Programming for Intel® Many Integrated Core

A Unified Approach to Heterogeneous programming with Intel® Multicore Processors and the Intel® Many Integrated Core Architecture
Agenda

• Introduction
• Knights Ferry SW Development Hardware
• Programming Environment for Intel® MIC
• Heterogeneous Programming / Offload Model
• Additional Developer Tools for Intel® MIC
• Summary
Tools need to access all three dimensions to deliver performance

<table>
<thead>
<tr>
<th>Core(s)</th>
<th>Intel® Xeon® processor 64-bit</th>
<th>Intel® Xeon® processor 5100 series</th>
<th>Intel® Xeon® processor 5500 series</th>
<th>Intel® Xeon® processor 5600 series</th>
<th>Intel® Xeon® processor code-named Sandy Bridge</th>
<th>Intel® Xeon® processor code-named Ivy Bridge</th>
<th>Intel® Xeon® processor code-named Haswell</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threads</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>12</td>
<td>16</td>
</tr>
<tr>
<td>SIMD Width</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>256</td>
<td>256</td>
<td>256</td>
</tr>
<tr>
<td>SSE2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSSE3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSE4.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSE4.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AVX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AVX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AVX2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FMA3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intel® MIC co-processor code-named Knights Ferry</td>
<td>32</td>
<td>&gt;50</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intel® MIC co-processor code-named Knights Corner</td>
<td>128</td>
<td>&gt;200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Software challenge: Develop scalable software
**Intel® MIC Architecture**

Intel’s Multi- and Many-Core Engines

**Intel® Xeon® processor:**
- Intel’s Foundation of HPC Performance
- Suited for full scope of workloads
- Industry leading performance/watt for serial & highly parallel workloads.

**Intel® MIC Architecture:**
- Optimized for highly parallelized compute intensive workloads
- Common programming model & S/W tools with Xeon processors, enabling efficient app readiness and performance tuning
- Launching on 22nm with >50 cores and required b/w to provide outstanding performance for highly parallel HPC uses
The Knights Family

Knights Corner

1st Intel® MIC product
22nm process
>50 Intel Architecture cores
Adds optimizations learned from Knights Ferry Platform for performance, programmability
Increased DP Floating Point Performance
Larger capacity of fast GDDR5
Additional RAS features

Knights Ferry

Future options subject to change without notice.
Intel® MIC Architecture
Developing Today on Knights Ferry – A Selection

[Various logos of institutions and companies]
Intel® MIC Architecture
More than one sustained TeraFLOP/sec (Double Precision)

ASCI Red: 1 TeraFLOP/sec
December 1996
1996 First System 1 TF/sec Sustained
7264 Intel® Pentium® Pro Processors**
OS: Cougar
72 Cabinets

Knights Corner: 1 TeraFLOP/sec
November 2011
2011 First Chip 1 TF/s Sustained
One 22nm Chip
OS: Linux*
One PCI Express* slot

** With 2/3 system built. Full system later upgraded to 3.1 TeraFlops/sec with 9298 Intel® Pentium II Xeon processors

Agenda

• Introduction
• Knights Ferry SW Development Hardware
• Programming Environment for Intel® MIC
• Heterogeneous Programming / Offload Model
• Additional Developer Tools for Intel® MIC
• Summary
“Knights Ferry” Software Development Platform

Growing availability through 2011
Aubrey Isle Co-Processor
Up to 32 cores, up to 1.2 GHz
Up to 128 threads at 4 threads / core
Up to 8MB shared coherent cache
Up to 2 GB GDDR5
Bundled with Intel HPC SW tools

Future options subject to change without notice.
**Aubrey Isle Co-Processor Architecture**

- **Multiple x86 cores**
  - In-order, short pipeline
  - Multi-thread support
- **16-wide vector units (512b)**
- **1024-bit ring bus**
- **GDDR5 memory**
- **Fully coherent caches**
- **Extended instruction set**

**Standard IA Cache Coherent Shared Memory Programming**

For illustration only. Future options subject to change without notice.
The Aubrey Isle co-processor core:
• Scalar pipeline derived from the dual-issue Pentium processor
• Short execution pipeline
• Fully coherent cache structure
• Significant modern enhancements such as multi-threading, 64-bit extensions, and sophisticated pre-fetching.
• 4 execution threads per core
• Separate register sets per thread
• Supports IEEE standards for floating point arithmetic
• Fast access to its 256KB local subset of a coherent L2 cache.
• 32KB instruction cache per core
• 32KB data cache for each core.

Enhanced x86 instructions set with:
• Over 100 new instructions,
• Wide vector processing operations
• Some specialized scalar instructions
• 3-operand, 16-wide vector processing unit (VPU)
• VPU executes integer, SP-float, and DP-float instructions

Interprocessor Network
1024 bits wide, bi-directional (512 bits in each direction)

Future options subject to change without notice.
Aubrey Isle Core VPU

Future options subject to change without notice.
New VPU Instructions

More than 100 new Instructions

512-bit SIMD
- 32x 512b vector-register, 8x 16b mask-register
- 16 FLOAT32, 8 FLOAT64 or 16 INT32 elements /vreg

Ternary, Multiply-Add (FMA)
- More flops in fewer ops (IEEE conform)

Load-op
- Third operand can be taken direct from memory

Broadcast/Swizzle/Format Conversion (on Load/Store)
- Float16, unorm8, etc. - allows more efficient use of caches

Predication/Masking on most Operations

Gather/Scatter

Future options subject to change without notice.
Multiply-Add

**Multiply-Add** (destination is also third source)

- `vmadd231ps v0, v5, v6 ; v0=v5*v6+v0`
- Operand 2 times operand 3 plus operand 1
**Predication/Masking**

**Predication** (mask the writing of the elements)
- `vmadd231ps v0 {k1}, v5, v6`
VGATHER/VSCATTER Operation

vgather v1{k2}, [rax+v3]

\[ v3 = \begin{bmatrix} 3 & 0 & 1 & 2 & 5 & 4 & 2 & 1 & 2 & 0 & 3 & 0 & 3 & 6 & 2 & 1 \end{bmatrix} \]

\[ k2 = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \]

\[ v1 = \begin{bmatrix} 8 & 5 & 6 & 0 & 0 & 9 & 7 & 0 & 7 & 5 & 0 & 0 & 0 & 0 & 7 & 6 \end{bmatrix} \]

vscatter [rax+v3]{k2}, v1 -- same as vgather, but in reverse
Vectorization for MIC
Not much different from Intel® SSE or AVX

**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
Intel® Cilk™ Plus Array Notation
Works for Intel® MIC as for SSE/AVX

\[
\begin{align*}
\text{A[ : ]} & \quad // \text{All of vector A} \\
\text{B[2:6]} & \quad // \text{Elements 2 to 7 of vector B} \\
\text{C[ : ]}[5] & \quad // \text{Column 5 of matrix C} \\
\text{D[0:3:2]} & \quad // \text{Elements 0,2,4 of vector D}
\end{align*}
\]

\[
\begin{align*}
\text{if (a[ : ] > b[ : ])} \\
\quad & \quad \text{c[ : ] = d[ : ] * e[ : ];} \\
\text{else} \\
\quad & \quad \text{c[ : ] = d[ : ] * 2;}
\end{align*}
\]

A simple and elegant solution: a language construct for vector level parallelism
Agenda

- Introduction
- Knights Ferry SW Development Hardware
- Programming Environment for Intel® MIC
- Heterogeneous Programming / Offload Model
- Additional Developer Tools for Intel® MIC
- Summary
Intel® MIC Programming
Programming and Tools
Architecting Scaling Programmability

One Programming Model
⇒ Heterogeneous Compilation
Expressing Parallelism [1]

Multiple Cores
Hardware Threads

Threads/Tasks
OpenMP*
Intel ® Cilk™ Plus
Intel ® Threading Building Blocks

SIMD-Data
Parallelism

Vectors
Array Notations
Elemental Functions
SIMD Pragma

Extract Parallelism from your applications
Express them as tasks with SIMD kernels
Expressing Parallelism [2]
Message Passing Based Parallelism

- Intel® MPI) for optimized application performance
  - Automatic tuning
  - Industry leading low latency
  - Multi-vendor interoperability
  - Performance optimized support for the latest OFED capabilities through DAPL 2.0

- Coarray Fortran
  - Transparently based on Intel® MPI
Execution Models

Multi-Core Hosted
General purpose serial and parallel computing

Offload
Codes with highly-parallel phases

Symmetric
Codes with balanced needs

Many Core Hosted
Highly-parallel codes

Multi-core Intel® Xeon® processor

Intel® MIC

Range of models to meet application needs
Programming and Tools
Intel Development Tools extend to Intel® MIC

Advanced Performance
C++ and Fortran Compilers, MKL/IPP Libraries & Analysis Tools for Windows*, Linux* developers on IA based multi-core node

Distributed Performance
MPI Cluster Tools with C++ and Fortran Compiler, MKL Libraries and Analysis Tools for Windows*, Linux* developers on IA based clusters
Agenda

• Introduction
• Knights Ferry SW Development Hardware
• Programming Environment for Intel® MIC
• Heterogeneous Programming / Offload Model
• Additional Developer Tools for Intel® MIC
• Summary
Heterogeneous Programming

Parallel programming is the same on MIC and CPU
**Heterogeneous Programming**

Parallel programming is the same on MIC and CPU.
Example: Computing PI
(for illustration purpose only)

```c
#define NSET 1000000
int main ( int argc, const char** argv )
{
    long int i;
    float num_inside, Pi;
    num_inside = 0.0f;
#pragma offload target (MIC)
#pragma omp parallel for reduction(+:num_inside)
    for( i = 0; i < NSET; i++ )
    {
        float x, y, distance_from_zero;
        // Generate x, y random numbers in [0,1)
        x = float(rand()) / float(RAND_MAX + 1);
        y = float(rand()) / float(RAND_MAX + 1);
        distance_from_zero = sqrt(x*x + y*y);
        if ( distance_from_zero <= 1.0f )
            num_inside += 1.0f;
    }
    Pi = 4.0f * ( num_inside / NSET );
    printf("Value of Pi = %f \n",Pi);
}
```

A one line addition to the CPU version

Future options subject to change without notice.
Heterogeneous Programming Model

Programmer designates code sections to run on MIC
- No further programming / API usage is needed
- Setup/teardown, data transfer, synchronization, are managed automatically by compiler and runtime

Offload is **optional**
- If MIC is missing or busy, program runs entirely on CPU
**Heterogeneous Memory Model (Non-shared Memory)**

**CPU and MIC do not share a common memory**

Two techniques are used to maintain program semantics with/without offload

- **Non-shared model**: Emulate shared data by copying back and forth at point of offload; supported for C/C++ and Fortran

- **Virtual-shared model**: Maintains coherence in a range of virtual addresses on CPU and MIC, automatically in software; C/C++ only

Both models might be used simultaneously in very same program but data maintained should be distinct
Offload Compile/Run Overview

**CPU Program**

```c
f()
{
    #pragma offload
    a = b + g();
}

__declspec (mic)
g()
{
}

h()
{
}
```

**Contents of MIC Program**

```c
f_part1()
{
    a = b + g();
}

__declspec (mic)
g()
{
}
```

**Execution**

At first offload, if MIC is installed and available, MIC program is loaded.

At each offload, if MIC is available, statement is run on MIC, else statement runs on CPU.

At program termination, MIC program is unloaded.
Offload Language Support

1: Offload Directives for marshalling
- Provides offload capability with pragma/directive
  - \#pragma offload in C/C++
  - !$dir omp offload directive in Fortran
- Offloaded data must be scalars, arrays, bit-wise copy-able structs (“flat data structures”)

2: Adds Offload Keywords and Virtual Shared Memory
- C/C++ keywords _Shared, _Offload(*) . All data types are supported
- Implemented with virtual shared-memory technology
- Enables pointer-based data, C structs and C++ classes, and locks, enabling a wider class of apps to be run on MIC

(*)names for keywords preliminary - might change for product release
# Offload Directives

## Non-Shared Model

<table>
<thead>
<tr>
<th></th>
<th>C/C++ Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>New offload pragma</td>
<td><code>#pragma offload &lt;clauses&gt;&lt;statement&gt;</code></td>
<td>Execute next statement on MIC (which could be an OpenMP parallel construct)</td>
</tr>
<tr>
<td>Compile function for CPU and MIC</td>
<td><code>__declspec ( target (MIC) )</code></td>
<td>Compile function for CPU and MIC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Fortran Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>New offload directive</td>
<td><code>!dir$ omp offload &lt;clauses&gt;</code></td>
<td>Execute next OpenMP parallel construct on MIC</td>
</tr>
<tr>
<td>Compile function for CPU and MIC</td>
<td><code>!dir$ attributes offload::&lt;MIC&gt; :: &lt;rtn-name&gt;</code></td>
<td>Compile function for CPU and MIC</td>
</tr>
</tbody>
</table>
### Offload Directives (contd.)

Variables restricted to scalars, structs, arrays and pointers to scalars / structs / arrays

<table>
<thead>
<tr>
<th>Clauses / Modifiers</th>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target specification</td>
<td><code>target ( name )</code></td>
<td>Where to run construct</td>
</tr>
<tr>
<td>Inputs</td>
<td><code>in (var-list modifiers_opt)</code></td>
<td>Copy CPU to target</td>
</tr>
<tr>
<td>Outputs</td>
<td><code>out (var-list modifiers_opt)</code></td>
<td>Copy target to CPU</td>
</tr>
<tr>
<td>Inputs &amp; outputs</td>
<td><code>inout (var-list modifiers_opt)</code></td>
<td>Copy both ways</td>
</tr>
<tr>
<td>Non-copied data</td>
<td><code>nocopy (var-list modifiers_opt)</code></td>
<td>Data is local to target</td>
</tr>
</tbody>
</table>

#### Modifiers

- **Specify pointer length**
  - `length (element-count-expr)`
  - Copy that many pointer elements

- **Control pointer memory allocation**
  - `alloc_if ( condition )`
  - Allocate new block of memory for pointer if condition is TRUE

- **Control freeing of pointer memory**
  - `free_if ( condition )`
  - Free memory used for pointer if condition is TRUE
**Offload Examples: OpenMP, Cilk**

### C/C++ OpenMP

```c
#pragma offload target (mic)
#pragma omp parallel for reduction(+:pi)
    for (i=0; i<count; i++) {
        float t = (float)((i+0.5)/count);
        pi += 4.0/(1.0+t*t);
    }
pi /= count
```

### C/C++ Cilk

```c
#pragma offload target (mic)
_Cilk_for (int i=0; i < count; i++)
{
    a[i] = b[i] * c + d;
}
```

### Fortran OpenMP

```fortran
!dir$ omp offload target (mic)
!omp parallel do
    do i=1,10
        A(i) = B(i) * C(i)
    enddo
!omp end parallel
```

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

2-D array \((a)\) is filled with data on the coprocessor

Data management handled automatically by the compiler
• Memory for \((a)\) allocated on coprocessor
• Private variables \((i, j, x)\) are created
• Result is copied back

```c
#include <omp.h> /* C example */
#include <stdlib.h>
#include <stdio.h>

int main() {
    const int n = 1024; /* Size of the array */
    float a[n][n]; /* Array */
    int i, j; /* Index variables */
    float x; /* Scalar */

    #pragma offload target(mic)
    #pragma omp parallel for shared(a), private(x),
    schedule(dynamic)
    for(i=0;i<n;i++)
        for(j=i;j<n;j++) {
            x = (float)(i + j);
            a[i][j] = x;
        }
}
```

Source: ACM paper: “Early Experiences with the Intel Many Integrated Cores Accelerated Computing Technology”
Offload Examples: Intel® TBB

```cpp
#pragma offload_attribute (push, target(mic))
#include "tbb/concurrent_hash_map.h"
...
#include "tbb/task_scheduler_init.h"
static void CountOccurrences() {
    StringTable table;
    parallel_for( blocked_range<string*>( Data, Data+N, 1000 ), Tally(table) );
    int n = 0;
    for( StringTable::iterator i=table.begin(); i!=table.end(); ++i )   {    ...    }
}

static void CreateData() {
    ...
    for( int i=0; i<N; ++i ) {
        Data[i] =
            Adjective[rand()%n_adjective];
        ...
    }
}

#pragma offload_attribute (pop)
int main() {
    #pragma offload target(mic) in(Verbose, NThread) {
    
        task_scheduler_init init(NThread);
        CreateData();
        CountOccurrences();
    }
}
```
Using Intel® MKL

Beside using Intel® MKL natively on Intel® MIC hardware, a “heterogeneous MKL” usage model is supported too

– There are no changes in MKL function names

C/C++ or Fortran code executing on the host, annotated for execution on Intel MIC card

```c
void foo()
{
    float *A, *B, *C; /* Matrices */
    #pragma offload target(mic)
        in(transa,transb,N,alpha,beta)
        in(A:length(matrix_elements))
        in(B:length(matrix_elements))
        in(C:length(matrix_elements))
        out(C:length(matrix_elements) alloc_if(0))
        sgemm(&transa, &transb, &N,&N,&N,&alpha,A,&N,B,&N,&beta,C, &N);
}
```
Offload Keyword Extension
Virtual Shared-Memory Model

MYO: Mine-Yours-Ours virtual shared memory implemented in software

Compiler allocates data exchanged between CPU and MIC in MYO shared memory

- Malloc’d memory must use MYO malloc/free routines
- Allows full C++ within limits of CPU and MIC object model overlap
- Pointers are no longer an issue when they point to shared data

MYO supports release consistency of shared memory between CPU and MIC

- Acquire/release of memory are automatically done around offloads
- Only modified data is transferred between CPU and MIC
# Keyword `_Shared` for Data and Functions

<table>
<thead>
<tr>
<th>What</th>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function available on both sides</td>
<td><code>int _Shared f(int x) { return x+1 }</code></td>
<td>The function may execute on either side</td>
</tr>
<tr>
<td>Global</td>
<td><code>_Shared int x = 0;</code></td>
<td>Visible on both sides</td>
</tr>
<tr>
<td>File/Function static</td>
<td><code>static _Shared int x;</code></td>
<td>Visible on both sides, only to code within the file/function</td>
</tr>
<tr>
<td>Class</td>
<td>`class _Shared x {...}</td>
<td>Static members are visible on both sides; methods and operators are available on both sides</td>
</tr>
<tr>
<td>Pointer to shared data</td>
<td><code>int _Shared *p;</code></td>
<td>P is allocated locally, can point to shared data</td>
</tr>
<tr>
<td>A shared pointer</td>
<td><code>int *__Shared p;</code></td>
<td>P is shared. (It should only point at shared data.)</td>
</tr>
</tbody>
</table>
## Offloading using `_Offload`

<table>
<thead>
<tr>
<th>Feature</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offloading a function call</td>
<td><code>x = _Offload func(y);</code></td>
<td>func executes on MIC</td>
</tr>
<tr>
<td>Offloading asynchronously</td>
<td><code>x = _Cilk_spawn _Offload func(y);</code></td>
<td>Non-blocking offload</td>
</tr>
<tr>
<td>Offload a parallel for-loop</td>
<td>`_Offload _Cilk_for (i = 0; i &lt; N; i++) {</td>
<td>Loop executes in parallel on MIC. The loop is implicitly outlined as a function call.</td>
</tr>
<tr>
<td></td>
<td>a[i] = b[i] + c[i]; }</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Examples of Offload Using Shared Data

typedef class _Shared
  _LinkedListNode{
    int data;
    class _LinkedListNode* next;
  } _LinkedListNode;

typedef _LinkedListNode LinkedList;

_Shared LinkedList listhead;
_Shared void calculateInLRB();

void initialize() {
  listhead.data = 5;
  listhead.next->data = 7;
  listhead.next->next->data = 9;
}

_Shared void calculateOnMIC() {
  listhead.data += 100;
  listhead.next->data += 100;
  listhead.next->next->data += 100;
}

main() {
  Initialize();
  _Offload calculateOnMIC();
  _Offload calculateOnMIC();
}

5 7 9
105 107 109
205 207 209
Summary Heterogeneous Compilation

A programming model + compiler for writing code to run on a CPU+ MIC system

Full integration into both C/C++ and Fortran makes the code easy to write

Enables use of our optimizing compilers on both CPU and MIC
- Vectorization
- Parallel programming with TBB, cilk, OpenMP

Enables *co-operative* processing between CPU and MIC
- Use both simultaneously for parallel processing
- Use CPU for serial code, MIC for data-parallel code

Using MIC is a simple extension of CPU programming
Agenda

• Introduction
• Knights Ferry SW Development Hardware
• Programming Environment for Intel® MIC
• Heterogeneous Programming / Offload Model
• Additional Developer Tools for Intel® MIC
• Summary
Intel® Vtune™ Amplifier for MIC

Intel® VTune™ Amplifier XE performance profiler to analyze performance of the Intel® MIC in offload mode

- Linux hosted command line tool that collects performance events (like cache misses) on Intel MIC card

- The VTune Amplifier GUI displays results collected in previous step highlighting bottlenecks, time spent and other details of performance
Intel® Debugger for MIC

- Debugging of host and target simultaneously
- If host application is being debugged, target application gets also debugged automatically
- Debugger runs on host for both host and target program
- Debugger halts and resumes both host and target program synchronously
- Full C++ and FORTRAN support on both sides
- Future debugger release goals are towards presenting only one virtual application view inside a single GUI
- Extendible to cover more than one offload card
Agenda

• Introduction
• Knights Ferry SW Development Hardware
• Programming Environment for Intel® MIC
• Heterogeneous Programming / Offload Model
• Additional Developer Tools for Intel® MIC
• Summary
Summary

• Intel tools and library take the parallel performance to extreme by using heterogeneous programming techniques on Intel multicore and Intel® Many Integrated Core Architecture

• Intel tools make it easy for software engineers to take an unified approach to compose, debug, profile and optimize parallel applications in any application domains
Legal Disclaimer & Optimization Notice

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

Copyright © , Intel Corporation. All rights reserved. Intel, the Intel logo, Xeon, Core, VTune, and Cilk
are trademarks of Intel Corporation in the U.S. and other countries. *Other names and brands may
be claimed as the property of others.

Optimization Notice

Intel’s compilers 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 SSE2, SSE3, and 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. Certain optimizations not specific to Intel microarchitecture are reserved for
Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information
regarding the specific instruction sets covered by this notice.

Notice revision #20110804