The Internet Streaming SIMD Extensions

Shreekant (Ticky) Thakkar, Microprocessor Products Group, Intel Corp.
Tom Huff, Microprocessor Products Group, Intel Corp.

ABSTRACT

The paper describes the development and definition of Intel's new Internet Streaming SIMD Extensions introduced on the Pentium® III processor. The extensions are divided into three categories: SIMD-FP, New Media, and Streaming Memory instructions. The new extensions accelerate the 3D geometry pipeline by nearly 2x that of the previous-generation processor while enabling new applications, such as real-time MPEG-2 encode. The Pentium III processor implementations achieved the desired goal at a modest 10% increase in die size. The definition achieved the short-term goal while still providing the performance scalability needed for future implementations.

INTRODUCTION

In late 1995, it was becoming clear that visual computing would assume an increasingly important role in the volume PC market segments. To address this need, Intel launched an initiative in visual computing aimed at the 1999 volume PC market segments. This required a balanced platform for 3D graphics performance in order to scale from arcade consoles to workstation applications. Floating-point (FP) computation is the heart of 3D geometry; thus, speeding up FP computation is vital to overall 3D performance.

An increase of 1.5 – 2x the native FP performance in IA-32 processors was required in order to have a visually perceptible difference in performance. 3D graphics applications require the same computation to be performed on 3D data types (vertices), making it amenable to a *Single Instruction Multiple Data* (SIMD) parallel computation model. This is the most cost-effective way of accelerating FP performance of 3D applications in general purpose processors, and it is similar to the acceleration for the class of integer applications provided by the Intel® MMX™ technology extensions [1]. Thus, a SIMD-FP model was selected for the IA-32 extension.

The Instruction Set Architecture (ISA) scope expanded beyond 3D geometry to include feedback on the usage of the MMX technology from independent software vendors (ISV), as well as support for 3D software rendering, video encode and decode, and speech recognition. Cacheability instructions were added to increase concurrency between execution and memory accesses. In all, 70 new instructions and a corresponding new state were added to IA-32 architecture; this is the first new state added since the Intel® i386™ processor added the x87-FP. This paper describes the architectural features and instructions selected as part of the IA-32 definition process.

ARCHITECTURE DEFINITION

2-Wide Versus 4-Wide SIMD-FP

The key component of the new extension was accelerating single precision floating-point computation, which involved the choice of either 2-wide or 4-wide 32-bit floating-point data parallel computations. This crucial decision is discussed later in this paper. This choice shaped key aspects of the new architecture.

An initial feasibility study of SIMD-FP in IA-32 done by the development team indicated that a new microarchitecture could perform 4-wide single precision floating-point operations per clock period, without adding significant complexity or cost to die size. The basic approach was to double-cycle existing 64-bit hardware. The performance benefit of selecting this format was to deliver a realized 1.5 - 2x (or greater) speedup for the geometry pipeline, which supports much greater levels of visual realism.

Another solution for achieving similar gains would be via a superscalar design, by adding execution units. Although this approach may be simpler for a programmer, it incurs a much larger area and timing cost, by increasing busses, register file ports, execution hardware, and scheduling complexity.

Implementing a datapath greater than 128 bits was also not viewed as a reasonable option, again due to balancing cost against performance benefits. Busses and registers were already 80 bits wide due to x87-FP; 128 bits represented only an incremental increase, whereas 256 bits would have a much larger impact. As mentioned, 128-bit execution is actually performed in 64-bit chunks and yet the peak rate of one 128-bit operation can be sustained if, as commonly occurs, instructions alternate between different execution units (i.e., add-multiply-add-multiply). Implementing a 256-bit wide SIMD unit would require doubling the width of execution units in order to still attain peak throughput in the same manner. Increasing SIMD-width beyond 128 bits would also require an increase in memory bandwidth in order to feed the wider execution units. There is a cost to this additional bandwidth, which may not follow Moore’s Law progression, required by other application areas. Also, since the primary focus for the extensions has been on 3D geometry, greater than 4-wide parallelism may offer diminishing returns, since triangular strip lengths in current desktop 3D applications tend to be fairly small (i.e., on the order of 20 vertices per strip).

Related to this decision were the following two issues:

- state space: overlap or new registers
- Pentium® III processor implementation

State Space

There were two choices: overlap the new state with the MMX/x87 FP registers or add a new state. One big advantage of the first choice is that it would not require any operating system (OS) changes, just like the MMX™ technology extension. However, there were many disadvantages with this choice. First, we could only implement four 4-wide 128-bit registers in the existing space since we only had eight 80-bit registers, or we could go to a 2-wide format, thus sacrificing potential performance gains. Second, we would be forced to share the state with MMX registers, which was an issue for the already register-starved IA-32 architecture. The complexity of adding another set of overlapped state was overwhelming.

Adding a new state had the advantage of reducing implementation complexity and easing programming model issues. SIMD-FP and MMX or x87 instructions can be used concurrently. This clearly eased OS Vendor and ISV concerns. The disadvantage of the second approach was that Intel had a dependency of not being able to use new features without OS support. However, Intel worked around this by implementing the new state store and restore instructions in an earlier implementation. Thus by the time the Pentium III processor was released, the new OS's supported this new state.

To ensure no unusual corner cases, all of the new state was separated from the x87-FP state. Figure 1 shows the new 128-bit registers. There is a new control/status register MXCSR which is used to mask/unmask numerical exception handling, to set rounding modes, to set flush-to-zero mode, and to view status flags.

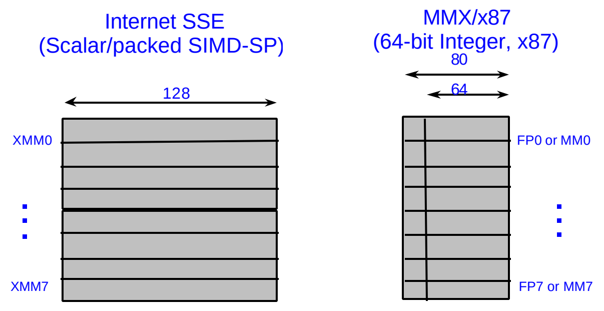

Figure 1: The Internet SSE 128-bit registers

There is also a new interrupt vector to handle SIMD-FP numeric exceptions.

Pentium® III Processor Implementation

The Pentium III processor implements each 4-wide computational macro-instruction as two 64-bit micro-instructions. However, since the processor is a superscalar implementation (i.e., two execution ports), a full 4-wide SIMD operation can also be done every clock cycle (assuming instructions alternate between execution units). With this approach, applications can theoretically achieve a full 4x performance gain; 2x is the realized gain on real applications in part because of micro-instruction pressure within the microarchitecture. A future 128-bit implementation can deliver a higher level of performance scaling.

Scalar Versus Packed Operations

We considered the inclusion of scalar floating instructions in the new SIMD-FP mode because applications often require both scalar and packed operations. It is possible to use x87-FP for scalar and the new registers just for SIMD-FP. However, this approach results in a cumbersome programming paradigm, since x87-FP is a stack register model while the SIMD-FP is a flat register model. Passing parameters would either require more conversion instructions or would be through memory, as currently implemented. Additionally, the results generated via x87-FP operations might differ from SIMD-FP results, due to differences between how computation is performed in the two paradigms (32 bit in SIMD-FP versus 80 bit in x87 FP).

Scalar implementation on the Pentium III processor was problematic because of its emulation of 4-wide SIMD-FP. Using packed instructions for scalar operations would impact performance since both 64-bit micro-instructions would still be executed. Also, it is particularly costly in terms of execution time for long latency de-pipelined packed operations, such as divide and square-root. Lastly, software would need to ensure that no faults occur in the unused slots. To address this issue, explicit scalar instructions were defined, which for the Pentium III processor execute only a single micro-instruction. The upper three components of the source register are passed directly to the destination register when a scalar operation is done; computation is performed only on the lower component pair (Figure 4). Thus, the Pentium III processor did not have to do any operation on the upper half of the data type.

While masked (selective) operations on SIMD-FP registers were another option, we decided against this on the grounds of design complexity and lack of compelling application requirements.

Improving Concurrency

High SIMD performance can only be achieved by balancing memory bandwidth and execution. Multimedia workloads such as 3D graphics and video are streaming applications that have situations where data are largely read once and then discarded. The caches local processors are not large enough to contain the entire data sets of these applications, which results in the execution units being stalled while data are fetched from memory. The out of order and speculative pipeline cannot hide the latency of these accesses without significantly increasing the hardware resources, which impacts die size and cost. A good alternative is to let the programmer overlap execution of one piece of data with the fetch of the next piece so that the latency of the fetch is hidden by the execution time. This works best if the algorithms have a compute-intensive component, such as 3D graphics, where scenes have multiple light sources. Thus we created cacheability hints that allow a programmer to prefetch the next data closer to the processor without touching the architectural state.

For these applications, programmers are the best judges of which data are going to be streaming and which are going to be reused. For example, in 3D graphics, the programmer wants code and transformation matrices to remain in the cache while the input display list and the output command list need to be streamed. This requires some primitives that allow a programmer to manage caching of the data and minimize cache pollution. Thus, the prefetching hints were expanded to let the programmer specify the cache hierarchy level where the prefetched data are going to be placed. Complementary instructions were added to perform non-allocating (streaming) stores so that needed data in the cache does not get replaced, and these stores do not generate unnecessary write-allocation.

The prefetch instructions do not update any architectural state. To some degree, the implementation is specific to each processor. The programmer may have to tune his/her application for each implementation to take full advantage of these instructions. However, it is a design goal to ensure that there are no performance glass jaws between implementations. These instructions merely provide a hint to the hardware: they do not generate exceptions or faults.

Figure 2 illustrates how the various features of the new extensions work together to improve concurrency and reduce total execution time. Prior to Internet Streaming SIMD Extensions, read miss latency and execution and subsequent store miss latency comprised total execution in a serial fashion. The extensions let read miss latency overlap execution via the use of prefetching, and they allowed store miss latency to be reduced and overlap execution via streaming stores. Moreover, SIMD-FP reduces the amount of time spent on execution.

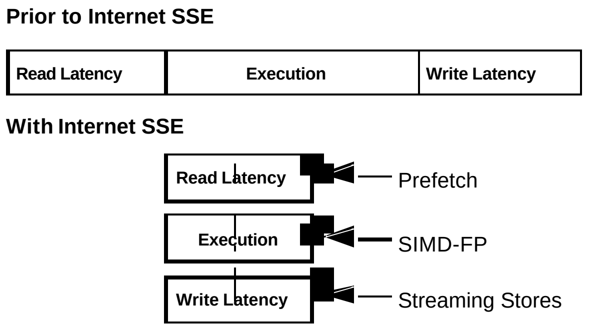

Figure 2: Increasing performance via concurrency

Data Alignment

Hardware support to efficiently handle memory accesses that are not aligned to a 16-byte (128-bit) boundary is expensive in both area and timing. Two options were explored: either detect and fix these cases using a micro-code assist, or generate a general protection fault. ISV feedback was unanimous in their desire to avoid the first option, which can silently introduce a degradation in performance that is difficult to track down. Instead, the ISVs preferred being alerted to misalignment via an explicit fault. As a result, all computation instructions that use a memory operand must be 16-byte aligned. Unaligned load and store instructions are also provided for cases where alignment cannot be guaranteed (i.e., video). These instructions are intended to operate as efficiently or more efficiently than would be the case if explicit code sequences were used to achieve alignment.

Flush-To-Zero and IEEE Modes

We decided to offer two modes of FP arithmetic: IEEE compliance for applications that need exact single-precision computation and a Flush-To-Zero (FTZ) mode for real-time applications. Full IEEE support ensures greater future applicability of the extensions for applications that require full precision and portability, while FTZ mode along with fast hardware support for masked exceptions enables high-performance execution. FTZ mode returns a zero result in an underflow situation during computation if the exceptions are masked. Most real-time 3D applications would use the FTZ mode since they are not sensitive to a slight loss in precision, especially if they can get faster execution by using the FTZ mode.

INSTRUCTION SET ARCHITECTURE

The Internet SSE supplies a rich set of instructions (shown in Table 1) that operate on either all, or the least significant pairs, of packed data operands in parallel. The packed instructions (with PS suffix) operate on a pair of operands as shown in Figure 3 while scalar instructions (with SS suffix) always operate on the least significant pair of the two operands as shown in Figure 4.

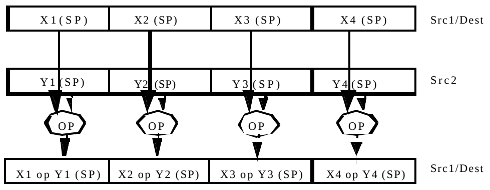

Figure 3: Packed operation

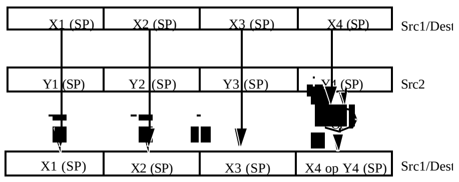

Figure 4: Scalar operation

		Packed Single	Scalar Single	Packed Integer
Arithmetic	ADD, SUB, MUL, DIV, MAX, MIN, RCP, RSQRT, SQRT	X	X	
Logical	AND, ANDN, OR, XOR	X		
Comparison	CMP, MAX, MIN	X	X	
	COMI, UCOMI		X	
Data Movement	MOV (load/store aligned),	X		
	MOVUPS (load/store unaligned), MOVLPS, MOVLHPS, MOVHPS, MOVHLPS (load/store), MOVMSKPS	X		
	MOVSS (load/store)		X	
Shuffle	SHUFPS, UNPCKHPS, UNPCKLPS	X		
Conversions	CVTSS2SI, CVTTSS2SI, CVTSI2SS		X	
	CVTPI2PS, CVTPS2PI, CVTTPS2PI	X		
State	FXSAVE, FXRSTOR, STMXCSR, LDMXCSR	X		
MMX^{TM} Tech Enhancements	PINSRW, PEXTRW, PMULHU, PSHUFW, PMOVMSKB, PSAD, PAVG, PMIN, PMAX			X
Streaming/ Prefetching	MASKMOVQ, MOVNTQ (aligned store)			X
	MOVTPS (aligned store)	X		
	PREFETCH			
	SFENCE			

Table 1: Internet SSE ISA

Basic Building Blocks

These include instructions such as load, store, addition, multiplication, subtraction, division, and square root as well as instructions to access the new Control/Status Register and Save/Restore new state.

Cacheability Hints

As mentioned earlier, achieving high performance for multimedia applications requires some degree of overlap between the execution of a block of data and the fetch of the next block of data. The PREFETCH instruction was added to the new extensions to let the programmer control overlap in the application. This instruction also allows control over data placement in the cache hierarchy and further allows programmers to distinguish between the locality of temporal (i.e., frequently used) cached data and non-temporal (i.e., read and used once before being discarded) data. There are four possible prefetches currently defined with room for future definitions. Note that these instructions can also be used for non-SIMD applications.

Streaming store instructions, MOVNTPS (Packed Single Precision FP) and MOVNTQ (Packed Integer) allow the programmer to specify a write-combining (WC) memory type on a per instruction basis. This is true even for memory otherwise assigned a writeback (WB) memory type via the Memory Type Range Register's (MTRRs) or Page Attribute Table (PAT). This allows the user to obtain all the benefits of a WC memory type (i.e., write-combining, write-collapsing, uncacheable, non-write-allocating) on a per-instruction granularity.

Fencing

In order to allow efficient software-controlled coherency, a light-weight fence (SFENCE) instruction was also included in the new extension; this instruction ensures that all stores that precede the fence are observed on the front-side bus before any subsequent stores are completed. SFENCE is targeted for uses such as writing commands from the processor to the graphics accelerator or to ensure observability between a producer and consumer where communication of data uses stores with a WC memory-type semantic.

Comparison and Conditional Flow

The basic single precision FP comparison instruction (CMP) is similar to existing MMX instruction variants (PCMPEQ, PCMPGT) in that it produces a redundant mask per float of all 1's or all 0's, depending upon the result of the comparison. This approach allows the mask to be used with subsequent logic operations (AND, ANDN, OR, XOR) in order to perform conditional moves. Additionally, four mask (most significant of each component) bits can be moved to an integer register using the MOVMSKPS/PMOVMSKB instructions. These instructions simplify data-dependent branching, such as the clip extent and front/back-face culling checks in 3D geometry, and they address a common desire registered by many ISVs.

Another important conditional usage model involves finding the maximum or minimum between two values (packed or scalar). While this can be done as just described for conditional moves, the MAX/MIN and PMIN/PMAX instructions perform this function in only one instruction by directly using the carry-out from the comparison subtraction to select which source to forward as a result. Within 3D geometry and rasterization, color clamping is an example that benefits from the use of MINPS/PMIN. Also, a fundamental component in many speech recognition engines is the evaluation of a Hidden-Markov Model (HMM); this function comprises upwards of 80% of execution time. The PMIN instruction improves this kernel performance by 33%, giving a 19% application gain.

To provide a complete set of comparisons for CMP, an 8-bit immediate is used to encode eight basic comparison predicates, EQ, LT, LE, UNORD, NEQ, NLT, and NLE. Another four can be obtained by using these predicates and swapping source operands. Using an immediate to encode the predicate greatly reduces the number of opcodes that are assigned to these comparison operations.

Data Manipulation

SIMD computation gains are only realized if data can be efficiently reorganized into an SIMD format. For example, 3D geometry transformation with 4-wide SIMD-FP format can be done per vertex or on four separate vertex components, where a vertex has four components (x, y, z, and w). The method of organizing 3D data structures on a per vertex basis is called Array-of-Structures (AOS) since the display list is an array of individual vertices. Organizing the display list for an ideal SIMD format is called Structure-of-Arrays (SOA) since the structure contains separate x, y, z, and w arrays. An AOS approach is less efficient for two reasons: 1) not all SIMD computation slots may be utilized (i.e., the w vertex component may not be needed); 2) horizontal reduction operations (i.e., dot products such as a * x + b * y + c * z) are typically needed, which use multiple SIMD slots to generate only one unique scalar result. This is exacerbated if other long-latency operations (i.e., square-root and division) are used in conjunction with the horizontal reduction.

Often, it may not be possible to statically reorganize data if for example, in 3D geometry, either a standard API or the rasterization graphics controller do not directly support SOA. In order to efficiently transpose data into the ideal SOA format or vice versa, the new extension supports a number of data manipulation instructions, including the following:

- UNPCKHPS/UNPCKLPS. These interleave floats from the high/low part of a register or memory operand, similar to the MMX unpack instructions.
- SHUFPS/PSHUFW. These support swizzling of data from source operands, including such operations as broadcast, rotate, swap, and reverse.
- MOVHPS/MOVLPS. When used in conjunction with SHUFPS, these 64-bit load/store instructions enable efficient gathering of four individual vertex components from four non-adjacent AOS data structures into a single 128-bit register (SOA); these instructions can be similarly used to de-swizzle SOA to AOS.
- PINSRW/PEXTRW. These support scatter and gather operations on words within an MMX register from memory or the 32-bit integer registers. Examples include gathering texture components and supporting SIMD lookup tables. The PINSRW instruction also gives a performance gain of 22% for the Hidden-Markov Model (HMM) based speech

recognition kernel and a 13% gain at the application level.

A number of experiments were run using various 3D transform/lighting building blocks as well as a more complete geometry pipeline. Of the approaches described for utilizing SIMD computation, the static SOA case achieves the best performance. Computing directly in AOS format achieves only half of the static SOA throughput for a geometry benchmark that implements a full lighting case (ambient, diffuse, specular) due to the reasons listed above. Dynamic reorganization from AOS to SOA and vice versa using a combination of SHUFPS/MOVHPS/MOVLSP instructions incurs a 20%-25% overhead compared to static SOA, which is a 6%-10% better performance than is possible with other methods. Note this overhead is constant and diminishes as more SIMD computation is performed (e.g., with additional lights). Computing directly in AOS may appear to provide a simpler programming model since most APIs handle display lists on a per vertex basis. In order to improve performance for the horizontal operations that can result from computing in an AOS format, several additional instructions, MOVLHPS/MOVHLPS, were added to the extensions. These instructions support emulating a full range of horizontal operations, including addition, subtraction, and logic operations. However, better performance can generally be achieved by computing in an SOA form, and the transpose code used with dynamic reorganization can be effectively hidden behind compiler pragmas or intrinsics.

Conversions

A large number of conversion operations are possible, including integer to/from FP, scalar and packed, source and destination of either register or memory, rounding mode contained implicitly within the instruction, and integer operand sizes (byte, word, double-word). A full set of all permutations is impractical and unnecessary since not all possible cases are common, and many others can be emulated by a sequence of instructions. The factors that motivated the final definition include the following:

- Basic operations between integer and FP are required with both SMID-FP and MMX^{TM} technology for packed data (CVTPI2PS, CVTPS2PI, CVTTPS2PI) and Scalar-FP and IA-32 Integer for scalar data (CVTSS2SI, CVTTSS2SI, CVTSI2SS).
- Destination is a register, since, if needed, the result can be explicitly moved to memory using a store.
- CVTTPS2PI/CVTTSS2SI implicitly encode truncation rounding to eliminate the common serialization performance penalty of changing the control register rounding mode when converting FP to integer.
- Internet SSEs support only conversions to/from double-words. Existing MMX pack and unpack instructions can be used to efficiently translate from double-words to/from words and bytes.

Reciprocal and Reciprocal Square Root

A basic building block operation in geometry involves computing divisions and square roots. For instance, transformation often involves dividing each x, y, z coordinate by the W perspective coordinate. Similarly, specular lighting contains a power function, which is often emulated using an approximation function that requires a division. Also, normalization is another common geometry operation, which requires the computation of 1/square-root. In order to optimize these cases, the new extensions introduce two approximation instructions: RCP and RSQRT. These instructions are implemented via hardware lookup tables and are inherently less precise (12 bits of mantissa) than the full IEEE-compliant DIV and SQRT (24 bits of mantissa). However, these instructions have the advantage of being much faster than the full precision versions. When greater precision is needed, the approximation instructions can be used with a single Newton-Raphson (N-R) iteration to achieve almost the same precision as the IEEE instructions (~22 bits of mantissa). This N-R iteration for the reciprocal operation involves two multiplies and a subtraction, so the overall latency and especially the throughput are lower than the IEEE instructions. For a basic geometry pipeline, these instructions can improve overall performance on the order of 15%.

Unsigned Multiply, Byte Mask Write

Discussions with the D3D team, among others, identified the lack of an unsigned MMX multiply operation as the reason for inefficiency in 3D rasterization performance. This function inherently works with unsigned pixel data, and the existing PMULHW instruction operates only on signed data. Providing an unsigned PMULHUW eliminates fix-up overhead required in using the signed version, creating an application-level performance gain of 8%-10%.

The byte-masked write instruction, MASKMOVQ, is aimed at specific rasterization and image processing applications. The instruction supports several beneficial features:

- A byte mask, either generated by a PCMPEQ/ PCMPGT instruction or loaded from memory, is used to selectively write bytes in the other source operand directly to memory. This mask is effectively transferred in a parallel fashion along with the data throughout the memory subsystem (i.e., write-combining buffers, bus queue entries, and bus byte enables). Alternative implementations with conditional moves or branches did not deliver as much of a performance gain because they introduce significantly more micro-operations into the execution pipeline as well as possible miss-predictions for the branch approach.
- Similar to other non-temporal streaming store cacheability instructions, MASKMOVQ implements a WC memory semantic, which eliminates unnecessary read-for-ownership bandwidth and disturbance of temporal cached data, since the cache is bypassed altogether.

Packed Average

Motion compensation is a key component of the MPEG-2 decode pipeline. It is the process of reconstituting each frame of the output picture stream by interpolating between key frames. This interpolation primarily consists of averaging operations between pixels from different macroblocks where a macroblock is a 16x16 pixel unit. The MPEG-2 specification requires that the result of the averaging operation be rounded to the nearest integer; values precisely at half way should be rounded away from zero. This is equivalent to operations with 9-bit precision. MMX instructions provide either 8 or 16 bits of accuracy, and it is desirable to use the 8-bit versions to increase the data parallelism. The PAVG instruction facilitates the use of 8-bit instructions by performing a 9-bit accurate averaging operation. The word version PAVGW provides higher accuracy for applications that accumulate a result using several computation instructions.

Currently, Motion Compensation of a DVD player on a Pentium® II processor-based system (266MHz) is evenly balanced between memory latency and execution. The PAVG instruction enabled a 25% kernel speedup. Instrumenting the motion compensation code in the player with the PAVG instruction provided a 4%-6% speedup at the application level (depending on the video clip chosen). The application level gain can increase to 10% for higher resolution HDTV digital television formats.

Packed Sum of Absolute Differences

Although the video encode pipeline is quite complex and involves many stages, the bulk of the execution is spent in the motion-estimation function (40%-70% at present). This stage compares a sub-block of the current frame with those in the same neighborhood of the previous and next frames in order to find the best match. Consequently, only a vector representing the position of this match, and the residual difference sub-block, needs to be included in the compressed output stream as opposed to the entire original sub-block.

There are two common comparison metrics that are used in motion-estimation: sum-of-square-differences (SSD) and sum-of-absolute-differences (SAD). Both have benefits and limitations in specific cases, although overall they are roughly comparable metrics in determining the quality of a match.

There is a factoring technique that allows SSD to be implemented using an unsigned multiply-accumulate (byte to word) operation; however, the accumulation range requires 24 bits of precision, which does not map neatly to a general purpose data-type. Instead, the PSADBW instruction retains byte-level parallelism of execution, working on 8 bytes at a time, and the accumulation does not exceed a 16-bit word. This single instruction replaces on the order of seven MMX instructions in the motion-estimation inner loop, largely because MMX technology does not support unsigned byte operations, which need to be emulated by zero extension to words and the use of word operations. Consequently, PSADBW has been found to increase motion-estimation performance by a factor of two.

CONCLUSION

The Internet Streaming SIMD Extensions enable an exciting new level of visual computing on the volume PC platform. The single precision SIMD-FP ISA will deliver the desired performance goal of 2x an increase in FP performance with the Pentium® III processor. This speedup will significantly improve the image quality for real-time 3D applications, and the Pentium III processor systems will enable real-time rendering of complex worlds. This instruction set represents a significant step forward for Intel in improving the performance of visualization on PC platforms.

The addition of SIMD-integer instructions for video will enable real-time video encoding in the MPEG-1 format, as well as the MPEG-2 format, with some trade-offs in visual quality and compression rates. The new extensions will also deliver DVD decode at 30 frames per second within the Pentium III processor timeframe, with good headroom for multitasking other processes. Increasing Pentium III processor frequency will subsequently enable 1M-pixel HDTV format decode. Initially this will require hardware motion compensation support, but with an incremental increase in processor frequency, this decode can be handled entirely in software. These instructions will also enable home video editing similar to that currently available for photographic editing.

A reduction in speech recognition error rates and an increase in dictionary size can be achieved with the use of the prefetching options and the new packed integer instructions. Concurrency of memory accesses and execution have also been enhanced across the range of multimedia applications via the new cacheability instructions.

The definition team delivered the new ISA in record time, working diligently to review all the requested instructions and analyzing the potential improvement in application performance. Intense scrutiny was applied to the definition by the three implementation teams to justify inclusion of instructions. A range of constraints drove the final definition, including performance benefits, die size, timing impact, and code portability. The Internet SSE implementation cost in the Pentium III processor was just around 10% of the die size. This is similar to the cost of including MMX^{TM} technology in the Pentium® and Pentium® II processors.

ACKNOWLEDGMENTS

We acknowledge the contribution of the entire definition team, in particular Srinivas Chennupaty, Patrice Roussel, Vladimir Pentkovski, and Mohammad Abdallah. We also acknowledge the adoption of IA-64 definition and semantics for a number of instructions to maintain consistency. We also acknowledge the guidance of Glenn Hinton, Bob Colwell, and Fred Pollack during the definition period.

REFERENCES

[1] Alex Peleg, et al, “MMX^{TM} Technology Extension to the Intel® Architecture” DTTC Proceedings 1996 (Internal Intel Document).

[2] Millind Mittal, et al, “MMX™ Technology Architecture Overview,” Intel Technology Journal, Q3, 1997, http://developer.intel.com/technology/itj/q31997.htm.

AUTHORS’ BIOGRAPHIES

Shreekant (Ticky) Thakkar is a principal processor architect in the Microprocessor Products Group. He led the Internet Streaming SIMD Extension development for the Pentium® III processor. Prior to that, Shreekant was responsible for the development of the Pentium® Pro multi-processor. His e-mail is ticky.thakkar@intel.com.

Tom Huff is an architect in the Microprocessor Product Group in Oregon. He was one of the architects in the core team that defined the Internet Streaming SIMD Extensions for the IA-32 architecture. He is currently working on multimedia performance analysis for the Willamette processor project. He holds M.S. and Ph.D. degrees in electrical engineering from the University of Michigan. His email is tom.huff@intel.com.