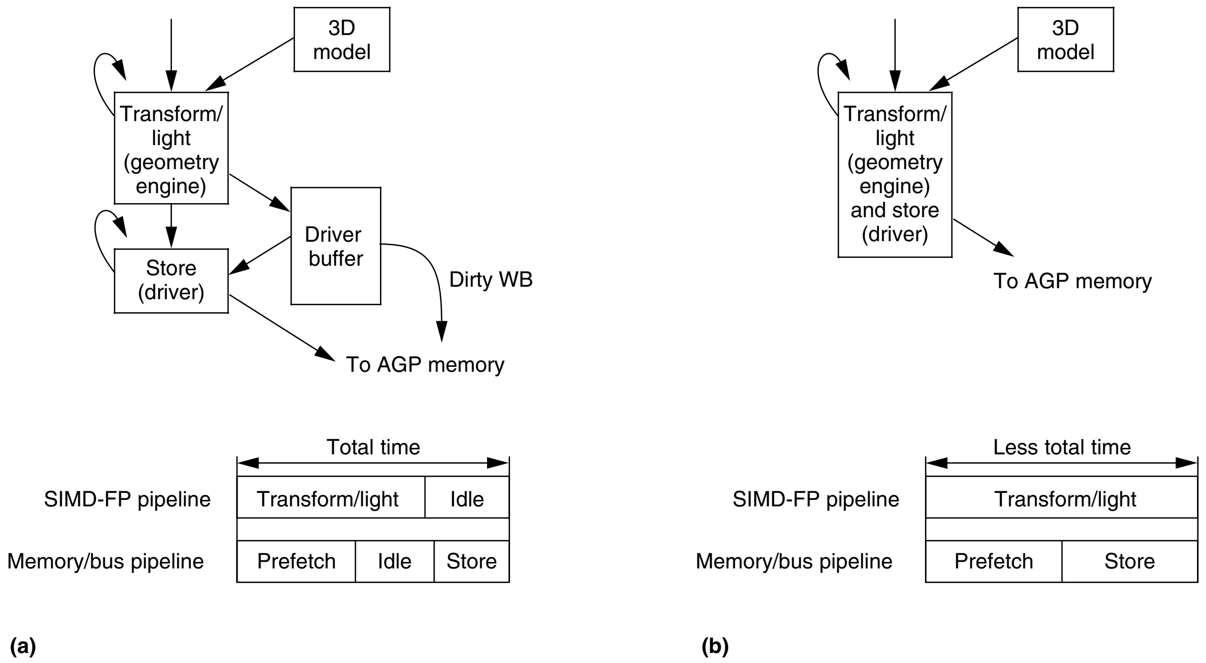


Figure 10. Improving 3D geometry engine throughput: conventional approach using offline driver (a); balanced approach using online driver (b).

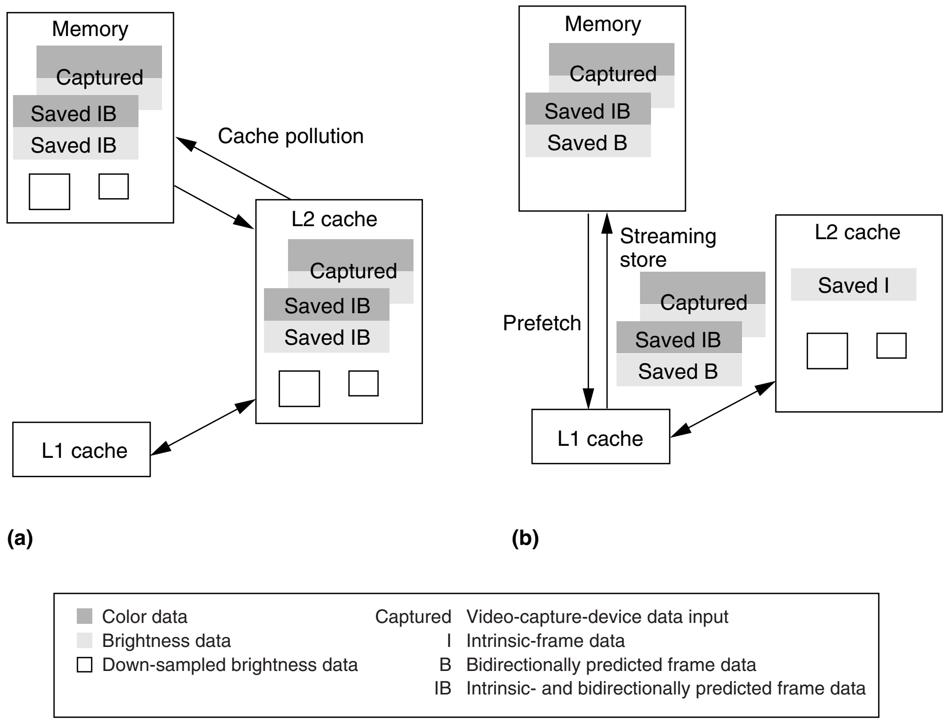


Figure 11. Encoder speedup via smart caching: uncontrolled caching (a); controlled caching (b).

build a caching strategy based on the frequency of reuse. The selective caching capability that the SSE makes available to the programmer is highly useful in building this strategy.

The Pentium III processor's cache control mechanism allows the programmer to explicitly control the location of data in different levels of the cache hierarchy. A typical video encoder application processes input color and brightness data and generates down-sampled data to speed the matching process. The application reuses some of these data—for example, intrinsic-frame brightness data and down-sampled data—much more often than the rest of the data. The programmer can use this reuse pattern as the basis for a caching strategy that keeps the intrinsic-frame brightness and down-sampled data in the cache and the rest in memory.

Figure 11a illustrates the data placement that results when the software does not take advantage of the Pentium III's selective caching capability. The major problem here is that cache pollution prevents the application from accessing frequently used data in the cache, resulting in cache misses. Figure 11b illustrates the use of the prefetch and streaming store instructions to follow a selective caching strategy. The result is that frequently used data are available in the cache, readily accessible for computations, and the miss penalty is significantly less. Following this approach on a video encoder application enabled us to support full MPEG-2 software real-time encoding with audio at 30 frames per second.

Performance

Two benchmarks illustrate the Pentium III's target application performance. The 3D Winbench 99 benchmark measures system-level 3D performance, including that of the processor and the graphics subsystem. To focus specifically on the processor's 3D performance, the benchmark suite includes the 3D lighting and transform test, which measures the CPU-intensive portion of the 3D-graphics pipeline. On this test, the Pentium III is faster than the Pentium II by 62% at the same frequency (450 MHz), and the performance scales almost linearly with further frequency increases.

The MultimediaMark 99 benchmark application suite from Futuremark Corporation tests multimedia performance of a modern PC in a real-world environment. The suite includes MPEG-1 video encoding, MPEG-1 video playback, image processing, and audio effects. On this benchmark, the Pentium III is faster than the Pentium II by 29% at the same frequency (450 MHz), and again its performance scales almost linearly with frequency.

More details on the Pentium III's performance are available at http://developer.intel.com/procs/perf/PentiumIII/index.htm.

The Pentium III processor was introduced in 1999 at frequencies up to 600 MHz. The product has been in production for more than a year and continues to exhibit excellent scaling in its operation frequency—quickly reaching and exceeding 1 GHz—and performance characteristics. Adding the 70 new SSE instructions resulted in only a small increase in die size (approximately 10%), while boosting operation frequency significantly beyond that of the Pentium II. The SSE features enable the processor to provide superior multimedia performance and a rich visualization experience for the end user. MICRO

Acknowledgments

This article represents the work of a large number of people at Intel. We thank Bob Colwell, Glenn Hinton, Dave Papworth, Ticky Thakkar, and Tom Huff for their input during definition and development of the product and the SSE definition team for their quick and diligent definition of the extensions. We also thank the entire Pentium III development team for their commitment and effort. A special thanks to the MPG-FM Architecture Group for making the product development journey an enriching experience.

References

1. T. Thakkar et al., "The Internet Streaming SIMD Extensions," *Intel Tech. J.*, Q2, 1999; also available at http://developer.intel.com/technology/itj/q21999/pdf/simd_ext.pdf.
2. *Intel Architecture Optimization Reference Manual*, 1999, http://developer.intel.com/design/pentiumii/manuals/245127.htm.
3. D.B. Papworth, "Tuning the Pentium Pro Microarchitecture," *IEEE Micro*, Vol. 16, No. 2, Apr. 1996, pp. 8-15.
4. *Intel Architecutre Software Developer's Manual*, *Vol. 2: Instruction Set Reference Manual*, 1999, http://developer.intel.com/design/pentiumii/manuals/243191.htm.
5. V. Pentkovski et al., "Architecture of a 3D Software Stack for Peak Pentium III Processor Performance," *Intel Tech. J.*, Q2, 1999, http://developer.intel.com/technology/itj/q21999/pdf/3d_stack.pdf.

Srinivas K. Raman is a project manager with Intel's Microprocessor Products Group, where he has contributed to the definition and development of Pentium II and Pentium III microprocessor products. Earlier, he defined processor upgrade products as an architect in the End-User Components Division. He originally joined Intel's ASIC core group in Chandler, Ariz., where he designed and developed modular microcontroller cores and core-based products. Raman has an MS in electrical engineering from Pennsylvania State University.

Vladimir Pentkovski is a principal engineer in Intel's Microprocessor Products Group. He led the development of the Pentium III processor architecture and was an architect on the team that defined the streaming SIMD extensions. Previously, he led the development of compilers, software, and programming languages for Elbrus multiprocessor computers in Russia. Pentkovski holds an MS in computer engineering from the Moscow Institute of Physics and Technology and a PhD and Doctor of Science from the Institute of Precise Mechanics and Computer Technology of the Russian Academy of Sciences. He is a member of the IEEE Computer Society and the ACM.

Jagannath Keshava works in Intel's MPG-Folsom Architecture Group on the definition of future microprocessor products. He led the Pentium III definition and microarchitecture validation teams. Earlier, he helped direct design, microarchitecture, and validation in the Pentium II and i960 microprocessor groups. Keshava has an MS in computer engineering from the University of Texas, Austin.

Send questions and comments about this article to Srinivas K. Raman, Intel Corp., FM5-162, 1900 Prairie City Rd., Folsom, CA 95630; srinivas.k.raman@intel.com.