The significance of SIMD, SSE and AVX

Stephen Blair-Chappell
Intel Compiler Labs
Agenda

• 1. Auto-Vectorisation
• 2. CPU Dispatch
• 3. Manual Processor Dispatch
• 4. A Case Study
“I must have the Intel compiler, it has sped up our application by two.”

A customer when moving from version 9.1 to version 10 of the Intel compiler
Auto-Vectorisation
Vector Processing

- A specific case of **data level parallelism** (DLP)

- Same operation simultaneously executed on $N > 1$ elements of a vector.

Scalar Processing

```
add.d r3, r1, r2
```

Vector Processing

```
addvec.d v3, v1, v2
```

VL = vector length
SIMD: Continuous Evolution

<table>
<thead>
<tr>
<th>Year</th>
<th>Technology</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1999</td>
<td>SSE</td>
<td>70 instr Single-Precision Vectors Streaming operations</td>
</tr>
<tr>
<td>2000</td>
<td>SSE2</td>
<td>144 instr Double-precision Vectors 8/16/32 64/128-bit vector integer</td>
</tr>
<tr>
<td>2004</td>
<td>SSE3</td>
<td>13 instr Complex Data</td>
</tr>
<tr>
<td>2006</td>
<td>SSSE3</td>
<td>32 instr Decode</td>
</tr>
<tr>
<td>2007</td>
<td>SSE4.1</td>
<td>47 instr Video Graphics building blocks Advanced vector instr</td>
</tr>
<tr>
<td>2008</td>
<td>SSE4.2</td>
<td>8 instr String/XML processing POP-Count CRC</td>
</tr>
<tr>
<td>2009</td>
<td>AES-NI</td>
<td>7 instr Encryption and Decryption Key Generation</td>
</tr>
<tr>
<td>2010\11</td>
<td>AVX</td>
<td>~100 new instr. ~300 legacy SSE instr updated 256-bit vector 3 and 4-operand instructions</td>
</tr>
</tbody>
</table>
SIMD Types in Processors from Intel [1]

**MMX™**
- Vector size: 64bit
- Data types: 8, 16 and 32 bit integers
- VL: 2, 4, 8
- For sample on the left: Xi, Yi 16 bit integers

**Intel® SSE**
- Vector size: 128bit
- Data types: 8, 16, 32, 64 bit integers, 32 and 64bit floats
- VL: 2, 4, 8, 16
- Sample: Xi, Yi bit 32 int / float
SIMD Types in Processors from Intel [2]

**Intel® AVX**
- Vector size: 256bit
- Data types: 32 and 64 bit floats
- VL: 4, 8, 16
- Sample: Xi, Yi 32 bit int or float

**Intel® MIC**
- Vector size: 512bit
- Data types: 32 and 64 bit integers, 32 and 64bit floats (some support for 16 bits floats)
- VL: 8,16
- Sample: 32 bit float
Scalar and Packed SSE Instructions

The “vector” form of SSE instructions operating on multiple data elements simultaneously are called packed - thus vectorized SSE code means use of packed instructions

- Most of these instructions have a scalar version too operating only one element only

\[
\text{add}^\text{ss} \quad \text{Scalar Single-FP Add}
\]

- Single precision FP data
- Scalar execution mode

\[
\text{add}^\text{ps} \quad \text{Packed Single-FP Add}
\]

- Single precision FP data
- Packed execution mode
Intel® AVX - Setting the Pace for Intel® Instruction Set

Now:
Improved upcoming Intel® microarchitectures: ~15% gain/year

Next:
Leapfrog with wide vectorization, ISA extensions: scalable performance & excellent power efficiency

Nehalem
- Intel® SSE4
- Memory latency, BW
- Fast Unaligned support

Westmere
- AES-NI
- Cryptographic Acceleration

Sandy Bridge
- Intel® AVX
- 2X FP Throughput
- 2X Load Throughput
- 3-Operand instructions

Future Extensions
- Hardware FMA
- Memory Latency/BW
- Many Other Features
## Key Intel® Advanced Vector Extensions (Intel® AVX) Features

<table>
<thead>
<tr>
<th>KEY FEATURES</th>
<th>BENEFITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wider Vectors</td>
<td></td>
</tr>
<tr>
<td>- Increased from 128 to 256 bit</td>
<td></td>
</tr>
<tr>
<td>- Two 128-bit load ports</td>
<td></td>
</tr>
<tr>
<td>Up to 2x peak FLOPs (floating point operations per second) output with good power efficiency</td>
<td></td>
</tr>
<tr>
<td>Enhanced Data Rearrangement</td>
<td></td>
</tr>
<tr>
<td>- Use the new 256 bit primitives to broadcast, mask loads and permute data</td>
<td></td>
</tr>
<tr>
<td>Organize, access and pull only necessary data more quickly and efficiently</td>
<td></td>
</tr>
<tr>
<td>Three and four Operands: Non Destructive Syntax for both AVX 128 and AVX 256</td>
<td></td>
</tr>
<tr>
<td>Fewer register copies, better register use for both vector and scalar code</td>
<td></td>
</tr>
<tr>
<td>Flexible unaligned memory access support</td>
<td></td>
</tr>
<tr>
<td>More opportunities to fuse load and compute operations</td>
<td></td>
</tr>
<tr>
<td>Extensible new opcode (VEX)</td>
<td></td>
</tr>
<tr>
<td>Code size reduction</td>
<td></td>
</tr>
</tbody>
</table>

Intel® AVX is a general purpose architecture, expected to supplant SSE in all applications used today.
A New 3- and 4-Operand Instruction Format

- Intel® Advanced Vector Extensions (Intel® AVX) has a distinct destination argument that results in fewer register copies, better register use, more load/op macro-fusion opportunities, and smaller code size.

- New 4-operand Blends example, implicit xmm0 not longer needed:
  - 1 less copy, 3 bytes smaller code size
  - 1 more load/op fusion opportunity, 4+ bytes smaller code size

\[
\begin{align*}
\text{xmm10} &= \text{xmm9} + \text{xmm1} \\
\text{movaps} \; \text{xmm10}, \; \text{xmm9} &\quad \Rightarrow \quad \text{vaddpd} \; \text{xmm10}, \; \text{xmm9}, \; \text{xmm1} \\
\text{addpd} \; \text{xmm10}, \; \text{xmm1} \\
\text{xmm10} &= \text{xmm9} + \text{m128} \\
\text{movups} \; \text{xmm10}, \; \text{m128} &\quad \Rightarrow \quad \text{vaddpd} \; \text{xmm10}, \; \text{xmm9}, \; \text{m128} \\
\text{addpd} \; \text{xmm10}, \; \text{xmm9}
\end{align*}
\]
**Intel® Microarchitecture (Sandy Bridge) Highlights**

- Instruction Fetch & Decode
- Allocate/Rename/Retire
- Zeroing Idioms

**Scheduler (Port names as used by IACA)**

- **Port 0**
  - ALU
  - VI MUL
  - SSE MUL
  - DIV *
  - AVX FP MUL
- **Port 1**
  - ALU
  - VI ADD
  - SSE ADD
  - AVX FP ADD
  - Imm Blend
- **Port 5**
  - ALU
  - VI ADD
  - SSE ADD
  - AVX FP ADD
  - Imm Blend
- **Port 2**
  - Load
  - Store Address
  - AVX/FP Shuf
  - AVX/FP Bool
  - Imm Blend
- **Port 3**
  - Load
  - Store Address
  - STD
- **Port 4**
  - Memory Control
  - 48 bytes/cycle

**L1 Data Cache**

- 48 bytes/cycle

- **1-per-cycle 256-bit multiply, add, and shuffle**
- **Load double the data**
  - with Intel microarchitecture (Sandy Bridge) !!!

*Not fully pipelined*
Auto-Vectorization

Transforming sequential code to exploit the vector (SIMD, SSE) processing capabilities

```c
for (i=0; i<MAX; i++)
    c[i]=a[i]+b[i];
```
# Many Ways to introduce SSE Vectorization

<table>
<thead>
<tr>
<th>Method</th>
<th>Programmer control</th>
<th>Ease of use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Performance Libraries (e.g. IPP and MKL)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compiler: Fully automatic vectorization</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cilk Plus Array Notation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compiler: Auto vectorization hints (#pragma ivdep, ...)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Mandated Vectorization (SIMD Directive)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manual CPU Dispatch (__declspec(cpu_dispatch ...))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIMD intrinsic class (F32vec4 add)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vector intrinsic (mm_add_ps())</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assembler code (addps)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
How do I know if a loop is vectorised?

- `-vec-report`

```
> icl /Qvec-report MultArray.c
MultArray.c(92): (col. 5) remark: LOOP WAS VECTORIZED.
```
Examples of Code Generation

```c
static double A[1000], B[1000],
    C[1000];
void add() {
    int i;
    for (i=0; i<1000; i++)
        if (A[i]>0)
            A[i] += B[i];
        else
            A[i] += C[i];
}
```

```
.B1.2::
    movaps    xmm2, A[rdx*8]
xorps      xmm0, xmm0
 cmp1tpd    xmm0, xmm2
 movaps     xmm1, B[rdx*8]
 andps      xmm1, xmm0
 andnps     xmm0, C[rdx*8]
orps       xmm1, xmm0
 addpd      xmm2, xmm1
 movaps     A[rdx*8], xmm2
 add         rdx, 2
 cmp         rdx, 1000
 jl          .B1.2

.B1.2::
    vmovaps   ymm3, A[rdx*8]
 vmovaps   ymm1, C[rdx*8]
 vcmpgtpd  ymm2, ymm3, ymm0
 vblendvd  ymm4, ymm1, B[rdx*8], ymm2
 vaddpd    ymm5, ymm3, ymm4
 vmovaps   A[rdx*8], ymm5
 add         rdx, 4
 cmp         rdx, 1000
 jl          .B1.2
```

```
.B1.2::
    movaps    xmm2, A[rdx*8]
 xorps      xmm0, xmm0
 cmp1tpd    xmm0, xmm2
 movaps     xmm1, B[rdx*8]
 andps      xmm1, xmm0
 andnps     xmm0, C[rdx*8]
orps       xmm1, xmm0
 addpd      xmm2, xmm1
 movaps     A[rdx*8], xmm2
 add         rdx, 2
 cmp         rdx, 1000
 jl          .B1.2
```
Vectorization Report

“Loop was not vectorized” because:

- “Existence of vector dependence”
- “Non-unit stride used”
- “Mixed Data Types”
- “Condition too Complex”
- “Condition may protect exception”
- “Low trip count”
- “Subscript too complex”
- ‘Unsupported Loop Structure”
- “Contains unvectorizable statement at line XX”
- “Not Inner Loop”
- “vectorization possible but seems inefficient”
- “Operator unsuited for vectorization”
Elemental Functions

- Use **scalar syntax** to describe an operation on a single element
- Apply operation to arrays in parallel
- Utilize both **vector parallelism** and **core parallelism**

```c
_declspec(vector)
double option_price_call_black_scholes
  (double S, double K, double r, double sigma, double time)
{
  double time_sqrt = sqrt(time);
  double d1 =
    (log(S/K)+r*time)/(sigma*time_sqrt)+0.5*sigma*time_sqrt;
  double d2 = d1-(sigma*time_sqrt);
  return S*N(d1) - K*exp(-r*time)*N(d2);
}

cilk_for (int i=0; i < NUM_OPTIONS; i++) {
  call_serial[i] = option_price_call_black_scholes(S[i], K[i], r, sigma, time[i]);
}
```
CPU-Dispatch

Adding Portability
“I’ve stopped using the Intel compiler. Each time I ship the product to a customer, they complain that applications crashes”!

A games developer at a recent networking event.
Imagine this scenario:

1. Your IT dept have just bought you the latest and greatest Intel based workstation.

2. You’ve heard auto-vectorisation can make a real difference to performance

3. You enable auto-vectorisation using -xhost

4. You boast to your colleagues, “my application runs faster than anything you can write…”

5. You send the application to a colleague – it refuses to run.
What might be the issue?

How can it be overcome?
Two Key Decisions to be Made:

1. How do we introduce the vector code?

2. How do we deal with the multiple SIMD instruction set extensions like SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, AVX ...?
Out-of-the-box behaviour – Intel Compiler

• Automatic-vectorisation is enabled by default
• (turn it off with -no-vec)

• The option -msse2 is used by default (as long as no x, ax or -m option has been used)

-msse2: “May generate Intel® SSE2 and SSE instructions. This value is only available on Linux systems”.
# Building for non-intel processors (-m)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sse4.1</td>
<td>May generate Intel® SSE4.1, SSSE3, SSE3, SSE2, and SSE instructions.</td>
</tr>
<tr>
<td>ssse3</td>
<td>May generate Intel® SSSE3, SSE3, SSE2, and SSE instructions.</td>
</tr>
<tr>
<td>sse2</td>
<td>May generate Intel® SSE2 and SSE instructions.</td>
</tr>
<tr>
<td>sse</td>
<td>This option has been deprecated; it is now the same as specifying ia32.</td>
</tr>
<tr>
<td>ia32</td>
<td>Generates x86/x87 generic code that is compatible with IA-32 architecture.</td>
</tr>
</tbody>
</table>

This option tells the compiler to generate code specialized for the processor that executes your program. Code generated with these options should execute on any compatible, non-Intel processor with support for the corresponding instruction set.
# Building for Intel processors (-x)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVX</td>
<td>AVX, SSE4.2, SSE4.1, SSSE3, SSE3, SSE2, and SSE instructions.</td>
</tr>
<tr>
<td>SSE4.2</td>
<td>SSE4 Efficient Accelerated String and Text Processing instructions supported by Intel® Core™ i7 processors. SSE4 .1, SSSE3, SSE3, SSE2, and SSE. May optimize for the Intel® Core™ processor family.</td>
</tr>
<tr>
<td>SSE4.1</td>
<td>SSE4 Vectorizing Compiler and Media Accelerator, SSSE3, SSE3, SSE2, and SSE. May optimize for Intel® 45nm Hi-k next generation Intel® Core™ microarchitecture.</td>
</tr>
<tr>
<td>SSE3_ATOM</td>
<td>MOVBE , (depending on -minstruction ), SSSE3, SSE3, SSE2, and SSE . Optimizes for the Intel® Atom™ processor and Intel® Centrino® Atom™ Processor Technology</td>
</tr>
<tr>
<td>SSSE3</td>
<td>SSSE3, SSE3, SSE2, and SSE. Optimizes for the Intel® Core™ microarchitecture.</td>
</tr>
<tr>
<td>SSE3</td>
<td>SSE3, SSE2, and SSE. Optimizes for the enhanced Pentium® M processor microarchitecture and Intel NetBurst® microarchitecture.</td>
</tr>
<tr>
<td>SSE2</td>
<td>SSE2 and SSE . Optimizes for the Intel NetBurst® microarchitecture.</td>
</tr>
</tbody>
</table>
Auto-Vectorization –Running on Sandy Bridge

-xAVX

for(i=0;i<NUM;i++) {
  j[i] = h[i] + i + 3
}

Running on a CPU supporting AVX
Auto-Vectorization

-xAVX

for(i=0;i<NUM;i++)
{
    j[i] = h[i] + i + 3
}

Fatal Error: This program was not built to run in your system.

Please verify that both the operating system and the processor support Intel(R) AVX.

Running on a CPU not supporting AVX
Using -ax compiler option ...

- Generates **multiple paths** if there is a performance benefit
- Generates a **base line** path
- **Other options** (e.g. -O3) control the base line path
- At **runtime** path chosen based on what processor code is running on
The Base line

• Use \texttt{-m} or \texttt{-x} to set base line

• \texttt{-m} for non-intel processors

• \texttt{-X} for intel processors

• If no \texttt{-m} or \texttt{-x}, compiler defaults to \texttt{-mSSE2}

• \texttt{-m} and \texttt{-x} are \textbf{mutually exclusive}
CPU Dispatching

- axAVX

```
for(i=0;i<NUM;i++) {
    j[i] = h[i] + i + 3
}
```

Base line (set with -m or -x option)
SSE2
Generic low-spec CPU (no support of AVX)

-axAVX

```c
for(i=0;i<NUM;i++)
{
    j[i] = h[i] + i + 3
}
```

Base line (set with -m or -x option)
SSE2
Sandy Bridge (supports AVX)

```c
for(i=0;i<NUM;i++) {
    j[i] = h[i] + i + 3
}
```

Base line
(set with `-m` or `-x` option)
SSE2
Running on Intel Processors

• If -ax and -x are used together
• Base line will execute on Intel compatible processors specified by the -x
Running on **Intel** and **non-Intel** processors

- If `-ax` and `-m` are used together
- Base line will execute on non-Intel processors compatible with the processor type specified by `-m`
What option do **AMD** recommend?

**AMD Opteron™ 6100 Series P**

**AMD Opteron™ 4100 Series P**

Compiler Options Quick Reference

---

**ICC**

Latest release: 12.0 update3, March 2011

http://software.intel.com

<table>
<thead>
<tr>
<th>Architecture</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Generate instructions specific to Magny-Cours</td>
<td>-msee3 (avoid -ax)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Optimization Levels</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Disable all optimizations</td>
<td>-00</td>
</tr>
</tbody>
</table>

Quiz – what option is best?

1. You application will only ever run on the same CPU as you development machine
2. Your application will run on a farm of AMD Opterons (4100) and Intel i7s
3. Your application will run on Sandy Bridge Machines and Core 2.
4. Your have no clue what machine the code will run on.
Benefit of CPU Dispatch

Code

- still works on older processors

- Works properly on non-intel CPUs
  - Non-intel processors will ALWAYS take the base-line

- Code can take advantage of latest generation of CPUs
Manual Processor Dispatch
Manual processor Dispatch

• Allows you to write processor-specific code

• Provide more than one version of code

• Use __declspec(cpu_dispatch(cpuid,cpuid...))
## CPUID Arguments

<table>
<thead>
<tr>
<th>Argument for cpuid</th>
<th>Processors</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>future_cpu_16</strong></td>
<td>2nd generation Intel® Core™ processor family with support for Intel® Advanced Vector Extensions (Intel® AVX).</td>
</tr>
<tr>
<td><strong>core_aes pclmulqdq</strong></td>
<td>Intel® Core™ processors with support for Advanced Encryption Standard (AES) instructions and carry-less multiplication instruction</td>
</tr>
<tr>
<td><strong>core_i7_sse4_2</strong></td>
<td>Intel® Core™ processor family with support for Intel® SSE4 Efficient Accelerated String and Text Processing instructions (SSE4.2)</td>
</tr>
<tr>
<td><strong>atom</strong></td>
<td>Intel® Atom™ processors</td>
</tr>
<tr>
<td><strong>core_2_duo_sse4_1</strong></td>
<td>Intel® 45nm Hi-k next generation Intel® Core™ microarchitecture processors with support for Intel® SSE4 Vectorizing Compiler and Media Accelerators instructions (SSE4.1)</td>
</tr>
<tr>
<td><strong>core_2_duo_ssse3</strong></td>
<td>Intel® Core™2 Duo processors and Intel® Xeon® processors with Intel® Supplemental Streaming SIMD Extensions 3 (SSSE3)</td>
</tr>
<tr>
<td><strong>pentium_4_sse3</strong></td>
<td>Intel® Pentium 4 processor with Intel® Streaming SIMD Extensions 3 (Intel® SSE3), Intel® Core™ Duo processors, Intel® Core™ Solo processors</td>
</tr>
<tr>
<td><strong>pentium_4</strong></td>
<td>Intel® Intel Pentium 4 processors</td>
</tr>
<tr>
<td><strong>pentium_m</strong></td>
<td>Intel® Pentium M processors</td>
</tr>
<tr>
<td><strong>pentium_iii</strong></td>
<td>Intel® Pentium III processors</td>
</tr>
<tr>
<td><strong>generic</strong></td>
<td>Other IA-32 or Intel 64 processors or compatible processors not provided by Intel Corporation</td>
</tr>
</tbody>
</table>
Manual Dispatch Example

```c
#include <stdio.h>
// need to create specific function versions
__declspec(cpu_dispatch(generic, future_cpu_16))
void dispatch_func() {}

__declspec(cpu_specific(generic))
void dispatch_func() {
    printf("Code for non-Intel processors\nand generic Intel\n");
}

__declspec(cpu_specific(future_cpu_16))
void dispatch_func() {
    printf("Code for 2nd generation Intel Core processors goes here\n");
}

int main() {
    dispatch_func();
    printf("Return from dispatch_func\n");
    return 0;
}
```
Questions to Ask

• Is my application going to run on a different CPU to my development platform?

• Is my application going to run on one specific generation of CPU?

• Is my application just going to run on just Intel CPUs?

• Will my application be running on non-intel processors?
A Case Study

An Engine Simulator
The Simulation Environment

ECM under test

www.pishurlok.com
The Simulation Frames

Tick
ADC Complete
Interrupt Request
Model
Logger
Script

Frame 1
Frame 2
Frame 3
Matlab design of the Engine Simulator

4 Cylinder SI Engine model
single throttle with idle speed bypass
# Results on 100k loop simulation

<table>
<thead>
<tr>
<th>CPU</th>
<th>No Auto-Vectorisation</th>
<th>With Auto-Vectorisation</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>P4</td>
<td>39.344</td>
<td>21.9</td>
<td>1.80</td>
</tr>
<tr>
<td>Core 2</td>
<td>5.546</td>
<td>0.515</td>
<td>10.77</td>
</tr>
<tr>
<td>Speedup</td>
<td>7.09</td>
<td>45.52</td>
<td>76</td>
</tr>
</tbody>
</table>
Vtune confirms reason for Speedup

<table>
<thead>
<tr>
<th>CPU EVENT</th>
<th>Without Vect</th>
<th>With Vect</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU_CLK_UNHALTED.CORE</td>
<td>16,641,000,448</td>
<td>1,548,000,000</td>
</tr>
<tr>
<td>INST RETIRED.ANY</td>
<td>3,308,999,936</td>
<td>1,395,000,064</td>
</tr>
<tr>
<td>X87_OPS RETIRED.ANY</td>
<td>250,000,000</td>
<td>0</td>
</tr>
<tr>
<td>SIMD_INST RETIRED</td>
<td>0</td>
<td>763,000,000</td>
</tr>
</tbody>
</table>

Summary of Simulation Performance Improvements

- Performance gains through migrating to newer silicon
- Performance gains by using Intel compiler.
Closing Remarks

• Try **Auto-vectorisation** – it can make a difference!

• **Out-of-the-box** use does not deliver the best optimisation

• If you are running on more than one generation of CPU use **–ax (CPU dispatching)**

• Use **–m** option on non-intel CPUs
Intel® compilers, associated libraries and associated development tools may include or utilize options that optimize for instruction sets that are available in both Intel® and non-Intel microprocessors (for example SIMD instruction sets), but do not optimize equally for non-Intel microprocessors. In addition, certain compiler options for Intel compilers, including some that are not specific to Intel micro-architecture, are reserved for Intel microprocessors. For a detailed description of Intel compiler options, including the instruction sets and specific microprocessors they implicate, please refer to the “Intel® Compiler User and Reference Guides” under “Compiler Options.” Many library routines that are part of Intel® compiler products are more highly optimized for Intel microprocessors than for other microprocessors. While the compilers and libraries in Intel® compiler products offer optimizations for both Intel and Intel-compatible microprocessors, depending on the options you select, your code and other factors, you likely will get extra performance on Intel microprocessors.

Intel® compilers, associated libraries and associated development tools may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include Intel® Streaming SIMD Extensions 2 (Intel® SSE2), Intel® Streaming SIMD Extensions 3 (Intel® SSE3), and Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3) instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors.

While Intel believes our compilers and libraries are excellent choices to assist in obtaining the best performance on Intel® and non-Intel microprocessors, Intel recommends that you evaluate other compilers and libraries to determine which best meet your requirements. We hope to win your business by striving to offer the best performance of any compiler or library; please let us know if you find we do not.
Legal Disclaimer

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, reference www.intel.com/software/products.


*Other names and brands may be claimed as the property of others.

Copyright © 2010. Intel Corporation.