

Faster Code.... Faster

Intel® Parallel Studio XE 2017

Dr.-Ing. Michael Klemm
Software and Services Group
michael.klemm@intel.com

Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

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

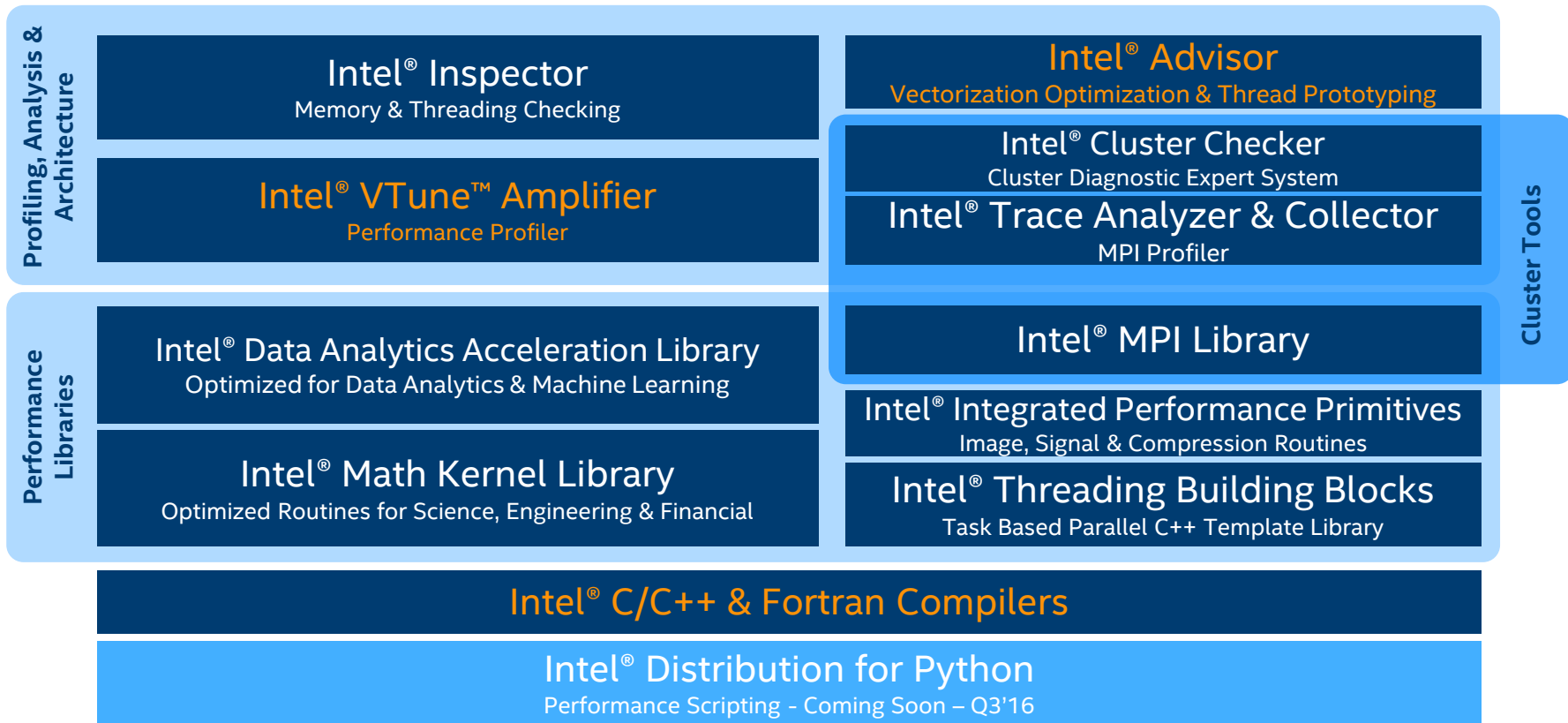
Copyright © 2016, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

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

Intel® Parallel Studio XE



Optimization Notice

Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



INTEL[®] COMPILERS

Intel® Compilers for Parallel Studio XE 2017

What's new in Intel® C++ 17.0 and Intel® Fortran 17.0

Productive language-level vectorization & parallelism models for advanced developers driving application performance

Common updates

- Enhanced support for the newest AVX2 and AVX512 instruction sets for the latest Intel® processors (including Intel® Xeon Phi)
- Enhanced optimization/vectorization reports – register allocation
- Tight integration with Intel® Advisor
- Initial support for OpenMP* 4.5, offering improved vectorization control, new SIMD instructions, and much more

Intel® C++ Compiler

- SIMD Data Layout Template to facilitate vectorization for your C++ code
- Virtual function vectorization capability
- Improved compiler loop and function alignment
- Full support for the latest C11 and C++14 standards

Intel® Fortran Compiler

- Substantial coarray performance improvement – up to **twice as fast** as previous versions on non-trivial coarray Fortran programs
- Almost complete Fortran 2008 support
- Further interoperability with C (part of draft Fortran 2015)

Impressive Performance Improvement

Intel® Compiler OpenMP* Explicit Vectorization

- Three lines added that take full advantage of both SSE or AVX
- Pragma's ignored by other compilers so code is portable

```
#pragma omp declare simd linear(z:40) uniform(L, N, Nmat) linear(k)
float path_calc(float *z, float L[][VLEN], int k, int N, int Nmat)
```

```
#pragma omp declare simd uniform(L, N, Nopt, Nmat) linear(k)
float portfolio(float L[][VLEN], int k, int N, int Nopt, int Nmat)
```

```
... ..
for (path=0; path<NPATH; path+=VLEN) {
```

```
/* Initialise forward rates */
z = z0 + path * Nmat;
```

```
#pragma omp simd linear(z:Nmat)
```

```
for(int k=0; k < VLEN; k++){
```

```
for(i=0;i<N;i++){
    L[i][k] = LO[i];
}
```

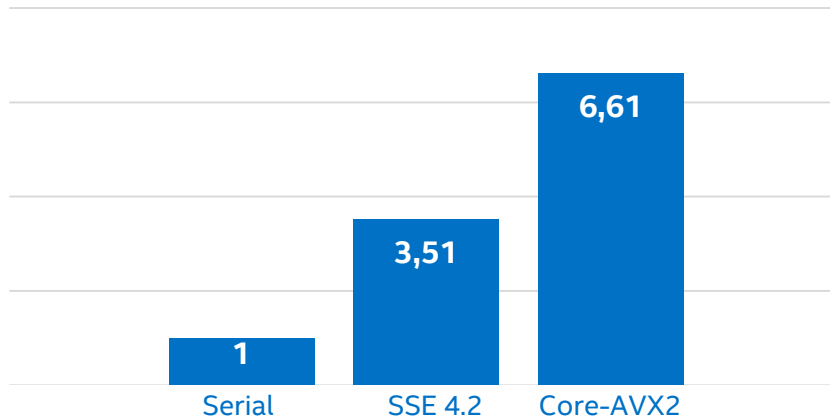
```
/* LIBOR path calculation */
```

```
float temp = path_calc(z, L, k, N, Nmat);
v[k+path] = portfolio(L, k, N, Nopt, Nmat);
```

```
/* move pointer to start of next block */
z += Nmat;
```

```
}
```

Libor calculation speedup
Normalized performance data – higher is better



Configuration: Intel® Xeon® CPU E3-1270 @ 3.50 GHz Haswell system (4 cores with Hyper-Threading On), running at 3.50GHz, with 32.0GB RAM, L1 Cache 256KB, L2 Cache 1.0MB, L3 Cache 8.0MB, 64-bit Windows® Server 2012 R2 Datacenter. Compiler options: SSE4.2: -O3 -Qopenmp-simd -QxSSE4.2 or AVX2: -O3 -Qopenmp-simd -QxCORE-AVX2. For more information go to <http://www.intel.com/performance>

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. * Other brands and names are the property of their respective owners. Benchmark. Source: Intel Corporation

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 SSE3.5 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.

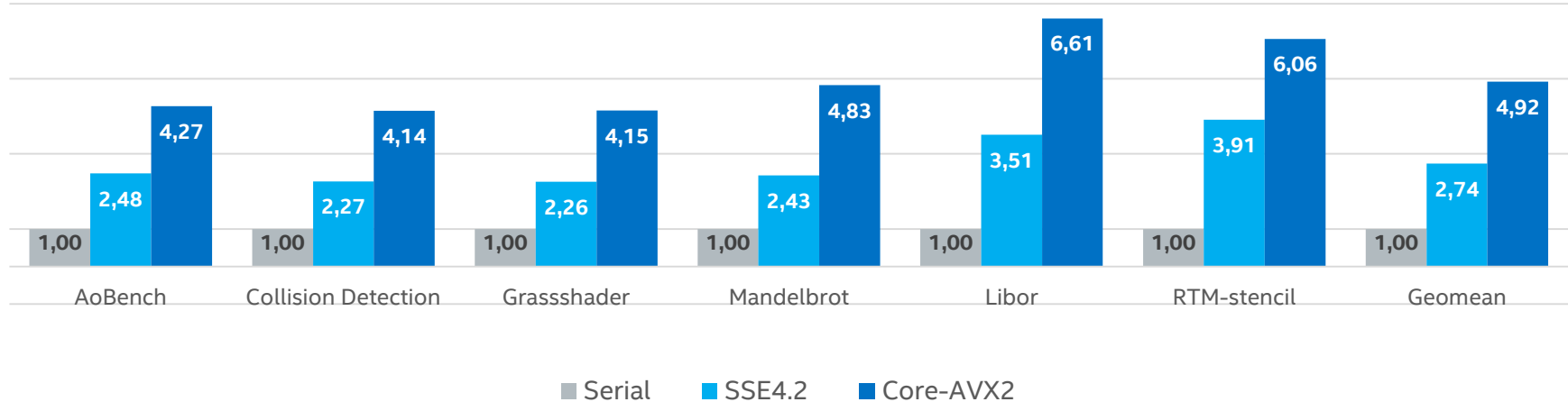
Optimization Notice

Impressive performance improvement

Intel C++ Explicit Vectorization using OpenMP* SIMD

SIMD Speedup on Intel® Xeon® Processor

Normalized performance data – higher is better



Configuration: Intel® Xeon® CPU E3-1270 @ 3.50 GHz Haswell system (4 cores with Hyper-Threading On), running at 3.50GHz, with 32.0GB RAM, L1 Cache 256KB, L2 Cache 1.0MB, L3 Cache 8.0MB, 64-bit Windows® Server 2012 R2 Datacenter. Compiler options: SSE4.2: -O3 -Qopenmp -simd -QxSSE4.2 or AVX2: -O3 -Qopenmp -simd -QxCORE-AVX2. For more information go to <http://www.intel.com/performance>

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. * Other brands and names are the property of their respective owners. Benchmark Source: Intel Corporation

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.

Optimization Notice

Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



INTEL SOFTWARE ANALYSIS TOOLS

Intel® VTune™ Amplifier XE Performance Profiler

Intel® Advisor XE Vectorization Optimization and Thread Prototyping

INTEL[®] VTUNE[™] AMPLIFIER XE PERFORMANCE PROFILER

Optimization Notice

Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Intel® VTune™ Amplifier

Faster, Scalable Code, Faster

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

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
- Graphical Interface & Command Line
- Local & Remote Data Collection
- Analyze Windows* & Linux* data on OS X²

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

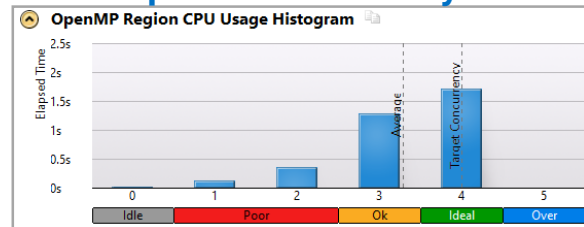
Quickly Find Tuning Opportunities

| Function / Call Stack | Effective Time by Utilization | Spin Time | Overhead Time |
|---|-------------------------------|-----------|---------------|
| FireObject::checkCollision | 4.507s | 0s | 0s |
| FireObject::ProcessFireCollisionsRange | 3.444s | 0s | 0s |
| NtWaitForSingleObject | 0s | 3.406s | 0s |
| std::basic_ifstream<char, struct std::char_traits | 3.359s | 0s | 0s |
| Ogre::FileSystemArchive::open | 3.359s | 0s | 0s |
| CBaseDevice::Present | 2.335s | 0.671s | 0s |

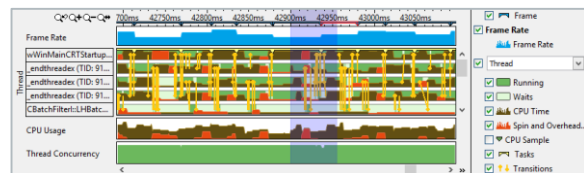
See Results On The Source Code

| Source Line | Source | CPU Time: Total by Utilization |
|-------------|---|--------------------------------|
| 81 | for (int i = 0; i < mem_array_i_max; i++) | 0.300s |
| 82 | { | |
| 83 | for (int j = 0; j < mem_array_j_max; j++) | 4.936s |
| 84 | { | |
| 85 | mem_array [j*mem_array_j_max+i] = *fill_val | 7.207s |

Tune OpenMP Scalability



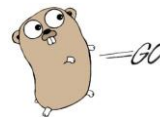
Visualize & Filter Data



Optimization Notice

Profile Python & Go!

And Mixed Python / C++ / Fortran



Low Overhead Sampling

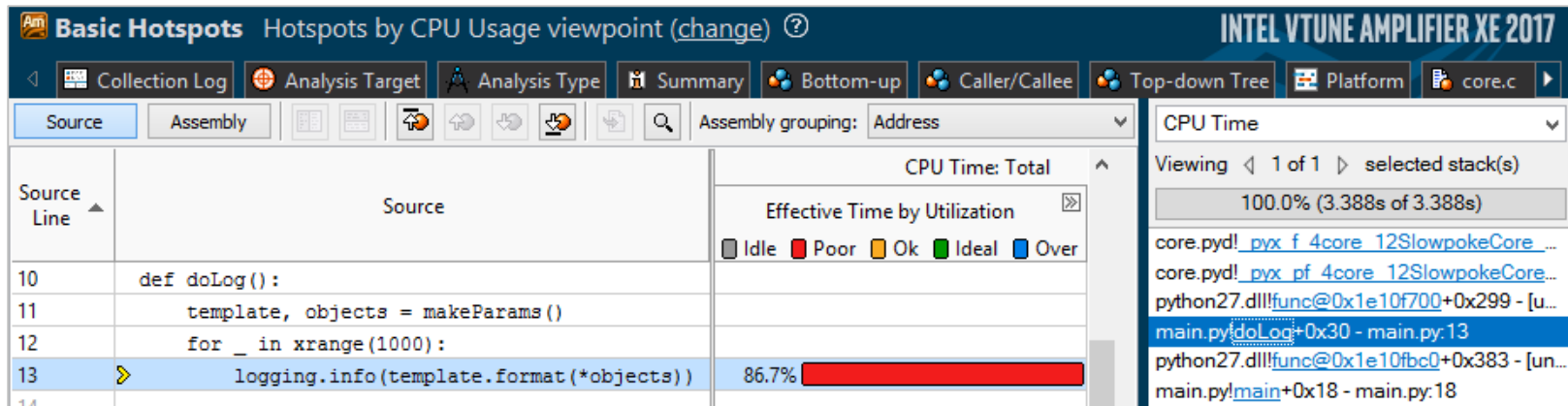
- Accurate performance data without high overhead instrumentation
- Launch application or attach to a running process

Precise Line Level Details

- No guessing, see source line level detail

Mixed Python / native C, C++, Fortran...

- Optimize native code driven by Python



Optimization Notice

Three Keys to HPC Performance:

Threading, Memory Access, Vectorization – Intel VTune™ Amplifier

Threading: CPU Utilization

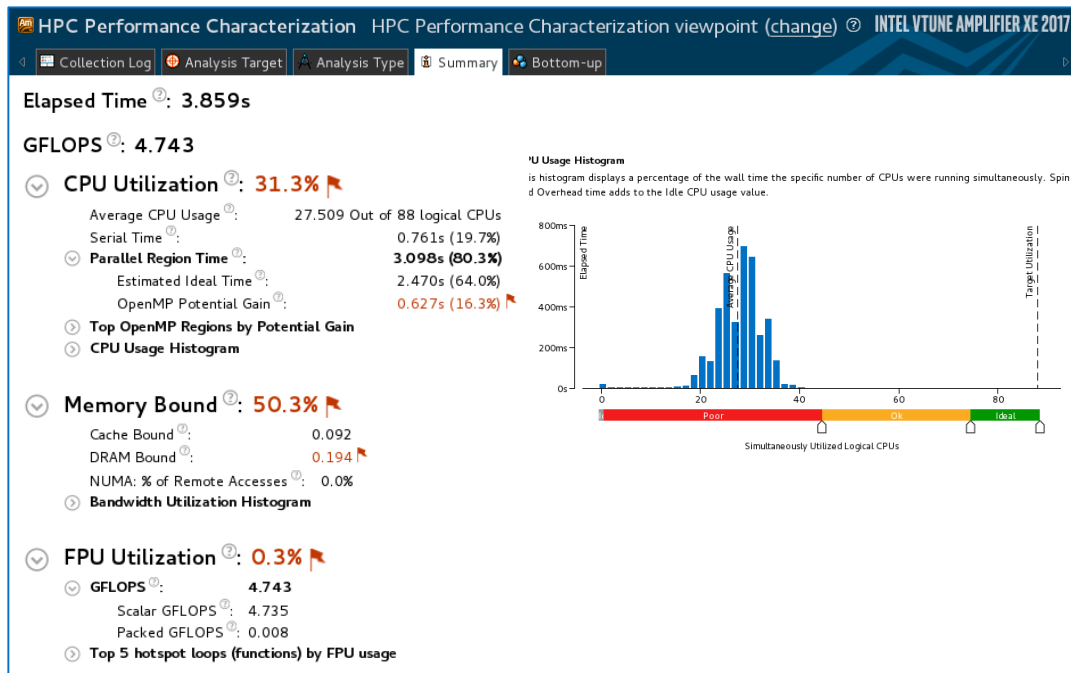
- Serial vs. Parallel time
- Top OpenMP regions by potential gain
- Tip: Use hotspot OpenMP region analysis for more detail

Memory Access Efficiency

- Stalls by memory hierarchy
- Bandwidth utilization
- Tip: Use Memory Access analysis

Vectorization: FPU Utilization

- FLOPS[†] estimates from sampling
- Tip: Use Intel Advisor for precise metrics and vectorization optimization



[†] For 3rd, 5th, 6th Generation Intel® Core™ processors and second generation Intel® Xeon Phi™ processor code named Knights Landing.

Optimize Memory Access

Memory Access Analysis - Intel® VTune™ Amplifier 2017

Tune data structures for performance

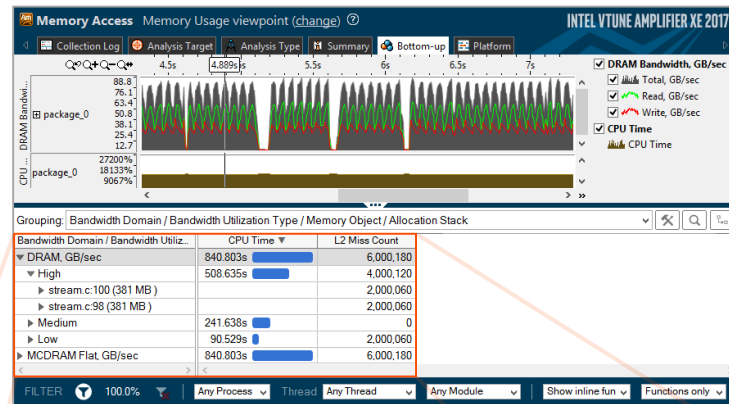
- Attribute cache misses to data structures (not just the code causing the miss)
- Support for custom memory allocators

Optimize NUMA latency & scalability

- True & false sharing optimization
- Auto detect max system bandwidth
- Easier tuning of inter-socket bandwidth

Easier install, latest processors

- No special drivers required on Linux*
- Intel® Xeon Phi™ processor MCDRAM (high bandwidth memory) analysis



| Bandwidth Domain / Bandwidth Utiliz... | CPU Time | L2 Miss Count |
|--|----------|---------------|
| ▼ DRAM, GB/sec | 840.803s | 6,000,180 |
| ▼ High | 508.635s | 4,000,120 |
| ▶ stream.c:100 (381 MB) | | 2,000,060 |
| ▶ stream.c:98 (381 MB) | | 2,000,060 |
| ▶ Medium | 241.638s | 0 |
| ▶ Low | 90.529s | 2,000,060 |
| ▶ MCDRAM Flat, GB/sec | 840.803s | 6,000,180 |

Storage Device Analysis (HDD, SATA or NVMe SSD)

Intel® VTune™ Amplifier

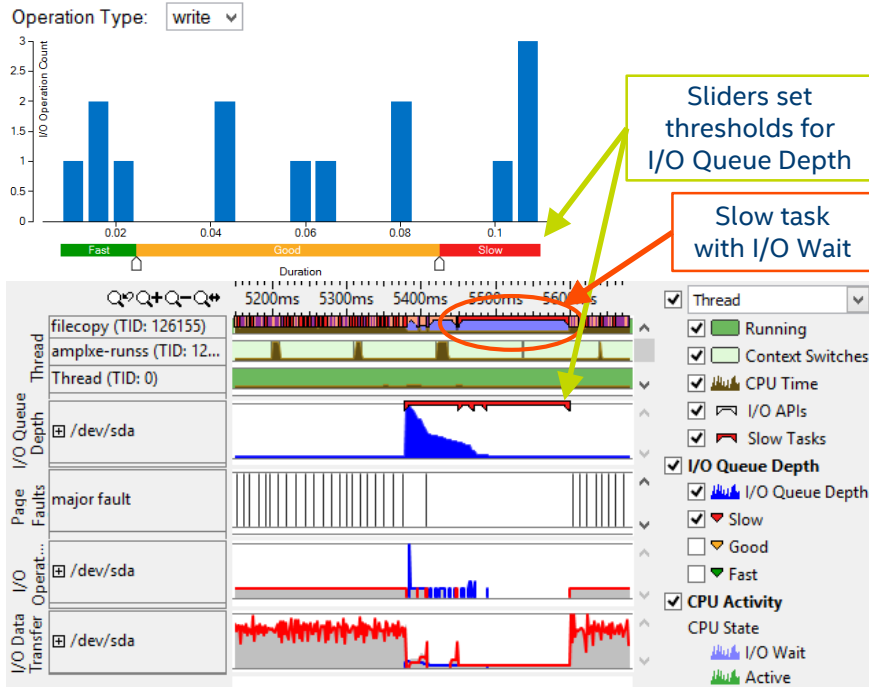
Are You I/O Bound or CPU Bound?

- Explore imbalance between I/O operations (async & sync) and compute
- Storage accesses mapped to the source code
- See when CPU is waiting for I/O
- Measure bus bandwidth to storage

Latency analysis

- Tune storage accesses with latency histogram
- Distribution of I/O over multiple devices

Disk Input and Output Histogram



Intel® Performance Snapshots

Three Fast Ways to Discover Untapped Performance

Is your application making good use of modern computer hardware?

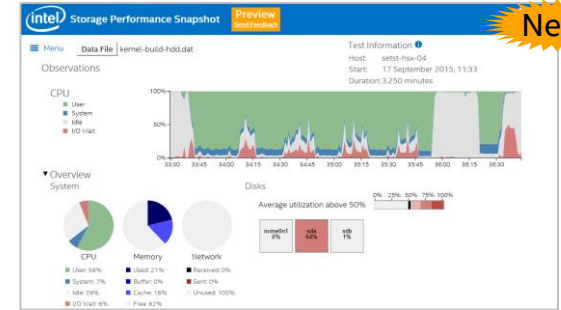
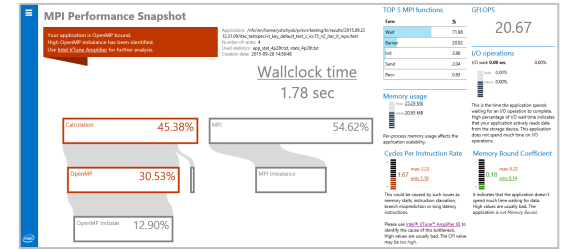
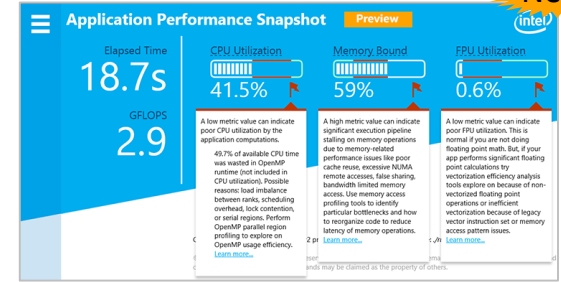
- Run a test case during your coffee break.
- High level summary shows which apps can benefit most from code modernization and faster storage.

Pick a Performance Snapshot:

- **Application** – for non-MPI apps
- **MPI** – for MPI apps
- **Storage** – for systems. Servers and workstations with directly attached storage.

Free download: <http://www.intel.com/performance-snapshot>

Also included with Intel® Parallel Studio and Intel® VTune™ Amplifier products.



Optimization Notice

Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



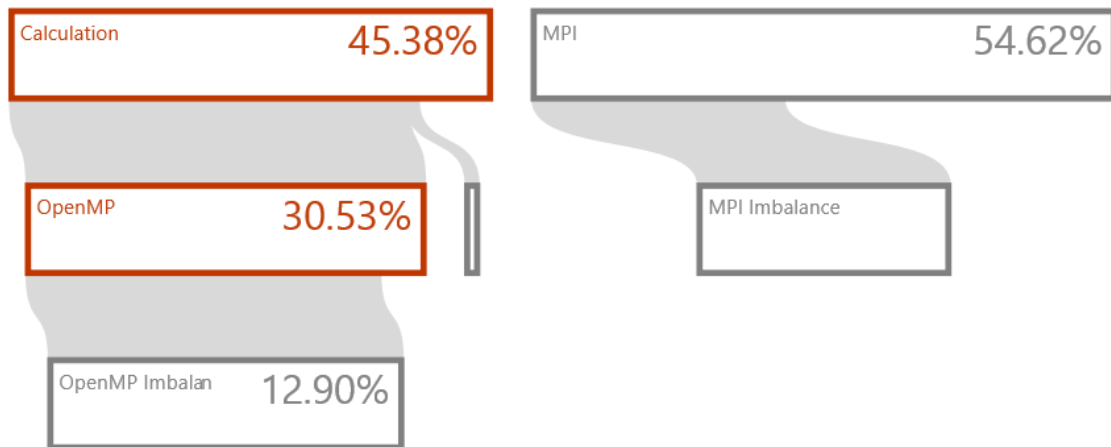
MPI Performance Snapshot

Your application is OpenMP bound.
High OpenMP imbalance has been identified.
Use [Intel VTune Amplifier](#) for further analysis.

Application: /nfs/inn/home/yshchyok/p/svn/testing/ts/results/2015.09.23
12.31.09/itac_testspec/vt_key_default_test_c_icc15_n2_itac_it_mps/test
Number of ranks: 4
Used statistics: app_stat_4p28t.txt, stats_4p28t.txt
Creation date: 2015-09-28 14:58:48

Wallclock time

1.78 sec



TOP 5 MPI functions

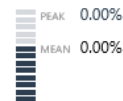
| Func | % |
|---------|-------|
| Wait | 71.98 |
| Barrier | 20.92 |
| Init | 3.98 |
| Send | 2.04 |
| Recv | 0.93 |

GFLOPS

20.67

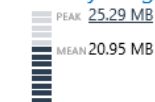
I/O operations

I/O wait: 0.00 sec 0.00%



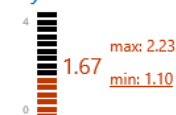
This is the time the application spends waiting for an I/O operation to complete. High percentage of I/O wait time indicates that your application actively reads data from the storage device. This application does not spend much time on I/O operations.

Memory usage



Per-process memory usage affects the application scalability.

Cycles Per Instruction Rate



This could be caused by such issues as memory stalls, instruction starvation, branch misprediction or long latency instructions.

Please use [Intel® VTune™ Amplifier XE](#) to identify the cause of this bottleneck. High values are usually bad. The CPI value may be too high.

Memory Bound Coefficient



It indicates that the application doesn't spend much time waiting for data. High values are usually bad. The application is *not* Memory Bound.

Free download: <http://www.intel.com/performance-snapshot>. Also included with Intel® Parallel Studio Cluster Edition.

Optimization Notice

Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



INTEL[®] ADVISOR XE

VECTORIZATION OPTIMIZATION AND THREAD PROTOTYPING FOR SOFTWARE ARCHITECTS

Get Faster Code Faster! Intel® Advisor

Vectorization Optimization

Have you:

- Recompiled for AVX2 with little gain
- Wondered where to vectorize?
- Recoded intrinsics for new arch.?
- Struggled with compiler reports?

Data Driven Vectorization:

New!

- What vectorization will pay off most?
- What's blocking vectorization? Why?
- Are my loops vector friendly?
- Will reorganizing data increase performance?
- Is it safe to just use pragma simd?

Where should I add vectorization and/or threading parallelism? Intel Advisor XE 2016

Summary Survey Report Refinement Reports Annotation Report Suitability Report

Elapsed time: 54.44s Vectorized Not Vectorized FILTER: All Modules All Sources

| Function Call Sites and Loop | Vector Issues | Self Time | Total Time | Trip Counts | Loop Type | Why No Vectorization? | Vectorized Loops |
|-------------------------------|-------------------------|-----------|------------|-------------|---------------|------------------------|---------------------|
| | | | | | | | Vecto... Efficiency |
| [loop at stl_algo.h:4740 i... | | 0.170s | 0.170s | | Scalar | non-vectorizable I ... | |
| [loop at loopstl.cpp:2449... | 2 Ineffective peeled... | 0.170s | 0.170s | 12; 4 | Collapse | Collapse | AVX ~100% |
| [loop at loopstl.cpp:2... | | 0.150s | 0.150s | 12 | Vectorized (B | | AVX |
| [loop at loopstl.cpp:2... | | 0.020s | 0.020s | 4 | Remainder | | |
| [loop at loopstl.cpp:7900... | | 0.170s | 0.170s | 500 | Scalar | vectorization possi... | |
| [loop at loopstl.cpp:35... | 1 High vector regi... | 0.160s | 0.160s | 12 | Expand | Expand | AVX ~69% |

"Intel® Advisor's Vectorization Advisor permitted me to focus my work where it really mattered. When you have only a limited amount of time to spend on optimization, it is invaluable."

Gilles Civario
Senior Software Architect
Irish Centre for High-End Computing

Faster Code Faster with Data Driven Design

Intel® Advisor – Vectorization Optimization and Thread Prototyping

Faster Vectorization Optimization:

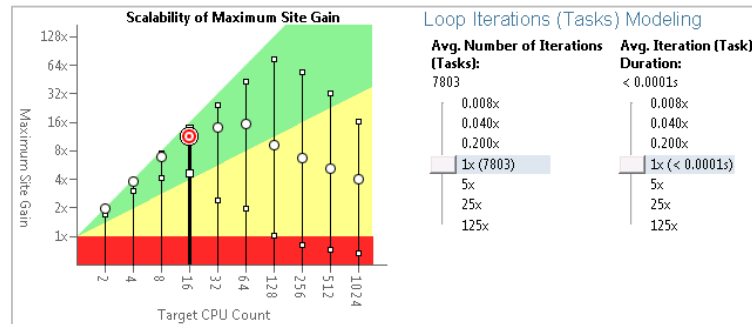
- Vectorize where it will pay off most
- Quickly ID what is blocking vectorization
- Tips for effective vectorization
- Safely force compiler vectorization
- Optimize memory stride

Breakthrough for Threading Design:

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

Less Effort, Less Risk and More Impact

| Function Call Sites and Loop | Vector Issues | Self Time | Total Time | Trip Counts | Loop Type | Why No Vectorization? | Vectorized Loops |
|-------------------------------|-----------------------|-----------|------------|-------------|---------------|------------------------|------------------|
| [loop at stl_algo.h:4740 i... | | 0.170s | 0.170s | 1 | Scalar | non-vectorizable l... | |
| [loop at loopstl.cpp:2449... | Ineffective peeled... | 0.170s | 0.170s | 12; 4 | Collapse | Collapse | AVX 100% |
| [loop at loopstl.cpp:2... | | 0.150s | 0.150s | 12 | Vectorized (B | | AVX |
| [loop at loopstl.cpp:2... | | 0.020s | 0.020s | 4 | Remainder | | |
| [loop at loopstl.cpp:7900... | | 0.170s | 0.170s | 500 | Scalar | vectorization possi... | |
| [loop at loopstl.cpp:35 ... | High vector regi... | 0.160s | 0.160s | 12 | Expand | Expand | AVX 69% |



Part of Intel® Parallel Studio for Windows* and Linux*

<http://intel.ly/advisor-xe>

Optimization Notice

Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Next Gen Intel® Xeon Phi™ Support

Vectorization Advisor runs on and optimizes for Intel® Xeon Phi

| Loops | Vector Issues | Self Time | Loop Type | Vectorized Loops | | | Instruction Set Analysis | | | |
|--------|------------------|-----------|-----------|--|------------|--------------|--------------------------|--------|------------------------------|--------------|
| | | | | Vector ISA | Efficiency | Gain Esti... | VL (V... | Traits | Data Types | |
| [Loop] | 3 Possible i... | 35.226s | 5.4% | Vectorized+Threaded (Body; Peeled; Re... | AVX512 | ~28% | 2.21x | 8 | Divisions; FMA; Gathers | Float32; ... |
| [Loop] | 2 Possible in... | 26.025s | 4.0% | Vectorized (Body)+Threaded (OpenMP) | AVX512 | | | 8 | Divisions; Gathers; FMA | Float32; ... |
| [Loop] | 1 High vecto... | 5.876s | | Vectorized (Peeled)+Threaded (OpenMP) | AVX512 | | | 8 | Divisions; Gathers; FMA | Float32; ... |
| [Loop] | 1 High vecto... | 3.324s | | Vectorized (Remainder)+Threaded (Open... | AVX512 | | | 8 | Divisions; Gathers; FMA | Float32; ... |
| [Loop] | | 34.599s | 5.3% | Vectorized (Body; Remainder) | AVX512 | ~70% | 5.64x | 8 | Divisions; FMA; Square Roots | Float32; ... |
| [Loop] | 1 Possible in... | 33.849s | 5.2% | Vectorized (Body; Peeled; Remainder) | AVX512 | ~28% | 2.24x | 8 | Divisions; FMA; Gathers | Float32; ... |
| [Loop] | | 19.839s | 3.1% | Vectorized (Body; Remainder) | AVX512 | 72% | 11.48x | 16; 8 | | Float32; ... |

AVX-512 ERI – specific to Intel® Xeon Phi

Efficiency (72%), Speed-up (11.5x), Vector Length (16)

Performance optimization problem and advice how to fix it

Source Top Down Loop Assembly Recommendations Computer Diagnostic Details

Issue: Possible inefficient memory access patterns present

Inefficient memory access patterns may result in significant vector code execution slowdown or block automatic vectorization by the compiler. Improve performance by investigating.

Recommendation: Confirm inefficient memory access patterns

There is no confirmation inefficient memory access patterns are present. To confirm: Run a [Memory Access Patterns analysis](#).

Confidence: Need More Data

Issue: Ineffective peeled/remainder loop(s) present

All or some [source loop](#) iterations are not executing in the [loop body](#). Improve performance by moving source loop iterations from [peeled/remainder](#) loops to the loop body.

Recommendation: Collect trip counts data

The Survey Report lacks [trip counts](#) data that might generate more precise recommendations. To fix: Run a [Trip Counts analysis](#).

Recommendation: Align data

Recommendation: Add data padding

The [trip count](#) is not a multiple of [vector length](#). To fix: Do one of the following:

- Increase the size of objects and add iterations so the trip count is a multiple of vector length.
- Increase the size of static and automatic objects, and use a compiler option to add data padding.

| Windows® OS | Linux® OS |
|---------------------------|---------------------------|
| /Oxpt-assume-safe-padding | -qopt-assume-safe-padding |

Program metrics

Elapsed Time: 142.79s

Vector Instruction Set: AVX, AVX2, AVX512, SSE, SSE2

Number of CPU Threads: 4

Loop metrics

| | | |
|------------------------------------|---------|--------|
| Total CPU time | 454.08s | 100.0% |
| Time in 88 vectorized loops | 41.86s | 9.2% |

Optimization Notice



Start Tuning for AVX-512 without AVX-512 hardware

Intel® Advisor - Vectorization Advisor

Use `-axCOMMON-AVX512 -xAVX` compiler flags to generate both code-paths

- AVX(2) code path (executed on Haswell and earlier processors)
- AVX-512 code path for newer hardware

Compare AVX and AVX-512 code with Intel Advisor

| Loops | Self Time | Loop Type | Vectorized Loops | | | | | Instruction Set Analysis | | | | Advanced |
|---|-----------|---------------------------------------|------------------|------------|---------|----------|----------------|--------------------------|-----------|-------------|------------------|--------------------|
| | | | Vect... | Efficiency | Gain... | VL (...) | Compiler Es... | Traits | Data T... | Vector W... | Instruction Sets | Vectorization D... |
| <input checked="" type="checkbox"/> [loop in s352_at loopstl.cpp:5939] | 0,641s | Vectorized (Body) | AVX2 | ~54% | 2,15x | 4 | 2,15x | FMA; Inserts | Float32 | 128 | AVX; FMA | |
| <input type="checkbox"/> [loop in s352_at loopstl.cpp:5939] | n/a | Remainder [Not Executed] | | | | 4 | | FMA | | | | |
| <input checked="" type="checkbox"/> [loop in s352_at loopstl.cpp:5939] | 0,641s | Vectorized (Body) | AVX2 | | | 4 | 2,15x | Inserts; FMA | | | | |
| <input type="checkbox"/> [loop in s352_at loopstl.cpp:5939] | n/a | Vectorized (Body) [Not Executed] | AVX512 | | | 16 | 3,20x | Gathers; FMA | | | | |
| <input type="checkbox"/> [loop in s352_at loopstl.cpp:5939] | n/a | Vectorized (Remainder) [Not Executed] | AVX512 | | | 16 | 2,70x | Gathers; FMA | | | | |
| <input checked="" type="checkbox"/> [loop in s125_ ASomp\$parallel_for@...] | 0,496s | Vectorized Versions | AVX2 | ~100% | 13,54x | 8 | <13,54x | FMA; NT-stores | | | | |
| <input type="checkbox"/> [loop in s125_ ASomp\$parallel_for@...] | n/a | Peeled [Not Executed] | | | | 8 | | FMA | | | | |
| <input type="checkbox"/> [loop in s125_ ASomp\$parallel_for@...] | n/a | Remainder [Not Executed] | | | | 8 | | FMA | | | | |
| <input checked="" type="checkbox"/> [loop in s125_ ASomp\$parallel_for@...] | 0,465s | Vectorized (Body) | AVX2 | | | 8 | 13,54x | | | | | |
| <input type="checkbox"/> [loop in s125_ ZSomp\$parallel_for@...] | n/a | Vectorized (Peeled) [Not Executed] | AVX512 | | | 16 | 6,77x | FMA | | | | |
| <input checked="" type="checkbox"/> [loop in s125_ ZSomp\$parallel_for@...] | n/a | Vectorized (Body) [Not Executed] | AVX512 | | | 32 | 30,61x | NT-stores | | | | |
| <input type="checkbox"/> [loop in s125_ ZSomp\$parallel_for@...] | n/a | Vectorized (Remainder) [Not Executed] | AVX512 | | | 16 | 9,78x | FMA | | | | |

Inserts (AVX2) vs. Gathers (AVX-512)

Speed-up estimate: 13.5x (AVX2) vs. 30.6x (AVX-512)

Optimization Notice



Precise Repeatable FLOPS Metrics

Intel® Advisor – Vectorization Optimization

- FLOPS by loop and function
- All recent Intel processors (not co-processors)
- Instrumentation (count FLOP) plus sampling (time with low overhead)
- Adjusted for masking with AVX-512 processors

| INTEL ADVISOR 2017 | | | | | | | | | |
|-------------------------------|---|-----------------------------------|---------|---------|---------|--------------------|-------|------------------------|-----|
| Function Call Sites and Loops | | FLOPS | | | | | | | |
| + | - | GFLOPS | AI | L1 GB/s | GFLOP | FLOP Per Iteration | L1 GB | L1 Bytes Per Iteration | |
| ⌵ | 🔄 | [loop in matvec at Multiply.c:69] | 0.826 0 | 0.1633 | 5.0586 | 3.0720 | 32 | 18.8160 | 196 |
| ⌵ | 🔄 | [loop in matvec at Multiply.c:60] | 0.912 0 | 0.1633 | 5.5853 | 3.0720 | 32 | 18.8160 | 196 |
| ⌵ | 🔄 | [loop in matvec at Multiply.c:69] | 1.248 0 | 0.2500 | 4.9920 | 1.3440 | 4 | 5.3760 | 16 |
| ⌵ | 🔄 | [loop in matvec at Multiply.c:60] | 1.592 0 | 0.2500 | 6.3699 | 1.3440 | 4 | 5.3760 | 16 |
| + | 🔄 | [loop in matvec at Multiply.c:69] | 3.055 0 | 0.2500 | 12.2205 | 0.0960 | 16 | 0.3840 | 64 |
| + | 🔄 | [loop in matvec at Multiply.c:60] | 6.282 0 | 0.2500 | 25.1279 | 0.0960 | 16 | 0.3840 | 64 |

Enhanced Memory Access Analysis

Are you bandwidth or compute limited?

Measure Footprint

- Compare to cache size
- Does it fit in cache?

Variable References

- Map data to variable names for easier analysis

Gather/Scatter

- Detect unneeded gather/scatters that reduce performance

| Site Location | Loop-Carried Dependencies | Strides Distribution ▲ | Access Pattern | Max. Site Footprint ▲ |
|--------------------------------------|---------------------------|------------------------|----------------|-----------------------|
| [loop in s4117_ at loopstl.cpp:76... | No information available | 50% / 50% / 0% | Mixed strides | 192B |
| [loop in s442_ at loopstl.cpp:6815] | No information available | 56% / 0% / 44% | Mixed strides | 256B |
| [loop in s272_ at loopstl.cpp:3447] | No information available | 60% / 0% / 40% | Mixed strides | 320B |

| ID | Stride | Type | Source | Nested Function | Variable references | Access Footprint | Mo |
|----|--------|---------------|------------------|-----------------|---------------------|------------------|------|
| P2 | | Gather stride | loopstl.cpp:3450 | | a, c, d | 320B | lcd_ |

```

3448     if (e[i_] >= *t)
3449     {
3450         a[i_] += c[i_] * d[i_];
3451         b[i_] += c[i_] * c[i_];
3452     }
    
```


| Line | Source | Stride |
|------|--|--------|
| 3450 | a[i_] += c[i_] * d[i_]; | [1] |
| 3451 | b[i_] += c[i_] * c[i_]; | |
| 3452 | } | |
| 3453 | dummy_ (ld, n, sa[1], sb[1], sc_[1], sd_ | |

| Address | Line | Assembly | Physical Stride |
|----------|------|---------------------------------------|-----------------|
| 0x43265a | 3450 | vgatherdpsz (%r8,%zmm0,4), %k1, %zmm2 | [1] |
| 0x432661 | 3403 | leaq (%r13,%rsi,1), %r8 | |
| 0x432666 | 3450 | vgatherdpsz (%r9,%zmm8,4), %k3, %zmm1 | [1] |

Details View

Gather (irregular) access

Operand Size (bits): 32
 Operand Type: bit*16,float32*16
 Vector Length: 16
 Memory access footprint: 320B

Gather/scatter details

Pattern: "Unit"

Instruction accesses values in contiguous memory throughout the loop:

- unit stride within instruction
- stride between iterations = vector length

Horizontal stride (bytes): 4
 Vertical stride (bytes): 64

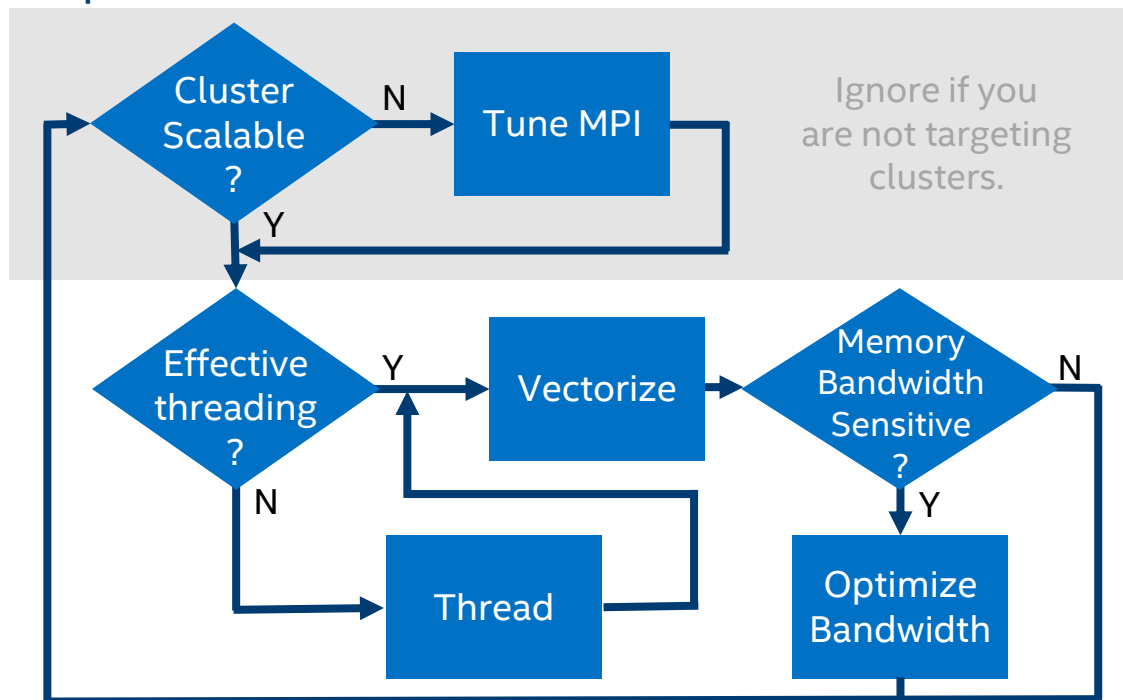


Speaker – the speaker notes are important for this presentation. Be sure to read them.

WHICH TOOL SHOULD I BE USING?

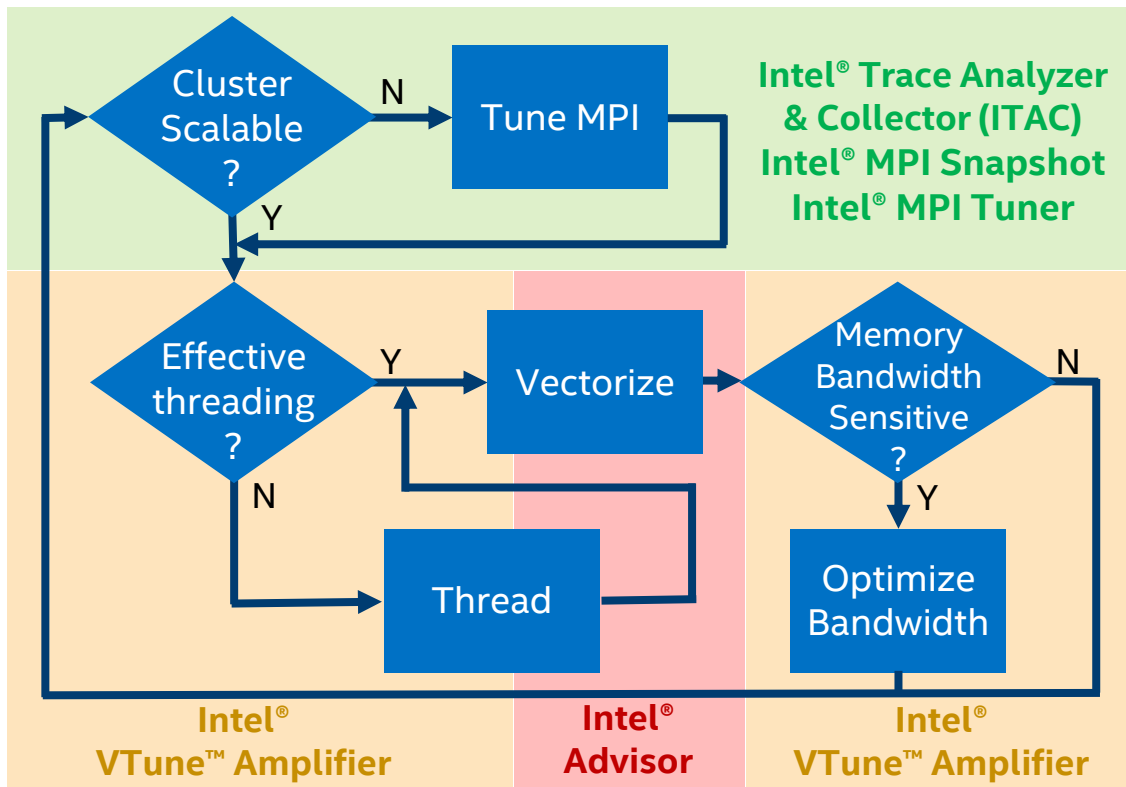
Optimizing Performance On Parallel Hardware

It's an iterative process...



Performance Analysis Tools for Diagnosis

Intel® Parallel Studio XE

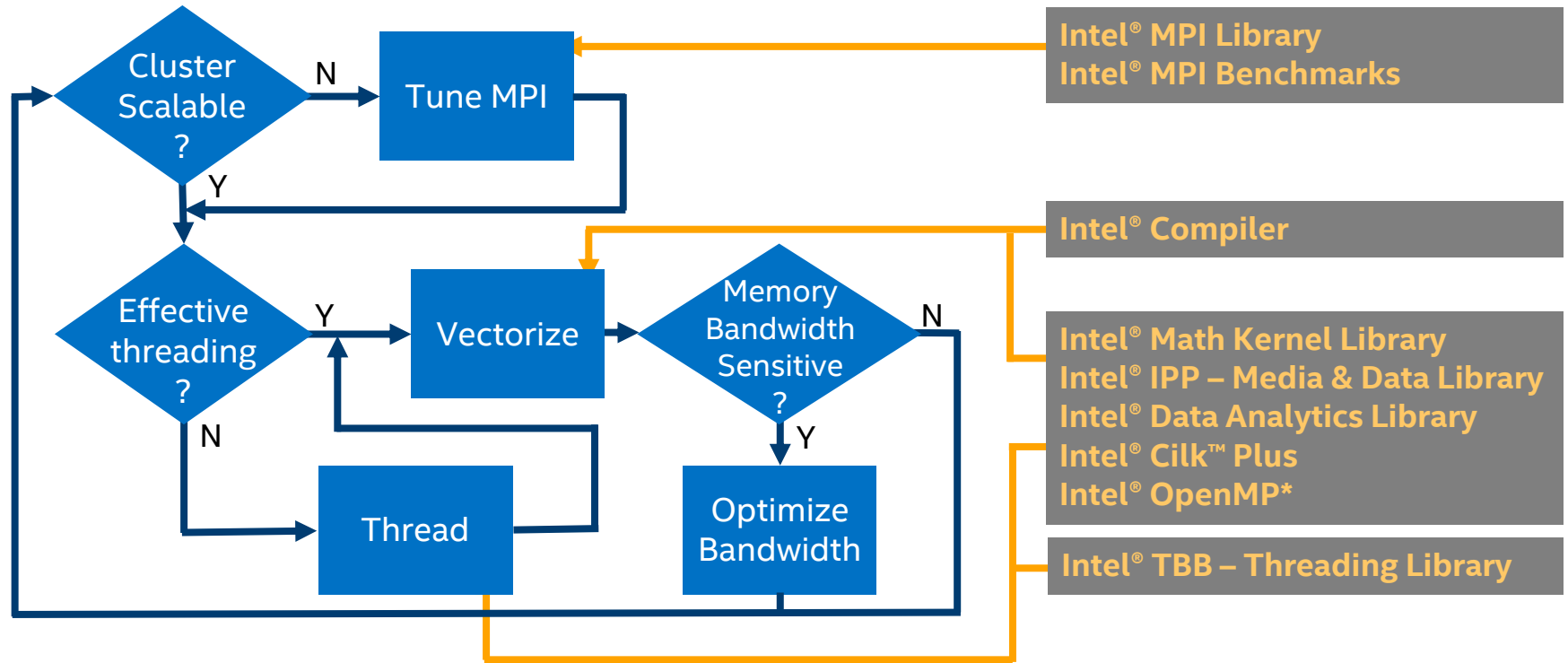


Optimization Notice

Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Tools for High Performance Implementation

Intel® Parallel Studio XE



Optimization Notice

Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



