Intel Software Tools

Stephen Blair-Chappell
Intel Compiler Labs
Intel® Parallel Studio XE 2013 and Intel® Cluster Studio XE 2013

Helping Developers Efficiently Produce Fast, Scalable and Reliable Applications
Intel® Parallel Studio XE 2013 and Intel® Cluster Studio XE 2013

More Cores
Multicore Many-core
Xeon Xeon Phi
50+ cores

Wider Vectors
128 Bits
256 Bits
512 Bits

Scaling Performance Efficiently
Serial Performance
Task & Data Parallel Performance
Distributed Performance

• Industry-leading performance from advanced compilers
• Comprehensive libraries
• Parallel programming models
• Insightful analysis tools
## Intel® Parallel Studio XE 2013 and Intel® Cluster Studio XE 2013 †

<table>
<thead>
<tr>
<th>Phase</th>
<th>Product</th>
<th>Feature</th>
<th>Benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Build</strong></td>
<td>Intel® Advisor XE</td>
<td>Threading design assistant (Studio products only)</td>
<td>• Simplifies, demystifies, and speeds parallel application design</td>
</tr>
<tr>
<td></td>
<td>Intel® Composer XE</td>
<td>• C/C++ and Fortran compilers</td>
<td>• Enabling solution to achieve the application performance and scalability benefits of multicore and forward scale to many-core</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Intel® Threading Building Blocks</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Intel® Cilk™ Plus</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Intel® Integrated Performance Primitives</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Intel® Math Kernel Library</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Intel® MPI Library‡</td>
<td>High Performance Message Passing (MPI) Library</td>
<td>• Enabling High Performance Scalability, Interconnect Independence, Runtime Fabric Selection, and Application Tuning Capability</td>
</tr>
<tr>
<td></td>
<td>Intel® VTune™ Amplifier XE</td>
<td>Performance Profiler for optimizing application performance and scalability</td>
<td>• Remove guesswork, saves time, makes it easier to find performance and scalability bottlenecks</td>
</tr>
<tr>
<td><strong>Verify &amp; Tune</strong></td>
<td>Intel® Inspector XE</td>
<td>Memory &amp; threading dynamic analysis for code quality</td>
<td>• Increased productivity, code quality, and lowers cost, finds memory, threading, and security defects before they happen</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Static Analysis for code quality</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Intel® Trace Analyzer &amp; Collector†</td>
<td>MPI Performance Profiler for understanding application correctness &amp; behavior</td>
<td>• Analyze performance of MPI programs and visualize parallel application behavior and communications patterns to identify hotspots</td>
</tr>
</tbody>
</table>

**Efficiently Produce Fast, Scalable and Reliable Applications**
## Top New Features

### Performance
- Improved compiler and library performance
  - Ivy Bridge microarchitecture
  - Haswell microarchitecture
  - Intel® Xeon Phi™ coprocessor

### Performance Profiling
- A dozen new analysis features
  - Low overhead Java* profiling
  - CPU Power Analysis

### Reliability
- Pointer checker
- Heap growth analysis
- Improved MPI fault tolerance†

### Reproducibility
- Conditional numerical reproducibility

### Standards
- Expanded C++ 11
- Expanded Fortran 2008
- MPI 2.2†

### Parallelism Assistance
- Analysis extended to include Linux*, Fortran and C# (in addition to Windows* and C/C++)

---

Efficiently produce fast, scalable and reliable applications running on Windows* and Linux*
A Story ...

A Bank near you!
The Reason Why

- Long overnight runtime
- Cost of renting space in data centres
- Power Consumption

Main bottleneck is here
One Line of Code (in a Monte Carlo Calculation) takes 50% of the time.
On the graphs, bigger is better.

The Same Source Change Improves Performance on Both Targets

HOW DO WE GET THERE?

Parallelization and vectorization together improve performance per second by >80X and by >50X.

The Same Source Change Improves Performance on Both Targets

Parallel

Vectorised
## Timing Summary

<table>
<thead>
<tr>
<th></th>
<th>initialization</th>
<th>calculation</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>MS VS 10 - CL, base line</td>
<td>1324</td>
<td>627</td>
<td>1951</td>
</tr>
<tr>
<td>ICL, base line</td>
<td>1172</td>
<td>487</td>
<td>1659</td>
</tr>
<tr>
<td>ICL, vectorized</td>
<td>1161</td>
<td>202</td>
<td>1363</td>
</tr>
<tr>
<td>ICL, vectorized + OMP threading</td>
<td>612</td>
<td>105</td>
<td>717</td>
</tr>
<tr>
<td>ICL, vectorized + Cilk tasking</td>
<td>608</td>
<td>117</td>
<td>725</td>
</tr>
<tr>
<td>ICL, vectorized + OMP + MKL VSL</td>
<td>99</td>
<td>103</td>
<td>202</td>
</tr>
</tbody>
</table>

Source code freely available: Please contact presenter in case you want to have the source code and build scripts to reproduce the measurements

ArraySection version available from software.intel.com (search for Black-Scholes)
Three Common Requests

“How can I make my program run faster?”

“How can I make my program parallel?”

“Will my code run on any CPU? - compatibility”
Programs on Xeon Phi . . .

Code must be highly Parallel effectively Vectorised
And three more questions of late . . .

Will my code run on a Xeon Phi?

Do I have to change my code to much?

What performance will I get?
Four Components

- Intel® Composer XE
  - Use to generate fast, safe, parallel code (C/C++, Fortran)

- Intel® VTune™ Amplifier XE
  - Find hotspots and bottlenecks in your code.

- Intel® Inspector XE
  - Use to find memory and threading errors

- Intel® Parallel Advisor
  - Use to model parallelism in your existing applications

+ Advisor
Three Common Requests

“How can I make my program run faster?”

“How can I make my program parallel?”

“Will my code run on any CPU? - compatibility”
The compiler uses many optimisation techniques

- profile guided optimisation
- architecture-specific optimisations
- auto-vectorisation
- auto-Parallelism
- function inlining
- fast floating point

- O1
- O2
- O3

optimise for size
inter procedural optimisation
optimised runtime functions

- fast intrinsic functions

Often we are happy with out-of-the-box experience.

When was the last time you looked at some documentation?
## SIMD Instruction Enhancements

<table>
<thead>
<tr>
<th>Year</th>
<th>Instruction Set</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1999</td>
<td>SSE</td>
<td>Single-precision vectors, Streaming operations</td>
</tr>
<tr>
<td>2000</td>
<td>SSE2</td>
<td>70 instructions, Double-precision vectors, Vector integer operations</td>
</tr>
<tr>
<td>2004</td>
<td>SSE3</td>
<td>144 instructions, Complex data, Video Graphics building blocks</td>
</tr>
<tr>
<td>2006</td>
<td>SSSE3</td>
<td>70 instructions, Double-precision vectors, Vector integer operations</td>
</tr>
<tr>
<td>2007</td>
<td>SSE4.1</td>
<td>47 instructions, String/XML processing, POPO-Count CRC</td>
</tr>
<tr>
<td>2008</td>
<td>SSE4.2</td>
<td>8 instructions, Encryption and Decryption, Key Generation</td>
</tr>
<tr>
<td>2009</td>
<td>AES-NI</td>
<td>7 instructions, Encryption and Decryption, Key Generation</td>
</tr>
<tr>
<td>2011</td>
<td>AVX</td>
<td>~100 new instructions, ~300 legacy SSE instructions, Vector shifts, Gather</td>
</tr>
<tr>
<td>2012</td>
<td>AVX2</td>
<td>Int. AVX expands to 256 bit, Improved bit manipulation, FMA</td>
</tr>
<tr>
<td>2012</td>
<td>MIC</td>
<td>512-bit vector</td>
</tr>
</tbody>
</table>

### Code Example

```c
for (i=0; i<MAX; i++)
    c[i] = a[i] + b[i];
```
**Other Ways of Inserting Vectorised Code**

- **Use Performance Libraries (e.g. IPP and MKL)**
- **Compiler: Fully automatic vectorization**
- **Cilk Plus Array Notation**
- **Compiler: Auto vectorization hints (#pragma ivdep, ...)**
- **User Mandated Vectorization (SIMD Directive)**
- **Manual CPU Dispatch (__declspec(cpu_dispatch ...))**
- **SIMD intrinsic class (F32vec4 add)**
- **Vector intrinsic (mm_add_ps())**
- **Assembler code (addps)**

**Ease of use**

- **Implicit**
- **Explicit**

**Programmer control**

**Instruction aware**
An example

<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>

Speedup by upgrading silicon

Verified using VTune

Faster Code

Speedup by swapping compiler

<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>

Three Common Requests

“How can I make my program run faster?”

“How can I make my program parallel?”

“Will my code run on any CPU? - compatibility”
Speedup using parallelism

Parallel Code

1. Analyze
   - Amplifier XE
   - Hotspot
   - EBS (XE only)

2. Implement
   - Composer XE
     - Compiler
     - Cilk Plus
     - OpenMP
     - Libraries
       - MKL
       - TBB
     - IPP

3. Debug
   - Inspector XE
     - Threads
     - Memory

4. Tune
   - Amplifier XE
     - concurrency
     - Locks & waits

Four Step Development
Language to help parallelism

Intel® Cilk™ Plus

OpenMP

Intel® Threading Building Blocks

Intel® MPI

Fortran Coarrays

OpenCL

Native Threads

cilk_for (int i = 0; i < max_row; i++)
{
    for (int j = 0; j < max_col; j++)
    {
        p[i][j] = mandel( complex(scale(i), scale(j)));
    }
}

#pragma omp parallel for
for(i=1;i<=4;i++)
{
    printf("Iter: %d", i);
}
An example ...

1. Hotspot Analysis
2. Implement
3. Find Threading Errors
4,5,6. Tune Parallelism

Three Common Requests

“How can I make my program run faster?”

“How can I make my program parallel?”

“Will my code run on any CPU? - compatibility”
Will my program run on any CPU?

Compatibility

- **run?**

Future Proofing

- **build?**

OS-agnostic
CPU-agnostic
Language / Standards
Tools
Scalability

- **Performance?**
Thank You
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, 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