Intel Developer Forum
Spring 2001

Optimizing for the Pentium® 4 Processor

Carl Dichter
AZ Site Mgr, Application Solution Center

Stephen Moore
ISV Performance Lab (MSL)

Intel Corporation

Intel Labs

Agenda

- Course Objectives
- Pentium® 4 processor Architecture
- Tools & Methods
- Streaming SIMD Extensions 2
- Tuning Tips
- Call to Action
- References

Course Objectives

- You should gain an understanding of:
  - the new instructions and architecture of the Pentium® 4 processor
  - pros/cons of each development option
  - tuning tips for how to get the most out of the Pentium 4 processor

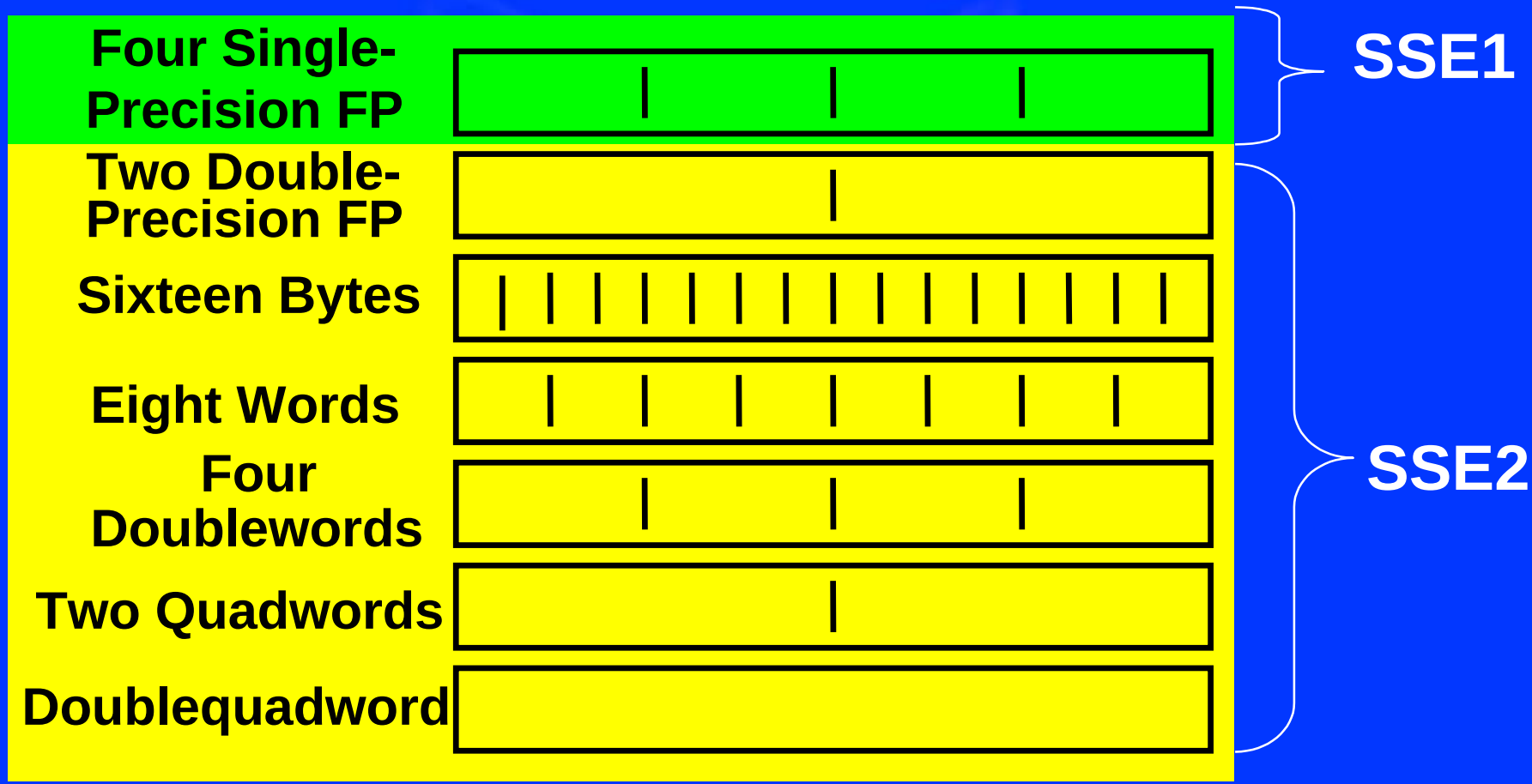
A new architecture

- P6 micro-architecture
  - Pentium® Pro processor some new instructions
  - Pentium® II processor included Pentium Pro processor instructions along with MMX technology
  - Pentium® III processor Streaming SIMD Extensions
- NetBurst™ micro-architecture
  - Pentium® 4 processor
    - Streaming SIMD Extensions 2
    - New register formats
    - Instruction trace cache
    - Faster bus bandwidths

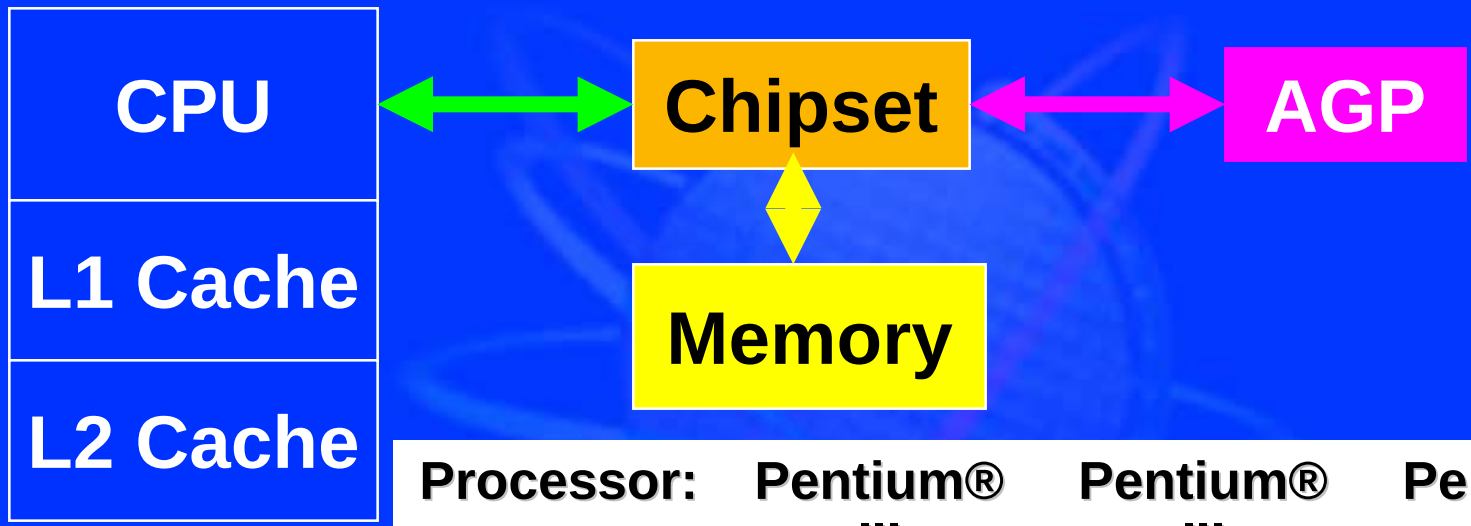
Key Capabilities

- New Processor Design
  - Building the foundation for present and future performance
    - Deeper pipeline enables higher speeds, more throughput
    - Better branch prediction
    - High performance multimedia unit
- More Bus Bandwidth
    - 400 MHz System bus supports 3.2 GB/sec
    - 64B cache line size
    - Better buffering technology
- Streaming SIMD Extensions 2
    - 128b Integer, 2x64b FP, More cache control

XMM Registers



New Bus Bandwidths



Processor:	Pentium® III	Pentium® III	Pentium® 4
Chipset:	440BX	820	850
FSB:	800MB/s	1GB/s	3.2GB/s
AGP:	500MB/s	1GB/s	1GB/s
Memory:	800MB/s	1.6GB/s	3.2GB/s

New Architecture and faster buses remove bottlenecks, change your "hotspots"

Intel Labs

Tools

- VTune™ Performance Enhancement Environment, Special Edition CD
  - Intel Compilers, VTune™ Analyzer, Assembler Macros, Performance Libraries, Intel® Architecture Performance Training Center
- Microsoft* Processor Pack
  - For Visual C++* 6.0 SP3 or part of 7.0
- NuMega Driver Studio w/SoftICE*

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

Let the tools do the work!

Intel Compilers

- **Intel® C/C++ and Fortran Compilers:**
  - **Integrates with Visual Studio***
  - **Supports inline assembler and intrinsics, vector class and performance libraries**
  - **Profile-Guided Optimizations**
  - **Vectorization**

VTune™ Analyzer

- Code Coach
  - Advisor for optimization
- Event-based sampling
  - Time or event-based, low intrusion
- Dynamic Analysis
  - Call graphing, more instrusive

Intel Performance Libs

- Intel® Performance Library Suite
- Highly-tuned libraries:
  - Math Kernels
  - Signal processing
  - Image processing
  - Speech recognition
  - JPEG encode/decode
- Tuned for each processor generation

IA Performance Training Center

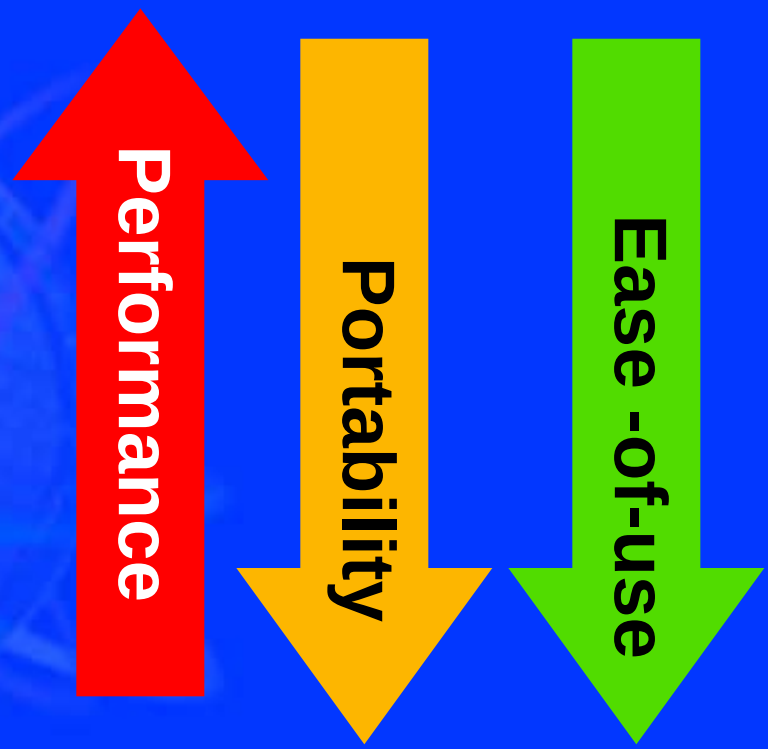
- Computer-based training for Pentium® 4 Processor's Streaming SIMD Extensions 2
- Pentium 4 Processor Optimization Manual
- Application notes showing tuned algorithms
- Processor documentation

Microsoft* Visual C++* Processor Pack

- Adds features to Visual C++ 6.0 SP3:
  - inline assembly and intrinsics
  - run-time exception handling
  - supports Streaming SIMD Extensions and Streaming SIMD Extensions 2
  - MASM and debugger support of new instructions and registers
  - Visual C++* 7.0 has this built in

Coding Alternatives

- Assembler
- Intrinsics
- C++ Vector Classes
- Vectorizing Compiler
- Performance Libraries
  - The exception– high performance with EOU and portability



Vectorization and Libraries: Easiest, Most Portable Way to Optimize!

New Instruction Overview

- Double Precision SIMD FP
- Extended SIMD Integer
- Cacheability
- Pause

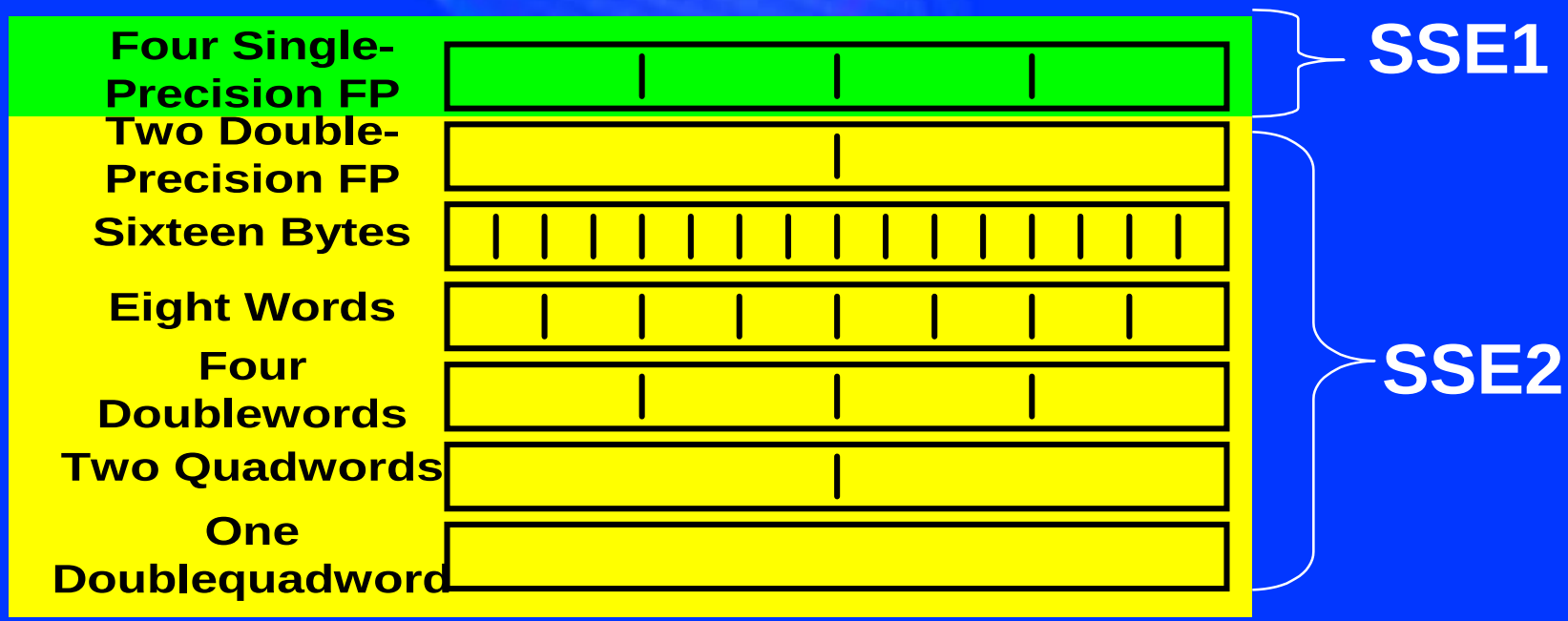
New instructions and architecture provide optimization opportunities

Double Precision SIMD FP

- Two x Double Precision Floating Point
  - in addition to the Four x Single Precision
- Same functionality as SIMD Single Precision Floating Point instructions
- Conversion instructions provided
  - Packed and scalar conversions for single precision to/from double precision

Extended SIMD Integer

- Uses XMM registers, like SIMD SP FP
  - Doesn't require EMMS instruction



Building block 128 bit instructions

- 128 bit Shift (bytewise)
- 128 bit Shuffle
- 128 bit Unpack
- 128 bit Moves
- 128 bit Conversion
- 64-bit Move from MM<->XMM

Cacheability

- CLFLUSH - Flush cache line
- LFENCE - Load fence
- MFENCE - Atomic combined load and store fence
- MOVNTI - Move integer non-temporal

New Moves

- Aligned/unaligned 128 bit moves
  - MOVDQA/MOVDQU, MOVAPD/MOVUPD
- Added SIMD Conditional Move
  - MASKMOVDQU - byte-wise 128-bit conditional move
- Added non-temporal moves (aligned)
  - MOVNTPD - move 2 double-precision FP
  - MOVNTDQ - move 2 quad word ints

PAUSE: for spin waits

- PAUSE
  - Backward compatible with all architectures
  - Makes spin-wait loops exit faster
  - Lowers power consumption
  - wait_loop: pause
    cmp eax, sync_var

The Pentium® 4 processor's new instructions and architecture provide optimization opportunities

Performance Counters

- Many new performance counters
- Described in hard cover manuals
  - Updated information via internet
- Vtune™ Analyzer has extensive support for these counters
  - No ‘skid’ for some counters

Great Performances

- New architecture calls for action
  - Learn implications of processor design choices
  - Take advantage of processor strengths
  - Use new instructions

Pipelining considered

- Why longer pipelines for Pentium® 4 processor?
  - Less work per stage means faster clock
  - More throughput
- But:
  - Cost of misprediction is greater
  - Latency of single instruction may be greater
    - Integer mul/shifts/rotates have longer latencies
- However:
  - Processor has better prediction mechanisms

The Road Not Taken

- Avoid branches altogether with:
  - SIMD and scale conditional move
  - SIMD average and sum Absolute Differences
  - Clamp/Saturate
  - Select values

Goto considered harmful

- Function Pointers (CALL instructions) are always a branch:
  - Major stall on first call
  - Predicted to take same path next time
    - Therefore only tiny stall
- Conclusion:
  - Avoid function pointers for dispatch
  - Suitable for implementing modes, other semi-static operations
  - Predictable pattern? Use if/else

Be Predictable

- Use static branch prediction rules:
  - IF / ELSE branch
    - IF path predicted taken
  - WHILE loop
    - Prediction is that the loop will continue

Unrolling your own

- Trace cache improves out-of-order execution
  - Less need to unroll loops (about 10 max)
  - Consider total size of unrolled loops
    - Excessive unrolling may clobber trace cache
- Good reasons to unroll
  - Improve branch prediction by unrolling irregular loops
  - Allow more aligned memory accesses (when some iterations hit unaligned data)
    - But consider using if/else instead

Let Someone Else Do It

- Profile-guided optimization
  - You can't control compiler by clause order
  - But the compiler can insert branch hints
  - The best predictor of branches is actual usage
- Three easy steps with the Intel compiler
  - Instrumented compile
  - Execute the instrumented code
  - Feedback Compilation

Two as Cheap as One

- SSE2 instructions increase compute power
  - 32 x 32 integer multiply (full 64 bit result)
  - 64 bit SIMD addition
    - Big speedup for RSA cryptography (4 to 10x)
  - SIMD Double Precision Floating Point
    - 2 64-bit operations in a 128-bit XMM register
    - Speed up technical computing apps (~1.5 to 2x)
  - Double Wide MMX logical operations
    - 128 bit MMX instructions
    - Speedup video, imaging (~1.1 to 1.6x)

PAUSE for a moment

- Use in spin-wait loops
  - Hints to hardware that program is in a spin loop
  - Reduces out-of-order execution loop exit penalty
  - Lowers power usage
- PAUSE is backward compatible
  - No CPUID required
  - PAUSE is a NOP on previous processors
- Microsoft* Windows 2000* is already PAUSE compliant

Movin' On

- Data Movement considerations
  - Prefetch where needed (1.1-1.3x gain)
  - Avoid partial writes (1.1-1.3x gain)
    - Insure full WC B/W to graphics H/W
  - Avoid stall cases (1.1-1.3x gain):
  - Avoid Store-to-load forwarding penalties
  - Avoid cache line splits

Getting things done early

- Prefetch hides latency behind computation
- PrefetchNTA is often best (1.1-1.15x gain)
  - Reduces cache evictions of useful data
  - Can maximize read B/W to system memory
- Does Pentium® 4 processor change this?
  - No. Pentium® III processor techniques work well.
    - But consider 64B (vs. 32) on Pentium® 4 processor
  - But the Pentium 4 processor has hardware prefetch that may diminish effect of software prefetch
  - Use type and location of prefetch in code
  - Increase fetch-ahead distance as memory-latency/computation increases

Exercising your writes

- Cache and buffering features make it profitable to eliminate partial writes
- 64B WC buffers vs. 32B PIII
  - May require code changes to PIII code to avoid partials.
- Full concurrency between read, write and compute (1.1-1.15x gain)

Partial Writes: W/C

Causes:

1) Too many WC streams

2) WB loads/stores contending for fill-buffers to access L2 cache or memory

Detection (VTune)

Event based sampling:

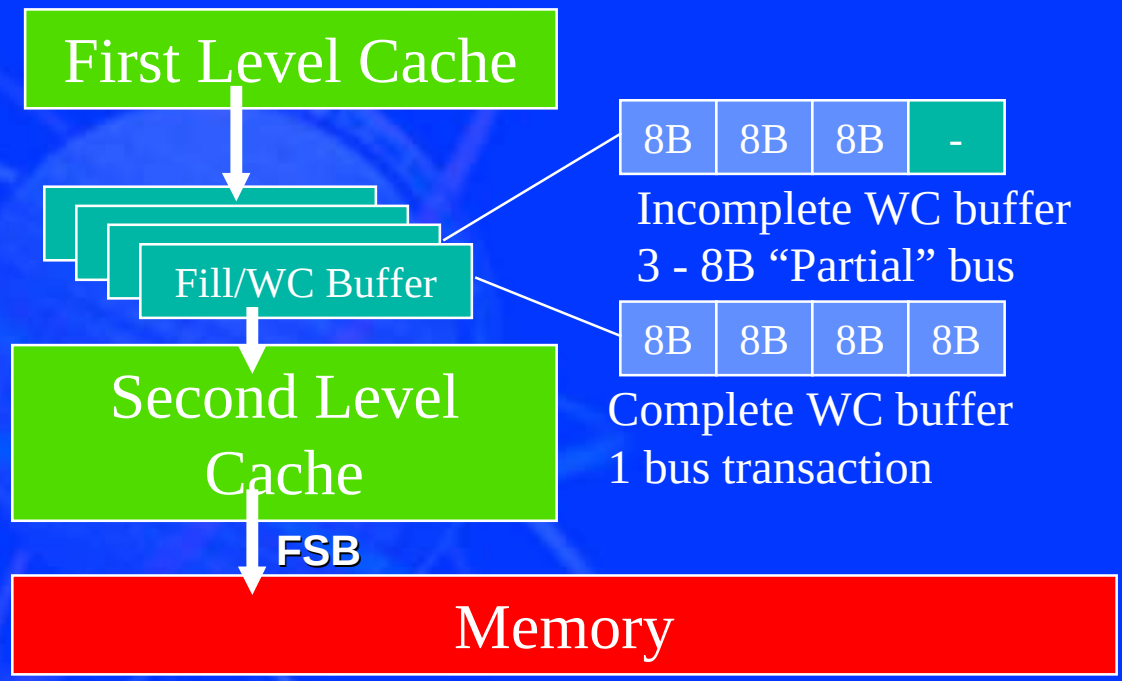
Ext. Bus Partial Write Trans.

Causes:

L2 Cache Request

Ext. Bus Burst Read Trans.

Ext. Bus RFO Trans.



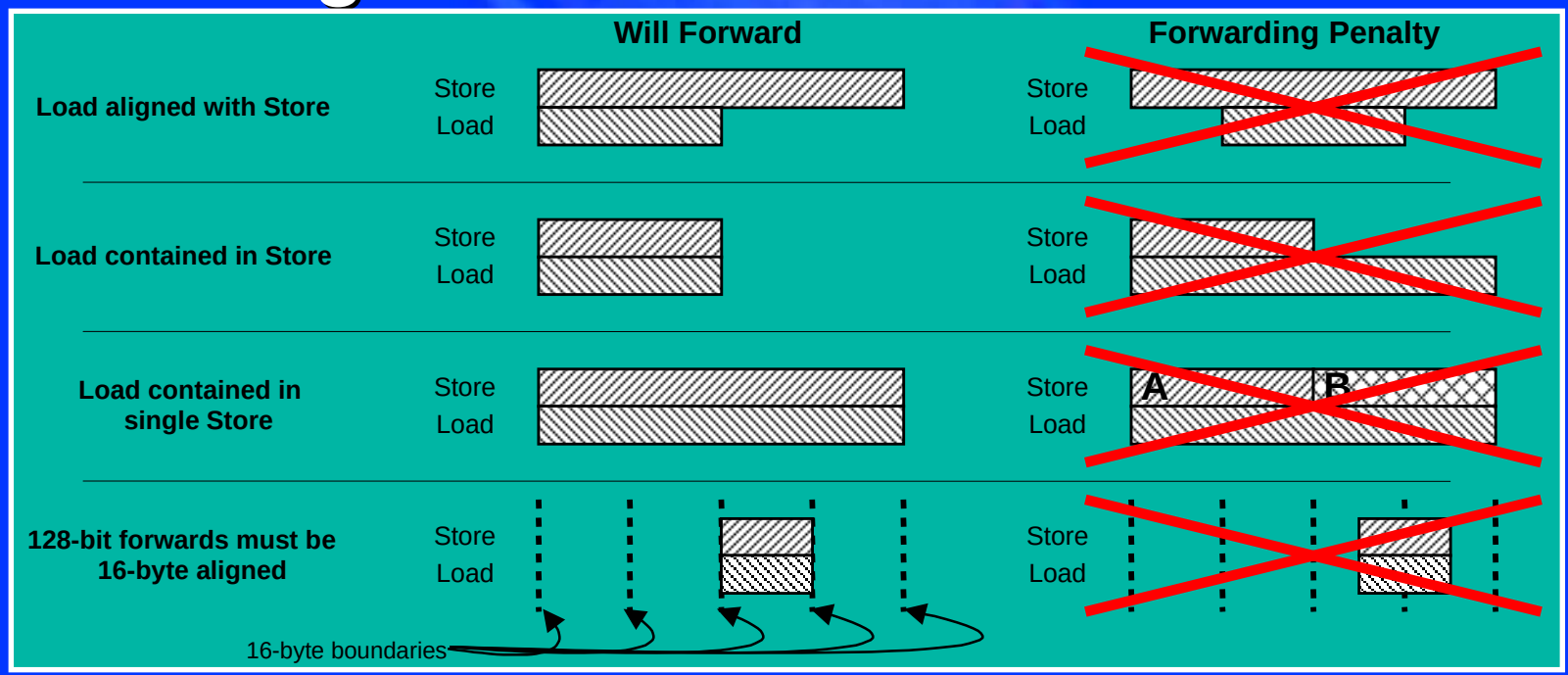
Partial writes reduce front-side bus B/W

- **~3x lower for Pentium® III processor**
- **~7x lower for ~Pentium® 4 processor (due to longer line size)**

Intel Labs

Store Forwarding

- Allows efficient load after store if you do it right



MSVC < 7.0 can have these penalties. Intel Compiler doesn't.

Doing Cache Business

- Avoid cache-line splits on loads (1.1-1.2x gain)
  - Align data structures to natural boundaries of access size
  - Select 8, 16, 32-byte alignments to avoid cache-line split accesses
    - Don't be fooled by SIMD accesses!
      - Element data type is immaterial – it's the access size!
  - Misalignment penalty is bigger on Pentium® 4 processor

Loose Ends

- Use DQWord loads/stores (1.1x gain):
  - Best use of cache and buffering resources
  - Good bandwidth for memory copies
- Use hybrid SOA data structures for greatest DRAM efficiency (1.1x gain)
  - Ensures better use of fetched lines, more important with increased cache line size
  - Less DRAM page misses
- Avoid sparse data structures

Cast away

- X87 casting is expensive
  - Rounding cheaper than truncation cast
    - i = (int) myFloat; // expensive
    - FISTP // cheaper
  - But rounding mode change may be expensive
- SIMD conversions are efficient
  - Examples: CVTTPD2DQ, CVTTPQ2PD

Slow Floats

- X87-Floating Point considerations
  - Transcendentals (FSIN, FCOS, etc) have longer latencies
    - Use Approximations (series expansion, Lookup)
  - Serialization penalty is greater for FLDCW (avoid changing for FP->Int conversions)
    - Flip-flopping between 2 values does not serialize

Denormal Exceptions

- In some cases we have values which are very close to zero
  - Arithmetic operation may give de-normal value result
  - When used subsequently causes exception
  - May also result from constants, coefficients, etc
  - Penalty hundreds of clocks
- Set appropriate mode (~1.1-1.5x gain)
  - FTZ: Flush output denormals To Zero
  - DAZ: Flush input denormals to zero

Pentium® 4 Processor Cheat Sheet

Intel Developer Forum Spring 2001

DO:

Use New ISA

- Double precision SIMD FP (use vectorizing compiler)
- Double-wide integer SIMD (XMM)
- 32x32 multiply for 'RSA-style' encryption

Exploit Data Movement

- Use hybrid SoA
- Hardware prefetcher
- 400 MHz FSB, AGP

Miscellaneous

- Use FTZ, DAZ FP modes
- Use PAUSE in spin loops

Tools

- Use Intel compilers or MSVC 7 beta
- Vtune™ Analyzer for Pentium 4 processor release

Avoid:

- Branch mispredictions (use logical ops if possible)
- Store forward problems
- Misaligned accesses and DCU splits
- Trace cache thrash, excessive unrolling
- Partial writes
- Sparse data structures
- SW Prefetches that are already handled by HW prefetcher
- Denormals
- Instructions with longer latencies…
  - int shifts/rotates
  - x87 transcendentals
- Changing FLDCW (for FP->int conv) (ping-ponging 2 values OK)

Intel Labs

Summary

- The Intel and Microsoft* tools provide easiest way to program for current and upcoming Intel processors
- New instructions and architecture provide optimization opportunities:
  - Faster buses remove bottlenecks, change your "hotspots"
  - Ways to avoid branches and casts
  - Efficient AGP & WC utilization
  - Avoid memory stalls
  - Avoid exceptional cases (ie, denormals)

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

Call to Action

- Start using Streaming SIMD extensions, and Streaming SIMD extensions 2 today!
- Try out the vectorizing and Profile-Guided Optimization features of the Intel Compilers

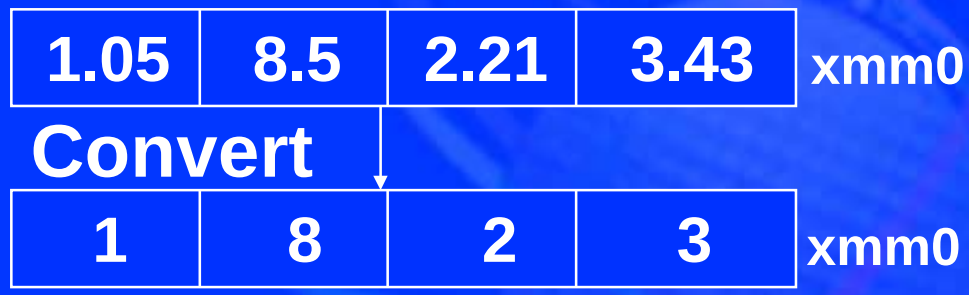
References

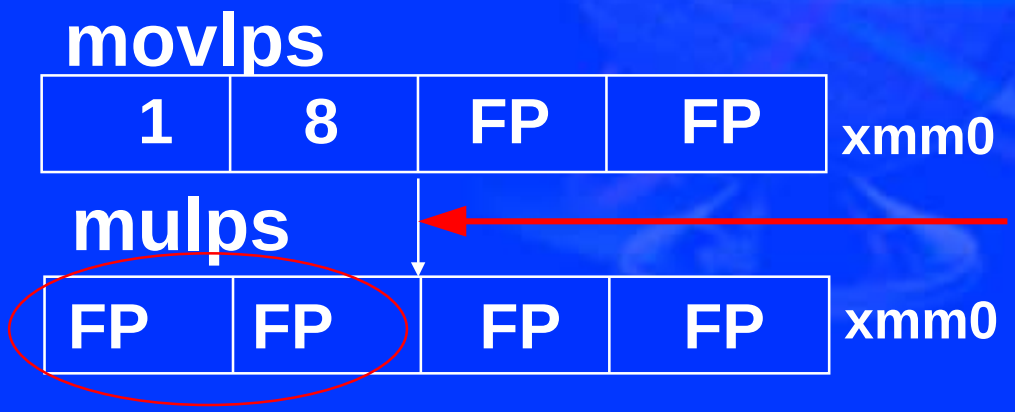
- Pentium® 4 Processor Software Developer's Guide
  - developer.intel.com
- VTune™ Performance Enhancement Environment


Intel
Developer
Forum
Spring 2001
intel

Backup

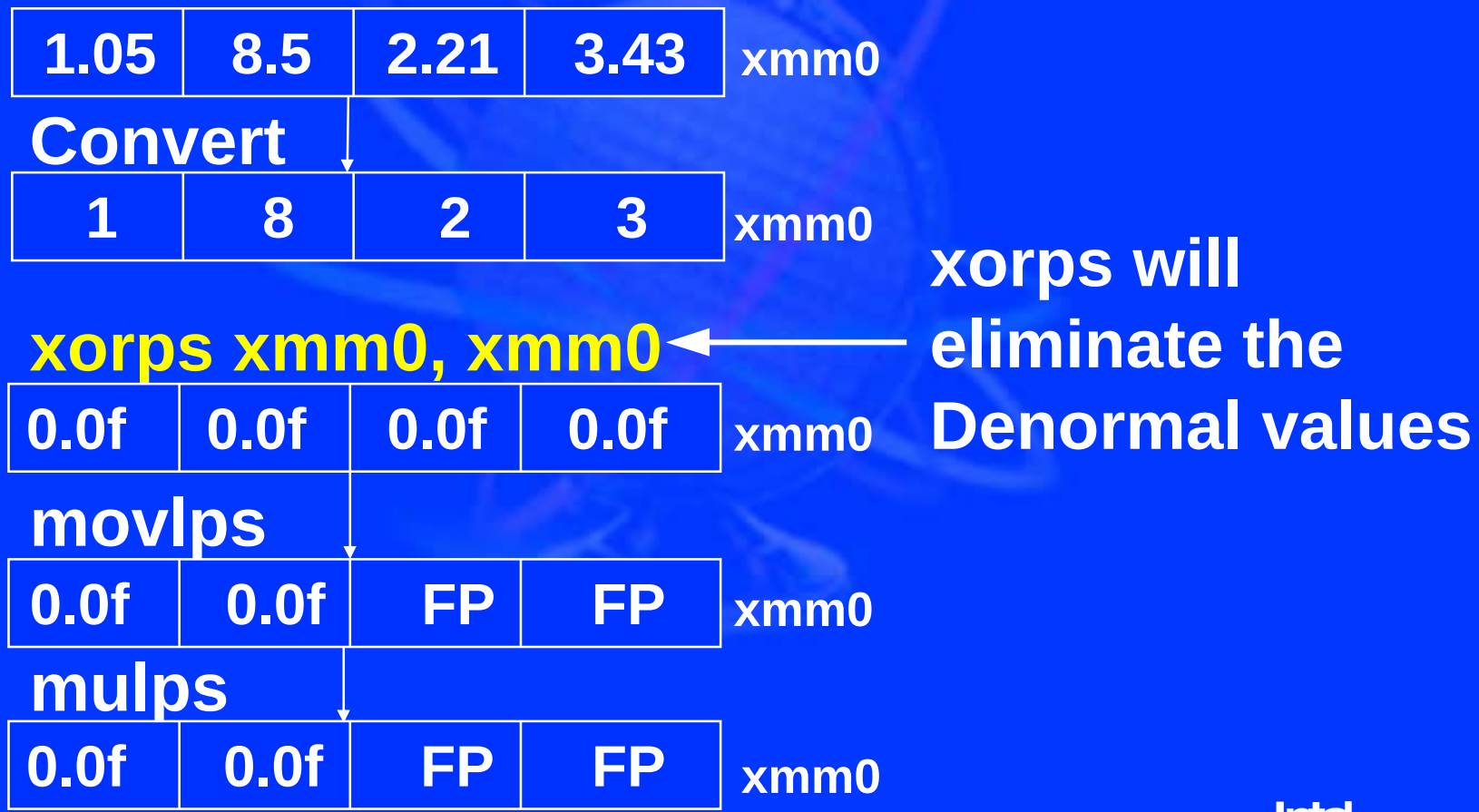
Denormal Input – When it can happen





Denormal input value

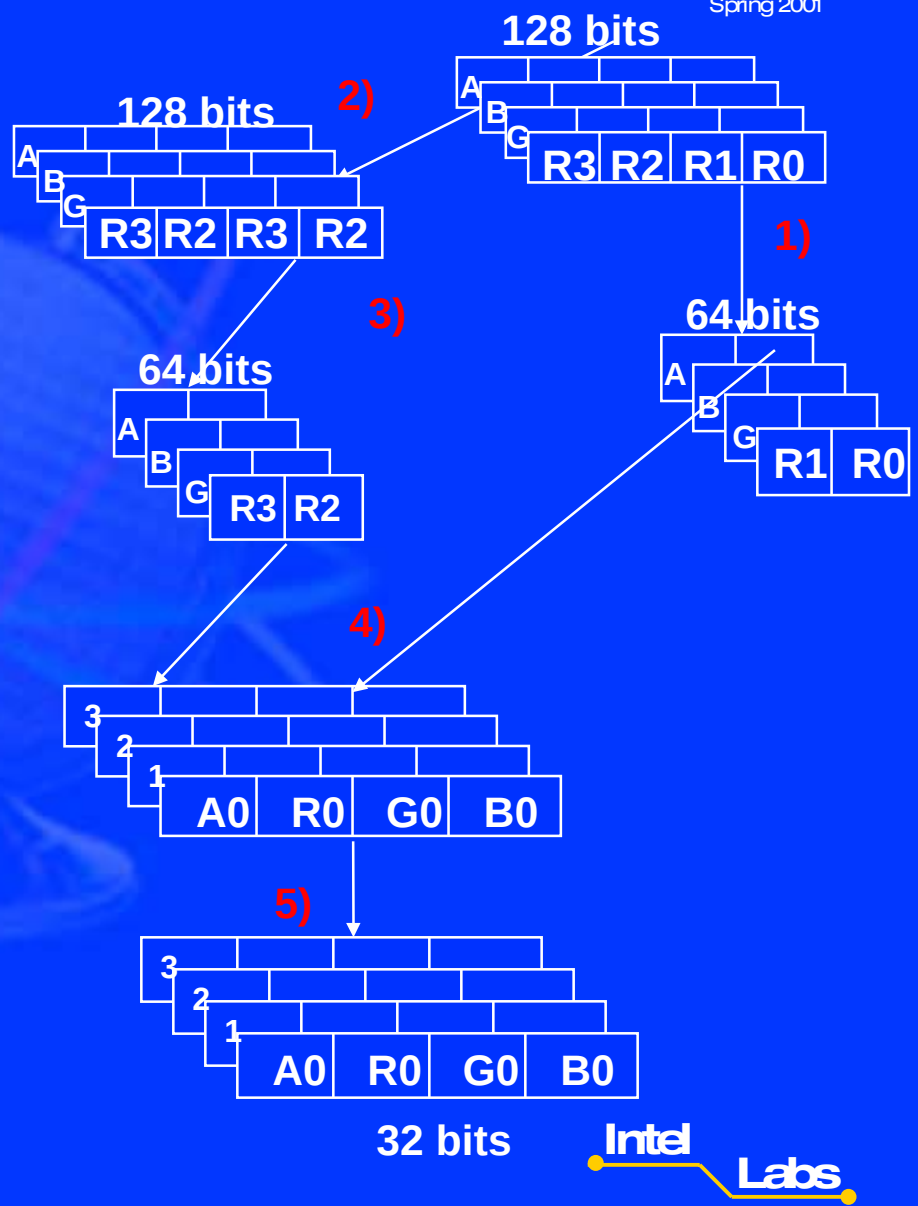
Denormal Input – How to solve it



xorps will eliminate the Denormal values

Color conversion without SSE2

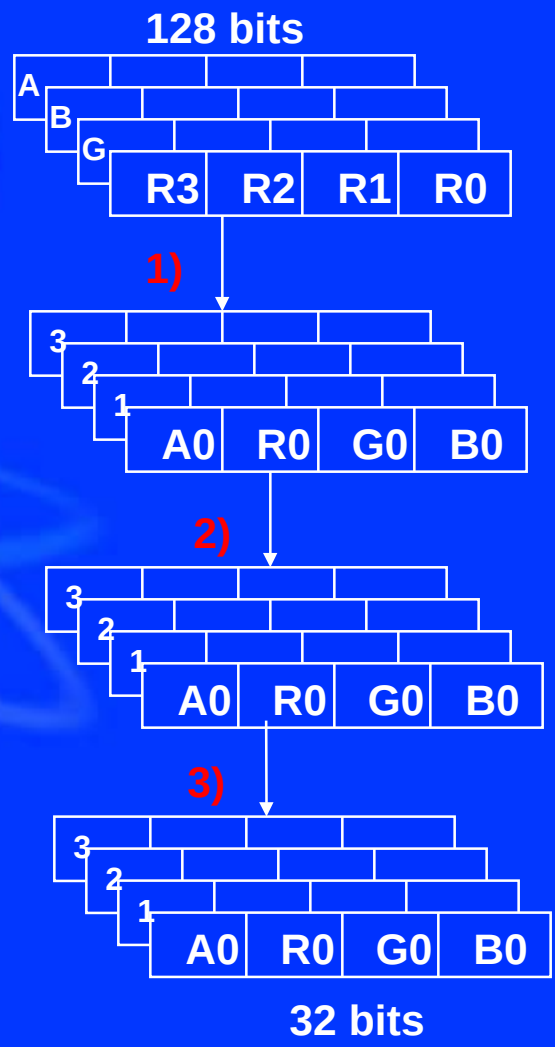
- In 3D Lighting, colors are converted from FP to packed byte format
- The algorithm for Intel® Pentium® III processors is:
  - Convert two color components
  - Shuffle the colors (High to low portions)
  - Convert two more
  - deSwizzle
  - Pack & Saturation



Color conversion: Pentium® 4 processor

The algorithm for the Intel® Pentium® 4 processor is:

- deSwizzle
- Convert color components for four vertices (using SSE2 instructions)
- Pack & Saturation (using SSE2 instructions)



Enabling FTZ & DAZ

- Example: enabling FZ:
  - set bit 15 in the SSE / SSE2 control / status register
  - The code for setting the mode*:
    - CSRReg = _mm_getcsr(void); // Get the MXCSR register
    - _mm_setcsr(CSRReg | 0x08000) // set the bit

    - // a macro to set the FZ mode on*
    - _MM_SET_FLUSH_ZERO_MODE(_MM_FLUSH_ZERO_ON)
- Similar approach for DAZ (check CPUID for support)

*You will need Intel® C/C++ Compiler

Case Study: Optimized Software MPEG-2 Video Decoder

Module/DLL				Function				Instr Bin				Optimized
Name	PIIIP	P4P	Scaling	Name	PIIIP	P4P	Scaling	EIP	PIIIP	P4P	Scaling	Scaling
decode.dll	60%	65%	1.2	VLD	5%	15%	1.1	0x00000-0x00040	5%	15%	1.1	1.4
grfx_driver.dll	25%	30%	0.5	memmove	25%	30%	0.5	0x4fc40-0x4fc80	25%	30%	0.5	1.4
app.exe	10%	4%	1.4	SplitStream	3%	1%	1.4	0x96540-0x96580	3%	1%	1.4	1.4
GDI32.DLL	5%	1%	1.2	SaveDC	1%	1%	1.2	0x21f40-0x21f80	1%	1%	1.2	1.2
Total	100%	100%	1.05									1.45

- Scaling shown in Excel for Module, Function & Instr Bin
- Quickly identifes key instr sections that are scaling poorly
- Estimate app-level gain for opt. these

PIIIP = Intel® Pentium® III processor.

AGP Considerations

- Ensure AGP enabled
  - New chipset .inf driver file for 850
- Pentium® 4 processor Bandwidths:
  - 2 GB/s WC write B/W on FSB to memory
  - CPU to AGP B/W:
    - ~700 MB/s with fast writes (1/3 Pentium 4 processor peak)
    - ~180 MB/s with fast writes off (1/10 Pentium 4 processor peak)
    - Enable Fast-Writes if supported

Matched to DMA model