INTEL® PARALLEL STUDIO XE
ANALYZERS AND CLUSTER TOOLS
OVERVIEW
For Distributed Performance
Agenda

Intel VTune Amplifier
- Overview, Predefined Analysis, VTune Platform Profiler, Demo

Intel Advisor
- Vectorization Advisor, Roofline, Demo

Intel Inspector
- Overview, Demo

Intel® MPI Library
- Basic Usage, Heterogeneous Jobs, Intel MPI Benchmarks, mpitune

Intel® Trace Analyzer and Collector
- Overview, Demo
Faster, Scalable Code, Faster

Intel® VTune™ Amplifier Performance Profiler

Accurate Data - Low Overhead
- CPU, GPU, FPU, threading, bandwidth...

Meaningful Analysis
- Threading, OpenMP region efficiency
- Memory access, storage device

Easy
- Data displayed on the source code
- Easy set-up, no special compiles

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

Claire Cates
Principal Developer
SAS Institute Inc.

http://intel.ly/vtune-amplifier-xe
# Two Great Ways to Collect Data

**Intel® VTune™ Amplifier**

<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*, KVM)</td>
</tr>
<tr>
<td>No driver required</td>
<td>Requires a driver - Easy to install on Windows - Linux requires root (or use default perf driver)</td>
</tr>
</tbody>
</table>

**No special recompiles - C, C++, C#, Fortran, Java, Assembly**
Find Answers Fast
Intel® VTune™ Amplifier

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

Scroll Bar “Heat Map” is an overview of hot spots
Click jump to scroll Asm
Timeline Visualizes Thread Behavior

Intel® VTune™ Amplifier

Optional: Use API to mark frames and user tasks
Optional: Add a mark during collection
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
VTune Predefined Analyses

Find your analysis direction

Hotspots
Want to find out where your application spends time and optimize your algorithms?

Microarchitecture
Want to see how efficiently your code is using the underlying hardware?

Parallelism
Want to assess the compute efficiency of your multi-threaded application?

Platform Analysis

- Platform Profiler (preview)
- CPU/GPU Concurrency
- GPU Compute/Media Hotspots
- Input and Output (preview)
- CPU/FPGA Interaction (preview)
VTune Command Line Analysis Configuration
How to Run VTune on MPI Applications

```bash
><mpi_launcher> – n N <vtune_command_line> ./app_to_run
  • >srun –n 48 -N 16 amplxe-cl –collect memory-access –trace-mpi –r result_dir ./my_mpi_app
  • >mpirun –n 48 -ppn 16 amplxe-cl –collect hotspots –r result_dir ./my_mpi_app

• Encapsulates ranks to per-node result directories suffixed with hostname
  • result_dir.hostname1 with 0-15, result_dir.hostname2 with 16-31, result_dir.hostname3 with 32-47

Add –trace-mpi option for VTune CL to enable per-node result directories for non-Intel MPIs

• Works for software and Intel driver-based collectors
```
VTune Command Line Analysis Configuration
Selective Rank Profiling

Superposition of application to launch and VTune command line for selective ranks to reduce trace size

Example: profile rank 1 from 0-15:
>`mpirun -n 1 ./my_app : -n 1 <vtune_command_line> -- ./my_app : -n 14 ./my_app`

• In the case of Intel MPI launcher –gtool option can be used:

Example: profile ranks 3, 7, 11-13 from 0-15:
>`mpirun –gtool “amplxe-cl –collect advanced-hotspots –r result_dir:3,7,11-13” ./my_app`
Finalization on KNL

Result finalization and viewing on KNL target might be slow

Use the recommended workflow:

1. Run collection on KNL deferring finalization to host:

   ```
   > amplxe-cl –collect memory-access –no-auto-finalize –r <my_result_dir> ./my_app
   ```

2. Finalize the result on the host

   - Provide search directories to the binaries of interest for resolving with –search-dir option

   ```
   > amplxe-cl –finalize –r <my_result_dir> –search-dir <my_binary_dir>
   ```

3. Generate reports, work with GUI

   ```
   > amplxe-cl –report hotspots –r <my_result_dir>
   ```
INTEL VTUNE™ AMPLIFIER – PLATFORM PROFILER
3-steps for optimizing complex workloads

Optimize
- How do I get the most out of my platform

Right-size
- What configuration changes should I make

Characterize
- What is my workload doing
- Where are the bottlenecks

system Focus
- Application Focus

Intel® VTune™ Amplifier
Configuration Matters!

5% performance drop when running from Socket 1 which is “far” from the storage device.
# The Long & Short of Performance Analysis

Get the big picture first with a Snapshot or Platform Profiler

## Snapshot

- Quickly size potential performance gain.
- Run a test “during a coffee break”.

<table>
<thead>
<tr>
<th>Application Focus</th>
<th>System Focus</th>
</tr>
</thead>
<tbody>
<tr>
<td>VTune Amplifier's Application Performance Snapshot</td>
<td>VTune Amplifier's Storage Performance Snapshot</td>
</tr>
</tbody>
</table>

## In-Depth

- Advanced collection & analysis.
- Insight for effective optimization.

<table>
<thead>
<tr>
<th>VTune Amplifier</th>
<th>Intel Advisor</th>
<th>ITAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Many profiles</td>
<td>Vectorization</td>
<td>MPI Optimization</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VTune Amplifier</th>
<th>- System-wide sampling</th>
<th>- Platform Profiler</th>
</tr>
</thead>
</table>

Maximum collection times:  
- **L** = long (hours)  
- **M** = medium (minutes)  
- **S** = short (seconds-few minutes)

---

Optimization Notice

Copyright © 2019, Intel Corporation. All rights reserved.  
*Other names and brands may be claimed as the property of others.*
Platform Profiler
Identify system performance & configuration issues and headroom

Target User
- Infrastructure Architects
- Software Architects & QA

Analyze performance
- Display current configuration
- ~150 hardware and ~20 OS metrics

Identify system configuration issues
- Inefficient memory module placements
- Need for faster storage, larger/faster memory

Identify potential software issues
- Inefficient CPU/storage/memory utilization
- Near vs. far memory accesses (NUMA)
Platform Profiler
Common use cases

Visualize workload behavior
- Very low overhead ~2%
- Long running workloads (hours)

Plan tuning strategy
- Which app needs it most?
- Which phase of the app?
- Optimize cache? I/O? Threading?

Optimize hardware configuration
- Add more I/O? Add memory?
Platform Profiler
Progressive disclosure methodology

Start with Overview and Topology

View different aspects of system

- Socket, Core, Memory, Disk, UPI, I/O

Drill down

- Socket → Core → Internal Caches
- Socket → Memory Link → Memory Module
Platform Profiler

Telemetry data collected (and displayed) at a granularity of 100ms

Analyze hours of execution

- No process-level info
- Average won’t hide erratic behavior

Select and zoom to desired execution region

- All graphs and averages adjust to selection
Platform Profiler

Data collection

Data Collector

Platform Profiler Analysis

Data Collection "Targets"
- Windows*
- Linux*

Analysis and visualization
- Web-based UI
- Works best with Chrome
INTEL ® ADVISOR
Vectorization of code

Transform sequential code to exploit vector processing capabilities (SIMD) of Intel processors

- Manually by explicit syntax
- Automatically by tools like a compiler

```
for(i = 0; i <= MAX; i++)
c[i] = a[i] + b[i];
```
Vectorization terms

Vector lanes

Vector length (VL): Elements of the vector

All elements of vector are of the same data types
Many Ways To Vectorize

- **Compiler:** Auto-vectorization (no change of code)
- **Compiler:** Auto-vectorization hints (`#pragma vector, ...`)
- **Compiler:** OpenMP* 4.0

- **SIMD intrinsic class**
  (e.g.: `F32vec`, `F64vec`, ...)

- **Vector intrinsic**
  (e.g.: `_mm_fmadd_pd(...), _mm_add_ps(...), ...`)

- **Assembler code**
  (e.g.: `[v]addps, [v]addss, ...`)

---

Ease of use

Programmer control

---
Both Skylake and Knights Landing processors have support for Intel® AVX-512 instructions. There are three ISA options in the Intel® Compiler:

- `xCORE-AVX512` : Targets Skylake, contains instructions not supported by Knights Landing
- `xCOMMON-AVX512` : Targets both Skylake and Knights Landing
- `xMIC-AVX512` : Targets Knights Landing, includes instructions not supported by Skylake

Intel® Compiler is conservative in its use of ZMM (512bit) registers so to enable their use with Skylake the additional flag `-qopt-zmm-usage=high` must be set.
Validating Vectorization Success I: Compiler report

- `qopt-report[]=n`: tells the compiler to generate an optimization report
  - `n`: (Optional) Indicates the level of detail in the report. You can specify values 0 through 5. If you specify zero, no report is generated. For levels n=1 through n=5, each level includes all the information of the previous level, as well as potentially some additional information. Level 5 produces the greatest level of detail. If you do not specify n, the default is level 2, which produces a medium level of detail.

- `qopt-report-phase[]=list`: specifies one or more optimizer phases for which optimization reports are generated.
  - `loop`: the phase for loop nest optimization
  - `vec`: the phase for vectorization
  - `par`: the phase for auto-parallelization
  - `all`: all optimizer phases

- `qopt-report-filter=string`: specified the indicated parts of your application, and generate optimization reports for those parts of your application.
Validating Vectorization Success II

- **-S**: assembler code inspection
  - Most reliable way and gives all details of course
  - Check for scalar/packed or (E)VEX encoded instructions:
    Assembler listing contains source line numbers for easier navigation
  - Compiling with `-qopt-report-embed` (Linux*, macOS*) helps interpret assembly code

- **Performance validation**
  - Compile and benchmark with `-no-vec -qno-openmp-simd` or on a loop by loop basis via
    `#pragma novector` or `#DIR$ NOVECTOR`
  - Compile and benchmark with selected SIMD feature
  - Compare runtime differences
Intel® Advisor
Boosting your application by threading and vectorization

Compiler will not always vectorize
- Check for Loop Carried Dependencies using Intel® Advisor
- All clear? Force vectorization. C++ use: pragma simd, Fortran use: SIMD directive

Not all vectorization is efficient vectorization
- Stride of 1 is more cache efficient than stride of 2 and greater. Analyze with Intel® Advisor.
- Consider data layout changes Intel® SIMD Data Layout Templates can help

Arrays of structures are great for intuitively organizing data, but are much less efficient than structures of arrays. Use the Intel® SIMD Data Layout Templates (Intel® SDLT) to map data into a more efficient layout for vectorization.
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:
- 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?

"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
4 Steps to Efficient Vectorization

Intel® Advisor – Vectorization Advisor

1. Compiler diagnostics + Performance Data + Roofline Model + SIMD efficiency information

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

3. Loop-Carried Dependency Analysis

4. Memory Access Patterns Analysis
Integrated Roofline model

In the Intel® Advisor Integrated Roofline chart the Arithmetic Intensity and memory traffic for each level of the memory hierarchy is represented separately.

You can visualize the levels that need further optimization.
Roofline compare

Visualize multiple roofline charts on the same graph.

Test optimization strategies and see how much progress your are making.
New and Improved Summary

Program metrics
- Elapsed Time: 7.71s
- Vector Instruction Set: AVX2, AVX
- Number of CPU Threads: 36

Performance characteristics
- Total CPU time
- Time in 2 vectorized loops
- Time in scalar code

Vectorization Gain/Efficiency
- Vectorized Loops Gain/Efficiency
- Program Approximate Gain

Effective OP/S and Bandwidth
- GFLOPS: 61.89
- GINTOPS: 1.29
- CPU <-> Memory [L1+NTS GB/s]: 203.4
- L2 Bandwidth [GB/s]: 105.9
- L3 Bandwidth [GB/s]: 69.07
- DRAM Bandwidth [GB/s]: 14.88

Informations on Operations and Memory transfers

Overall metrics
- GFLOP Count: 477,428
- FP Arithmetic Intensity: 0.30422
- GINTOPS: 1.29

Ratio of vectorized/unvectorized code
- Total: 100%
- 167.59s: 65.3%
- 109.30s: 34.7%
- 58.20s: 80%
- 6.87x
- 4.83x
# Column Configurator

**Customize view**

![Column Configurator Screenshot]

The Column Configurator is a tool for optimizing and visualizing computational performance. It allows users to customize the view of performance metrics, such as CPU time, vectorized loops, and compute performance metrics. The tool provides detailed insights into function call sites and loops, helping users identify inefficiencies and improve overall performance.

### Performance Issues

<table>
<thead>
<tr>
<th>Function Call Sites and Loops</th>
<th>Performance Issues</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Loop in main at Driver.c171]</td>
<td>1 Possible inefficiency</td>
</tr>
<tr>
<td>![Loop in main at Driver.c158]</td>
<td>1 Possible inefficiency</td>
</tr>
<tr>
<td>![Loop in main at Driver.c164]</td>
<td>1 Potential underflow</td>
</tr>
<tr>
<td>![Loop in main at Driver.c155]</td>
<td>1 Data type conversion</td>
</tr>
<tr>
<td>![sct_common_main seh]</td>
<td>1 Data type conversion</td>
</tr>
<tr>
<td>![printf]</td>
<td>n/a</td>
</tr>
<tr>
<td>![Loop in main at Driver.c63]</td>
<td>n/a</td>
</tr>
</tbody>
</table>

### CPU Time

- **Self Time**: Represents the time spent on the current function or loop, excluding any time spent in external functions.
- **Total Time**: Represents the time spent on the current function or loop, including any time spent in external functions.

### Vectorized Loops

- **Vectorized Loops** section includes details on vectorized loops such as Vectorized (Body), Vectorized (Function), Scalar, and inner loop already vectorize.
- **AVX2** and **AVX** indicate the vectorization technology used.
- **Gain Es**: Represents the gain in efficiency when vectorization is applied.
- **VL (Vec)**: Represents the vector length.
- **Self GFLOPS** and **Self Al**: Represent the self-generate floating-point operations per second and self-accumulated operations, respectively.

### Compute Performance

- **Top 5.00%**: Indicates the top 5% of performance metrics.
- **View Layout**: Default view configuration is shown.

The Column Configurator is a powerful tool for optimizing computational performance, offering detailed insights and customizable views to identify and improve inefficiencies.
Visualize Parallelism—Interactively Build, Validate & Analyze Algorithms

Intel® Advisor—Flow Graph Analyzer (FGA)

Design mode

- Allows you to create a graph topology interactively
- Validate the graph and explore what-if scenarios
- Add C/C++ code to the node body
- Export C++ code using Threading Building Blocks (TBB) flow graph API

Analysis mode

- Compile your application (with tracing enabled)
- Capture execution traces during the application run
- Visualize/analyze in Flow Graph Analyzer

Use Intel® TBB or OpenMP* 5 (draft) OMPT APIs
Intel® Advisor – Flow Graph Analyzer (Analysis mode)

- Trace program execution
- Show correlation
- Trace Playback
How to set it up (command line) ?

Before running an analysis, run:
- $ export ADVIXE_EXPERIMENTAL=int_roofline
- Use –integrated option

Run the survey
- advixe-cl -collect survey -integrated ........ -- ./my_application param1 param2 ...

Run the trip count and flop
- advixe-cl -collect tripcounts -flop -integrated -enable-cache-simulation ........ -- ./my_application param1 param2 ...
- Or Run Roofline analysis
- advixe-cl -collect roofline -integrated... -- ./my_application param1 param2
How to display the Integrated Roofline chart

You can select which memory level you want to display. Each memory level selected display an additional dot for every kernel. Each dot of the same kernel has the same performance but different Arithmetic Intensities. Here we selected CARM, L2, L3 and DRAM.
INTEL® INSPECTOR
Memory and thread debugger
Motivation for Intel® Inspector

Memory Errors

- Invalid Accesses
- Memory Leaks
- Uninitialized Memory Accesses

Threading Errors

- Data Races
- Deadlocks
- Cross Stack References

Multi-threading problems

- Hard to reproduce,
- Difficult to debug
- Expensive to fix

Let the tool do it for you
Intel® Inspector: Dynamic Analysis

Inspection scope

Dynamic binary instrumentation

Inspector Collector

Results

Application process

Binary

Source

Optimization Notice

Copyright © 2019, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.
How it Looks: Visual Studio* Integration

- **Run analysis from toolbar**
- **Problems found:** memory leaks
- **Memory allocation site in source code**
- **Call stack**
- **Choose existing project, no special configuration**
Standalone GUI: Windows* and Linux*
Workflow: manage results

- Code locations grouped into Problems to simplify results management
- Double click on Problem to navigate to source
- Powerful filtration feature
Workflow: navigate to sources

Call stacks

Problematic line in source code

All code locations for a problem

Switch to disassembly for more details
Workflow: timeline view

Individual Code Locations are seen in Timeline view in the context of their respective threads

Hover gives details
Analyze Memory Growth

During Analysis:

- Set Start Point
- Set End Point

Analysis Results:

- Memory Growth Problem Set
- Code location for each block of memory that was allocated but not de-allocated during the time period
On-demand leak detection

- Check code regions between points 'A' and 'B' for leaks
- Check daemon processes for leaks
- Check crashing processes for leaks

Analysis Results:
- Memory Leak shown during runtime

Set Start Point

Set End Point
Define analysis scope in source code

```c
#include <ittnotify.h>

void ProcessPipeline()
{
    __itt_heap_reset_detection(__itt_heap_leaks);  // Start measuring memory leaks
    pipeline_stage1();  // Run pipeline stage 1
    __itt_heap_record(__itt_heap_leaks);  // Report leaks in stage 1

    DoSomeOtherWork();

    __itt_heap_reset_detection(__itt_heap_growth);  // Start measuring memory growth
    pipeline_stage2();  // Run pipeline stage 2
    __itt_heap_record(__itt_heap_growth);  // Report memory growth in stage 2
}
```
Correctness analyses overhead

Inspector XE tracks
- Thread and Sync APIs
- Memory accesses

Inspector performs binary instrumentation using PIN
- Dynamic instrumentation system provided by Intel
  (http://www.pintool.org)
- Injected code used for observing the behavior of the running process
- Source modification/recompilation is not needed

Increases execution time and memory consumed (potentially significantly)

The Inspector XE dilates both time and memory consumed significantly!
Workload guidelines

Use small data set
- Smaller number of threads
- Minimize data set size (e.g. smaller image sizes)
- Minimize loop iterations or time steps
- Minimize update rates (e.g. lower frames per second)

Use small but representative data set
- Only **actually executed** code paths are analyzed

Scale down workload to speed up analysis!
Include and Exclude modules

1. There are two options:
   - Include modules of interest
   - Exclude unnecessary modules

2. Press Modify

3. Choose modules you want to include or exclude from analysis
Debugger integration

Break into debugger

- Analysis can stop when it detects a problem
- User is put into a standard debugging session

Windows*
- Microsoft* Visual Studio Debugger

Linux*
- gdb
Debug this problem

Right click on a problem

Inspector will set breakpoint, and launch debug session at the place of the problem occurrence

Powerful debugger integration
Debug this problem

Problematic code location with context values

Inspector XE problem context

Local variable values
Collect results and create baseline

```bash
inspxe-cl -collect mi1 -r r002mi1 -- D:\tests\my_app.exe

inspxe-cl -collect mi1
-module-filter module1.dll,module2.dll -module-filter-
    mode exclude -- D:\tests\my_app.exe

inspxe-cl -collect mi1 -executable-of-interest
    mem_error.exe -- D:\tests\startup_script.bat

inspxe-cl -create-suppression-file "D:\tests\mySup"
    -result-dir r002mi1

inspxe-cl -collect mi1 -suppression-file "D:\tests\mySup"
    -- D:\tests\my_app.exe

inspxe-cl -collect mi1 -baseline-result mi1_base --
    D:\tests\my_app.exe
```
Intel Inspector: User APIs

Enable you to

- Control collection, limit analysis scope
- Specify non-standard synchronization primitives
- Specify custom memory allocation primitives

To use user APIs:

- Include `ittnotify.h`, located at `<install_dir>/include`
- Insert `__itt_*` notifications in your code
- Link to the `libittnotify.lib` file located at `<install_dir>/ <lib32|lib64`
- Available for C/C++ and Fortran
## Collection control APIs

<table>
<thead>
<tr>
<th>API</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void __itt_suppress_push(unsigned int etype)</code></td>
<td>Stop analyzing for errors on the current thread</td>
</tr>
<tr>
<td><code>void __itt_suppress_pop (void)</code></td>
<td>Resume analysis</td>
</tr>
<tr>
<td><code>void __itt_suppress_mark_range ( __itt_suppress_mode_t mode, unsigned int etype, void * address, size_t size);</code></td>
<td>Suppress or unsuppress error detection for the specific memory range (object).</td>
</tr>
<tr>
<td><code>void __itt_suppress_clear_range ( __itt_suppress_mode_t mode, unsigned int etype, void * address, size_t size);</code></td>
<td>Clear the marked memory range</td>
</tr>
</tbody>
</table>
Using the Intel® Inspector with MPI

Compile the `inspector_example.c` code with the MPI scripts

Use the command-line tool under the MPI run scripts to gather report data

```
mpirun -n 4 inspxe-cl --result-dir insp_results
   -collect mi1 -- ./insp_example.exe
```

Output is: a results directory for each MPI rank in the job

```
ls | grep inspector_results on Linux
```

Launch the GUI and view the results for each particular rank

```
inspxe-gui inspector_results.<rank#> on Linux
```
INTEL® PARALLEL STUDIO XE CLUSTER EDITION OVERVIEW
For Distributed Performance
Intel® Parallel Studio XE

Optimization Notice
Copyright © 2019, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
Standards Based Optimized MPI Library for Distributed Computing

- Built on open source MPICH Implementation
- Tuned for low latency, high bandwidth & scalability
- Multi-fabric support for flexibility in deployment

What’s New in 2019 Release

- New MPI code base- MPI-CH4 (on the path to Exascale & beyond)
- Greater scalability & shortened CPU paths
- Superior MPI Multi-threaded performance
- Supports the latest Intel® Xeon® Scalable processor

Learn More: software.intel.com/intel-mpi-library
Superior MPI Performance with Intel® MPI Library 2019 on Linux* 64
1,280 Processes, 32 Xeon nodes (Intel® Omni-Path) Linux* 64

Performance results are based on testing as of Sept. 5, 2018 and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure. 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 complete information, see Performance Benchmark Test Disclosure.

Configuration: Testing by Intel as of Sept. 5, 2018. Hardware: Intel® Xeon®Gold 6148 CPU @ 2.40GHz; 192 GB RAM. Interconnect: Intel® Omni-Path Host Fabric Interface Software: RHEL* 7.4; IFS 10.7.0.0.145; Libfabric internal; Intel® MPI Library 2019; Intel® MPI Benchmarks 2019 (built with Intel® C++ Compiler XE 18.0.2.199 for Linux*);

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.

For more complete information about compiler optimizations, see our Optimization Notice.
Command Line Argument Set

May set both global and local variables for each host

No limit to number of different host or executables

For high numbers of hosts a configuration file is more convenient...

$ mpirun --genv OMP_NUM_THREADS 4 -n 6 -host node1 ./exe1 -n 4 -host node2 ./exe2:
- n 6 -host node4 ./exe4
Configuration File

Configuration file allows flexibility and automation

Notice commented out line – simple to change host assignment

```
$ cat theconfigfile
-genv OMP_NUM_THREADS 4
-n 6 –host node1 ./exe1
-n 4 –host node2 ./exe2
# -n 4 –host dead_node3 ./exe3
-n 6 –host node4 ./exe4
```

Launching job is straightforward

```
$ mpirun –configfile theconfigfile
```
Understanding MPI and Launcher Behavior

I_MPI_DEBUG=<level>

Debug Levels (cumulative):

- 0 – Default, no debug information
- 1 – Verbose error diagnostics
- 2 – Fabric selection process
- 3 – Rank, PID, node mapping
- 4 – Process pinning [recommended]
- 5 – Display Intel® MPI Library environment variables
- 6 – Collective operation algorithm controls

I_MPI_HYDRA_DEBUG=1

Turns on Hydra debug output

- Extremely verbose output
- Only turn on if needed
Fabric Selection (v2018 & earlier)

I_MPI_FABRICS=<intranode fabric>:<internode fabric> or <fabric>

- shm – Shared Memory (only valid for intranode)
- dapl – Direct Access Provider Library*
- ofa – Open Fabric Alliance (OFED* verbs)
- tmi – Tag Matching Interface
- tcp – Ethernet/Sockets
- ofi – OpenFabrics Interfaces*

Default behavior goes through a list to find first working fabric combination

If you specify a fabric, fallback is disabled, I_MPI_FALLBACK=1 to re-enable
Fabric Selection (v2018 vs v2019)

I_MPI_FABRICS=<intranode fabric>:<internode fabric> or <fabric>

- ofi – OpenFabrics Interfaces*

Default behavior goes through a list to find first working fabric combination
If you specify a fabric, fallback is disabled, I_MPI_FALLBACK=1 to re-enable
Intel® MPI Support of Hybrid Codes

Intel® MPI is strong in mapping and pinning support for MPI processes

Sophisticated defaults or user controlled:

- For pure MPI codes use `I_MPI_PIN_PROCESSOR_LIST`
- For hybrid codes (default, takes precedence over `I_MPI_PIN_PROCESSOR_LIST`):

  
  \[
  \text{\texttt{I_MPI_PIN_DOMAIN}} = \langle \text{size}\rangle[:\langle \text{layout}\rangle]
  \]

  \[
  \begin{align*}
  \langle \text{size}\rangle &= \text{omp} & \text{Adjust to OMP_NUM_THREADS} \\
  & = \text{auto} & \text{#CPUs/#MPIprocs (default)} \\
  & = \langle n\rangle & \text{Number}
  \\
  \langle \text{layout}\rangle &= \text{platform} & \text{According to BIOS numbering} \\
  & = \text{compact} & \text{Close to each other} \\
  & = \text{scatter} & \text{Far away from each other}
  \end{align*}
  \]

  Defines mapping and pinning for MPI processes, leaves room for threads on remaining cores!

NB: Naturally extends to hybrid codes on Intel® Xeon Phi™
$ export I_MPI_DEBUG=4
$ mpirun -machinefile .hosts.txt -n 8 ./mpi_hello

[0] MPI_startup(): Multi-threaded optimized library
[0] MPID_nem_ofi_init(): used OFI provider: psm2
...
[0] MPI_startup(): shm and ofi data transfer modes
...
[0] MPI_startup(): Rank Pid Node name Pin cpu
[0] MPI_startup(): 0 121023 node0 {0,1,2,3,4,5,6,7,8,9,40,41,42,43,44,45,46,47,48,49}
[0] MPI_startup(): 1 121024 node0 {10,11,12,13,14,15,16,17,18,19,50,51,52,53,54,55,56,57,58,59}
[0] MPI_startup(): 2 121025 node0 {20,21,22,23,24,25,26,27,28,29,60,61,62,63,64,65,66,67,68,69}
[0] MPI_startup(): 3 121026 node0 {30,31,32,33,34,35,36,37,38,39,70,71,72,73,74,75,76,77,78,79}
[0] MPI_startup(): 4 246334 node1 {0,1,2,3,4,5,6,7,8,9,40,41,42,43,44,45,46,47,48,49}
[0] MPI_startup(): 5 246335 node1 {10,11,12,13,14,15,16,17,18,19,50,51,52,53,54,55,56,57,58,59}
[0] MPI_startup(): 6 246336 node1 {20,21,22,23,24,25,26,27,28,29,60,61,62,63,64,65,66,67,68,69}
[0] MPI_startup(): 7 246337 node1 {30,31,32,33,34,35,36,37,38,39,70,71,72,73,74,75,76,77,78,79}

Hi from MPI task 0
...

Default Binding

Fabric provider

Active transfer modes

Equal distribution of cores among MPI ranks
Using I_MPI_PIN_PROCESSOR_LIST

$ export I_MPI_DEBUG=4
$ export I_MPI_PIN_PROCESSOR_LIST=0,1,20,21
$ mpirun -machinefile ./hosts.txt -n 8 ./mpi_hello

[0] MPI startup(): Multi-threaded optimized library
[0] MPID_nem_ofi_init(): used OFI provider: psm2
...[0] MPI startup(): shm and ofi data transfer modes
...
[0] MPI startup(): Rank  Pid    Node name  Pin  cpu
[0] MPI startup(): 0  121023 node0   {0}
[0] MPI startup(): 1  121024 node0   {1}
[0] MPI startup(): 2  121025 node0   {20}
[0] MPI startup(): 3  121026 node0   {21}
[0] MPI startup(): 4  246334 node1   {0}
[0] MPI startup(): 5  246335 node1   {1}
[0] MPI startup(): 6  246336 node1   {20}
[0] MPI startup(): 7  246337 node1   {21}

Hi from MPI task 0
...
Using I_MPI_PIN_DOMAIN

Set binding to OMP range

Each MPI task floats on OMP_NUM_THREADS logical processors

Careful in HT systems!

Hi from MPI task 0 OMP thread 0
Intel® MPI Benchmarks 2019

Standard benchmarks with OSI-compatible CPL license

- Enables testing of interconnects, systems, and MPI implementations
- Comprehensive set of MPI kernels that provide performance measurements for:
  - Point-to-point message-passing
  - Global data movement and computation routines
  - One-sided communications
  - File I/O
  - Supports MPI-1.x, MPI-2.x, and MPI-3.x standards

What’s New:

Introduction of new benchmarks

- Added benchmarks to test new multi-threaded support (IMB-MT) and shared memory transport (IMB-P2P)
Measuring Communication Speed with IMB

The simplest benchmark in IMB is called PingPong.

Data packages of different size are sent from rank 0 to rank 1 and back:

```
$ mpirun -n 2 IMB-MPI1 pingpong
```
What is mpitune?

Tool to optimize Intel® MPI Library settings

Walks through [application specific] search space of settings and tests performance

Writes out a configuration file with the best settings found

Works in two different modes:

- Cluster Specific
- Application Specific
Cluster Specific Tuning with mpitune

Find optimal values for library tuning knobs on the particular cluster or application environment.

- Run it once after installation and each time after a cluster configuration change
- Best configuration is recorded for each combination of communication device, number of nodes, MPI ranks and the process distribution model
- Configuration is stored in Intel® MPI folders and available to all users

Collect configuration values:
$ mpitune [options]

Reuse recorded values:
$ mpirun –tune ./application
Application Specific Tuning with mpitune

Find optimal values for library tuning knobs on the particular cluster or application environment

- Run it for each application and after application- or cluster configuration change
- Best configuration is recorded for each combination of communication device, number of nodes, MPI ranks and the process distribution model
- Configuration is stored in user’s home

Collect configuration values:

```
$ mpitune [options] \
   --application "mpirun application"
```

Reuse recorded values:

```
$ mpirun --tune ./app.conf \
   ./application
```
Example: Cluster Specific MPITUNE

MPITUNE is an executable script. The easiest way is to simply run:

```
$ mpitune
```

We may restrict MPITUNE on full nodes and the default fabric

```
$ mpitune -pr 8:8 -fl shm:dapl
```

hosts should be taken from provided hostfile or the batch system

As the search space is very wide, you may want to limit the number of minimum and maximum ranks per node as well as the number of nodes and the fabric used:

```
$ mpitune -pr <min_ppn>:<max_ppn> \ 
  -hr <min_nodes>:<max_nodes> \ 
  -fl <fabric> \ 
  -a "mpirun ...
```
**MPITUNE – Output**

<table>
<thead>
<tr>
<th>Message size (bytes)</th>
<th>Initial time</th>
<th>Tuned vs initial (ex)</th>
<th>Validated time</th>
<th>Validated vs initial (ex)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Alg 1</td>
<td>Alg 2</td>
<td>Alg 3</td>
<td>Alg 4</td>
</tr>
<tr>
<td>0</td>
<td>0.38</td>
<td>0.40</td>
<td>0.35*</td>
<td>0.38</td>
</tr>
<tr>
<td>4</td>
<td>3.46</td>
<td>51.77</td>
<td>13.81</td>
<td>3.45</td>
</tr>
<tr>
<td>8</td>
<td>4.05</td>
<td>52.01</td>
<td>13.78</td>
<td>*4.14</td>
</tr>
<tr>
<td>16</td>
<td>4.00</td>
<td>52.04</td>
<td>13.98</td>
<td>4.47</td>
</tr>
<tr>
<td>32</td>
<td>4.46</td>
<td>52.54</td>
<td>15.25</td>
<td>5.54</td>
</tr>
<tr>
<td>64</td>
<td>4.85</td>
<td>52.88</td>
<td>16.29</td>
<td>8.22</td>
</tr>
<tr>
<td>128</td>
<td>5.16</td>
<td>52.98</td>
<td>17.29</td>
<td>11.47</td>
</tr>
<tr>
<td>256</td>
<td>5.77</td>
<td>54.86</td>
<td>8.04</td>
<td>25.84</td>
</tr>
<tr>
<td>512</td>
<td>6.07</td>
<td>54.89</td>
<td>8.42</td>
<td>26.49</td>
</tr>
<tr>
<td>1024</td>
<td>9.14</td>
<td>56.15</td>
<td>8.66*</td>
<td>27.60</td>
</tr>
<tr>
<td>2048</td>
<td>11.03</td>
<td>57.73</td>
<td>8.86*</td>
<td>29.85</td>
</tr>
<tr>
<td>4096</td>
<td>11.26</td>
<td>65.04</td>
<td>9.71*</td>
<td>33.99</td>
</tr>
<tr>
<td>8192</td>
<td>18.05</td>
<td>73.03</td>
<td>11.41*</td>
<td>41.31</td>
</tr>
<tr>
<td>16384</td>
<td>28.02</td>
<td>81.30</td>
<td>14.29*</td>
<td>55.85</td>
</tr>
<tr>
<td>32768</td>
<td>50.47</td>
<td>160.89</td>
<td>20.71*</td>
<td>61.36</td>
</tr>
<tr>
<td>65536</td>
<td>93.98</td>
<td>247.74</td>
<td>30.63*</td>
<td>82.26</td>
</tr>
<tr>
<td>131072</td>
<td>163.70</td>
<td>325.70</td>
<td>59.16*</td>
<td>136.41</td>
</tr>
<tr>
<td>262144</td>
<td>460.85</td>
<td>742.38</td>
<td>970.05*</td>
<td>426.22</td>
</tr>
<tr>
<td>524288</td>
<td>823.88</td>
<td>1746.14</td>
<td>818.26*</td>
<td>1368.98</td>
</tr>
<tr>
<td>1048576</td>
<td>898.08</td>
<td>3351.08</td>
<td>5261.42</td>
<td>970.05*</td>
</tr>
<tr>
<td>2097152</td>
<td>2003.62</td>
<td>14957.62</td>
<td>6862.75</td>
<td>13167.39</td>
</tr>
<tr>
<td>4194304</td>
<td>3582.84</td>
<td>38222.43</td>
<td>33904.29</td>
<td>4914.07</td>
</tr>
<tr>
<td>AVG</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
</tbody>
</table>
MPITUNE – Result File

```bash
$ cat mpiexec_shm_nn_1_np_32_ppn_32.conf

-genv I_MPI_ADJUST_BARRIER '7'
-genv I_MPI_ADJUST_GATHER '1:0-1;4:2-12;1:13-64;4:65-947;3:948-2147483647'
```

Note that results files may be empty if the default settings are optimal.
PROFILE & ANALYZE HIGH PERFORMANCE MPI APPLICATIONS WITH INTEL® TRACE ANALYZER & COLLECTOR 2019

Profile, Analyze & Visualize MPI Applications

Part of Intel® Parallel Studio XE Cluster Edition and Available Individually
Intel® Trace Analyzer and Collector Overview

Helps the developer to:
- 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

API and -tcollect

Intel® Trace Collector

Trace File (.stf)

Intel® Trace Analyzer

Source Code
- Compiler
- Objects
- Linker
- Binary
- Runtime
- Output
How to Use Intel® Trace Analyzer and Collector
It’s Easy...

Step 1

Run your binary and create a tracefile:

$ mpirun -trace -n 2 ./test

Step 2

View the results:

$ traceanalyzer &
Compare the event timelines of two communication profiles

Blue = Computation
Red = Communication

Chart showing how the MPI processes interact
Summary page shows computation vs. communication breakdown

Is your application MPI-bound?

Largest MPI consumers

Resource usage

Is your application CPU-bound?

Next Steps
Views and Charts

- Helps navigating through the trace data and keep orientation.
- Every View can contain several Charts.
- All Charts in a View are linked to a single:
  - Time-span
  - Set of threads
  - Set of functions
- All Charts follow changes to View (e.g., zooming)
Event Timeline

- Get detailed impression of program structure.
- Display functions, messages, and collective operations for each process/thread along time axis.
- Retrieve detailed event information.

### Event Timeline

<table>
<thead>
<tr>
<th>7.482</th>
<th>500 s</th>
<th>7.489</th>
<th>500 s</th>
<th>7.486</th>
<th>500 s</th>
<th>7.490</th>
<th>500 s</th>
<th>7.491</th>
<th>500 s</th>
<th>7.492</th>
<th>500 s</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>MPI</td>
<td>RELAX</td>
<td>CONVERGENCE</td>
<td>MPI</td>
<td>RELAX</td>
<td>CONVERGENCE</td>
<td>MPI</td>
<td>RELAX</td>
<td>CONVERGENCE</td>
<td>MPI</td>
<td>RELAX</td>
</tr>
<tr>
<td>P1</td>
<td>MPI</td>
<td>RELAX</td>
<td>CONVERGENCE</td>
<td>MPI</td>
<td>RELAX</td>
<td>CONVERGENCE</td>
<td>MPI</td>
<td>RELAX</td>
<td>CONVERGENCE</td>
<td>MPI</td>
<td>RELAX</td>
</tr>
<tr>
<td>P2</td>
<td>MPI</td>
<td>RELAX</td>
<td>CONVERGENCE</td>
<td>MPI</td>
<td>RELAX</td>
<td>CONVERGENCE</td>
<td>MPI</td>
<td>RELAX</td>
<td>CONVERGENCE</td>
<td>MPI</td>
<td>RELAX</td>
</tr>
<tr>
<td>P3</td>
<td>MPI</td>
<td>RELAX</td>
<td>CONVERGENCE</td>
<td>MPI</td>
<td>RELAX</td>
<td>CONVERGENCE</td>
<td>MPI</td>
<td>RELAX</td>
<td>CONVERGENCE</td>
<td>MPI</td>
<td>RELAX</td>
</tr>
<tr>
<td>P4</td>
<td>MPI</td>
<td>RELAX</td>
<td>CONVERGENCE</td>
<td>MPI</td>
<td>RELAX</td>
<td>CONVERGENCE</td>
<td>MPI</td>
<td>RELAX</td>
<td>CONVERGENCE</td>
<td>MPI</td>
<td>RELAX</td>
</tr>
<tr>
<td>P5</td>
<td>MPI</td>
<td>RELAX</td>
<td>CONVERGENCE</td>
<td>MPI</td>
<td>RELAX</td>
<td>CONVERGENCE</td>
<td>MPI</td>
<td>RELAX</td>
<td>CONVERGENCE</td>
<td>MPI</td>
<td>RELAX</td>
</tr>
<tr>
<td>P6</td>
<td>MPI</td>
<td>RELAX</td>
<td>CONVERGENCE</td>
<td>MPI</td>
<td>RELAX</td>
<td>CONVERGENCE</td>
<td>MPI</td>
<td>RELAX</td>
<td>CONVERGENCE</td>
<td>MPI</td>
<td>RELAX</td>
</tr>
<tr>
<td>P7</td>
<td>MPI</td>
<td>RELAX</td>
<td>CONVERGENCE</td>
<td>MPI</td>
<td>RELAX</td>
<td>CONVERGENCE</td>
<td>MPI</td>
<td>RELAX</td>
<td>CONVERGENCE</td>
<td>MPI</td>
<td>RELAX</td>
</tr>
</tbody>
</table>
Quantitative Timeline

- Get impression on parallelism and load balance.
- Show for every function how many threads/processes are currently executing it.
# Flat Function Profile

## Statistics About Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>TTotal</th>
<th>TTotal / TSelf</th>
<th>TSelf / TTotal</th>
<th>#Calls</th>
<th>TSelf / Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Send</td>
<td>0.913 s</td>
<td>0.913 s</td>
<td>0.913 s</td>
<td>1</td>
<td>0.913 s</td>
</tr>
<tr>
<td>Process 31 Thread 0</td>
<td>0.913 s</td>
<td>0.913 s</td>
<td>0.913 s</td>
<td>1</td>
<td>0.913 s</td>
</tr>
<tr>
<td>Process 24 Thread 0</td>
<td>0.766 s</td>
<td>0.766 s</td>
<td>0.766 s</td>
<td>1</td>
<td>0.766 s</td>
</tr>
<tr>
<td>Process 23 Thread 0</td>
<td>0.721 s</td>
<td>0.721 s</td>
<td>0.721 s</td>
<td>1</td>
<td>0.721 s</td>
</tr>
<tr>
<td>Process 27 Thread 0</td>
<td>0.711 s</td>
<td>0.711 s</td>
<td>0.711 s</td>
<td>1</td>
<td>0.711 s</td>
</tr>
<tr>
<td>Process 7 Thread 0</td>
<td>0.643 s</td>
<td>0.643 s</td>
<td>0.643 s</td>
<td>1</td>
<td>0.643 s</td>
</tr>
<tr>
<td>Process 15 Thread 0</td>
<td>0.597 s</td>
<td>0.597 s</td>
<td>0.597 s</td>
<td>1</td>
<td>0.597 s</td>
</tr>
<tr>
<td>Process 16 Thread 0</td>
<td>0.628 s</td>
<td>0.628 s</td>
<td>0.628 s</td>
<td>1</td>
<td>0.628 s</td>
</tr>
<tr>
<td>Process 6 Thread 0</td>
<td>0.610 s</td>
<td>0.610 s</td>
<td>0.610 s</td>
<td>1</td>
<td>0.610 s</td>
</tr>
<tr>
<td>Process 8 Thread 0</td>
<td>0.596 s</td>
<td>0.596 s</td>
<td>0.596 s</td>
<td>1</td>
<td>0.596 s</td>
</tr>
<tr>
<td>Process 4 Thread 0</td>
<td>0.584 s</td>
<td>0.584 s</td>
<td>0.584 s</td>
<td>1</td>
<td>0.584 s</td>
</tr>
<tr>
<td>Process 20 Thread 0</td>
<td>0.575 s</td>
<td>0.575 s</td>
<td>0.575 s</td>
<td>1</td>
<td>0.575 s</td>
</tr>
<tr>
<td>Process 25 Thread 0</td>
<td>0.573 s</td>
<td>0.573 s</td>
<td>0.573 s</td>
<td>1</td>
<td>0.573 s</td>
</tr>
<tr>
<td>Process 26 Thread 0</td>
<td>0.571 s</td>
<td>0.571 s</td>
<td>0.571 s</td>
<td>1</td>
<td>0.571 s</td>
</tr>
<tr>
<td>Process 11 Thread 0</td>
<td>0.565 s</td>
<td>0.565 s</td>
<td>0.565 s</td>
<td>1</td>
<td>0.565 s</td>
</tr>
<tr>
<td>Process 30 Thread 0</td>
<td>0.547 s</td>
<td>0.547 s</td>
<td>0.547 s</td>
<td>1</td>
<td>0.547 s</td>
</tr>
<tr>
<td>Process 29 Thread 0</td>
<td>0.547 s</td>
<td>0.547 s</td>
<td>0.547 s</td>
<td>1</td>
<td>0.547 s</td>
</tr>
<tr>
<td>Process 18 Thread 0</td>
<td>0.540 s</td>
<td>0.540 s</td>
<td>0.540 s</td>
<td>1</td>
<td>0.540 s</td>
</tr>
<tr>
<td>Process 16 Thread 0</td>
<td>0.510 s</td>
<td>0.510 s</td>
<td>0.510 s</td>
<td>1</td>
<td>0.510 s</td>
</tr>
<tr>
<td>Process 6 Thread 0</td>
<td>0.466 s</td>
<td>0.466 s</td>
<td>0.466 s</td>
<td>1</td>
<td>0.466 s</td>
</tr>
<tr>
<td>Process 2 Thread 0</td>
<td>0.486 s</td>
<td>0.486 s</td>
<td>0.486 s</td>
<td>1</td>
<td>0.486 s</td>
</tr>
<tr>
<td>Process 12 Thread 0</td>
<td>0.485 s</td>
<td>0.485 s</td>
<td>0.485 s</td>
<td>1</td>
<td>0.485 s</td>
</tr>
<tr>
<td>Process 5 Thread 0</td>
<td>0.480 s</td>
<td>0.480 s</td>
<td>0.480 s</td>
<td>1</td>
<td>0.480 s</td>
</tr>
<tr>
<td>Process 21 Thread 0</td>
<td>0.474 s</td>
<td>0.474 s</td>
<td>0.474 s</td>
<td>1</td>
<td>0.474 s</td>
</tr>
<tr>
<td>Process 6 Thread 0</td>
<td>0.466 s</td>
<td>0.466 s</td>
<td>0.466 s</td>
<td>1</td>
<td>0.466 s</td>
</tr>
<tr>
<td>Process 18 Thread 0</td>
<td>0.482 s</td>
<td>0.482 s</td>
<td>0.482 s</td>
<td>1</td>
<td>0.482 s</td>
</tr>
<tr>
<td>Process 1 Thread 0</td>
<td>0.448 s</td>
<td>0.448 s</td>
<td>0.448 s</td>
<td>1</td>
<td>0.448 s</td>
</tr>
<tr>
<td>Process 10 Thread 0</td>
<td>0.392 s</td>
<td>0.392 s</td>
<td>0.392 s</td>
<td>1</td>
<td>0.392 s</td>
</tr>
<tr>
<td>Process 22 Thread 0</td>
<td>0.360 s</td>
<td>0.360 s</td>
<td>0.360 s</td>
<td>1</td>
<td>0.360 s</td>
</tr>
<tr>
<td>Process 14 Thread 0</td>
<td>0.377 s</td>
<td>0.377 s</td>
<td>0.377 s</td>
<td>1</td>
<td>0.377 s</td>
</tr>
<tr>
<td>Process 17 Thread 0</td>
<td>0.377 s</td>
<td>0.377 s</td>
<td>0.377 s</td>
<td>1</td>
<td>0.377 s</td>
</tr>
<tr>
<td>Process 9 Thread 0</td>
<td>0.357 s</td>
<td>0.357 s</td>
<td>0.357 s</td>
<td>1</td>
<td>0.357 s</td>
</tr>
<tr>
<td>Process 19 Thread 0</td>
<td>0.245 s</td>
<td>0.245 s</td>
<td>0.245 s</td>
<td>1</td>
<td>0.245 s</td>
</tr>
<tr>
<td>Process 17 Thread 0</td>
<td>0.245 s</td>
<td>0.245 s</td>
<td>0.245 s</td>
<td>1</td>
<td>0.245 s</td>
</tr>
</tbody>
</table>

*Other names and brands may be claimed as the property of others.*
Call-Tree and Call-Graph

Function Statistics Including Calling Hierarchy

- **Tree: Call-stack**
- **Graph: Calling dependencies**
Communication Profiles

- Statistics about point-to-point or collective communication
- Generic matrix supports grouping by several attributes in each dimension: Sender, receiver, data volume per message, tag, communicator, type
- Available attributes: Count, bytes transferred, time, transfer rate
BACKUP
Intel® VTune Amplifier

Platform Profiler
Collect and manage data from multiple targets easily

Data Collector

Platform Profiler Server
Web-based user interface

Data Collection “Targets”

Platform Profiler Server
Web-based UI
Chrome works best
Workflow overview

1. Install VTune and launch the Platform Profiler Server

2. Install data collector on “targets” (Linux or Windows) Ivy Bridge or Later

3. Collect data

4. Upload data

5. View and analyze results

Intel® VTune Amplifier – Platform Profiler

Data Collection “Targets”

Platform Profiler Server

Web-based UI Chrome works best
Legal Disclaimer & Optimization Notice

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 complete information visit www.intel.com/benchmarks.

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.

Copyright © 2018, 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