High Performance Parallel Programming

Multicore development tools with extensions to many-core. Investment protection. Scale Forward.
Enabling & Advancing Parallelism
High Performance Parallel Programming

Intel tools, libraries and parallel models extend to multicore, many-core and heterogeneous computing.

Use One Software Architecture Today. Scale Forward Tomorrow.
Intel® Software Development Products
Deliver Application Performance

**Advanced Performance**

- Intel® C/C++ and Fortran Compilers w/OpenMP
- Intel® MKL, Intel® Cilk™ Plus, Intel® TBB Library, Intel® IPP Library
- Intel® Inspector XE, Intel® VTune™ Amplifier XE, Intel® Advisor

**Cluster Performance**

- Intel® MPI Library
- Intel® Trace Analyzer and Collector
- Intel® Parallel Studio XE

**Foundation of Performance, Productivity, and Standards**
A Family of Parallel Programming Models

Developer Choice

**Intel® Cilk™ Plus**
- C/C++ language extensions to simplify parallelism
- Open sourced
- Also an Intel product

**Intel® Threading Building Blocks**
- Widely used C++ template library for parallelism
- Open sourced
- Also an Intel product

**Domain-Specific Libraries**
- Intel® Integrated Performance Primitives
- Intel® Math Kernel Library

**Established Standards**
- Message Passing Interface (MPI)
- OpenMP*
- Coarray Fortran
- OpenCL*

**Research and Development**
- Intel® Concurrent Collections
- Offload Extensions
- Intel® SPMD Parallel Compiler

Choosing from high-performance parallel programming models

Applicable to Multicore and Many-core Programming
Invest in Common Tools and Programming Models

**Multicore**

Intel® Xeon® processors are designed for intelligent performance and smart energy efficiency.

Continuing to advance Intel® Xeon® processor family and instruction set (e.g., Intel® AVX, etc.)

**Code**

Use One Software Architecture

**Many-core**

Intel® Xeon Phi™ coprocessors are ideal for highly parallel computing applications.

Software development platforms ramping now

*Use One Software Architecture Today. Scale Forward Tomorrow.*
Go Parallel with Intel® Cilk™ Plus

Proven Cilk parallel model, teachable in one minute

Parallelism in Three Key Words:
- `cilk_spawn`
- `cilk_sync`
- `cilk_for`

Cilk™ Plus: an open specification

Recently placed into open source by Intel for the advancement of parallel programming

// Parallel function invocation, in C
```
cilk_for (int i=0; i<n; ++i){
    Foo(a[i]);
}
```

// Parallel spawn in a recursive fibonacci computation, in C
```
int fib (int n) {
    if (n < 2) return 1;
    else {
        int x, y;
        x = cilkSpawn fib(n-1);
        y = fib(n-2);
        cilkSync;
        return x + y;
    }
}
```

Learn more at http://cilkplus.org

Intel® Cilk™ Plus is Applicable to Multicore and Many-core Programming
Go Parallel with Intel® Cilk™ Plus

Data and Task Parallelism as first class citizens in C and C++

Vectorization via intuitive notations that automatically span MMX, SSE, AVX, and wider widths in the future including those in the Intel® Xeon Phi™ coprocessors

- array notations
- #pragma SIMD controls
- elemental functions

// Simplify operation using array notations in C/C++:

    a[:] = b[:] + c[:];

// Elemental functions, in C, using Cilk Plus:

    __declspec (vector)
    void saxpy(float a, float x, float &y)
    {
        y += a * x;
    }

//pragma SIMD: User-mandated // vectorization

    #pragma simd
    for (i=0; i<n; i++) {
    }

Intel® Cilk™ Plus is Applicable to Multicore and Many-core Programming

Learn more at http://cilkplus.org
Go Parallel with Intel® Threading Building Blocks (Intel® TBB)

A popular parallel abstraction for C++ developers

- A C++ template library
- Scalable memory allocation
- Load-balancing
- Work-stealing task scheduling
- Thread-safe pipeline
- Concurrent containers
- High-level parallel algorithms
- Numerous synchronization primitives

Intel remains a leading participant and contributor in the TBB open source project as well as a leading supplier of TBB support and supporting tools.

Intel® TBB is Applicable to Multicore and Many-core Programming

Learn more at http://threadingbuildingblocks.org
Support TBB 4.2: The latest Intel architectures

- Transactional Synchronization Extensions (TSX)
- Intel® Xeon Phi™ coprocessor for Windows

Android* OS support
Windows Store support
Lower memory overhead
Improved handling of large memory requests
Better fork support
Parallel Patterns Library (PPL)*
Compatibility
Go Parallel with Message Passing Interface (MPI)
Intel® Message Passing Interface (Intel® MPI)

Extend your cluster solutions to the Intel® Xeon Phi™ coprocessor

E.g., Intel Xeon Phi™ coprocessor in every node of the cluster using Intel® MPI and Intel® Threading Building Blocks and/or Intel® Cilk™ Plus on nodes

Same model as an Intel® Xeon processor based cluster.

Intel is a leading vendor of MPI implementations and tools

Learn more at http://intel.com/go/mpi

MPI is applicable to Multicore and Many-core Programming
Go Parallel with Coarray Fortran

Intel® Fortran Compiler

A standard, explicit notation for data decomposition, such as that often used in message-passing models, expressed in a natural Fortran-like syntax.

For parallel programming on both shared memory and distributed memory systems

Learn more at http://intel.com/software/products

Coarray Fortran is Applicable to Multicore and Many-core Programming
Go Parallel with Coarray Fortran
Intel® Fortran Compiler

• Three coarray execution models
  • Images run on host with offload regions
  • Images run on both coprocessor and host
  • Images run natively on the coprocessor

• Last 2 models requires manual upload of referenced shared object libs including MPI (impi.so) and coarray (libicaf.so)
Go Parallel with OpenMP®
Intel® C/C++ and Fortran Compilers

<table>
<thead>
<tr>
<th>//C/C++ OpenMP® Pragma</th>
<th>!$Fortran OpenMP®</th>
</tr>
</thead>
<tbody>
<tr>
<td>#pragma omp parallel for reduction(+:pi)</td>
<td>!$omp parallel do</td>
</tr>
<tr>
<td>for (i=0; i&lt;count; i++) {</td>
<td>do i=1,10</td>
</tr>
<tr>
<td>float t = (float)((i+0.5)/count);</td>
<td>A(i) = B(i) * C(i)</td>
</tr>
<tr>
<td>pi += 4.0/(1.0+t*t);</td>
<td>enddo</td>
</tr>
<tr>
<td>}</td>
<td>!$omp end parallel do</td>
</tr>
<tr>
<td>pi /= count;</td>
<td></td>
</tr>
</tbody>
</table>

A flexible interface for developing parallel applications

★ An abstraction for multi-threaded solutions

OpenMP® is a standard used by many parallel applications

★ Supported by every major compiler for Fortran, C, and C++

Learn more at http://openmp.org

OpenMP® is Applicable to Multicore and Many-core programming
Go Parallel with OpenMP*
Intel® C/C++ and Fortran Compilers
(C Example)

```c
main()
{   double pi = 0.0f; long i;

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

printf("pi = %f\n",pi/N);
}
```

One Line Change to Offload to the Intel® Xeon Phi™ coprocessor

OpenMP* is Applicable to Multicore and Many-core Programming
Go Parallel with OpenMP*

Intel® C/C++ and Fortran Compilers
(Fortran Example)

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

One Line Change to Offload to the Intel® Xeon Phi™ coprocessor

OpenMP* is Applicable to Multicore and Many-core Programming
Partial OpenMP* 4.0
Intel® Composer XE 2013 SP1

- New directives to enable vectorization and offloading of execution to attached devices (i.e., coprocessors or accelerators) for C++ and Fortran
- **TARGET Constructs** enable creation of a data environment for attached devices, movement of data between host and devices, and execution of constructs on devices
- **SIMD Constructs** enable loops and functions to be executed concurrently by a thread team using SIMD vector instructions
- See [http://openmp.org](http://openmp.org) and the OpenMP API Specification Version 4.0 RC2 for our current implementation of the supported features.
- Changes to features since the OpenMP 4.0 RC2 spec are not yet supported
Other Supported OpenMP* 4.0 features

- OpenMP taskgroup directive construct for syncing child tasks
- Atomic clause seq_cst (sequential consistency) provides an implicit flush after all atomic operations
- OMP_PLACES - processor list available to the execution environment. Allows thread affinity control
- omp_get_proc_bind() - API to find thread affinity policy of next parallel region
- Extended support for Fortran 2003
Go Parallel with C/C++ Language Extensions

Simple Keyword Language Extensions to control offloading to Intel Xeon Phi™ coprocessor

C/C++ Language Extensions

```cpp
class _Shared common {
    int data1;
    char *data2;
    class common *next;
    void process();
};

_Sharde class common obj1, obj2;
...

_Cilk_spawn _Offload obj1.process();
_Cilk_spawn obj2.process();
...
```
Go Parallel with High Performance Math Kernel Library

Intel® Math Kernel Library (Intel® MKL)

void foo() /* Intel® Math Kernel Library */
{
    float *A, *B, *C; /* Matrices */
    sgemm(&transa, &transb, &N, &N, &N, &alpha, A, &N, B, &N, &beta, C, &N);
}

Implicit automatic offloading requires no code changes, simply link with the offload MKL Library

Intel® Xeon® processor

Intel® Xeon Phi™ coprocessor

Intel High Performance Math Kernel Library is Applicable to Multicore and Many-core Programming
Conditional Numerical Reproducibility (CNR) for unaligned memory

- Balances performance with reproducible results by allowing greater flexibility in code branch choice and by ensuring algorithms are deterministic. More information: training site or the Intel® MKL User’s Guide).

- This release extends the feature to remove memory alignment requirements

- Memory alignment is still recommended for best performance

C/C++ Offload Pragma
#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;

MKL Implicit Offload
//MKL implicit offload requires no source code changes, simply link with the offload MKL Library.

MKL Explicit Offload
#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);

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

C/C++ Language Extensions
class _Shared common {
    int data1;
    char *data2;
    class common *next;
    void process();
};

_Shiped class common obj1, obj2;
...
_Cilk_spawn _Offload obj1.process();
_Cilk_spawn obj2.process();
...
Parallelism with OpenCL*
Intel® OpenCL SDK

OpenCL* is a framework for writing programs that execute across heterogeneous platforms (e.g., CPUs, GPUs, many-core)

```c
//Simple per element multiplication using OpenCL*:

kernel void dotprod( global const float *a,
                     global const float *b,
                     global float *c)
{
    int myid = get_global_id(0);
    c[myid] = a[myid] * b[myid];
}
```

Learn more at http://intel.com/go/opencl

Intel is a leading participant in the OpenCL* standard efforts, and a vendor of solutions and related tools with early implementations available today.

OpenCL* addresses the needs of customers in specific segments

OpenCL is applicable to multicore and many-core programming
**SIMD Types for Intel® Architecture**

**AVX**
Vector size: **256 bit**
Data types:
- 8, 16, 32, 64 bit integer
- 32 and 64 bit float
VL: 4, 8, 16, 32

**Intel® MIC**
Vector size: **512 bit**
Data types:
- 32 bit integer
- 32 and 64 bit float
VL: 8, 16

Illustrations: Xi, Yi & results 32 bit integer
### Running your Application

#### Execution on the host and Intel® Xeon Phi™ coprocessor

<table>
<thead>
<tr>
<th>Without: Intel® Xeon Phi™ coprocessor(s) are absent</th>
<th>With: Intel® Xeon Phi™ coprocessor(s) are present</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application starts and executes on host</td>
<td>Application starts on host and executes portions on Intel® Xeon Phi™ coprocessor(s)</td>
</tr>
<tr>
<td></td>
<td>At runtime, if Intel® Xeon Phi™ coprocessor(s) are available, the target binary is loaded</td>
</tr>
<tr>
<td>At each offload, the construct runs on host cores/threads</td>
<td>At each offload, the construct runs on the Intel® Xeon Phi™ coprocessor(s)</td>
</tr>
<tr>
<td>Normal program termination on host</td>
<td>At program termination, target binary is unloaded</td>
</tr>
</tbody>
</table>
Using the Intel® Debugger

Overview

Debugging of host and target simultaneously

If host application is being debugged, target application is 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, C++ and Fortran support on both sides

Future: debugger presents view of one virtual application inside a single GUI

Extensible to cover more than one offload card
Analyzing your Application
Performance Analysis Tools

Intel® VTune™ Amplifier XE performance profiler

- Analyze your multicore and many-core performance
  - Analyze performance of the application in offload mode
  - Support for Intel® Xeon Phi™ coprocessors includes:
    - A Linux* hosted command line tool that collects events
    - The VTune™ Amplifier XE graphical user interface to display results collected in previous step highlighting bottlenecks, time spent and other details of performance.
Preserve Your Development Investment
Common Tools and Programming Models for Parallelism

C/C++
- Intel® Cilk Plus
- OpenCL*
- OpenMP*
- Intel® TBB
- Offload Pragmas
- Intel® C/C++ Compiler

Fortran
- Intel® Fortran Compiler
- Coarray
- Offload Directives
- OpenMP*

Offload Pragmas
- Intel® MKL
- Intel® MPI

Multicore
Heterogeneous Computing
Many-core

Develop Using Parallel Models that Support Heterogeneous Computing
Invest in Common Tools and Programming Models

**Multicore**

Intel® Xeon® processors are designed for intelligent performance and smart energy efficiency.

Continuing to advance Intel® Xeon® processor family and instruction set (e.g., Intel® AVX, etc.)

**Code**

Use One Software Architecture

**Many-core**

Intel® Xeon Phi™ coprocessors are ideal for highly parallel computing applications.

Software development platforms ramping now

Use One Software Architecture Today. Scale Forward Tomorrow.
Call to Action

• Evaluate the Intel® Software Development Products, including the family of Parallel Programming Models, for your High Performance needs:

http://www.intel.com/software/products/eval

Note: The Intel® Parallel Studio XE 2013 and Intel® Cluster Studio XE 2013 products include support for Intel® Xeon Phi™ coprocessors prior to the coprocessors being generally available.

• For product information see:

http://www.intel.com/software/products/products
Performance Caveats and Notes

Performance varies with each application, regardless of the technology and methods used.

Certain types of HPC applications are amenable to acceleration and it is important to understand their characteristics.

Once an application is identified to take advantage of acceleration, the high level and low level techniques are expected to work equally well.
## Using Language Extensions for Intel® MIC

Simple Offload Extensions with the Intel® Compilers

<table>
<thead>
<tr>
<th>C/C++ Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>New offload pragma</td>
<td>Execute next statement on target (which could be an OpenMP* parallel construct)</td>
</tr>
<tr>
<td>#pragma offload ( clauses )</td>
<td></td>
</tr>
<tr>
<td>Place function on target</td>
<td>Compile function for host and target</td>
</tr>
<tr>
<td>__declspec( target ( x ) )</td>
<td></td>
</tr>
<tr>
<td>Place on data target</td>
<td>Two arrays are created, one on the host and one on the Intel® Xeon Phi™ coprocessor</td>
</tr>
<tr>
<td>__declspec(target(MIC)) float array [8000];</td>
<td></td>
</tr>
</tbody>
</table>

### Fortran Syntax

<table>
<thead>
<tr>
<th>Fortran Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>New offload directive</td>
<td>Execute next OpenMP* parallel construct on target</td>
</tr>
<tr>
<td>!dir$ omp offload &lt;clauses&gt;</td>
<td></td>
</tr>
<tr>
<td>Place function on target</td>
<td>Compile function for host and target</td>
</tr>
<tr>
<td>!dir$ attributes offload:&lt;x&gt; :: &lt;rtn-name&gt;</td>
<td></td>
</tr>
</tbody>
</table>
Using Language Extensions (contd.)

Variables restricted to scalars, arrays and pointers to scalars/arrays, structs (which can be bitwise copied)

<table>
<thead>
<tr>
<th>What</th>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target specification</td>
<td>target ( name )</td>
<td>Where to run construct</td>
</tr>
<tr>
<td>Inputs</td>
<td>in (var-list modifiers&lt;sub&gt;opt&lt;/sub&gt;)</td>
<td>Copy CPU to target</td>
</tr>
<tr>
<td>Outputs</td>
<td>out (var-list modifiers&lt;sub&gt;opt&lt;/sub&gt;)</td>
<td>Copy target to CPU</td>
</tr>
<tr>
<td>Inputs &amp; outputs</td>
<td>inout (var-list modifiers&lt;sub&gt;opt&lt;/sub&gt;)</td>
<td>Copy both ways</td>
</tr>
<tr>
<td>Non-copied data</td>
<td>nocopy (var-list modifiers&lt;sub&gt;opt&lt;/sub&gt;)</td>
<td>Data is local to target</td>
</tr>
<tr>
<td>Modifiers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specify pointer length</td>
<td>length (element-count-expr)</td>
<td>Copy that many pointer elements</td>
</tr>
<tr>
<td>Control pointer memory allocation</td>
<td>alloc_if ( condition )</td>
<td>Allocate new block of memory for pointer if condition is TRUE</td>
</tr>
<tr>
<td>Control freeing of pointer memory</td>
<td>free_if ( condition )</td>
<td>Free memory used for pointer if condition is TRUE</td>
</tr>
</tbody>
</table>

Variables restricted to scalars, arrays and pointers to scalars/arrays, structs (which can be bitwise copied)
# Using other constructs for Intel® Xeon Phi™ coprocessor

Enhancements in control layers provide additional flexibility.

<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><code>func</code> executes on Intel Xeon Phi™ coprocessor</td>
</tr>
<tr>
<td>Offloading asynchronously</td>
<td><code>x = __cilk_spawn __Offload f(y);</code></td>
<td>Non blocking offload</td>
</tr>
<tr>
<td>Data available on both sides</td>
<td><code>_Shared int x = 0;</code></td>
<td>Allocated in the shared memory area, can be synchronized</td>
</tr>
<tr>
<td>Function available on both sides</td>
<td><code>int __Shared f(int x)</code></td>
<td>The function can execute on either side</td>
</tr>
<tr>
<td>Function call from Intel Xeon Phi™ coprocessor to the CPU</td>
<td><code>x = __Borrow func(y);</code></td>
<td>The caller on the Intel® Xeon Phi™ coprocessor, the callee on the CPU. (use the waiting thread on the CPU)</td>
</tr>
</tbody>
</table>
| Offload a parallel for loop      | `__Offload __cilk_for (i = 0; i < N; i++) {  
  a[i] = b[i] + c[i];
}                                   | Loop executes in parallel. The loop is implicitly outlined as a function call. (borrow inside the loop disallowed) |
| Offload array expressions        | `__Offload a[:,:] = b[:,: ] <op> c[:,:];`    | Array operations execute in parallel.                                      |
|                                  | `__Offload a[:,:] = elemental_func(b[:,:]);`|                                                                             |
Using the Intel® Math Kernel Library (Intel® MKL)

Heterogeneous Intel® Math Kernel Library (Intel® MKL) automatically extends existing Intel® MKL functions to use Intel® Xeon Phi™ coprocessor to accelerate computations.
Using Array Notation Parallel Constructs
Part of the Intel® Cilk™ Plus Parallel Model
Making it easier to express and exploit vectorization opportunities and wider SIMD units on IA and Intel® Xeon Phi™ coprocessor

- Deterministic vectorization
- Predictable performance
- Freely mixable with scalar C/C++
### ISV Guidance: Tools for Parallelism

<table>
<thead>
<tr>
<th>Memory</th>
<th>Lang</th>
<th>Model and Capability</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared</td>
<td>C++</td>
<td>Cilk™ Plus</td>
<td>• Language extensions for task and vector parallelism</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Serial semantics capability = low overhead &amp; powerful</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TBB</td>
<td>• Widely used C++ template library for task parallelism</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Contains a rich feature set for general purpose parallelism</td>
</tr>
<tr>
<td>C</td>
<td>Cilk™ Plus</td>
<td></td>
<td>• Language extensions for task and vector parallelism</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Serial semantics capability = low overhead &amp; powerful</td>
</tr>
<tr>
<td></td>
<td>OpenGL*</td>
<td></td>
<td>• Emerging industry standard for hybrid (CPU+GPU) computing</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Is low level – requires deep expertise and advanced knowledge</td>
</tr>
<tr>
<td>C or Fortran</td>
<td>OpenMP*</td>
<td></td>
<td>• Industry standard compiler based language with roots in HPC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Thread based with many controls to tweak behavior and get performance</td>
</tr>
<tr>
<td>Distributed</td>
<td>C++, C or Fortran</td>
<td>MPI</td>
<td>• Library based capability</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Enables apps to run on clusters as well as shared memory</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Works with all above models</td>
</tr>
</tbody>
</table>

Select from a variety of powerful tools to aid parallelism
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.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Copyright ©, Intel Corporation. All rights reserved. Intel, the Intel logo, Xeon, Core, Xeon Phi, VTune, and Cilk are trademarks of Intel Corporation in the U.S. and other countries.

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