OPTIMIZING THREADED CODE
PERFORMANCE AND SCALABILITY

AGENDA

- Which tools should I use for threading and scalability?
  - Intel® Performance Snapshot
  - Intel® VTune™ Amplifier
  - Some examples and solutions
  - What’s new in 2018?
WHICH TOOL SHOULD I USE FOR THREADING AND SCALABILITY?

• Intel® Performance Snapshots
  • 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 tools should I use for threading and scalability?

• Intel® Performance Snapshot

• Intel® VTune™ Amplifier

• Some examples and solutions

• What’s new in 2018?
BEFORE DIVE TO A PARTICULAR TOOL...

• How to assess that I have potential in performance tuning?
• Which tool should I use first?
• 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?
• Answer: try Application Performance Snapshot 2018
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® Parallel Studio XE 2018
  • Separate free download from Performance Snapshot page
Application Performance Snapshot

Application: heart_demo_aux_2
Number of ranks: 22
Used statistics: /home/vtune/dprohoroi/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)

MPI Time
62.60% of Elapsed Time
(12.66s)

OpenMP Imbalance
4.03% of Elapsed Time
(0.81s)

Memory Footprint
Cache Stalls
24.45% of cycles
DRAM Stalls
0.01% of cycles
NUMA
16.03% of remote accesses

Memory Stalls
23.33% of pipeline slots

FPU Utilization
0.90%

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%

© 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.
APS USAGE

Setup Environment
• source <APS_Install_dir>/apsvars.sh

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

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

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

• Which tools should I use for threading and scalability?
• Intel® Performance Snapshot
• Intel® VTune™ Amplifier
• Some examples and solutions
• What’s new in 2018?
**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
**Intel® VTune™ Amplifier XE**

Tune Applications for Scalable Multicore Performance

- **Fast, Accurate Performance Profiles**
  - Hotspot (Statistical call tree)
  - Call counts (Statistical)
  - Hardware-Event Sampling

- **Thread Profiling**
  - Visualize thread interactions on timeline
  - Balance workloads

- **Easy set-up**
  - Pre-defined performance profiles
  - Use a normal production build

- **Find Answers Fast**
  - Filter extraneous data
  - View results on the source / assembly

- **Compatible**
  - Microsoft, GCC, Intel compilers
  - C/C++, Fortran, Assembly, .NET, Java
  - Latest Intel® processors and compatible processors¹

- **Windows or Linux**
  - Visual Studio Integration (Windows)
  - Standalone user i/f and command line
  - 32 and 64-bit

¹ IA32 and Intel® 64 architectures. Many features work with compatible processors. Event based sampling requires a genuine Intel® Processor.
## INTEL® VTUNE™ AMPLIFIER XE

### Analysis Types (based on technology)

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

Software or hardware collector?

List of hardware counters used
INTEL® VTUNE™ AMPLIFIER XE

Get a quick snapshot
INTEL® VTUNE™ AMPLIFIER XE
Look for Common Patterns

- Coarse Grain Locks
- High Lock Contention
- Load Imbalance

Low Concurrency
INTEL® VTUNE™ AMPLIFIER XE

Identify hotspots

Hottest Functions

Quickly identify what is important

Hottest Call Stack
INTEL® VTUNE™ AMPLIFIER XE
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
• Optional: Use API to mark frames and user tasks
• Optional: Add a mark during collection
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.

© 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.
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
User API

Enable you to

• control 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[]) {
    doSomeInitializationWork();

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

    __itt_pause();

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

• Which tools for threading and scalability?
• Intel® Performance Snapshot
• Intel® VTune™ Amplifier
• Some examples and solutions
• What’s new in 2018?
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] = fib(i);
}
```

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

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

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] = fib(i);
}

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

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

2. Identify allocation site, object size and average latency

3. Look into the code

This structure seems to be responsible
What is false sharing?

Cache line

Thread 0

New struct starts here

Here the structure is 64 bytes (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_elens;
    long long SX;
    long long SY;
    long long SXX;
    long long SYY;
    long long SXY;
} lreg_args;
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_eles;
    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 tools for threading and scalability?
• Intel® Performance Snapshot
• Intel® VTune™ Amplifier
• Some examples and solutions
• What’s new in 2018?
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.
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*
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.
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 © 2015, 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