Intel on HPC: great hardware with a twist: best support for standards and programmability

James Reinders, Intel
March 24, 2015
Naval Air Station Patuxent River

james.r.reinders@intel.com
James Reinders
james.r.reinders@intel.com

lots of cores.com ← slides

Photo (c) 2014, James Reinders; used with permission; Yosemite Half Dome rising through forest fire smoke 11am on September 10, 2014
Twice: More than one sustained TeraFlop/sec

ASCI Red: 1 TeraFlop/sec
December 1996

- 1 TF/s
- 7264 Intel® Pentium Pro processors
- 72 Cabinets

Knights Corner: 1 TeraFlop/sec
November 2011

- 1 TF/s
- one chip
- 22nm
- One PCI express slot

Twice: More than one sustained TeraFlop/sec

More than three sustained TeraFlop/sec

ASCI Red: 1 TeraFlop/sec
December 1996

1 TF/s
7264 Intel® Pentium Pro processors

1999: upgraded to 3.1 TF/s with
9298 Intel® Pentium II Xeon processors

72 Cabinets

Knights Corner
November 2011

1 TF/s
one chip

22nm
One PCI express slot

Twice: More than one sustained TeraFlop/sec
Twice: More than three sustained TeraFlop/sec

ASCI Red: 1 TeraFlop/sec
December 1996

1 TF/s
7264 Intel® Pentium Pro processors
1999: upgraded to 3.1 TF/s with
9298 Intel® Pentium II Xeon processors
72 Cabinets

Knights Corner
November 2011

1 TF/s
one chip
22nm
One PCI express slot

Knights Landing
2015

3 TF/s
14nm
One Processor

Parallelism is the Future
Processor Clock Rate over Time

Growth halted around 2005

Source: © 2014, James Reinders, Intel, used with permission
Transistors per Processor over Time

Continues to grow exponentially (Moore's Law)

Source: © 2014, James Reinders, Intel, used with permission
Moore’s Law

Number of components (transistors) doubles about every 18-24 months.

Source: Intel Corp., used with permission
Parallelism is key +
Exploit locality of data
Parallelism is key
I want you to understand why
How we proceed impacts portability & programmability
Design Question: Computation?

A few powerful vs. Many less powerful.

Diagrams for discussion purposes only, not a precise representation of any product of any company.
Design Question

A few powerful vs. Many much less powerful and very restrictive.

Diagrams for discussion purposes only, not a precise representation of any product of any company.
Design Question

A few powerful vs. Many less powerful.

Same programming models, languages, optimizations and tools.

Diagrams for discussion purposes only, not a precise representation of any product of any company.
Design Question

A few powerful vs. Many less powerful.

Same programming models, languages, optimizations and tools.

Diagrams for discussion purposes only, not a precise representation of any product of any company.
vision
span from few cores to many cores
with consistent models, languages, tools, and techniques
Optimizations for Intel® Xeon® and Intel® Xeon Phi™ products share the same:

- Languages
- Directives
- Libraries
- Tools
Picture worth many words

© 2013, James Reinders & Jim Jeffers, diagram used with permission
Intel® Xeon Phi™ Coprocessor
Intel® Xeon Phi™ Coprocessor

- NOT a GPU
- Many core x86 (a real general purpose (co-)processor!)
Intel® Xeon Phi™ Coprocessors

Up to 61 cores, 1.1 GHz, 244 threads.

Up to 16GB memory.

Up to 352 GB/s bandwidth.

Runs Linux OS.

Standard tools, models, languages.

1 TFLOP/s DP FP peak.

Better for parallelism than processor...
Up to 2.2X performance
Up to 4X more power efficient

© 2015, Intel Corporation. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi are trademarks of Intel Corporation in the U.S. and/or other countries. Other names and brands may be claimed as the property of others.
it is an SMP-on-a-chip running Linux
Based on an actual customer example. Shown to illustrate a point about common techniques. Your results may vary!

Fortran code using MPI, single threaded originally. Run on Intel® Xeon Phi™ coprocessor natively (no offload).
Illustrative example

- Untuned Performance on Intel® Xeon® processor
- Untuned Performance on Intel® Xeon Phi™ coprocessor
- TUNED Performance on Intel® Xeon Phi™ coprocessor

Yeah!
Illustrative example

Common optimization techniques... “dual benefit”
Illustrative example

Common optimization techniques... “dual benefit”
Intel® Xeon Phi™ Coprocessor
**LAMMPS**

**Liquid Crystal Benchmark; 524K Atoms**

---

**Application:** LAMMPS*


**Availability:**

**Usage Model:** Load balancer offloads part of neighbor-list and non-bond force calculations to Intel® Xeon Phi™ coprocessor for concurrent calculations with CPU.

**Highlights:** Improved results with Intel® Xeon® processor E5-2697 v3 and Intel® Xeon PHI™ coprocessor 7120A. Dynamic load balancing allows for concurrent:
- Data transfer between host and coprocessor.
- Calculations of neighbor-list, non-bond, bond, and long-range terms.

Same routines in LAMMPS Intel Package also run faster on CPU.

**Results:** Up to 5.4X performance improvement utilizing Intel® Xeon® processors and Intel® Xeon Phi™ coprocessors with application optimization on a single node compared to the baseline configuration. Performance gains continue to hold at 5.3x when scaling up to 32 nodes.

---

**LAMMPS* Liquid Crystal Benchmark Performance (Mixed Precision)**

<table>
<thead>
<tr>
<th>Comparative Performance</th>
<th>1 Node (524K Atoms)</th>
<th>32 Nodes (16.8M Atoms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2S Intel® Xeon® processor E5-2697v3 (LAMMPS Baseline)</td>
<td>4.5X</td>
<td>1</td>
</tr>
<tr>
<td>2S Intel® Xeon® processor E5-2697 v3 (LAMMPS IA Package)</td>
<td>5.4X</td>
<td>1</td>
</tr>
<tr>
<td>2S E5-2697 v3 + Intel® Xeon Phi™ coprocessor 7110P/7120A Turbo Off (LAMMPS IA Package)</td>
<td>3.66X</td>
<td>1</td>
</tr>
</tbody>
</table>

For configuration details, go [here](#).
**NAMD* 2.10 Pre-Release**

**STMV**

**Application:** NAMD 2.10 pre-release; STMV

**Description:** A parallel, object-oriented molecular dynamics code designed for high-performance simulation of large biomolecular systems. More at [http://www.ks.uiuc.edu/Research/namd/](http://www.ks.uiuc.edu/Research/namd/)

**Availability:**

- **Code:** Intel® Xeon Phi™ coprocessor support is available as pre-release at [http://www.ks.uiuc.edu/Development/Download/download.cgi?PackageName=NAMD](http://www.ks.uiuc.edu/Development/Download/download.cgi?PackageName=NAMD). Use the nightly build.


**Usage Model:** Single rank on host with 47 threads. Various computations are offloaded to Intel® Xeon Phi™ coprocessor from each thread.

**Highlights:** Intel® Xeon Phi™ coprocessor support is now in the development branch of NAMD 2.10 pre-release.

**Results:** For the STMV workload, the Intel® Xeon® processor E5-2697 v3 and the Intel® Xeon Phi™ coprocessor (32 nodes, 55 PPN) improved performance by up to 32X compared to the baseline processor (1 node, 47 PPN).

---

![NAMD* 2.10 (pre-release) Cluster Performance Increase STMV (~1M atoms)](image)

For configuration details, see [here](#).

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSpark 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. See benchmark tests and configurations in the speaker notes. For more information go to [http://www.intel.com/performance](http://www.intel.com/performance).

© 2015, Intel Corporation. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi are trademarks of Intel Corporation in the U.S. and/or other countries. *Other names and brands may be claimed as the property of others.*
Monte Carlo* RNG
European Options; Double Precision

**Application:** Monte Carlo* RNG European option pricing; double precision.


**Availability:**
- **Code:** [https://software.intel.com/protected-download/267270/517449](https://software.intel.com/protected-download/267270/517449)

**Usage Model:** Hybrid or Asynchronous offload on one shared memory host node with up to two coprocessor devices.

**Highlights:**
- One of two most computational intensive workloads in FSI benchmark suites.
- More operations per option data set.
- Benefits from Intel® AVX2 FMA.

**Results:**
The baseline Intel® Xeon® processor E5-2697 v2 is 1.39X higher performance than Intel® Xeon® processor E5-2697 v2.

Adding one coprocessor card nearly doubles the baseline performance adding two coprocessors nearly triples it.

For configuration details, go here.
Application: Binomial European Option Pricing.

Description: Binomial Options is a popular derivative pricing benchmark widely used by investment banks such as UBS*, Bank of America* and Goldman-Sachs*. One of two most computational intensive workloads in FSI benchmark suites.

Availability:
- Code: https://software.intel.com/sites/default/files/managed/ff/05/Binomialsrc.tar

Usage Model: Hybrid or Asynchronous offload on one shared memory host node with up to two coprocessor devices.

Highlights:
- More operations per option data set.
- Benefits from Intel® AVX2 FMA.

Results: The baseline Intel® Xeon® processor E5-2697 v3 is nearly double the performance of the Intel® Xeon® processor E5-2697 v2. Adding one coprocessor card increases the baseline performance by over 50%; adding two coprocessors more than doubles it.

For configuration details, go here.
Intel® Xeon Phi™ out scales Intel® Xeon® processors

Intel® Xeon Phi™ coprocessor vs. 2S Intel® Xeon® Processor

Higher is Better

Intel Measured Results: Different hardware architectures may require different source code. Results are based on Intel’s best efforts to use code optimized to run best on all architectures and perform the same work. Future code optimizations may result in different results.

© 2015, Intel Corporation. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi are trademarks of Intel Corporation in the U.S. and/or other countries. *Other names and brands may be claimed as the property of others.
Intel® Xeon Phi™ beats NVIDIA*
“Modernize” Code (for Parallelism)
What are the chances?

A Code (probably in FORTRAN) written for the CPU on the left perform well on those to the right?

photos not to same scale
Not Good, It Turns Out

Monte Carlo DP SS and VS

Vectorized and Single threaded

Scalar and Single threaded

2007
Intel Xeon Processor X5472 (formerly codenamed Harpertown)

2009
Intel Xeon Processor X5570 (formerly codenamed Nehalem)

2010
Intel Xeon Processor X5680 (formerly codenamed Westmere)

2012
Intel Xeon Processor 2600 family (formerly codenamed Sandy Bridge)

2013
Intel Xeon Processor 2600 v2 family (formerly codenamed Ivy Bridge)

Intel® and Xeon® are trademarks of Intel Corporation.

Configurations: as listed on Slide 78. Performance measured in Intel Labs by Intel employees. For more information go to http://www.intel.com/performance.
How much potential lies untapped today?

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. Configurations: as listed on Slide 78. Performance measured in Intel Labs by Intel employees. For more information go to http://www.intel.com/ performance.
Modernizing the Hydro CFD benchmark

Optimization approach delivered .5x Xeon in 18 months; modernization delivered >10x in 6 months, but required substantially different code.
Sequential code used 1 core, 1 SIMD lane

Not Good, It Turns Out
Using a single vector lane
Permission to use all lanes
Inspired by 61 Cores
“Inspired by 61 cores”

A key realization while preparing our recent book.

Over and over and over again...

“Inspired by 61 cores” was the biggest reason for people to work on scaling and vectorization... while benefiting processors and Intel® Xeon Phi™ coprocessors BOTH!
From ‘Correct’ to ‘Correct & Efficient’: a Hydro2D case study with Godunov’s scheme

Guillaume Colin de Verdière and Jason D. Sewall

- Real-world code developed in CEA
- Poetically notes, “a rising tide lifts all boats”, using “a common set of optimizations [that] benefit both general-purpose Intel® Xeon® processors and more specialized Intel® Xeon Phi™ accelerators”
  - 12x increase on Intel Xeon Phi coprocessors
  - 5x increase on Intel Xeon processors
The thinking process and techniques used in this chapter have wide applicability:

• Focus on data locality
• Then apply threading and vectorization techniques

Observed 3x performance improvement after:

• Re-structuring the code for better data locality
• Exploiting the available threads and SIMD lanes
Gilles Civario and Michael Lysaght

“The newly released OpenMP 4.0 standard, supporting an offload mode, can take advantage of emerging many-core coprocessors such as the Intel® Xeon Phi™ Coprocessor.”

The authors demonstrate how to take advantage of the OpenMP 4.0 standard on processors and coprocessors to portably and efficiently maximize an N-body kernel.

- The sample code can be used as a template applicable for countless codes.
- OpenMP 4.0 can unleash the full hardware potential in a very simple and straightforward manner.
The chapter authors discuss the hardware heterogeneity found in modern clusters along with the positioning an MPI task within the node.

- The performance through different communication pathways is highlighted using micro benchmarks.
- A hybrid Lattice Boltzmann Method application is configured with optimal MPI options
- Scaling performance is evaluated both with and without proxy-based communications.

“Using the Intel MPI provided proxy is critical to obtaining optimal communication performance across multiple coprocessors”
Our heterogeneous offload solution implemented for the NWChem CCSD(T) method shows that obtaining outstanding performance across a large-scale cluster is not only possible but also feasible by relying on well-known standard programming languages (Fortran) and parallel programming models (OpenMP).

“Our analysis methodology to determine offload candidates leads to a straightforward porting process.”

- Specifically: find hotspots, common anchor points for offloads, and perform loop analysis.
- Allows informed design decisions to be made about:
  - Where to place offload directives to determine the best location for data and offload transfers.
  - The structure of the compute kernels and their potential for vectorization and parallelization.
At the task level, ray tracing is trivially parallel …
- Rays associated with different pixels can be traced independently
- Unfortunately, efficiently exploiting the vector units of modern CPUs for ray tracing is much more challenging
- Stepping a ray through a hierarchical spatial data structure requires fine-grained data-dependent branching and irregular memory access patterns
  - Inhibits auto-vectorization
  - Worse, the optimal mapping of such a kernel is not always obvious, even for an experienced programmer
Image Rendering - Trend 1: Increasing Data Size

• increasingly complex phenomena, outgrowing peripheral memory
• moving big data over a WAN, security concerns, interactivity demands
• increased spatial / temporal resolution, more challenging to visually interpret

Trend 2: Increasing Shading Complexity

• Improved illumination effects (shadows) needed to aid in scene understanding
• Volumetric effects are also increasingly common (e.g., oil and gas applications)
• Difficult to compute efficiently in OpenGL but intrinsic to ray tracing / ray casting

Data: Florida International University
Increasing Image Quality

Increasing Data Size

Data Size < GPU Memory

Simple Shading

OpenGL*
(many existing apps in this space)

Data Size > GPU Memory

Advanced Shading

OpenGL or Ray Tracing

Ray Tracing

Ray Tracing

Render Method and Target Architecture
Performance: Embree vs. NVIDIA* OptiX*

Frames Per Second (Higher is Better)

- Headlight (0.8M Tris)
- Bentley (2.3M Tris)
- Crown (4.8M Tris)
- Dragon (7.4M Tris)
- Karst Fluid Flow (8.4M Tris)

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.

Source: Intel
At the task level, ray tracing is trivially parallel …
- Rays associated with different pixels can be traced independently
- Unfortunately, efficiently exploiting the vector units of modern CPUs for ray tracing is much more challenging
- Stepping a ray through a hierarchical spatial data structure requires fine-grained data-dependent branching and irregular memory access patterns
  - Inhibits auto-vectorization
  - Worse, the optimal mapping of such a kernel is not always obvious, even for an experienced programmer
For this reason they designed a usable and efficient data structure for vectorized sparse computations on multi-core architectures with vector processing capabilities - like Intel® Xeon Phi™ coprocessors.

This data structure helps with the difficulties in achieving a high performance for sparse matrix–vector (SpMV) multiplications caused by a low flop-to-byte ratio and inefficient cache use.

“Current hardware trends lead to an increasing width of vector units as well as to decreasing effective bandwidth-per-core. For sparse computations these two trends conflict.” [Emphasis added]
MORTON ORDER IMPROVES PERFORMANCE

Kerry Evans

“There are many facets to performance optimization but three issues to deal with right from the beginning are memory access, vectorization, and parallelization. Unless we can optimize these, we cannot achieve peak performance.”

Examines a method of mapping multidimensional data into a single dimension while maintaining locality using Morton, or Z-curve ordering. [Emphasis added]

- And look at the effects it has on performance of two common linear algebra problems: matrix transpose and matrix multiply
- Next, the transpose and multiply codes are tuned to take advantage of the processor and coprocessor cache, vector hardware, and threading

Morton order for 8 × 8 2D grid, subpartitioned into 4 × 4 blocks.
Because performance and programming matters

Optimizations for Intel® Xeon® and Intel® Xeon Phi™ products share the same:

- Languages
- Directives
- Libraries
- Tools
Hardware roadmap

Photo (c) 2014, James Reinders; used with permission; Yosemite Half Dome rising through forest fire smoke 11am on September 10, 2014

© 2015, Intel Corporation. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi are trademarks of Intel Corporation in the U.S. and/or other countries. *Other names and brands may be claimed as the property of others.
Intel® Xeon Phi™ Coprocessors

Up to 61 cores, 1.1 GHz, 244 threads.

Up to 16GB memory.

Up to 352 GB/s bandwidth.

Runs Linux OS.

Standard tools, models, languages.

1 TFLOP/s DP FP peak.

Better for parallelism than processor...
Up to 2.2X performance
Up to 4X more power efficient
# Parallel is the Path Forward

Intel® Xeon® and Intel® Xeon Phi™ Product Families are both going parallel

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Core(s)</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>12</td>
<td>18</td>
<td>61</td>
<td>≥ 61</td>
</tr>
<tr>
<td>Threads</td>
<td>2</td>
<td>2</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td>24</td>
<td>36</td>
<td>244</td>
<td>≥ 244</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>
<td>512</td>
<td>512</td>
</tr>
</tbody>
</table>

*Product specification for launched and shipped products available on [ark.intel.com](http://ark.intel.com).  
1. Not launched or in planning.

**More cores → More Threads → Wider vectors**

2008

45nm

Intel® Core™ Microarchitecture
First high-volume server Quad-Core CPUs

2009

Nehalem Microarchitecture
Up to 6 cores and 12MB Cache

2010

32nm

Sandy Bridge Microarchitecture
Up to 8 cores & 20MB Cache, AVX, AES-NI

2012

22nm

Haswell Microarchitecture
AVX2, FMA3, TSX

2014

14nm

TOCK

★ Major Developer SW Tools: Support new microarchitecture, available ~2Q before CPU

Early access to latest features w/ Intel Developer Tools
Parallel is the Path Forward
Intel® Xeon® and Intel® Xeon Phi™ Product Families are both going parallel

<table>
<thead>
<tr>
<th>Core(s)</th>
<th>Threads</th>
<th>SIMD Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>128</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>128</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>128</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>128</td>
</tr>
<tr>
<td>8</td>
<td>16</td>
<td>256</td>
</tr>
<tr>
<td>12</td>
<td>24</td>
<td>256</td>
</tr>
<tr>
<td>18</td>
<td>36</td>
<td>256</td>
</tr>
</tbody>
</table>

More cores → More Threads → Wider vectors

*Product specification for launched and shipped products available on ark.intel.com. 1. Not launched or in planning.
Next Intel® Xeon Phi™ Processor: Knights Landing

14nm process

3 TFlop/s standalone CPU or PCIe coprocessor

integrated on-package memory

option: integrated fabric

All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.
NERSC – 1st Announced
(~9,300 nodes)

“NERSC, Cray, Intel Partner on Next-gen Extreme-Scale Computing System... named Cori”

Scientific Computing – Apr 30, 2014

“... our goal is to enable performance that is portable across systems and will be sustained in future supercomputing architectures.”

Scientific Computing – Apr 30, 2014

“... users will be able to retain the MPI/OpenMP programming model...”

EurekAlert! – Apr 29, 2014

Trinity – 2nd Announced

“... multipetaflop supercomputer called Trinity”

PCWorld – Jul 10, 2014

“...powered by future Intel Xeon and Intel Xeon Phi processors, will deliver great application performance for a wide set of codes while the binary compatibility between the processors will allow the NNSA to reuse existing codes.”

insideHPC – Jul 10, 2014
Software Transition from KNC to KNL

Native or Symmetric or Offload

Recompile Only

MKL
OpenMP
AVX-512
KNL Enabled Compilers

MPI
Cilk Plus™
Cache Mode For
High Bandwidth Memory

TBB
OpenCL
KNL Enabled Performance Libraries & Runtimes

KNL Enabled Compilers

Using Intel® SW Development Tools

MOST benefits gained with a recompile

Incremental tuning benefits

Application & Execution Model Tuning
Memory Enhancements

MOST
Software Tools Overview
Intel® Parallel Studio XE

ACCELERATE

Improve application performance, scalability and reliability
Helping Software Developers Make High Performance Software

- Industry leading performance
- Advanced Compilers and Libraries
- Standards Support
- Linux*, Windows*, and OS X*

Parallel programming models

- Task and Data Parallel Performance
- Distributed Performance

Analysis

- Parallelism Assistant
- Thread and Memory Debug
- Performance Analysis
How Intel® Parallel Studio XE 2015 helps make *Faster Code Faster* for HPC

Foundational compilers, libraries and programming models...

Composer Edition

- Intel® C++ and Fortran compilers
- Parallel models (e.g., OpenMP*)
- Optimized libraries
How Intel® Parallel Studio XE 2015 helps make Faster Code Faster for HPC

Plus Analysis tools...

Professional Edition
- Threading design & prototyping
- Parallel performance tuning
- Memory & thread correctness

Composer Edition
- Intel® C++ and Fortran compilers
- Parallel models (e.g., OpenMP*)
- Optimized libraries
How Intel® Parallel Studio XE 2015 helps make **Faster Code Faster** for HPC
Short list: TRY THESE FIRST

**Higher Performance**
1. Recompile with Intel Compilers
2. Use Intel® Math Kernel Library (MKL)
3. Scale...
   - Use OpenMP directives
   - (C++ developers: use Intel® Threading Building Blocks – TBB)
4. Vectorization:
   - New optimization reports
   - New “OMP SIMD”

**Help with Optimization and Debug**
1. Intel® Advisor –
   - Scaling Analysis
   - Vectorization Advice
2. Intel® Inspector
   - Test for Threading Errors
   - Test for Memory Leaks
3. Intel® VTune™ Amplifier
   - Find Hotspots
4. Intel® Trace Analyze and Collector
   - MPI tuning
PART 2 - 90 minutes

Tools intro (roadmap) – do in Part 1?

Things to try out the next few weeks?

Call to Action Summary:
• Scale – OpenMP/TBB
• Vectorize – pragma SIMD

OpenMP/TBB/MPI standards
• How to choose, 3 hybrid options (MPI+OpenMP/TBB, CAF, MPI+MPI SHM)

Compiler (hand out optimization cards)
• Optimization reports
• Switches

Performance Tuning
• VTune
• MPI

Vectorization
• AVX, AVX2, AVX-512
• Design Advice (Advisor w/vector assistant)

Error Analysis/Debug (Inspector w/Mem, thread checking)

Developer Resources
Repeat Call to Action Summary
Boost C++ application performance on Windows* & Linux* using Intel® C++ Compiler
(higher is better)

<table>
<thead>
<tr>
<th>Floating Point</th>
<th>Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Linux</td>
</tr>
<tr>
<td>Visual C++ 2013</td>
<td>Intel C++ 15.0</td>
</tr>
<tr>
<td>1.23</td>
<td>1.24</td>
</tr>
</tbody>
</table>

Boost Fortran application performance on Windows* & Linux* using Intel® Fortran Compiler
(lower is better)

<table>
<thead>
<tr>
<th>Fortran</th>
<th>Windows</th>
<th>Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Fortran</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.54</td>
<td>Intel Fortran 13.0</td>
<td>PGI Fortran 14.7</td>
</tr>
</tbody>
</table>

Relative geometric mean performance, SPEC* benchmark - higher is better

Boost application performance on Windows* and Linux*

Intel® C++ and Fortran Compilers

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 #20110B04.
GDB Debugger now standard

Intel provides enhanced GDB as standard debug solution with Intel® Parallel Studio XE

- Increase C++ and now Fortran application reliability

Faster debug cycles for hybrid programming through full simultaneous debug support across host and Intel® Xeon Phi™ coprocessor targets – on Linux* and Windows* host

Intel® MPX and Intel® AVX-512 support for more robust and faster applications

Fast bug fixing through Intel® Processor Trace support

Fast and efficient analysis and debugging of past program execution
New Optimization Reports in Intel® Compilers 15.0

Photo (c) 2014, James Reinders; used with permission; Yosemite Half Dome rising through forest fire smoke 11am on September 10, 2014
Easier to Use Optimization Reports
Intel Compilers

```c
int size();
void foo(double *restrict a, double *b){
  int i;
  for (i=0;i<size();i++){
    a[i] += b[i];
  }
}
```

```bash
icpc -c -O3 -restrict -opt-report x.cpp
```

**14.0 compiler:**
x.cpp(6) (col. 15) remark: loop was not vectorized: unsupported loop structure

**15.0 compiler:**
LOOP BEGIN at x.cpp(6,15)
  remark #15523: loop was not vectorized: cannot compute loop iteration count before executing the loop.
  LOOP END
“We heard your feedback”

Improve the user experience

• Make the report easier to read and understand
  • A single, unified report
  • Loop-based reporting
• Focus on user actionable information
• Make it easy to select the desired information
• Expand the range of output modes

Make the report’s information accessible

• In a text file
• In the assembly listing
• Through the Microsoft* Visual Studio* IDE
• Through other Intel software tools
Introduced in Intel® Compiler v15.0 for C, C++ and Fortran
• for Windows*, Linux* and OS X*

Main options:
/Qopt-report:N (Windows), -qopt-report=N (Linux and OS X)
    N = 1-5 for increasing levels of detail, (default N=2)
/Qopt-report-phase:str[,str1,...] -qopt-report-phase= [,str1,...]
    str = loop, par, vec, openmp, ipo, pgo, cg, offload, tcollect, all
/Qopt-report-file: [stdout | stderr | filename]
- qopt-report-file = [stdout | stderr | filename]
Report Output

Output goes to a text file by default, no longer stderr

- File extension is .optrpt, root is same as object file
- One report file per object file, in object directory
- Created from scratch or overwritten (no appending)

/Qopt-report-file:stderr gives old behavior (to stderr)
-qopt-report-file=stderr

: or =filename to change default file name

/Qopt-report-format:vs format for Visual Studio* IDE

For debug builds, (-g on Linux* or OS X*, /Zi on Windows*), assembly code and object files contain loop optimization info

• /Qopt-report-embed to enable this for non-debug builds
Filtering Report Output

The optimization report can be large

Filtering can restrict the content to the most performance-critical parts of an application

[-q | /Q]opt-report-routine[: | =]<function1>[,<function2>,...]  
“function1” can be a substring of function name  
or a regular expression

   can also restrict to a particular range of line numbers, e.g.:
icl /Qopt-report-filter="test.cpp,100-300" test.cpp
ifort –qopt-report-filter="test.f90,100-300" test.f90

Also select the optimization phase(s) of interest with option  
-qopt-report-phase or /Qopt-report-phase
Loop, Vectorization and Parallelization Phases

Hierarchical display of loop nest

- Easier to read and understand
- For loops for which the compiler generates multiple versions, each version gets its own set of messages

Where code has been inlined, caller/callee info available

The “loop” phase (formerly hlo) includes messages about memory and cache optimizations, such as blocking, unrolling and prefetching

- Now integrated with vectorization & parallelization reports
Hierarchically Presented
Loop Optimization Report

double a[1000][1000], b[1000][1000], c[1000][1000];

void foo() {
    int i, j, k;
    for (i=0; i<1000; i++) {
        for (j=0; j< 1000; j++) {
            c[j][i] = 0.0;
            for (k=0; k<1000; k++) {
                c[j][i] = c[j][i] + a[k][i] * b[j][k];
            }
        }
    }
}

Copyright © 2015, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.*
By comparison: 14.0 vs 15.0 reports

HPO VECTORIZER REPORT (foo) LOG OPENED ON Mon Feb 24 12:10:15 2014

HLO REPORT LOG OPENED ON Mon Feb 24 12:10:15 2014

Ordered Reporting of Transformations
Multi-versioned loops: Peel loop, remainder loop and kernel

LOOP BEGIN at ggFineSpectrum.cc(124,5) inlined into ggFineSpectrum.cc(56,7)
remark #15018: loop was not vectorized: not inner loop

LOOP BEGIN at ggFineSpectrum.cc(138,5) inlined into ggFineSpectrum.cc(60,15)

**Peeled**
remark #25460: Loop was not optimized
LOOP END

LOOP BEGIN at ggFineSpectrum.cc(138,5) inlined into ggFineSpectrum.cc(60,15)
remark #15145: vectorization support: unroll factor set to 4
remark #15002: LOOP WAS VECTORIZED
LOOP END

LOOP BEGIN at ggFineSpectrum.cc(138,5) inlined into ggFineSpectrum.cc(60,15)

**Remainder**
remark #15003: REMAINDER LOOP WAS VECTORIZED
LOOP END
LOOP END
Annotated Assembly Listings

.L11: # optimization report
# LOOP WAS INTERCHANGED
# loop was not vectorized: not inner loop
xorl %edi, %edi #38.3
movsd b.279.0.2(%rax,%rsi,8), %xmm0 #41.32
unpcklpd %xmm0, %xmm0 #41.32
# LOE rax rcx rbx rsi rdi r12 r13 r14 r15 edx xmm0
..B1.11: # Preds ..B1.11 ..B1.10
..L12: # optimization report
# LOOP WAS INTERCHANGED
# LOOP WAS VECTORIZED
# VECTORIZATION HAS UNALIGNED MEMORY REFERENCES
# VECTORIZATION SPEEDUP COEFFICIENT 2.250000
movaps a.279.0.2(%rcx,%rdi,8), %xmm1 #41.22
movaps 16+a.279.0.2(%rcx,%rdi,8), %xmm2 #41.22
movaps 32+a.279.0.2(%rcx,%rdi,8), %xmm3 #41.22
movaps 48+a.279.0.2(%rcx,%rdi,8), %xmm4 #41.22
mulpd %xmm0, %xmm1 #41.32
mulpd %xmm0, %xmm2 #41.32

L4:: ; optimization report
; PEELED LOOP FOR VECTORIZATION
$LN36:
$LN37:
  vaddss xmm1, xmm0, DWORD PTR [r8+r10*4] ;4.5
snip snip snip

L5:: ; optimization report
; LOOP WAS VECTORIZED
; VECTORIZATION HAS UNALIGNED MEMORY REFERENCES
; VECTORIZATION SPEEDUP COEFFICIENT 8.398438
$LN46:
  vaddps ymm1, ymm0, YMMWORD PTR [r8+r9*4] ;4.5
snip snip snip

L6:: ; optimization report
; LOOP WAS VECTORIZED
; REMAINDER LOOP FOR VECTORIZATION
; VECTORIZATION HAS UNALIGNED MEMORY REFERENCES
; VECTORIZATION SPEEDUP COEFFICIENT 2.449219
$LN78:
  add r10, 4 ;3.3
Vectorization – report levels

-qopt-report-phase:vec  -qopt-reportN
/Qopt-report-phase:vec   /Qopt-report:N

N specifies the level of detail; default N=2 if N omitted

Level 0: No vectorization report
Level 1: Reports when vectorization has occurred.
Level 2: Adds diagnostics why vectorization did not occur.
Level 3: Adds vectorization loop summary diagnostics.
Level 4: Additional detail, e.g. on data alignment
Level 5: Adds detailed data dependency information
Actionable Messages Example (1)

```
$ icc -c -opt-report=4 -opt-report-phase=loop,vec -opt-report-file=stderr foo.c

Begin optimization report for: foo

Report from: Loop nest & Vector optimizations [loop, vec]

LOOP BEGIN at foo.c(4,3)

**Multiversioned v1**

remark #25231: Loop multiversioned for Data Dependence
remark #15135: vectorization support: reference theta has unaligned access
remark #15135: vectorization support: reference sth has unaligned access
remark #15127: vectorization support: unaligned access used inside loop body
remark #15145: vectorization support: unroll factor set to 2
remark #15164: vectorization support: number of FP up converts: single to double precision 1
remark #15165: vectorization support: number of FP down converts: double to single precision 1
remark #15002: **LOOP WAS VECTORIZED**
remark #36066: unmasked unaligned unit stride loads: 1
remark #36067: unmasked unaligned unit stride stores: 1
... (loop cost summary) ....
remark #25018: Estimate of max trip count of loop=32

```
More Information on Optimization Reports

• Video recorded Webinar “Getting the most out of your compiler with the new Optimization Reports

• Slides for “Getting the most out of your compiler with the new Optimization Reports”
Updated standards support in Intel Compilers 15.0 and MPI
OpenMP* 4.0 Support

• Everything in 4.0 now supported except for user-defined reductions
• CANCEL directive
  ▪ Requests cancellation of the innermost enclosing region
• CANCELLATION POINT directive
  ▪ Defines a point at which implicit or explicit tasks check to see if cancellation has been requested
• DEPEND clause on TASK directive
  ▪ Enforces additional constraints on the scheduling of a task by enabling dependences between sibling tasks in the task region.
• Combined constructs (TEAMS DISTRIBUTE, etc.)
New OpenMP* Support

- Fortran WORKSHARE can go parallel (sometimes)
  - Simple array assignments such as \( A = B + C \) parallelize.
  - Simple array assignments with overlap such as \( A = A + B + C \) parallelize.
  - Array assignments with user-defined function calls parallelize such as \( A = A + F(B) \). \( F \) must be ELEMENTAL.
  - Array assignments with array slices on the right hand side of the assignment such as \( A = A + B(1:4) + C(1:4) \) parallelize. If the lower bound of the left hand side or the array slice lower bound or the array slice stride on the right hand side is not 1, then the statement does not parallelize.
  - Assigning into array slices does not parallelize.
  - Scalar assignments do not parallelize – there is no work that needs to be done in parallel.
  - FORALL and WHERE constructs do not parallelize.
-ansi-alias now default on Linux* C++

-ansi-alias enabled by default at -O2 or -O3 on Linux* (gcc* enables -fstrict-aliasing at -O2 and -O3)

-ansi-alias asserts that code follows ANSI aliasing rules, allowing the compiler to make more aggressive optimizations

This can improve performance if this is true

If not true, this can result in bad behavior

Alias checking at compile time may catch this

But it may not resulting in bad runtime results

If in doubt, use -no-ansi-alias to disable

with gcc, -fno-strict-aliasing
Intel® C++ version 15.0 completes all C++11 standard language features!

- virtual overrides
- inheriting constructors, i.e.:
  ```
  struct Derived { using Base::Base; }
  ```
- deprecation of exception specifications
- user defined literals
- thread_local (C++11 semantics) (Linux only)

C++11 library features are dependent on the support provided by the standard C++ library on the platform:

- Windows*: msvcrt/libcm, Linux*: libstdc++, OS X*: libc++/libstdc++
GNU* Compatibility

• To enable c++11 support you need to use 
  –std=c++11 (or –std=c++0x) option

• We currently support all c++11 features used in the GNU 4.8 versions of the headers enabled when you use the option

• Depending upon the GNU on your system (i.e. g++ in your PATH) you may get different features enabled
  • Support of C++11 features requires support from C++ header files included with GNU C/C++ installation – these features vary by version.
    • Toolchain for Intel® Many Integrated Core Architecture specifically has issues. See release notes for details.
  • Recommend use of GNU 4.8 or newer packages
Microsoft* compatibility/Microsoft* standard library

Microsoft* implementation of C++11 features in latest Microsoft Visual Studio* version 2013

C++11 support in VS2012 and older versions depends on the features Microsoft included in their C++ header files

No special command line switch or feature macro used in standard lib headers to access C++11 functionality

**Intel® C++ compiler is compatible by default** (i.e. whatever C++11 features are provided by the reference Microsoft compiler are available with Intel Windows* compiler)

To get additional C++11 functionality with our compiler use Intel-specific /Qstd=c++0x or /Qstd=c++11 switch

Intel C++ Composer XE 2015 is fully compatible with the Microsoft Visual C++ 2013 with respect to C++11 functionality
Fortran Standard Features

- Full language Fortran 2003 support
  - Parameterized Derived Types
  - Expanded support of intrinsics in specification and constant expressions
- BLOCK from Fortran 2008
F03: Parameterized Derived Types

- Allows programmer to create a template for a type that can have KIND and length parameters deferred

- KIND type parameters are compile-time constants, length parameters can be run-time. Example:

```fortran
TYPE humongous_matrix(k, d)
  INTEGER, KIND :: k = kind(0.0)
  INTEGER(selected_int_kind(12)), LEN :: d
  REAL(k) :: element(d,d)
END TYPE
giant = TYPE(humongous_matrix(8,10000000))
```

F08: BLOCK Construct

- An executable construct that may contain declarations
- Variables declared within the construct are local to that scope
- No COMMON, EQUIVALENCE, NAMELIST, IMPLICIT
- SAVE allowed, local to that construct
- SAVE in outer scope does not affect BLOCK
- Labels and formats are not local
- Useful with DO CONCURRENT for threadlocals
F08: DO CONCURRENT with BLOCK

DO CONCURRENT (I = 1:N)
    BLOCK
        REAL T
        T = A(I) + B(I)
        C(I) = T + SQRT(T)
    END BLOCK
END DO

Without BLOCK, no way to create an iteration-local (threadprivate) temporary variable
Cluster Level Programming

More choices exist

MPI ranks on every core (or hardware thread)

*This usually runs into trouble as nodes have high core counts.*

"Hybrid" programming offers other options.

MPI + OpenMP*

MPI + TBB (*preferred for C++*)

MPI + MPI Shared Memory (MPI SHM: new in MPI 3.0)

Coarray Fortran ("PGAS" inside Fortran 2008 standard)
Cool new options in Intel Compilers 15.0
New Compiler Options

- `/Qinit:keyword` or `-init keyword` ([no]arrays, [no]snan, [no]zero)
  - Initializes **static local variables** to signaling NaN or zero (replaces `/Qzero`)
  - Default is to initialize scalars only
  - `/Qsave` also needed, typically
  - Only REAL and COMPLEX can be sNAN initialized
  - No EQUIVALENCE, no derived types, no automatic or allocatable variables
  - Sets `-fpe0` – warning if `-fpe3` explicitly specified
New Compiler Options

- `/Qopt-dynamic-align[-] (-q[no-]opt-dynamic-align)
  - Allows user to turn off conditional code paths based on alignment, for better run-to-run reproducibility without larger performance impact of other options

- `/Qprof-gen:[no]threadsafe
  - Accurately collect block counts while multiple threads are updating the same counter.

- `-f[no-]fat-lto-objects (Linux only)
  - Determines whether a fat link-time optimization (LTO) object, containing both intermediate language and object code, is generated during –ipo compile

- `-xcore-avx512 (in 15.0 Update 1, fall 2014)
Other compiler changes

• DWARF3 is now the default debug format
• -stdlib=libc++ default on OS X*
• Specific GNU standard C headers provided with compiler
• -fast and –Ofast now implies /fp:fast=2 or –fp-model fast=2
• __intel_simd_lane() intrinsic for SIMD enabled functions
• gcc-compatible function multiversioning
• Microsoft vectorcall calling convention supported
• New aligned_new header for C++11 type alignment
• /Qcheck-pointers-narrowing-, -no-check-pointers-narrowing to relax Pointer Checker analysis of struct fields
• Improved debugging information for C++11 lambda functions
• #pragma offload now permits non-contiguous data
• _Simd, _Safelen, and _Reduction keywords for explicit vectorization
• /Qno-builtint-<name> option added for Windows* to disable intrinsic functions by name
Other compiler changes

- Arithmetic or logical operators usable for SIMD data types (like __m128)
- -fno-fat-lto-objects to separate IPO-generated IL from object
- inline-max-per-routine, inline-max-total-size pragmas to control inlining per function
- INTEL_PROF_DYN_PREFIX environment variable to specify custom .dyn filename prefix for PGO profiles
- /Qprof-gen: threadsafe or -prof-gen=threadsafe to provide threadsafe PGO instrumentation
- Inlining limit diagnostic remarks added
Explicit Vectorization: A “take control” alternative to implicit vectorization (code & hope?)

Auto-vectorization solved years ago, still improving, and never enough? Make room for explicit vectorization
Parallel first

Vectorize second
What is a Vector?
Vector of numbers

\[
\begin{bmatrix}
4.4 & 1.1 & 3.1 & -8.5 & -1.3 & 1.7 & 7.5 & 5.6 & -3.2 & 3.6 & 4.8
\end{bmatrix}
\]
Vector addition

\[
\begin{bmatrix}
4.4 & 1.1 & 3.1 & -8.5 & -1.3 & 1.7 & 7.5 & 5.6 & -3.2 & 3.6 & 4.8 \\
-0.3 & -0.5 & 0.5 & 0 & 0.1 & 0.8 & 0.9 & 0.7 & 1 & 0.6 & -0.5 \\
\end{bmatrix}
\]

\[+
\]

\[
\begin{bmatrix}
4.1 & 0.6 & 3.6 & -8.5 & -1.2 & 2.5 & 8.4 & 6.3 & -2.2 & 4.2 & 4.3 \\
\end{bmatrix}
\]
...and Vector multiplication

<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>-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>

<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>-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>-1.32</td>
<td>-0.55</td>
<td>1.55</td>
<td>0</td>
<td>-0.13</td>
<td>1.36</td>
<td>6.75</td>
<td>3.92</td>
<td>-3.2</td>
<td>2.16</td>
<td>-2.4</td>
</tr>
</tbody>
</table>
An example
vector data operations:
data operations done in parallel

```c
void v_add (float *c,
            float *a,
            float *b)
{
    for (int i=0; i<= MAX; i++)
        c[i]=a[i]+b[i];
}
```
vector data operations: data operations done in parallel

```c
void v_add (float *c, float *a, float *b)
{
    for (int i=0; i<= MAX; i++)
        c[i]=a[i]+b[i];
}
```

**Loop:**
1. LOAD a[i] -> Ra
2. LOAD b[i] -> Rb
3. ADD Ra, Rb -> Rc
4. STORE Rc -> c[i]
5. ADD i + 1 -> i
vector data operations: data operations done in parallel

void v_add (float *c, float *a, float *b)

Loop:
1. LOADv4 a[i:i+3] -> Rva
2. LOADv4 b[i:i+3] -> Rvb
3. ADDv4 Rva, Rvb -> Rvc
4. STOREv4 Rvc -> c[i:i+3]
5. ADD i + 4 -> i

Loop:
1. LOAD a[i] -> Ra
2. LOAD b[i] -> Rb
3. ADD Ra, Rb -> Rc
4. STORE Rc -> c[i]
5. ADD i + 1 -> i
vector data operations:

data operations done in parallel

We call this “vectorization”

void v_add (float *c, float *a, float *b)
{
    for (int i = 0; i <= MAX; i++)
    {
        c[i] = a[i] + b[i];
    }
}

Loop: 1. LOAD a[i] -> Ra
2. LOAD b[i] -> Rb
3. ADD Ra, Rb -> Rc
4. STORE Rc -> c[i]
5. ADD i + 1 -> i

Loop: 1. LOADv4 a[i:i+3] -> Rva
2. LOADv4 b[i:i+3] -> Rvb
3. ADDv4 Rva, Rvb -> Rvc
4. STOREv4 Rvc -> c[i:i+3]
5. ADD i + 4 -> i
vector data operations: data operations done in parallel

```c
void v_add (float *c, float *a, float *b)
{
    for (int i=0; i<= MAX; i++)
        c[i]=a[i]+b[i];
}
```
vector data operations: data operations done in parallel

```c
void v_add (float *c, float *a, float *b) {
    for (int i=0; i<= MAX; i++)
        c[i]=a[i]+b[i];
}
```

**PROBLEM:**
This LOOP is NOT LEGAL to (automatically) VECTORIZE in C / C++ (without more information).

Arrays *not* really in the language
Pointers are, evil pointers!
Choice 1: use a compiler switch for auto-vectorization (and hope it vectorizes)
Choice 2: give your compiler hints (and *hope* it vectorizes)
C99 `restrict` keyword

```c
void v_add (float *restrict c,
            float *restrict a,
            float *restrict b)
{
    for (int i=0; i<= MAX; i++)
        c[i]=a[i]+b[i];
}
```
**IVDEP** (ignore assumed vector dependencies)

```c
void v_add (float *c,
             float *a,
             float *b)
{
    #pragma ivdep
    for (int i=0; i<= MAX; i++)
        c[i]=a[i]+b[i];
}
```
Choice 3: code explicitly for vectors

(mandatory vectorization)
void v_add (float *c, 
    float *a, 
    float *b)
{
    #pragma omp simd 
    for (int i=0; i<= MAX; i++)
        c[i]=a[i]+b[i];
}
OpenMP* 4.0: #pragma omp declare simd

```c
#pragma omp declare simd
void v1_add (float *c,
        float *a,
        float *b)
{
    *c=*a+*b;
}
```
void v_add (float *c, float *a, float *b)
{
    __m128* pSrc1 = (__m128*) a;
    __m128* pSrc2 = (__m128*) b;
    __m128* pDest = (__m128*) c;
    for (int i=0; i<= MAX/4; i++)
        *pDest++ = _mm_add_ps(*pSrc1++, *pSrc2++);
}
array operations (Cilk™ Plus)

```c
void v_add (float *c, float *a, float *b)
{
    c[0:MAX]=a[0:MAX]+b[0:MAX];
}
```

Challenge: long vector slices can cause cache issues; fix is to keep vector slices short.

Cilk™ Plus is supported in Intel compilers, and gcc (4.9).
vectorization solutions

1. auto-vectorization (use a compiler switch and hope it vectorizes)
   - sequential languages and practices get in the way

2. give your compiler hints and hope it vectorizes
   - C99 restrict (implied in FORTRAN since 1956)
   - #pragma ivdep

3. code explicitly
   - OpenMP 4.0 #pragma omp simd
   - Cilk™ Plus array notations
   - SIMD instruction intrinsics
   - Kernels: OpenMP 4.0 #pragma omp declare simd; OpenCL; CUDA kernel functions
vectorization solutions

1. auto-vectorization (use a compiler switch and hope it vectorizes)
   - sequential languages and practices get in the way

2. give your compiler hints and hope it vectorizes
   - C99 restrict (implied in FORTRAN since 1956)
   - #pragma ivdep

3. code explicitly
   - OpenMP 4.0 #pragma omp simd
   - Cilk™ Plus array notations
   - SIMD instruction intrinsics
   - Kernels: OpenMP 4.0 #pragma omp declare simd; OpenCL; CUDA kernel functions

---

Best at being
Reliable, predictable and portable
OpenMP 4.0

Based on a proposal from Intel based on customer success with the Intel® Cilk™ Plus features in Intel compilers.

```
#pragma omp simd reduction(+:val) reduction(+:val2)
for(int pos = 0; pos < RAND_N; pos++) {
    float callValue =
        expectedCall(Sval,Xval,MuByT,VBySqrtT,l_Random[pos]);
    val += callValue;
    val2 += callValue * callValue;
}
```
**simd construct**  
(OpenMP 4.0)

**Summary**

The `simd` construct can be applied to a loop to indicate that the loop can be transformed into a SIMD loop (that is, multiple iterations of the loop can be executed concurrently using SIMD instructions).

```c
#pragma omp simd [clause[,] clause] ... new-line
for-loops
```

where `clause` is one of the following:

- `safelen(length)`
- `linear(list[:linear-step])`
- `aligned(list[:alignment])`
- `private(list)`
- `lastprivate(list)`
- `reduction(reduction-identifier:list)`
- `collapse(n)`

The `simd` directive places restrictions on the structure of the associated `for-loops`. Specifically, all associated `for-loops` must have **canonical loop form** (Section 2.6 on page 51).

**Note:** per the OpenMP standard, the “for-loop” must have canonical loop form.

**C/C++**

```c
!$omp simd [clause[,] clause ...]
do-loops
!$omp end simd/
```

**Fortran**

where `clause` is one of the following:

- `safelen(length)`
- `linear(list[:linear-step])`
- `aligned(list[:alignment])`
- `private(list)`
- `lastprivate(list)`
- `reduction(reduction-identifier:list)`
- `collapse(n)`

If an `end simd` directive is not specified, an `end simd` directive is assumed at the end of the `do-loops`.

All associated `do-loops` must be **do-constructs** as defined by the Fortran standard. If an `end simd` directive follows a **do-construct** in which several loop statements share a DO termination statement, then the directive can only be specified for the outermost of these DO statements.
**Summary**

The `declare simd` construct can be applied to a function (C, C++ and Fortran) or a subroutine (Fortran) to enable the creation of one or more versions that can process multiple arguments using SIMD instructions from a single invocation from a SIMD loop. The `declare simd` directive is a declarative directive. There may be multiple `declare simd` directives for a function (C, C++, Fortran) or subroutine (Fortran).

**C/C++**

```c
#pragma omp declare simd[clause[[,] clause] ...] new-line
```

**Fortran**

```fortran
!omp declare simd(proc-name) [clause[[,] clause] ...]
```

where `clause` is one of the following:

- `simdlen(length)`
- `linear(argument-list[,constant-linear-step])`
- `aligned(argument-list[,alignment])`
- `uniform(argument-list)`
- `inbranch`
- `notinbranch`
Loop SIMD construct
(OpenMP 4.0)

Summary
The loop SIMD construct specifies a loop that can be executed concurrently using SIMD
instructions and that those iterations will also be executed in parallel by threads in the
team.

Syntax

C/C++

```c
#pragma omp for simd [clause[[],] clause] ...) new-line
for-loops
```

Fortran

```fortran
!$omp do simd [clause[[],] clause] ...) do-loops
[$omp end do simd [nowait]]
```

where `clause` can be any of the clauses accepted by the `for` or `simd` directives with
identical meanings and restrictions.

If an `end do simd` directive is not specified, an `end do simd` directive is assumed at the end of the do-loop.
You like directives?

- Yes → Use OpenMP 4.0
- No → You are not alone.
for your consideration:

Intel 15.0 Compilers support **keywords** as an alternative

- Keyword versions of SIMD pragmas added: 
  - _Simd, _Safelen, _Reduction
- __intel_simd_lane() intrinsic for SIMD enabled functions

Keywords / library interfaces being discussed for SIMD constructs in C and C++ standards
### Intel Instruction Set Vector Extensions from 1997-2008

<table>
<thead>
<tr>
<th>Year</th>
<th>Technology</th>
<th>Instructions</th>
<th>Bits</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>1997</td>
<td>MMX™ technology</td>
<td>57 new instructions</td>
<td>64 bits</td>
<td>Overload FP stack, Integer only media extensions</td>
</tr>
<tr>
<td>1998</td>
<td>Intel® SSE</td>
<td>70 new instructions</td>
<td>128 bits</td>
<td>4 single-precision vector FP, scalar FP instructions, cacheability instructions, control &amp; conversion instructions, media extensions</td>
</tr>
<tr>
<td>1999</td>
<td>Intel® SSE2</td>
<td>144 new instructions</td>
<td>128 bits</td>
<td>2 double-precision vector FP, 8/16/32/64 vector integer, 128-bit integer memory &amp; power management</td>
</tr>
<tr>
<td>2004</td>
<td>Intel® SSE3</td>
<td>13 new instructions</td>
<td>128 bits</td>
<td>FP vector calculation, x87 integer conversion, 128-bit integer unaligned load, thread sync.</td>
</tr>
<tr>
<td>2006</td>
<td>Intel® SSSE3</td>
<td>32 new instructions</td>
<td>128 bits</td>
<td>Enhanced packed integer calculation</td>
</tr>
<tr>
<td>2007</td>
<td>Intel® SSE4.1</td>
<td>47 new instructions</td>
<td>128 bits</td>
<td>Packed integer calculation &amp; conversion, better vectorization by compiler, load with streaming hint</td>
</tr>
<tr>
<td>2008</td>
<td>Intel® SSE4.2</td>
<td>7 new instructions</td>
<td>128 bits</td>
<td>String (XML) processing, POP-Count, CRC32</td>
</tr>
</tbody>
</table>
# Intel Instruction Set Vector Extensions since 2011

<table>
<thead>
<tr>
<th>Year</th>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2011</td>
<td>Intel® AVX</td>
<td>Co-processor only 512 bits</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Promotion of 128 bit FP vector instructions to 256 bit</td>
</tr>
<tr>
<td>2011</td>
<td></td>
<td>Hundreds of new 512 bit vector instructions only available for MIC architecture - not supported by and not compatible to x86 architecture</td>
</tr>
<tr>
<td>2012</td>
<td>“AVX-1.5”</td>
<td>7 new instructions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16 bit FP support</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RDRAND</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>2013</td>
<td>Intel® AVX-2</td>
<td>Promotion of integer instruction to 256 bit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- FMA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Gather</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- TSX/RTM</td>
</tr>
<tr>
<td>TBD</td>
<td>Intel® AVX-512</td>
<td>Promotion of vector instructions to 512 bits</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Much more</td>
</tr>
</tbody>
</table>
Growth is in vector instructions

Disclaimer: Counting/attributing instructions is in inexact science. The exact numbers are easily debated, the trend is quite real regardless.
Motivation for AVX-512 Conflict Detection
Sparse computations are common in HPC, but hard to vectorize due to race conditions

Consider the “histogram” problem:

```cpp
for(i=0; i<16; i++) { A[B[i]]++; }
```

- Code above is wrong if any values within B[i] are duplicated
  - Only one update from the repeated index would be registered!
- A solution to the problem would be to avoid executing the sequence 
gather-op-scatter with vector of indexes that contain conflicts
Motivation for AVX-512 Conflict Detection

Sparse computations are common in HPC, but hard to vectorize due to race conditions

Consider the “histogram” problem:

```
for(i=0; i<16; i++) { A[B[i]]++; }
```

- Code above is wrong if any values within B[i] are duplicated
  - Only one update from the repeated index would be registered!
- A solution to the problem would be to avoid executing the sequence gather-op-scatter with vector of indexes that contain conflicts
Conflict Detection Instructions in AVX-512
improve vectorization!

VPCONFLECT instruction detects elements with previous conflicts in a vector of indexes

- Allows to generate a mask with a subset of elements that are guaranteed to be conflict free
- The computation loop can be re-executed with the remaining elements until all the indexes have been operated upon

```c
index = vload u8[B[i] // Load 16 B[i]
pending_elem = 0xFFFF; // all still remaining
do {
    curr_elem = get_conflict_free_subset(index, pending_elem)
    old_val = vgather {curr_elem} A, index // Grab A[B[i]]
    new_val = vadd old_val, +1.0 // Compute new values
    vscatter A {curr_elem}, index, new_val // Update A[B[i]]
    pending_elem = pending_elem ^ curr_elem // remove done idx
} while (pending_elem)
```

for illustration: this not even the fastest version

<table>
<thead>
<tr>
<th>CDI instr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>VPCONFLECT(D,Q) zmm1(k1), zmm2/mem</td>
</tr>
<tr>
<td>VPBROADCAST(W2D,B2Q) zmm1, k2</td>
</tr>
<tr>
<td>VPTESTNM(D,Q) k2(k1), zmm2, zmm3/mem</td>
</tr>
<tr>
<td>VPLZCNT(D,Q) zmm1 (k1), zmm2/mem</td>
</tr>
</tbody>
</table>

© 2015, Intel Corporation. All rights reserved. Intel, the Intel logo, Intel Inside, Intel Xeon, and Intel Xeon Phi are trademarks of Intel Corporation in the U.S. and/or other countries. *Other names and brands may be claimed as the property of others.
Vectorization Advisor in Intel® Advisor XE
Vectorization Advisor

NEW in Intel® Advisor XE this year (beta spring 2015)

High ROI for Vectorization Effort

- What vectorization will pay off the most?
- What is blocking vectorization?
- What is the benefit to reorganization?

Vectorize Your Code

Faster with More Confidence

```
559       for (Index_type k=0 ; k<len ; k++) |
560         x[k] = y[k+1] - y[k];
```

© 2015, Intel Corporation. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi are trademarks of Intel Corporation in the U.S. and/or other countries. *Other names and brands may be claimed as the property of others.
Libraries
Intel® Threading Building Blocks (TBB)

What
- Widely used C++ template library for task parallelism.
- Features
  - Parallel algorithms and data structures.
  - Threads and synchronization primitives.
  - Scalable memory allocation and task scheduling.

Benefit
- Rich feature set for general purpose parallelism.
- Available as an open source and a commercial license.
- Supports C++, Windows*, Linux*, OS X*, other Os's.
- Commercial support for Intel® Atom™, Core™, Xeon® processors, and for Intel® Xeon Phi™ coprocessors

Also available as open source at threadingbuildingblocks.org
https://software.intel.com/intel-tbb
"Intel® TBB provided us with optimized code that we did not have to develop or maintain for critical system services. I could assign my developers to code what we bring to the software table."

Michaël Rouillé, CTO, Golaem
## What’s New

Intel® Threading Building Blocks 4.3

<table>
<thead>
<tr>
<th>Feature</th>
<th>Benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Allocator Improvements</td>
<td>Improved tbbmalloc – increases performance and scalability for threaded applications</td>
</tr>
<tr>
<td>Improved Intel® TSX Support</td>
<td>Applications that use read-write locks can take additional advantage of Intel TSX via tbb::speculative_spin_rw_mutex</td>
</tr>
<tr>
<td>Improved C++ 11 support</td>
<td>Improved compatibility with C++ 11 standard.</td>
</tr>
<tr>
<td>Tasks arenas</td>
<td>Improved control over workload isolation and the degree of concurrency with new class tbb::task_arena</td>
</tr>
<tr>
<td>Latest and Future Intel® Architecture Support</td>
<td>Supports¹ latest Intel® Architecture. Future proof with the next generation.</td>
</tr>
</tbody>
</table>

**New Flow Graph Designer tool (currently in Alpha)**

Visualize graph execution flow that allows better understanding of how Intel® TBB flow graph works.

Available today on [http://whatif.intel.com](http://whatif.intel.com)

¹See Intel® TBB release notes for hardware support matrix
Resources and Availability
Intel® Threading Building Blocks (Intel® TBB)

Resources:

The Open-Source Community Site: www.threadingbuildingblocks.org


Flow Graph Designer: https://software.intel.com/en-us/articles/flow-graph-designer


Availability


Also available as open source at threadingbuildingblocks.org
Intel® Integrated Performance Primitives (IPP)

A software developer's competitive edge

Optimized performance focused on the compute-intensive tasks that matter to you

Easy to use building blocks to create high performance workflows in String Processing, Data Compression, Image Processing, Cryptography, Signal Processing, & Computer Vision

Unleash your potential through access to silicon

Consistent C APIs span multiple generations of Intel’s processors and SoC solutions, removing the need to develop for specific architecture optimizations

Included in Intel® Parallel Studio XE Suites

The optimizations you need, available where you need them
Wide platform support from a single set of APIs

IPP automatically selects the optimizations based on the processor running your application
## Intel® IPP Programming Domains

<table>
<thead>
<tr>
<th>Signal Processing</th>
<th>Image Processing</th>
<th>Computer Vision</th>
<th>String Processing</th>
<th>Data Compression</th>
<th>Cryptography</th>
</tr>
</thead>
<tbody>
<tr>
<td>One dimensional input data processing</td>
<td>2D input data processing including color conversion support</td>
<td>Includes optimizations that accelerate common OpenCV functions</td>
<td>String manipulation and regular expression functionality</td>
<td>Huffman, VLC and Dictionary compression techniques</td>
<td>Support for standard cryptographic algorithms</td>
</tr>
</tbody>
</table>

Domains previously marked for deprecation are no longer in the default IPP package. Please see the IPP documentation for additional details.

* Cryptography domain may not be available in all geographies
Intel® Math Kernel Library (MKL)

- Speeds math processing in scientific, engineering and financial applications
- Functionality for dense and sparse linear algebra (BLAS, LAPACK, PARDISO), FFTs, vector math, summary statistics and more
- Provides scientific programmers and domain scientists
  - Interfaces to de-facto standard APIs from C++, Fortran, C#, Python and more
  - Support for Linux®, Windows® and OS X® operating systems
  - The ability to extract great parallel performance with minimal effort
- Unleash the performance of Intel® Core, Intel® Xeon and Intel® Xeon Phi™ product families
  - Optimized for single core vectorization and cache utilization
  - Coupled with automatic OpenMP*-based parallelism for multi-core, manycore and coprocessors
  - Scales to PetaFlop (1015 floating-point operations/second) clusters and beyond
- Included in Intel® Parallel Studio XE Suites

**Used on the World’s Fastest Supercomputers**

**http://www.top500.org**

Copyright © 2015, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.
Intel® MKL is a Computational Math Library

Mathematical problems arise in many scientific disciplines

These scientific applications areas typically involve mathematics ...

- Differential equations
- Linear algebra
- Fourier transforms
- Statistics

Intel® MKL can help solve your computational challenges
## Optimal Mathematical Building Blocks

### Intel® MKL

<table>
<thead>
<tr>
<th>Linear Algebra</th>
<th>Fast Fourier Transforms</th>
<th>Vector Math</th>
<th>And More</th>
</tr>
</thead>
<tbody>
<tr>
<td>🟦 BLAS</td>
<td>🟦 Multidimensional</td>
<td>🟦 Trigonometric</td>
<td>🟦 Splines</td>
</tr>
<tr>
<td>🟦 LAPACK</td>
<td>🟦 FFTW interfaces</td>
<td>🟦 Hyperbolic</td>
<td>🟦 Interpolation</td>
</tr>
<tr>
<td>🟦 Sparse Solvers</td>
<td>🟦 Cluster FFT</td>
<td>🟦 Exponential, Log</td>
<td>🟦 Trust Region</td>
</tr>
<tr>
<td>🟦 Iterative</td>
<td></td>
<td>🟦 Power / Root</td>
<td>🟦 Fast Poisson Solver</td>
</tr>
<tr>
<td>🟦 Pardiso*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>🟦 ScaLAPACK</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Vector RNGs

- Congruential
- Wichmann-Hill
- Mersenne Twister
- Sobol
- Neiderreiter
- Non-deterministic

### Summary Statistics

- Kurtosis
- Variation coefficient
- Order statistics
- Min/max
- Variance-covariance

### And More

- Splines
- Interpolation
- Trust Region
- Fast Poisson Solver
Intel® MKL is a Performance Library

We go to extremes to get the most performance from the available resources

- Core: vectorization, prefetching, cache utilization
- Multicore (processor/socket) level parallelization
- Multi-socket (node) level parallelization
  - Cluster scaling
  - Data locality is key

Automatic scaling from multicore to manycore and beyond
Immediate Performance Benefit to Applications

Intel® MKL

Significant LAPACK Performance Boost using Intel® Math Kernel Library versus ATLAS*

DGETRF on Intel® Xeon® E5-2690 Processor

The latest version of Intel® MKL unleashes the performance benefits of Intel architectures
Analysis Tools

Intel® VTune™ Amplifier XE
Performance Profiler

Intel® Inspector XE
Memory & Thread Debugger

Intel® Advisor XE
Threading Design & Prototyping

Photo (c) 2014, James Reinders; used with permission; Yosemite Half Dome rising through forest fire smoke 11am on September 10, 2014

© 2015, Intel Corporation. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi are trademarks of Intel Corporation in the U.S. and/or other countries. *Other names and brands may be claimed as the property of others.
Intel® VTune™ Amplifier XE
Tune Applications for Scalable Multicore Performance

Intel® VTune™ Amplifier XE Performance Profiler

Is your application slow?

Does its speed scale with more cores?

Tuning without data is just guessing

- Accurate CPU, GPU1 & threading data
- Powerful analysis & filtering of results
- Easy set-up, no special compiles

"Last week, Intel® VTune™ Amplifier XE helped us find almost 3X performance improvement. This week it helped us improve the performance another 3X."

Claire Cates
Principal Developer
SAS Institute Inc.

For Windows* and Linux* From $899
(GUI only now available on OS X*)

1 Windows* only.
Tune Applications for Scalable Multicore Performance
Intel® VTune™ Amplifier XE Performance Profiler

Get the Data You Need
- Hotspot (Statistical call tree), Call counts (Statistical)
- Thread Profiling – Concurrency and Lock & Waits Analysis
- Cache miss, Bandwidth analysis\(^1\)
- GPU Offload and OpenCL* Kernel Tracing on Windows

Find Answers Fast
- View Results on the Source / Assembly
- OpenMP Scalability Analysis, Graphical Frame Analysis
- Filter Out Extraneous Data - Organize Data with Viewpoints
- Visualize Thread & Task Activity on the Timeline

Easy to Use
- No Special Compiles – C, C++, C#, Fortran, Java, ASM
- Visual Studio* Integration or Stand Alone on Windows* or Linux*
- Graphical Interface & Command Line
- Local & Remote Data Collection
- New! Analyze Windows* & Linux* data from OS X*\(^2\)

\(^1\) Events vary by processor.  \(^2\) No data collection on OS X*
Good Tuning Data Gets Good Results

“We achieved a significant improvement (almost 2x) even on one core by optimizing the code based on the information provided by Intel® VTune™ Amplifier XE.”

Alexey Andrianov, R&D Director Deputy Mechanical Analysis Division Mentor Graphics Corporation

“The new VTune™ Amplifier XE brings even more capability to an already indispensable tool. The sampling based call stack hotspots is excellent and alone is worthy of the upgrade. We have also been impressed by how the concurrency and Locks and Waits analysis can even provide useful data on complex applications such as Premiere Pro.”


“Intel® VTune™ Amplifier XE analyzes complex code and helps us identify bottlenecks rapidly. By using it and other Intel® Software Development Tools, we were able to improve PIPESIM performance up to 10 times compared with the previous software version.”

Rodney Lessard Senior Scientist Schlumberger

Copyright © 2015, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.
What’s New in Intel® VTune™ Amplifier XE 2015?

Performance Profiler

Powerful Data Analysis
- Tune OpenMP threading & scalability
- New timeline & grid data groupings
- Correlate imported app logging data with other data

Easier to use
- Analyze Linux* or Windows* data on your Mac¹
- Easy collection from remote systems
- Auto select correct processor metrics
- Fewer driver hassles on Linux*

More Profiling Data for CPU & GPU
- Tune OpenCL™ kernels & GPU offload on Windows*
- TSX transactional analysis
- Reduce overhead with selectable stack depth

Support for the Latest Processors & OSs

¹ No data collection on OS X*

Copyright © 2015, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.
# Two Great Ways to Collect Data

## Intel® VTune™ Amplifier XE

<table>
<thead>
<tr>
<th>Software Collector</th>
<th>Hardware Collector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uses OS interrupts</td>
<td>Uses the on chip Performance Monitoring Unit (PMU)</td>
</tr>
<tr>
<td>Collects from a single process tree</td>
<td>Collect system wide or from a single process tree.</td>
</tr>
<tr>
<td>~10ms default resolution</td>
<td>~1ms default resolution (finer granularity - finds small functions)</td>
</tr>
<tr>
<td>Either an Intel® or a compatible processor</td>
<td>Requires a genuine Intel® processor for collection</td>
</tr>
<tr>
<td>Call stacks show calling sequence</td>
<td>Optionally collect call stacks</td>
</tr>
<tr>
<td>Works in virtual environments</td>
<td>Works in a VM only when supported by the VM (e.g., vSphere* 5.1)</td>
</tr>
<tr>
<td>No driver required</td>
<td>Requires a driver</td>
</tr>
<tr>
<td></td>
<td>- Easy to install on Windows</td>
</tr>
<tr>
<td></td>
<td>- Linux requires root</td>
</tr>
<tr>
<td></td>
<td>(or use default perf driver without stacks)</td>
</tr>
</tbody>
</table>

No special recompiles - C, C++, C#, Fortran, Java, Assembly
## A Rich Set of Performance Data

### Intel® VTune™ Amplifier XE

<table>
<thead>
<tr>
<th>Software Collector</th>
<th>Hardware Collector</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Basic Hotspots</strong></td>
<td><strong>Advanced Hotspots</strong></td>
</tr>
<tr>
<td>Which functions use the most time?</td>
<td>Which functions use the most time? Where to inline? – Statistical call counts</td>
</tr>
<tr>
<td><strong>Concurrency</strong></td>
<td><strong>General Exploration</strong></td>
</tr>
<tr>
<td>Tune parallelism. Colors show number of cores used.</td>
<td>Where is the biggest opportunity? Cache misses? Branch mispredictions?</td>
</tr>
<tr>
<td><strong>Locks and Waits</strong></td>
<td><strong>Advanced Analysis</strong></td>
</tr>
<tr>
<td>Tune the #1 cause of slow threaded performance: – waiting with idle cores.</td>
<td>Dig deep to tune access contention, etc.</td>
</tr>
</tbody>
</table>

Any IA86 processor, any VM, no driver

Higher res., lower overhead, system wide

---

**No special recompiles - C, C++, C#, Fortran, Java, Assembly**
Intel® VTune™ Amplifier XE
Tune Applications for Scalable Multicore Performance

Agenda

- Data Collection – Rich set of performance data
- Data Analysis - Find answers fast
- Flexible workflow –
  - User i/f and command line
  - Compare results
  - Remote collection
- New for 2015!
- Summary
Find Answers Fast
Intel® VTune™ Amplifier XE

Adjust Data Grouping
- Function - Call Stack
- Module - Function - Call Stack
- Source File - Function - Call Stack
- Thread - Function - Call Stack
... (Partial list shown)

Double Click Function to View Source

Click [-] for Call Stack

Filter by Timeline Selection (or by Grid Selection)

Filter by Process & Other Controls
Tuning Opportunities Shown in Pink. Hover for Tips
See Profile Data On Source / Asm
Double Click from Grid or Timeline

View Source / Asm or both
CPU Time
Right click for instruction reference manual

Quick Asm navigation:
Select source to highlight Asm

Quickly scroll to hot spots. Scroll Bar
“Heat Map” is an overview of hot spots

Click jump to scroll Asm
Visualize Parallel Performance Issues
Look for Common Patterns

- Coarse Grain Locks
- High Lock Contention
- Load Imbalance

Low Concurrency
Timeline Visualizes Thread Behavior

Intel® VTune™ Amplifier

Optional: Use API to mark frames and user tasks

Optional: Add a mark during collection
Intel® VTune™ Amplifier
Tune Applications for Scalable Multicore Performance

Agenda

- Data Collection – Rich set of performance data
- Data Analysis - Find answers fast
- Flexible workflow –
  - User i/f and command line
  - Compare results
  - Remote collection
- New for 2015!
- Summary
Command Line Interface
Automate analysis

amplxe-cl is the command line:
- **Windows:** `C:\Program Files (x86)\Intel\VTune Amplifier XE \bin[32|64]\amplxe-cl.exe`
- **Linux:** `/opt/intel/vtune_amplifier_xe/bin[32|64]/amplxe-cl`

Help: `amplxe-cl -help`

Use UI to setup
1) Configure analysis in UI
2) Press “Command Line...” button
3) Copy & paste command

Great for regression analysis – send results file to developer
Command line results can also be opened in the UI
Interactive Remote Data Collection
Performance analysis of remote systems just got a lot easier

Interactive analysis
1) Configure SSH to a remote Linux* target
2) Choose and run analysis with the UI

Command line analysis
1) Run command line remotely on Windows* or Linux* target
2) Copy results back to host and open in UI

Conveniently use your local UI to analyze remote systems
Compare Results Quickly - Sort By Difference
Intel® VTune™ Amplifier XE

Quickly identify cause of regressions.
- Run a command line analysis daily
- Identify the function responsible so you know who to alert

Compare 2 optimizations – What improved?

Compare 2 systems – What didn't speed up as much?
Intel® VTune™ Amplifier XE
Tune Applications for Scalable Multicore Performance

Agenda

- Data Collection – Rich set of performance data
- Data Analysis - Find answers fast
- Flexible workflow –
  - User i/f and command line
  - Compare results
  - Remote collection
- New for 2015!
- Summary
What’s New in Intel® VTune™ Amplifier XE 2015?

Performance Profiler

Powerful Data Analysis

- Tune OpenMP* threading & scalability
- New timeline & grid data groupings
- Correlate imported app logging data with other data

Easier to use

- Analyze Linux* or Windows* data on your Mac*¹
- Easy collection from remote systems
- Auto select correct processor metrics
- Fewer driver hassles on Linux*

More Profiling Data for CPU & GPU

- Tune OpenCL™ kernels & GPU offload on Windows*
- Transactional analysis for Intel® TSX²
- Reduce overhead with selectable stack depth

Support for the Latest Processors & OSs

º No data collection on OS X* ²Intel® Transactional Synchronization Extensions (Intel® TSX)
OS X* Host Support
Intel® VTune™ Amplifier XE

Host Runs on OS X*
- Analyze data from Linux*
- Analyze data from Windows*
- No local OS X data collection

No Extra Charge
- Separate download
- Uses your Windows or Linux license

Easy Remote Collection
- SSH to remote Linux
OpenMP* Scalability Analysis
Intel® VTune™ Amplifier XE

Identify serial time and load imbalance

- Are you spending time in serial regions?
- Are some threads finishing before others in a parallel region?

Find & tune slow instances in a region

Intel® Xeon® and Xeon Phi® systems

Intel and gcc* runtimes

Great Tuning Data for More OpenMP Performance!
Tune GPU Compute Performance
Intel® VTune™ Amplifier for Windows*

**New!**

**Tune OpenCL™ Kernels & GPU offload**
On newer processors, optionally collect GPU data. Correlate GPU and CPU activities. (Windows* only.)

**Opportunities Highlighted**
The cell is highlighted (pink) when there is a potential tuning opportunity. Hover to get suggestions.

**Tune for the whole processor, CPU + GPU**
Fewer Driver Hassles on Linux*
Intel® VTune™ Amplifier XE

Auto-rebuild Intel EBS driver
- Does advanced analysis stop working when an OS update is installed?
- Do you have to ask IT to rebuild the driver?
- No longer! Just setup the driver to auto-rebuild when the OS is updated.

Auto-disable NMI watchdog
- Tired of turning off NMI watchdog to run advanced EBS profiling?
- Now you don’t have to. We turn it off, then put it back the way it was.

Use pre-installed perf driver
- IT won’t install the Intel driver?
- Use perf!
- Intel EBS driver does provide additional features not available in perf:
  - Stacks
  - Uncore events
  - Multiple precise events
  - New events for the latest processors, even on an older OS

Easier access to the on-chip PMU for advanced performance profiling
Interactive Remote Data Collection
Performance analysis of remote systems just got a lot easier

Interactive analysis
1) Configure SSH to a remote Linux* target
2) Choose and run analysis with the UI

Command line analysis
1) Run command line remotely on Windows* or Linux* target
2) Copy results back to host and open in UI

Conveniently use your local UI to analyze remote systems
Intel® VTune™ Amplifier XE
Tune Applications for Scalable Multicore Performance

Agenda

- Data Collection – Rich set of performance data
- Data Analysis - Find answers fast
- Flexible workflow –
  - User i/f and command line
  - Compare results
  - Remote collection
- New for 2015!

- Summary
Intel® VTune™ Amplifier XE
Tune Applications for Scalable Multicore Performance

Get the Data You Need
- Hotspot (Statistical call tree), Call counts (Statistical)
- Thread Profiling – Concurrency and Lock & Waits Analysis
- Cache miss, Bandwidth analysis...¹
- GPU Offload and OpenCL™ Kernel Tracing on Windows

Find Answers Fast
- View Results on the Source / Assembly
- OpenMP Scalability Analysis, Graphical Frame Analysis
- Filter Out Extraneous Data - Organize Data with Viewpoints
- Visualize Thread & Task Activity on the Timeline

Easy to Use
- No Special Compiles – C, C++, C#, Fortran, Java*, ASM
- Visual Studio* Integration or Stand Alone on Windows* or Linux*
- Graphical Interface & Command Line
- Local & Remote Data Collection
- New! Analyze Windows* & Linux* data on OS X*²

¹ Events vary by processor. ² No data collection on OS X*

Quickly Find Tuning Opportunities

See Results On The Source Code

Timeline Visualizes & Filters
Frame Analysis in Intel® VTune™ Amplifier XE
Filter Data - Get Actionable Information
Intel® VTune™ Amplifier

Frame Analysis –
Analyze Long Latency Activity

Frame: a region executed repeatedly (non-overlapping)

- API marks start and finish or
- Auto detect DirectX* frames

Examples:
- Game – Compute next graphics frame
- Simulator – Time step loop
- Computation – Convergence loop

Application
void algorithm_1();
void algorithm_2(int myid);
double GetSeconds();
DWORD WINAPI do_xform (void * lpmyid);
bool checkResults();
__itt_domain *fPtr;
fPtr = __itt_domain_create("My Domain");

Region (Frame)
while( gRunning ) {
    __itt_frame_begin_v3(fPtr, NULL);
    //Do Work
    __itt_frame_end_v3(fPtr, NULL);
}

for (int k = 0; k < N; ++k) {
    int ik = i*N + k;
    int kj = k*N + j;
c2[ij] += a[ik]*b[kj];
}
Find Slow Frames With One Click
Intel® VTune™ Amplifier

(1) Regroup Data

Function - Call Stack
Module - Function - Call Stack
Source File - Function - Call Stack
Thread - Function - Call Stack
Function - Thread - Call Stack
OpenMP Region - Function - Call Stack
Task Type - Function - Call Stack
Frame Domain - Frame - Function - Call Stack

Before: List of Functions Taking Time

After: List of Slow Frames
Find Slow Functions in Slow Frames
Turn raw data into actionable information

(1) Only show slow frames

Result:
Functions taking a lot of time in slow frames

(2) Regroup: Show functions
Intel® InspectorXE

Photo (c) 2014, James Reinders; used with permission; Yosemite Half Dome rising through forest fire smoke 11am on September 10, 2014
Find & Debug Memory & Threading Errors
Intel® Inspector XE – Memory & Thread Debugger

Correctness Tools Increase ROI By 12%-21%1

- Errors found earlier are less expensive to fix
- Several studies, ROI% varies, but earlier is cheaper

Diagnosing Some Errors Can Take Months

- Races & deadlocks not easily reproduced
- Memory errors can be hard to find without a tool

Debugger Integration Speeds Diagnosis

- Breakpoint set just before the problem
- Examine variables & threads with the debugger

Diagnose in hours instead of months

1 Cost Factors – Square Project Analysis
CERT: U.S. Computer Emergency Readiness Team, and Carnegie Mellon CyLab
NIST: National Institute of Standards & Technology : Square Project Results

Intel® Inspector XE dramatically sped up our ability to track down difficult to isolate threading errors before our packages are released to the field.

Peter von Kaenel, Director, Software Development, Harmonic Inc.

http://intel.ly/inspector-xe
Correctness Tools Increase ROI By 12%-21%

Cost Factors – Square Project Analysis
CERT: U.S. Computer Emergency Readiness Team, and Carnegie Mellon CyLab
NIST: National Institute of Standards & Technology : Square Project Results

Size and complexity of applications is growing

Reworking defects is 40%-50% of total project effort

Correctness tools find defects during development prior to shipment

Reduce time, effort, and cost to repair

Find errors earlier when they are less expensive to fix
Race Conditions Are Difficult to Diagnose
They only occur occasionally and are difficult to reproduce

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
<th>Shared Counter</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Read count</td>
<td>←</td>
<td>0</td>
</tr>
<tr>
<td>Increment</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Write count</td>
<td>→</td>
<td>1</td>
</tr>
<tr>
<td>Read count</td>
<td>←</td>
<td>1</td>
</tr>
<tr>
<td>Increment</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Write count</td>
<td>→</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
<th>Shared Counter</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Read count</td>
<td>←</td>
<td>0</td>
</tr>
<tr>
<td>Increment</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Read count</td>
<td>←</td>
<td>0</td>
</tr>
<tr>
<td>Increment</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Write count</td>
<td>→</td>
<td>1</td>
</tr>
<tr>
<td>Increment</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Write count</td>
<td>→</td>
<td>1</td>
</tr>
</tbody>
</table>
“We struggled for a week with a crash situation, the corruption was identified but the source was really hard to find. Then we ran Intel® Inspector XE and immediately found the array out of bounds that occurred long before the actual crash. We could have saved a week!”

Mikael Le Guerroué, Senior Codec Architecture Engineer, Envivio

“Intel® Inspector XE is quite fast and intuitive compared to products we have used in the past. We can now run our entire batch of test cases (~750) which was not feasible previously. Intel® Inspector XE easily completed tests that failed due to lack of virtual memory on another product.”

Gerald Mattauch, Senior Software Developer, Siemens AG, Healthcare Sector

Intel® Inspector XE has dramatically sped up our ability to find/fix memory problems and track down difficult to isolate threading errors before our packages are released to the field.

Peter von Kaenel, Director, Software Development, Harmonic Inc.
Dramatically Faster Thread Checking on Windows* and Linux* with Intel® Inspector XE 2015

**New!**

**Faster Race & Deadlock Analysis, Linux***  
(Lower is Better)

- **7zip**: 10.2x  
- **blender**: 6.3x  
- **firefox**: 1.6x

**Faster Race & Deadlock Analysis, Windows***  
(Lower is Better)

- **7zip**: 1.8-16x  
- **blender**: 6.8x  
- **firefox**: 16.7x

+ On open source applications 7zip, blender and firefox. Runtime improvements will vary by application and OS.

View configuration information at end of this presentation: Click to view

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.
Incrementally Diagnose Memory Growth
Intel® Inspector XE 2015

As your app is running...

- Memory usage graph plots memory growth
- Select a cause of memory growth
- See the code snippet & call stack
Data Driven Threading Design
Intel® Advisor XE – Thread Prototyping for Software Architects

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
Design Then Implement
Intel® Advisor XE Thread Prototyping

Design Parallelism
- No disruption to regular development
- All test cases continue to work
- Tune and debug the design before you implement it

Implement Parallelism

Less Effort, Less Risk, More Impact
Prototyping Speeds Effective Threading Design

“Intel® Advisor XE has allowed us to quickly prototype ideas for parallelism, saving developer time and effort, and has already been used to highlight subtle parallel correctness issues in complex multi-file, multi-function algorithms.”

Matt Osterberg
Senior Software Engineer
Vickery Research Alliance

“Intel® Advisor XE can be invaluable in developing the understanding required to parallelize existing code. It assists with identifying opportunities, designing tests, modeling scenarios and revealing flaws.”

“Intel® Advisor XE has been extremely helpful in identifying the best pieces of code for parallelization. We can save several days of manual work by targeting the right loops. At the same time, we can use Advisor to find potential thread safety issues to help avoid problems later on.”

Carlos Boneti
HPC software engineer,
Schlumberger

Simon Hammond
Senior Technical Staff
Sandia National Laboratories

More Case Studies
Vectorization Design
Intel® Advisor XE Future Release

Currently Under Development
- Beta test during 2014
- Targeting launch in 2015

High ROI for Vectorization Effort
- What vectorization will pay off the most?
- What is blocking vectorization?
- What is the benefit to reorganization?

Vectorize Your Code Faster with More Confidence
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

Part of Intel® Parallel Studio
For Windows* and Linux* From $1,599

Add Parallelism with Less Effort, Less Risk and More Impact
Threading Advisor in Intel® Advisor XE
Design Then Implement

Intel® Advisor XE Thread Prototyping

Design Parallelism

- No disruption to regular development
- All test cases continue to work
- Tune and debug the design before you implement it

Design Then Implement

Implement Parallelism

Less Effort, Less Risk, More Impact
Threading Advisor
Threading design and prototyping for software architects

**Iteration Space Modeling**
Move the sliders to see what happens when you change the number and duration of tasks.

**Info Zone**
High-level break-down of parallelism performance losses: imbalance, contention and parallel runtime overheads impact

**Fast Prototyping for Better Software Design**
Threading Advisor

- New Target Platforms option – See modeling based on:
  - Intel® Xeon® processors or
  - Intel® Xeon Phi™ coprocessors

Make better design decisions with more confidence
Cluster Tools – MPI tools!
**Reduced Latency Means Faster Performance**

**Intel® MPI Library**

### Superior Performance with Intel® MPI Library 5.0

192 Processes, 8 nodes (InfiniBand + shared memory), Linux* 64

Relative (Geometric) MPI Latency Benchmarks (Higher is Better)

<table>
<thead>
<tr>
<th>Speedup (times)</th>
<th>4 bytes</th>
<th>512 bytes</th>
<th>16 Kbytes</th>
<th>128 Kbytes</th>
<th>4 Mbytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel MPI 5.0</td>
<td>3.1</td>
<td>2.5</td>
<td>3.4</td>
<td>2.9</td>
<td>2.2</td>
</tr>
<tr>
<td>Platform MPI 9.1.2 CE</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MVAPICH2 2.0rc2</td>
<td>2.0</td>
<td>1.9</td>
<td>1.7</td>
<td>1.6</td>
<td>1.8</td>
</tr>
<tr>
<td>OpenMPI 1.7.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Configuration:**
- CPU: Dual Intel® Xeon® E5-2697v2@2.7GHz 64 GB RAM.
- Interconnect: Mellanox Technologies* MT27500 Family (ConnectX®-3) FDR,
  Mellanox OFED 3.2-1, Mellanox LibrED 2.2.0-rc2 Intel® MPI Library 5.0 Intel® MPI Benchmarks 2.2.0-rc2
- Hardware: Intel® Xeon® CPU E5-2697 v2 @ 2.7GHz; RAM 64GB; Interconnect: InfiniBand, ConnectX adapters, FDR, PCIe: C0-IBC 1338095 kHz; 61 cores. RH 64-bit 64xGB RHEL 6.2 OFED 3.2-1; MVAPICH2-2.0rc2 Intel® C/C++ Compiler XE 13.1.1; Intel® MPI Benchmarks 2.2.0-rc2

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.

---

### Superior Performance with Intel® MPI Library 5.0

64 Processes, 8 nodes (InfiniBand + shared memory), Linux* 64

Relative (Geometric) MPI Latency Benchmarks (Higher is Better)

<table>
<thead>
<tr>
<th>Speedup (times)</th>
<th>4 bytes</th>
<th>512 bytes</th>
<th>16 Kbytes</th>
<th>128 Kbytes</th>
<th>4 Mbytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel MPI 5.0</td>
<td>2.0</td>
<td>1.9</td>
<td>1.7</td>
<td>1.6</td>
<td>1.8</td>
</tr>
<tr>
<td>MVAPICH2 2.0rc2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Configuration:**
- CPU: Dual Intel® Xeon® E5-2680v2@2.7GHz 64 GB RAM.
- Interconnect: Mellanox Technologies* MT27500 Family (ConnectX®-3) FDR,
  Mellanox OFED 3.2-1, Mellanox LibrED 2.2.0-rc2 Intel® MPI Library 5.0 Intel® MPI Benchmarks 2.2.0-rc2
- Hardware: Intel® Xeon® CPU E5-2680v2 @ 2.7GHz; RAM 64GB; Interconnect: InfiniBand, ConnectX adapters, FDR, PCIe: C0-IBC 1338095 kHz; 61 cores. RH 64-bit 64xGB RHEL 6.2 OFED 3.2-1; MVAPICH2-2.0rc2 Intel® C/C++ Compiler XE 13.1.1; Intel® MPI Benchmarks 2.2.0-rc2

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.
Overview
Intel® Trace Analyzer and Collector

Intel® Trace Analyzer and Collector helps the developer:
- Visualize and understand parallel application behavior
- Evaluate profiling statistics and load balancing
- Identify communication hotspots

Features
- Event-based approach
- Low overhead
- Excellent scalability
- Powerful aggregation and filtering functions
- Idealizer
Overview

Intel® Trace Analyzer and Collector

Source Code → Compiler → Objects → Linker → Binary → Runtime → Output

API and -tcollect

Intel® Trace Collector

Trace File (.stf)

Intel® Trace Analyzer
Intel® Trace Analyzer and Collector

Compare the event timelines of two communication profiles

Blue = computation
Red = communication

Chart showing how the MPI processes interact
What’s New
Intel® Trace Analyzer and Collector

MPI Communications Profile Summary

Overview

Expanded Standards Support with MPI 3.0

Automatic Performance Assistant

- Detect common MPI performance issues
- Automated tips on potential solutions

Automatically detect performance issues and their impact on runtime
Improving Load Balance: Real World Case

Collapsed data per node and coprocessor card

Host
16 MPI procs x 1 OpenMP thread

Coprocessor
8 MPI procs x 28 OpenMP threads

Too high load on Host = too low load on coprocessor
Improving Load Balance: Real World Case

Collapsed data per node and coprocessor card

Host
16 MPI procs x 1 OpenMP thread

Coprocessor
24 MPI procs x 8 OpenMP threads

Too low load on Host = too high load on coprocessor
Improving Load Balance: Real World Case

Collapsed data per node and coprocessor card

Host
16 MPI procs x
1 OpenMP thread

Coprocessor
16 MPI procs x
12 OpenMP thrds

Perfect balance
Host load = Coprocessor load
Ideal Interconnect Simulator (Idealizer)
# Product Briefs, Evaluation Guides, White Paper

## Product Briefs
- Intel® Cluster Studio XE 2013
- Intel® Parallel Studio XE 2013
- Intel® Composer XE 2013
- Intel® VTune™ Amplifier XE 2013
- Intel® Inspector XE 2013
- Intel® Advisor XE 2013
- Intel® Math Kernel Library 11.0
- Intel® Integrated Performance Primitives 7.1 Library
- Intel® Threading Building Blocks 4.1
- Intel® MPI Library 4.1
- Intel® Graphics Performance Analyzers 2012
- Intel® SDK for OpenCL Applications 2012
- Intel® Media SDK 2012
- Intel® System Studio 2013 for Linux* OS
- Intel® Perceptual Computing SDK

## Evaluation Guides
- Evaluation Guide Portal
- Get an easy Performance Boost even with Unthreaded Apps
- A Simple Path to Parallelism with Intel® Cilk™ Plus
- Efficiently Introduce Threading using Intel® TBB 4.1
- Design Parallel Performance with Less Risk and More Impact
- Resolve Resource Leaks to Improve Program Stability
- Eliminate Threading Errors to Improve Program Stability
- Eliminate Memory Errors to Improve Program Stability
- Improve C++ Code Quality with Static Analysis
- Improve Fortran Code Quality with Static Analysis

## White Papers
- The ROI from Optimizing Software Performance with Intel Parallel Studio XE
- A Concise Guide to Parallel Programming Tools for Intel Xeon Processors
- Java support in Intel VTune Amplifier XE
- An Introduction to Vectorization with the Intel® C++ Compiler
- An Introduction to Vectorization with the Intel® Fortran Compiler
- Xeon Phi MIC Developer home page
- Programming for Multicore & Many-core
- Xeon Phi Solution Brief: Parallel Processing, Unparalleled Discovery
- Xeon Phi Webinar (Slides & Videos)
- Beyond Offloading: Programming Models for Xeon Phi (IDF deck)

---

All the links on this page are public that can be accessed from the internet
### Case Studies

<table>
<thead>
<tr>
<th>CAE/Manufacturing</th>
<th>HPC</th>
<th>Image and Video</th>
<th>Gaming and Digital Content Creation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Flow Science uses Cluster Studio XE (ITAC, MPI) for CFD</strong>&lt;br&gt;Altair Speeds Complex Simulation w/ Xeon Phi&lt;br&gt;Altair uses Intel compilers and MPI for CAE and CFD software&lt;br&gt;MSC Software SimXpert w/ TBB&lt;br&gt;SIMULIA turns to Parallel Studio XE&lt;br&gt;Altair crash simulation w/ Intel SW tools&lt;br&gt;ESI Group achieves up to 450% faster performance</td>
<td><strong>RWTH Aachen University adopts Parallel Studio</strong>&lt;br&gt;Comparing Arrays of Structures &amp; Structures of Arrays on Xeon vs Xeon Phi&lt;br&gt;ISPP: eLearning Software earns A+&lt;br&gt;Kyoto University: Xeon and Cluster Studio XE</td>
<td><strong>NEC used Intel compilers for Video Conversion</strong>&lt;br&gt;Fixstars High Speed CG Renderer using Parallel Studio XE&lt;br&gt;Nik Software rendering speed of HDR by 1.3x&lt;br&gt;Envivio* video encoding w/ Parallel Studio</td>
<td><strong>USC Gaming students use GPA, TBB and Parallel Studio XE</strong>&lt;br&gt;Golaem uses TBB, PSXE for crowd control&lt;br&gt;Dreamworks uses Intel MKL for Dazzling Special Effects&lt;br&gt;Geomerics removes Bakeware from the Runtime using Intel GPA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Financial</th>
<th>Education</th>
<th>AEROSPACE</th>
<th>Medical</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Black Scholes w/ Xeon Phi</strong>&lt;br&gt;Monte Carlo w/ Xeon Phi&lt;br&gt;DCSG: Thomson Reuters delivers real-time financial information&lt;br&gt;DRD: Computing Black Scholes w/ Intel AVX</td>
<td><strong>Parallel Performance for University of Florence and Avio</strong>&lt;br&gt;Aerospace Supercomputing Parallelism Advantage</td>
<td><strong>Aerospace Supercomputing Parallelism Advantage</strong>&lt;br&gt;Comparing Arrays of Structures &amp; Structures of Arrays on Xeon vs Xeon Phi</td>
<td><strong>Massachusetts General Hospital achieves 20x increase</strong>&lt;br&gt;Massachusetts General Hospital achieves 20x increase&lt;br&gt;Massachusetts General Hospital achieves 20x increase&lt;br&gt;Massachusetts General Hospital achieves 20x increase</td>
</tr>
</tbody>
</table>

**How to access links:**
- Click on link in Slideshow mode, OR
- Right click on link for Hyperlink options

---

**Optimization Notice**

Copyright © 2015, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.
# Technical Computing, Enterprise & HPC Tools

## Bundled Suites
- Intel® Parallel Studio XE 2013, Intel® Cluster Studio XE 2013
- Getting Started Tutorial
- Learn: Product Training

## Threading Prototyping Tool
- Intel® Advisor XE 2013
- Getting Started Tutorial
- Learn: Product Training

## Error Checking
- Intel® Inspector XE 2013
- Getting Started Tutorial
- Learn: Product Training

## Compilers
- Intel® Composer XE 2013
- Getting Started Tutorial for C++ and Fortran
- Learn: Product Training

## Libraries
- Intel® Math Kernel Library 11.0
- Learn: Product Training
- Intel® Integrated Performance Primitives
- Learn: Product Training

## Programming Models
- Intel® Cilk Plus
- Intel® Threading Building Blocks
- Intel® OpenMP®
- Intel® Coarray Fortran
- Intel® SDK for OpenCL Apps

---

All the links on this page are public that can be accessed from the internet.

---

How to access links:
- Click on link in Slideshow mode, OR
- Right click on link for Hyperlink options

---

Copyright © 2015, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.
Parallel Universe Magazine

Quarterly. Packed with insights on using Intel tools, and interesting industry innovations!

Published more than 5 years now.

Issue 21 included MPI SHM (shared memory) article.

http://tinyurl.com/intel-PUM
ixpug

Intel® Xeon Phi™ Users Group

ixpug.org
69 contributors
Real world applications.
High performance on Intel® Xeon processors and Intel® Xeon Phi™ coprocessors.

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

Short list: TRY THESE FIRST

**Higher Performance**
1. Recompile with Intel Compilers
2. Use Intel® Math Kernel Library (MKL)
3. Scale...
   - Use OpenMP directives
   - (C++ developers: use Intel® Threading Building Blocks – TBB)
4. Vectorization:
   - New optimization reports
   - New “OMP SIMD”

**Help with Optimization and Debug**
1. Intel® Advisor –
   - Scaling Analysis
   - Vectorization Advice
2. Intel® Inspector
   - Test for Threading Errors
   - Test for Memory Leaks
3. Intel® VTune™ Amplifier
   - Find Hotspots
4. Intel® Trace Analyze and Collector
   - MPI tuning
Short list: TRY THESE FIRST

Higher Performance
1. Recompile with Intel Compilers
2. Use Intel® Math Kernel Library (MKL)
3. Scale…
   • Use OpenMP directives
   • (C++ developers: use Intel® Threading Building Blocks – TBB)
4. Vectorization:
   • New optimization reports
   • New “OMP SIMD”

Help with Optimization and Debug
1. Intel® Advisor –
   • Scaling Analysis
   • Vectorization Advice
2. Intel® Inspector –
   • Test for Threading Errors
   • Test for Memory Leaks
3. Intel® VTune™ Amplifier –
   • Find Hotspots
4. Intel® Trace Analyze and Collector –
   • MPI tuning

james.r.reinders@intel.com
slides on lotsofcores.com
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.

Figures, diagrams, photos used from the book (High Performance Parallelism Pearls) used with the permission of the publisher and are available for download from www.lotsofcores.com for additional uses as well.

Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

Other names and brands may be claimed by others.

Copyright © 2014, 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.

### 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