Intel® Advisor XE
Vectorization Optimization
Performance is a Proven Game Changer
It is driving disruptive change in multiple industries

Protecting buildings from extreme events
Sophisticated mechanics simulations are performed to identify innovative ways to protect infrastructure from extreme events, such as natural disasters.

Solving Austin, Texas’s traffic problem
Running advanced traffic simulations to improve the models used to plan infrastructure and traffic control changes.

New possible treatments for Parkinson’s
Extensive calculations performed at supercomputer helped researchers to learn more about the protein structure’s evolution.

Click on a picture for details
The “Free Lunch” is over, really
Processor clock rate growth halted around 2005

Software must be parallelized to realize all the potential performance
Moore’s Law Is Going Strong
Hardware performance continues to grow exponentially

“We think we can continue Moore's Law for at least another 10 years.”

Intel Senior Fellow Mark Bohr, 2015
Changing Hardware Impacts Software
More cores → More Threads → Wider vectors

High performance software must be both:

- Parallel (multi-thread, multi-process)
- Vectorized
Vector Instructions are Dramatically Faster
Multiple arithmetic operations with a single instruction

Adding 2 vectors

<table>
<thead>
<tr>
<th></th>
<th>4.4</th>
<th>1.1</th>
<th>3.1</th>
<th>-8.5</th>
<th>-1.3</th>
<th>1.7</th>
<th>7.5</th>
<th>5.6</th>
<th>-3.2</th>
<th>3.6</th>
<th>4.8</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-0.3</td>
<td>-0.5</td>
<td>0.5</td>
<td>0</td>
<td>0.1</td>
<td>0.8</td>
<td>0.9</td>
<td>0.7</td>
<td>1</td>
<td>0.6</td>
<td>-0.5</td>
</tr>
<tr>
<td>=</td>
<td>4.1</td>
<td>0.6</td>
<td>3.6</td>
<td>-8.5</td>
<td>-1.2</td>
<td>2.5</td>
<td>8.4</td>
<td>6.3</td>
<td>-2.2</td>
<td>4.2</td>
<td>4.3</td>
</tr>
</tbody>
</table>

- These instructions are also referred to as Single Instruction Multiple Data (SIMD instructions)
Intel® Advanced Vector Extensions (Intel® AVX)

Intel® AVX
- 8x floats
- 4x doubles

Intel® AVX2
- 32x bytes
- 16x 16-bit shorts
- 8x 32-bit integers
- 4x 64-bit integers
- 2x 128-bit(!) integer

Vector length – the number of elements that can be processed.
Don’t use a single Vector lane!
Un-vectorized and un-threaded software will under perform
Permission to Design for All Lanes
Threading and Vectorization needed to fully utilize modern hardware
Untapped Potential Can Be Huge!

Threaded + Vectorized can be much faster than either one alone

Threaded | Vectorized
---|---
✓ | ✓
✓ | X
X | ✓
X | X

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. For more information go to [http://www.intel.com/performance](http://www.intel.com/performance)
Data-Driven Threading Design
Intel® Advisor XE – Thread Prototyping

Have you:

- Tried threading an app, but seen little performance benefit?
- Hit a “scalability barrier”? Performance gains level off as you add cores?
- Delayed a release that adds threading because of synchronization errors?

Breakthrough for threading design:

- Quickly prototype multiple options
- Project scaling on larger systems
- Find synchronization errors before implementing threading
- Separate design and implementation - Design without disrupting development

Add Parallelism with Less Effort, Less Risk and More Impact

http://intel.ly/advisor-xe
Data Driven Vectorization Design
Intel® Advisor XE – Vectorization Advisor

Have you:

- Recompiled with AVX2, but seen little benefit?
- Wondered where to start adding vectorization?
- Recoded intrinsics for each new architecture?
- Struggled with cryptic compiler vectorization messages?

Breakthrough for vectorization design

- What vectorization will pay off the most?
- What is blocking vectorization and why?
- Are my loops vector friendly?
- Will reorganizing data increase performance?
- Is it safe to just use pragma simd?

More Performance
Fewer Machine Dependencies
The Right Data At Your Fingertips
Get all the data you need for high impact vectorization

Filter by which loops are vectorized!
Trip Counts
What prevents vectorization?
Focus on hot loops
What vectorization issues do I have?
Which Vector instructions are being use?
How efficient is the code?
1. Compiler diagnostics + Performance Data + SIMD efficiency information

2. Guidance: detect problem and recommend how to fix it

3. Loop-Carried Dependency Analysis

4. Memory Access Patterns Analysis
### 1. Compiler diagnostics + Performance Data + SIMD efficiency information

<table>
<thead>
<tr>
<th>Function Call Site and Loops</th>
<th>Self Time</th>
<th>Total Time</th>
<th>Compiler Vectorization</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>loop in main()cache</code></td>
<td>0.019s</td>
<td>0.019s</td>
<td>Scalar</td>
</tr>
<tr>
<td><code>loop in main()cache</code></td>
<td>0.019s</td>
<td>0.019s</td>
<td>Vectorized</td>
</tr>
</tbody>
</table>

- **Vectorized SSE**: SIMD loop processing Float32, Float64 data types; handling Divisions, Square Roots operations
- **Pseudo loop**: Loop starts are excluded

<table>
<thead>
<tr>
<th>Loop Type</th>
<th>Why No Vectorization?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalar</td>
<td>vector dependence prevents vectorization</td>
</tr>
</tbody>
</table>

- Loop Type: Scalar
- Why No Vectorization?: vector dependence prevents vectorization
Efficiently Vectorize your code
Intel Advisor XE – Vectorization Advisor

Where should I add vectorization and/or threading parallelism?

Summary  Survey Report  Refinement Reports  Annotation Report  Suitability Report

Elapsed times 54.44s  Vectorized  Not Vectorized  FILTERS: All Modules  All Sources

Function Cell Sites and Loops

Vector Issues  Self Time  Total Time  Trip Counts  Loop Type  Why No Vectorization?  Vectorized Loops

Objective to add vectorization to the following loops:

1. for (nl = 1; nl <= i_L; ++nl)

2. for (i_L = 1; i_L <= i_L; ++i_L)

File: loops1.cpp:3509 in s273

Line  Source  Total Time  %  Loop Time  %

3504  forntime_ |(c1)|;
3505  i_L = *ntimes;
3506  for (nl = 1; nl <= i_L; ++nl)

[loop at loops1.cpp:3506 in s273]
Scalar Loop. Not vectorized: inner loop was already vectorized
No loop transformations were applied

3507  {
3508  i_L = *nt;
3509  for (i_L = 1; i_L <= i_L; ++i_L)

[loop at loops1.cpp:3509 in s273]
Vectorized AVX Loop processing Float32; Float64; Int32 data type(s) having Ineters; Extractors; Masked

Selected (Total Time): 0.010s
Background on loop vectorization

A typical vectorized loop consists of

Main vector body
• Fastest among the three!

Optional peel part
• Used for the unaligned references in your loop. Uses Scalar or slower vector

Remainder part
• Due to the number of iterations (trip count) not being divisible by vector length. Uses Scalar or slower vector.

Larger vector register means more iterations in peel/remainder
• Make sure you Align your data!
• Make the number of iterations divisible by the vector length!
Intel Advisor XE shows how much time you are spending in the various parts of your loops!
1. Compiler diagnostics + Performance Data + SIMD efficiency information

2. Guidance: detect problem and recommend how to fix it

- **Issue:** Peeled/Remainder loop(s) present
  - All or some source loop iterations are not executing in the kernel loop. Improve performance by moving source loop iterations from peeled/remainder loops to the kernel loop. Read more at Vector Essentials, Utilizing Full Vectors.
  - **Recommendation:** Align memory access
    - **Projected maximum performance gain:** High
    - **Projection confidence:** Medium
    - The compiler created a peeled loop because one of the memory accesses in the source loop does not start at a data boundary. Align the memory access and tell the compiler your memory access is aligned. This example aligns memory using a 32-byte boundary:
      ```
      float *array;
      array = (float *)malloc(ARRAY_SIZE*sizeof(float), 32);
      // Somewhere else
      assume_aligned(array, 32);
      // Use array in loop
      ```
Get Specific Advice For Improving Vectorization

Intel® Advisor XE – Vectorization Advisor

Advisor XE shows hints to move iterations to vector body.
Don’t Just Vectorize, Vectorize Efficiently
See detailed times for each part of your loops. Is it worth more effort?

[Image of a software interface showing vectorization analysis]

<table>
<thead>
<tr>
<th>Function Call Sites and Loops</th>
<th>Vector Issues</th>
<th>Self Time</th>
<th>Total Time</th>
<th>Loop Type</th>
<th>Why No Vectorization?</th>
</tr>
</thead>
<tbody>
<tr>
<td>[loop at fractal.cpp:179 in &lt;lambda1&gt;]: op ...</td>
<td>High vector</td>
<td>0,013s</td>
<td>12,020s</td>
<td>Collapse</td>
<td>Collapse</td>
</tr>
<tr>
<td>[loop at fractal.cpp:179 in &lt;lambda1&gt;]: o ...</td>
<td>Serialized use</td>
<td>0,013s</td>
<td>11,281s</td>
<td>Vectorized (Body)</td>
<td></td>
</tr>
<tr>
<td>[loop at fractal.cpp:179 in &lt;lambda1&gt;]: o ...</td>
<td>Data type conversion</td>
<td>0,000s</td>
<td>0,163s</td>
<td>Peeled</td>
<td></td>
</tr>
<tr>
<td>[loop at fractal.cpp:179 in &lt;lambda1&gt;]: o ...</td>
<td>Data type conversion</td>
<td>0,000s</td>
<td>0,576s</td>
<td>Remainder</td>
<td></td>
</tr>
<tr>
<td>[loop at fractal.cpp:177 in &lt;lambda1&gt;]: oper ...</td>
<td>Data type conversion</td>
<td>0,010s</td>
<td>12,030s</td>
<td>Scalar</td>
<td></td>
</tr>
</tbody>
</table>
Critical Data Made Easy
Loop Trip Counts

Knowing the time spent in a loop is not enough!

Check actual trip counts

Loop is iterating 101 times but called > million times

Since the loop is called so many times it would be a big win if we can get it to vectorize.
1. Compiler diagnostics + Performance Data + SIMD efficiency information

<table>
<thead>
<tr>
<th>Function Call Site and Loops</th>
<th>Self Time</th>
<th>Total Time</th>
<th>Compiler Vectorization</th>
<th>Loop Type</th>
<th>Why No Vectorization?</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="0x0">loop in main:Complex loops</a></td>
<td>0.00ms-0.004ms</td>
<td>0.00ms-0.006ms</td>
<td>0.10ms-0.274ms</td>
<td>Scale</td>
<td>vector dependence prevents vectorization</td>
</tr>
<tr>
<td><a href="0x0">loop in main:Complex loops</a></td>
<td>0.00ms-0.004ms</td>
<td>0.00ms-0.006ms</td>
<td>0.10ms-0.274ms</td>
<td>Scale</td>
<td>inner loop was already vectorized</td>
</tr>
</tbody>
</table>

- Vectorizer SSE: SSE2 loop processing float4/ float64 data type(s); handling divisions; Square Roots operations
- Vectorizer Loop: Loops were vectorized

2. Guidance: detect problem and recommend how to fix it

**Issue:** Peeled/Remainder loop(s) present

All or some source loop iterations are not executing in the kernel loop. Improve performance by moving source loop iterations from peeled/remainder loops to the kernel loop. Read more at Vector Essentials, Utilizing Full Vectors.

**Recommendation:** Align memory access

Projected maximum performance gain: High

Projection confidence: Medium

The compiler created a peeled loop because one of the memory accesses in the source loop does not start at a data boundary. Align the memory access and tell the compiler your memory access is aligned. This example aligns memory using a 32-byte boundary:

```c
float *array;
array = (float *)memalloc(ARRAY_SIZE*sizeof(float), 32);
// Somewhere else
assume_aligned(array, 32);
// Use array in loop
```

3. Loop-Carried Dependency Analysis

<table>
<thead>
<tr>
<th>Problems and Messages</th>
<th>ID</th>
<th>Type</th>
<th>Site Name</th>
<th>Sources</th>
<th>Modules</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1 Parallel site information site2</td>
<td>dgtest2.cpp</td>
<td>dgtest2</td>
<td>✔ Not a problem</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2 Read after write dependency site2</td>
<td>dgtest2.cpp</td>
<td>dgtest2</td>
<td>New</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P3 Read after write dependency site2</td>
<td>dgtest2.cpp</td>
<td>dgtest2</td>
<td>New</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P4 Write after write dependency site2</td>
<td>dgtest2.cpp</td>
<td>dgtest2</td>
<td>New</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P5 Write after write dependency site2</td>
<td>dgtest2.cpp</td>
<td>dgtest2</td>
<td>New</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P6 Write after read dependency site2</td>
<td>dgtest2.cpp</td>
<td>dgtest2</td>
<td>New</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P7 Write after read dependency site2</td>
<td>dgtest2.cpp, idle.h</td>
<td>dgtest2</td>
<td>New</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Is It Safe to Vectorize?

Loop-carried dependencies analysis verifies correctness

Select loop for Correct Analysis and press play!

Vector Dependence prevents Vectorization!
Data Dependencies – Tough Problem #1

Is it safe to force the compiler to vectorize?

Data dependencies

DO I = 1, 10000
  A(I) = B(I) * 17
  X(I+1) = X(I) + A(I)
ENDDO

// Need the ability to check if it
// it is safe to force the compiler
// the compiler to vectorize!

Issue: Assumed dependency present

The compiler assumed there is an anti-dependency (Write after read – WAR) or true dependency (Read after write – RAW) in the loop. Improve performance by investigating the assumption and handling accordingly.

Enable vectorization

The Correctness analysis shows there is no real dependency in the loop for the given workload. Tell the compiler it is safe to vectorize using the restrict keyword or a directive.

<table>
<thead>
<tr>
<th>ICL/ICC/ICPC Directive</th>
<th>IFORT Directive</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>#pragma simd or #pragma omp simd</td>
<td>!DIRS SIMD or !$OMP SIMD</td>
<td>Ignores all dependencies in the loop</td>
</tr>
<tr>
<td>#pragma ivdep</td>
<td>!DIRS IVDEP</td>
<td>Ignores only vector dependencies (which is safest)</td>
</tr>
</tbody>
</table>

Read More:

- User and Reference Guide for the Intel C++ Compiler 15.0 > Compiler Reference > Pragmas > Intel-specific Pragma Reference >
  - ivdep
  - omp simd
Correctness – Is It Safe to Vectorize?

Loop-carried dependencies analysis

Received recommendations to force vectorization of a loop:

1. Mark-up the loop and check for the presence of REAL dependencies

2. Explore dependencies in more details with code snippets

In this example 3 dependencies were detected

- RAW – Read After Write
- WAR – Write After Read
- WAW – Write After Write

This is NOT a good candidate to force vectorization!
1. Compiler diagnostics + Performance Data + SIMD efficiency information

- Self Time
- Total Time
- Compiler Vectorization
- Loop Type
- Why No Vectorization?

2. Guidance: detect problem and recommend how to fix it

- Issue: Peeled/Remainder loop(s) present
  - All or some source loop iterations are not executing in the kernel loop. Improve performance by moving source loop iterations from peeled/remainder loops to the kernel loop. Read more at Vector Essentials, Utilizing Full Vectors...

- Recommendation: Align memory access
  - Projected maximum performance gain: High
  - Proportion confidence: Medium
  - The compiler created a peeled loop because one of the memory accesses in the source loop does not start at a data boundary. Align the memory access and tell the compiler your memory access is aligned. This example aligns memory using a 32-byte boundary:

```c
float *array;
array = (float*)malloc(ARRAY_SIZE*sizeof(float), 32);
// Somewhere else
assume_aligned(array, 32);
// Use array in loop
```

3. Loop-Carried Dependency Analysis

<table>
<thead>
<tr>
<th>ID</th>
<th>Type</th>
<th>Site Name</th>
<th>Sources</th>
<th>Modules</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>∙</td>
<td>Par.sit inf</td>
<td>s2</td>
<td>dgt2.cpp</td>
<td>dgt2</td>
</tr>
<tr>
<td>P2</td>
<td>∙</td>
<td>Read after write</td>
<td>s2</td>
<td>dgt2.cpp</td>
<td>dgt2</td>
</tr>
<tr>
<td>P3</td>
<td>∙</td>
<td>Read after write</td>
<td>s2</td>
<td>dgt2.cpp</td>
<td>dgt2</td>
</tr>
<tr>
<td>P4</td>
<td></td>
<td></td>
<td>s2</td>
<td>dgt2.cpp</td>
<td>dgt2</td>
</tr>
<tr>
<td>P5</td>
<td></td>
<td></td>
<td>s2</td>
<td>dgt2.cpp</td>
<td>dgt2</td>
</tr>
<tr>
<td>P6</td>
<td></td>
<td></td>
<td>s2</td>
<td>dgt2.cpp</td>
<td>dgt2</td>
</tr>
<tr>
<td>P7</td>
<td></td>
<td></td>
<td>s2</td>
<td>dgt2.cpp</td>
<td>dgt2</td>
</tr>
</tbody>
</table>

4. Memory Access Patterns Analysis
Non-Contiguous Memory – Tough Problem #2
Potential to vectorize but may be inefficient

- Non-unit strided access to arrays

```c
for (i=0; i<N; i+=2) //Incrementing “i” by 2 is not unit stride
    //We need a way to check how we are
    //accessing memory.
```

- Indirect reference in a loop

```c
for (i=0; i<N; i++)
    A[B[i]] = C[i]*D[i]; //We have to decode B[i] to find out
    //which element of A to reference
```
Improve Vectorization
Memory Access pattern analysis

Run Memory Access Patterns analysis, just to check how memory is used in the loop and the called function.
Find vector optimization opportunities
Memory Access pattern analysis

All memory accesses are uniform, with zero unit stride, so the same data is read in each iteration.

We can therefore declare this function using the `omp` syntax:

```c
#pragma omp declare simd uniform(x0
```
Quickly Find Loops with Non-optimal Stride
Memory Access pattern analysis

- Quickly identify loops that are good, bad or mixed.
- Unit stride memory accesses are preferable.
- Find unaligned data
Vectorization is a tough problem
It is decomposable into tractable steps
Get help at each step:
  ▪ Find the best opportunities
  ▪ Improve vectorization effectiveness
  ▪ Assure correctness

Download the Beta Today!
Intel® Parallel Studio XE 2016

Download Today
Google: “Intel Parallel Studio 2016”
Or go directly to:
Intel® Parallel Studio XE
Faster code faster!

Vectorizing Compiler
Squeeze all the performance out of the latest instruction set

Threaded Performance Libraries
Pre-vectorized, pre-threaded, pre-optimized

High Level Parallel Models
Productive solutions for thread, process & vector parallelism

Parallel Performance Profilers
Quickly discover bottlenecks and tune for high performance

Thread Debugger
Find and debug non-deterministic threading errors

Vectorization Optimization and Thread Prototyping
Data driven design tools help you vectorize & thread effectively

Download Today
Google: “Intel Parallel Studio 2016”
Or go directly to: https://software.intel.com/en-us/articles/intel-parallel-studio-xe-2016-beta
Additional Resources

All links start with: https://software.intel.com/


For Intel® Xeon Phi™ coprocessors, but also applicable:

Intel® Composer XE  User and Reference Guides:
https://software.intel.com/compiler_15.0_ug_c
https://software.intel.com/compiler_15.0_ug_f

Compiler User Forums:  http://software.intel.com/forums
### Configurations for Binomial Options SP

![Graph showing Binomial Options SP (Higher is Better)](image)

#### 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

Performance measured in Intel Labs by Intel employees

### Platform Hardware and Software Configuration

<table>
<thead>
<tr>
<th>Platform</th>
<th>Unscaled Core Frequency</th>
<th>Cores/Socket</th>
<th>Num Sockets</th>
<th>L1 Data Cache</th>
<th>L1 I Cache</th>
<th>L2 Cache</th>
<th>L3 Cache</th>
<th>Memory</th>
<th>Memory Frequency</th>
<th>Memory Access</th>
<th>H/W Prefetchers Enabled</th>
<th>HT Enabled</th>
<th>Turbo Enabled</th>
<th>C States</th>
<th>O/S Name</th>
<th>Operating System</th>
<th>Compiler Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® Xeon™ 5472 Processor</td>
<td>3.0 GHZ</td>
<td>4</td>
<td>2</td>
<td>32K</td>
<td>32K</td>
<td>12 MB</td>
<td>None</td>
<td>32 GB</td>
<td>800 MHZ</td>
<td>UMA</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Disabled</td>
<td>Fedora 20</td>
<td>3.11.10-301.fc20</td>
<td>icc version 14.0.1</td>
</tr>
<tr>
<td>Intel® Xeon™ X5570 Processor</td>
<td>2.93 GHZ</td>
<td>4</td>
<td>2</td>
<td>32K</td>
<td>32K</td>
<td>256K</td>
<td>8 MB</td>
<td>48 GB</td>
<td>1333 MHZ</td>
<td>NUMA</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Disabled</td>
<td>Fedora 20</td>
<td>3.11.10-301.fc20</td>
<td>icc version 14.0.1</td>
</tr>
<tr>
<td>Intel® Xeon™ X5680 Processor</td>
<td>3.33 GHZ</td>
<td>6</td>
<td>2</td>
<td>32K</td>
<td>32K</td>
<td>256K</td>
<td>12 MB</td>
<td>48 MB</td>
<td>1333 MHZ</td>
<td>NUMA</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Disabled</td>
<td>Fedora 20</td>
<td>3.11.10-301.fc20</td>
<td>icc version 14.0.1</td>
</tr>
<tr>
<td>Intel® Xeon™ E5 2690 Processor</td>
<td>2.9 GHZ</td>
<td>8</td>
<td>2</td>
<td>32K</td>
<td>32K</td>
<td>256K</td>
<td>20 MB</td>
<td>64 GB</td>
<td>1600 MHZ</td>
<td>NUMA</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Disabled</td>
<td>Fedora 20</td>
<td>3.11.10-301.fc20</td>
<td>icc version 14.0.1</td>
</tr>
<tr>
<td>Intel® Xeon™ E5 2697v2 Processor</td>
<td>2.7 GHZ</td>
<td>12</td>
<td>2</td>
<td>32K</td>
<td>32K</td>
<td>256K</td>
<td>30 MB</td>
<td>64 GB</td>
<td>1867 MHZ</td>
<td>NUMA</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Disabled</td>
<td>Fedora 20</td>
<td>3.11.10-301.fc20</td>
<td>icc version 14.0.1</td>
</tr>
<tr>
<td>Codename Haswell</td>
<td>2.2 GHZ</td>
<td>14</td>
<td>2</td>
<td>32K</td>
<td>32K</td>
<td>256K</td>
<td>35 MB</td>
<td>64 GB</td>
<td>2133 MHZ</td>
<td>NUMA</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Disabled</td>
<td>Fedora 20</td>
<td>3.13.5-202.fc20</td>
<td>icc version 14.0.1</td>
</tr>
</tbody>
</table>
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 © 2015, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

<table>
<thead>
<tr>
<th>Optimization Notice</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
</tr>
</tbody>
</table>

Notice revision #20110804