INTEL® VTUNE™ AMPLIFIER XE
FOR TUNING OF HPC APPLICATIONS

Intel Software Developer Conference – Frankfurt, 2017
Klaus-Dieter Oertel, Intel
• Which performance analysis tool should I use first?
  • Intel® Application Performance Snapshot
  • Intel® VTune™ Amplifier
  • Some examples and solutions
  • What’s new in 2018?
  • Resources
BEFORE DIVE TO A PARTICULAR TOOL...

• How to assess easily that I have potential in performance tuning?
• What to use on big scale not be overwhelmed with huge trace size, post processing time and collection overhead?
  • On a KNL cluster customers can end-up with more than 1000 ranks on just 8 nodes
• How to quickly evaluate environment settings or incremental code changes?
• Which tool should I use first?
• **Answer: try Application Performance Snapshot 2018**
WHICH PERFORMANCE ANALYSIS TOOL SHOULD I USE FIRST?

• Intel® Application Performance Snapshot
  • Provides high level and easy to understand metrics
  • Highlight the main bottlenecks
  • Can be easily integrated in the build chain to provide feedback to developers

• Intel® Vtune™ Amplifier
  • Go deeper, get detailed information about source lines
  • Dedicated analysis to target a specific aspect (threading, memory, etc.)
AGENDA

• Which performance analysis tool should I use first?

• Intel® Application Performance Snapshot

• Intel® VTune™ Amplifier

• Some examples and solutions

• What’s new in 2018?

• Resources
APPLICATION PERFORMANCE SNAPSHOT (APS)

- High-level overview of application performance
- Identify primary optimization areas and next steps in analysis
- Easy to use
- Detailed reports available via command line
- Scales to large jobs
- Multiple methods to obtain
  - Part of Intel® Vtune™ Amplifier 2018
  - Separate free download from Performance Snapshot page
APS HTML REPORT

Application Performance Snapshot

Application: heart_demo_aux_2
Number of ranks: 22
Used statistics: /home/vtune/dprohorov/apps/Cardiac/Cardiac
/build/stat_20170605
Creation date: 2017-06-05 21:33:32

20.22s
Elapsed Time

60.81 SP FLOPS
1.12 CPI (MAX 1.13, MIN 1.12)

MPL Time
62.60% of Elapsed Time (12.66s)

OpenMP Imbalance
4.03% of Elapsed Time (0.81s)

Memory Imbalance
53.16% of Elapsed Time (10.75s)

TOP 5 MPI Functions %
Waitall 55.30
Barrier 5.80
Isend 0.28
Irecv 0.15
Scatterv 0.01

Memory Footprint
Per node: AVG 11055.40 MB, PEAK 11055.40 MB
Per rank: AVG 502.52 MB, PEAK 610.43 MB

Your application is MPI bound. This may be caused by high busy wait time inside the library (imbalance), non-optimal communication schema or MPI library settings. Use MPI profiling tools like Intel® Trace Analyzer and Collector to explore performance bottlenecks.

- **MPL Time**: 62.60% <15%
- **OpenMP Imbalance**: 4.03% <10%
- **Memory Stalls**: 23.33% <20%
- **FPU Utilization**: 0.90% >50%
- **I/O Bound**: 0.00% <10%

For more complete information about compiler optimizations, see our Optimization Notice.
APS USAGE

Setup Environment
• source <APS_Install_dir>/apsvars.sh

Run Application
• mpirun <mpi options> aps <application and args>

Generate Report on Results
• aps-report <result folder>

Generate advanced CL reports on Results
• aps-report –<option> <result folder>
AGENDA

• Which performance analysis tool should I use first?
• Intel® Application Performance Snapshot
• Intel® VTune™ Amplifier
• Some examples and solutions
• What’s new in 2018?
• Resources
INTEL® VTUNE™ AMPLIFIER XE
Performance Profiler

Where is my application...

Spending Time?
- Focus tuning on functions taking time
- See call stacks
- See time on source

Wasting Time?
- See cache misses on your source
- See functions sorted by # of cache misses

Waiting Too Long?
- See locks by wait time
- Red/Green for CPU utilization during wait

- Windows & Linux
- Low overhead
- No special recompiles

Advanced Profiling For Scalable Multicore Performance

© 2017 Intel Corporation. All rights reserved. Intel and the Intel logo are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries. *Other names and brands may be claimed as the property of others. For more complete information about compiler optimizations, see our Optimization Notice.
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...
- 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
- Local & Remote Data Collection, Command Line
- Analyze Windows* & Linux* data on OS X*

Quickly Find Tuning Opportunities

See Results On The Source Code

Tune OpenMP Scalability

Visualize & Filter Data

Optimization Notice
Events vary by processor. No data collection on OS X*. Other names and brands may be claimed as the property of others.
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.
## Analysis Types (based on technology)

<table>
<thead>
<tr>
<th>Software Collector</th>
<th>Hardware Collector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any x86 processor, any virtual, no driver</td>
<td>Higher res., lower overhead, system wide</td>
</tr>
<tr>
<td><strong>Basic Hotspots</strong></td>
<td></td>
</tr>
<tr>
<td>Which functions use the most time?</td>
<td>Which functions use the most time?</td>
</tr>
<tr>
<td></td>
<td>Where to inline? – Statistical call counts</td>
</tr>
<tr>
<td><strong>Concurrency</strong></td>
<td></td>
</tr>
<tr>
<td>Tune parallelism.</td>
<td></td>
</tr>
<tr>
<td>Colors show number of cores used.</td>
<td>General Exploration</td>
</tr>
<tr>
<td></td>
<td>Where is the biggest opportunity?</td>
</tr>
<tr>
<td></td>
<td>Cache misses? Branch mispredictions?</td>
</tr>
<tr>
<td><strong>Locks and Waits</strong></td>
<td></td>
</tr>
<tr>
<td>Tune the #1 cause of slow threaded performance – waiting with idle cores.</td>
<td>Advanced Analysis</td>
</tr>
<tr>
<td></td>
<td>Dig deep to tune bandwidth, cache misses, access contention, etc.</td>
</tr>
</tbody>
</table>
INTEL® VTUNE™ AMPLIFIER XE

Software or hardware collector?

List of hardware counters used
Intel® VTune™ Amplifier XE

Identify hotspots

<table>
<thead>
<tr>
<th>Function / Call Stack</th>
<th>CPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>grid_intersect</td>
<td>6.767s</td>
</tr>
<tr>
<td>shader</td>
<td>4.208s</td>
</tr>
<tr>
<td>trace &lt; shade_reflect</td>
<td>1.474s</td>
</tr>
<tr>
<td>render_one_pixel_d</td>
<td>0.741s</td>
</tr>
<tr>
<td>grid_intersect</td>
<td>0.343s</td>
</tr>
<tr>
<td>sphere_intersect</td>
<td>3.778s</td>
</tr>
<tr>
<td>grid_bounds_intersect</td>
<td>0.414s</td>
</tr>
<tr>
<td>GdiDrawImagePointRect</td>
<td>0.384s</td>
</tr>
<tr>
<td>shader</td>
<td>0.113s</td>
</tr>
</tbody>
</table>

Quickly identify what is important

Hottest Functions

Hottest Call Stack
TUNE OPENMP FOR EFFICIENCY AND SCALABILITY

See the wall clock impact of inefficiencies, identify their cause

Focus On What's Important

- What region is inefficient?
- Is the potential gain worth it?
- Why is it inefficient?
  - Imbalance? Scheduling? Lock spinning?
- Intel® Xeon Phi systems supported
VISUALIZE PARALLEL PERFORMANCE ISSUES

Look for Common Patterns

- Coarse Grain Locks
- High Lock Contention
- Load Imbalance
- Low Concurrency
**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

*Other names and brands may be claimed as the property of others.
STORAGE DEVICE ANALYSIS
HDD, SATA or NVMe SSD

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
**FIND ANSWERS FAST**

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
USER API

Enable you to

- control data collection
- set marks during the execution of the specific code
- specify custom synchronization primitives implemented without standard system APIs

To use the user APIs, do the following:

- Include `ittnotify.h`, located at `<install_dir>/include`
- Insert `__itt_*` notifications in your code
- Link to the `libittnotify.lib` file located at `<install_dir>/lib`
USER API

Collection control and threads naming

**Collection Control APIs**

void **__itt_pause** (void)

Run the application without collecting data. VTune™ Amplifier XE reduces the overhead of collection, by collecting only critical information, such as thread and process creation.

void **__itt_resume** (void)

Resume data collection. VTune™ Amplifier XE resumes collecting all data.

**Thread naming APIs**

void **__itt_thread_set_name** (const __itt_char *name)

Set thread name using char or Unicode string, where *name* is the thread name.

void **__itt_thread_ignore** (void)

Indicate that this thread should be ignored from analysis. It will not affect the concurrency of the application. It will not be visible in the Timeline pane.
**USER API**

Collection Control Example

```c
int main(int argc, char* argv[]) {
    __itt_pause();
    doSomeInitializationWork();

    __itt_resume();
    while(gRunning) {
        doSomeDataParallelWork();
    }
    __itt_pause();

    doSomeFinalizationWork();
    return 0;
}
```
AGENDA

• Which performance analysis tool should I use first?

• Intel® Application Performance Snapshot

• Intel® VTune™ Amplifier

• Some examples and solutions

• What’s new in 2018?

• Resources
Fibonacci and scheduling
Thread scheduling issue with OMP

• Very naïve implementation (just want to show a common pattern)
  • We want to fill an array with numbers from the Fibonacci suite

```c
#pragma omp parallel for
for (int i=0; i<SIZE; i++){
    fib_array[i] = fibonacci(i);
}
```

```c
int fibonacci(int i){
    if(i==0) return 0;
    if(i==1) return 1;
    return fibonacci(i-1) + fibonacci(i-2);
}
```

By default, OMP uses a static scheduling. Each thread will do the same number of iterations.
Thread scheduling issue with OMP

CPU Usage Histogram
This histogram displays a percentage of the wall time the specified

Very poor threading
Fib(0) is much faster to compute than Fib(50) !!!!
A static scheduling creates a very high Load imbalance.
Thread scheduling issue with OMP

• Very naïve implementation (just want to show a common pattern)
  • We want to fill an array with numbers from the Fibonacci suite

```c
#pragma omp parallel for schedule(guided)
for (int i=0; i<SIZE; i++){
    fib_array[i] = fibonacci(i);
}

int fibonacci(int i){
    if(i==0) return 0;
    if(i==1) return 1;
    return fibonacci(i-1) + fibonacci(i-2);
}
```
Thread scheduling issue with OMP

**CPU Usage Histogram**

This histogram displays a percentage of the wall time the spe

Just changing the scheduling provides an important speedup, around 2x for Fib(50)
Linear regression and false sharing identification
What is false sharing?

- 2 or more threads reading/writing the same cache line
  - At least 1 thread is writing data
  - Other threads want to read another data in the same cache line

- Linear regression sample (available in Vtune’s package)

Running the memory analysis shows a bottleneck on the L1 cache system.
What is false sharing?

1- Look for memory object responsible for latency

<table>
<thead>
<tr>
<th>Memory Object</th>
<th>Total Latency</th>
<th>Loads</th>
<th>Stores</th>
<th>LLC Miss Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear_regression pthread.c:136 (512 B)</td>
<td>64.4%</td>
<td>14,058,042,174</td>
<td>4,998,074,970</td>
<td>0</td>
</tr>
<tr>
<td>[Unknown]</td>
<td>28.7%</td>
<td>19,104,057,312</td>
<td>202,803,042</td>
<td>0</td>
</tr>
<tr>
<td>linear_regression pthread.c:118 (54 MB)</td>
<td>6.0%</td>
<td>10,536,031,608</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>[Stack]</td>
<td>0.0%</td>
<td>0</td>
<td>2,400,036</td>
<td>0</td>
</tr>
</tbody>
</table>

2- Identify allocation site, object size and average latency

<table>
<thead>
<tr>
<th>Memory Object</th>
<th>Function</th>
<th>Allocation Stack</th>
<th>Loads</th>
<th>Stores</th>
<th>Average Latency (cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Unknown]</td>
<td></td>
<td></td>
<td>19,104,057,312</td>
<td>202,803,042</td>
<td>8</td>
</tr>
<tr>
<td>linear_regression pthread.c:136 (512 B)</td>
<td></td>
<td></td>
<td>14,058,042,174</td>
<td>4,998,074,970</td>
<td>37</td>
</tr>
<tr>
<td>linear_regression pthread.c:118 (54 MB)</td>
<td></td>
<td></td>
<td>10,536,031,608</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>[Stack]</td>
<td></td>
<td></td>
<td>0</td>
<td>2,400,036</td>
<td>0</td>
</tr>
</tbody>
</table>

3- Look into the code

```c
135  req_units = n / num_threads;
136  tid_args = (lreq_args *call) : sizeof(lreq_args), num_procs);
```

This structure seems to be responsible
What is false sharing?

Cache line

Thread 0

New struct starts here

Thread 1

Here the structure is 64bytes (same as cache line) But depending on alignment, 2 lreg_args objects can Share the same cache line.

typedef struct {
    pthread_t tid;
    POINT_T *points;
    int num_elms;
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
    long long SX,
    long long SY,
What is false sharing?

- To solve the false sharing, we can add an array that will pad our structure and avoid having data of 2 lreg_args objects sharing the same cache line.

```c
typedef struct {
    char pad[80];
    pthread_t tid;
    POINT_T *points;
    int num_elems;
    long long SX;
    long long SY,
    long long SXX,
    long long SYY,
    long long SXY;
} lreg_args;
```

Bonus, not explained in the sample!

In this test, aligning the data to a 64 bytes boundary can also solve the problem!
AGENDA

• Which performance analysis tool should I use first?

• Intel® Application Performance Snapshot

• Intel® VTune™ Amplifier

• Some examples and solutions

• What’s new in 2018?

• Resources
APPLICATION PERFORMANCE SNAPSHOT ADDS MPI
All the data in one place: MPI + OpenMP + Memory + Floating Point

Quick & easy performance overview
- Does the app need performance tuning?
- MPI and non-MPI Apps†
- Distributed MPI with or without threading
- Shared memory applications

Popular MPI implementations supported
- Intel® MPI
- MPICH and Cray MPI

Richer metrics on computation efficiency
- CPU (processor stalls, memory access)
- FPU (vectorization metrics)

† MPI supported only on Linux*. 
MORE COMPLETE HPC PERFORMANCE OVERVIEW

MPI metrics added to HPC analysis

**MPI Imbalance Metric**
- Metric for performance of rank on critical path
- Computational bottlenecks and outlier rank behavior now available in VTune Amplifier
- For communication pattern problems between ranks use Intel® Trace Analyzer and Collector (ITAC)

**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.
WHAT’S USING ALL THE MEMORY?
Memory Consumption Analysis

See What Is Allocating Memory
- Lists top memory consuming functions and objects
- View source to understand cause
- Filter by time using the memory consumption timeline

- Standard & Custom Allocators
  - Recognizes libc malloc/free, memkind and jemalloc libraries
  - Use custom allocators after markup with ITT Notify API

Languages
- Python*
- Linux*: Native C, C++, Fortran

Native language support is not currently available for Windows*
TUNE THREADED PYTHON* PERFORMANCE
Visualize parallel performance issues

Locks and Waits Analysis
- Python
- Mixed Python / Native code
- Native code

Optional Call Stacks

Coarse Grain Locks

Quickly see patterns in the timeline that indicate low concurrency

High Lock Contention

Load Imbalance
OPTIMIZE PRIVATE CLOUD-BASED APPLICATIONS
Profile native & Java apps in containers

Profile Enterprise Applications
- Native C, C++, Fortran
- Attach to running Java services (e.g., Mail)
- Profile Java daemons without restart

Accurate low-overhead data collection
- Advanced hotspots and hardware events
- Memory analysis
- Accurate stack information for Java and HHVM

Popular containers supported
- Docker*
- Mesos*
- LXC*

Software collectors (e.g. Locks & Waits) and Python profiling are not currently available for containers.
IN-KERNEL GPU PROFILING

Tune Inefficient Kernel Algorithms with Intel® VTune™ Amplifier

Analyze GPU Kernel Execution

- Find memory latency or inefficient kernel algorithms
- See the hotspot on the OpenCL™ source & assembly code
- Analyze DMA packet execution
  - Packet Queue Depth histogram
  - Packet Duration histogram

```
12 __kernel void workload(int nIter, __global float* result) {
13     float r = 0.0;
14     for (int i = 1; i <= nIter; i++) {
15         r += 1.0 / factor(i);
16     }
17     *result = r;
18 }
```
EASIER PROFILING OF REMOTE LINUX SYSTEMS
Automated install of performance collectors on a remote Linux target

Just Specify an SSH Connection and Install Directory
- No separate download and install required
- Always get the correct version of the collectors
AGENDA

• Which performance analysis tool should I use first?
• Intel® Application Performance Snapshot
• Intel® VTune™ Amplifier
• Some examples and solutions
• What’s new in 2018?
• Resources
RESOURCES

Intel® VTune™ Amplifier – Performance Profiler

- Product page – overview, features, FAQs...
- Training materials – tech briefs, documentation, eval guides...
- Reviews
- Support – forums, secure support...

Additional Analysis Tools

- Intel® Inspector – memory and thread checker/ debugger
- Intel® Advisor – vectorization optimization and thread prototyping
- Intel® Trace Analyzer and Collector – MPI Analyzer and Profiler

Additional Development Products

- Intel® Software Development Products
- Intel® Distribution for Python* – accelerated Python distribution

Webinars
Free in-depth presentations
- Register
- View Archives

What's New?
Purchase includes a year of updates. Check out the latest improvements.
CODE THAT PERFORMS AND OUTPERFORMS

Download a free, 30-day trial of Intel® Parallel Studio XE 2018 today

software.intel.com/en-us/intel-parallel-studio-xe

AND DON’T FORGET...

To fill out the evaluation survey via a URL that will be provided at the end of the day
OR
Watch your email for a link to the survey

P.S.

Everyone who fills out the survey will receive a personalized certificate indicating completion of the training!
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