HPC APPLICATIONS NEED HIGH-PERFORMANCE ANALYSIS

Jackson Marusarz – Developer Products Division
AGENDA

- Performance Analysis Accessibility: The Current State
- Segment Specific Performance Analysis: HPC Characterization
- HPC Characterization Metrics
- Examples
- Summary & Next Steps
PERFORMANCE ANALYSIS ACCESSIBILITY: THE CURRENT STATE

- One size fits all solutions
  - Hotspots, top, SDM/perf metrics, etc...
- One size fits ONE solutions
  - printf, timing APIs, app-specific benchmarks
- What is useful vs. what is easy
  - Use an ax or reinvent the scalpel

SEGMENT SPECIFIC METHODOLOGIES ARE RARE
Segment Specific Performance Analysis: HPC Characterization

- HPC applications exhibit common behaviors and performance issues
  - Highly parallel, heavy resource demands, “by any means necessary”
- Targeted monitoring and analysis
  - Pinpoint the intersection of important, understandable, and actionable performance data
- Provide expert analysis and advice
  - Metric thresholds, understandable explanations and advice

We know our enemy, how do we defeat it?
SEGMENT SPECIFIC PERFORMANCE ANALYSIS: HPC CHARACTERIZATION

THREE METRICS CLASSES

- **CPU Utilization**
  - Logical core % usage
  - Includes parallelism and OpenMP information

- **Memory Bound**
  - Break down each level of the memory hierarchy

- **FPU Utilization**
  - Floating point GFLOPS and density

---

*In general* HPC Applications care less about power and response (mobile/client) or multi-job throughput and peak load limiting (server/real time).
SEGMENT SPECIFIC PERFORMANCE ANALYSIS: HPC CHARACTERIZATION
RUNNING THE TOOL

- Setup analysis with the GUI
- Or Easy command line collection
  - `>amplxe-cl --collect hpc-performance --data-limit=0 --r result_dir ./my_app`
HPC CHARACTERIZATION: CPU UTILIZATION

CPU Utilization
- % of “Effective” logical CPU usage by the application under profiling (threshold 90%)
  - Under assumption that the app should use all available logical cores on a node
  - Subtracting spin/overhead time spent in MPI and threading runtimes based on event IPs

Metrics in CPU utilization section
- Average CPU Utilization – based on CPU_CLK_TICK events
- Additional MPI and OpenMP scalability metrics impacting effective CPU utilization
- CPU utilization histogram

WHEN CORES SIT IDLE, PERFORMANCE IS LOST.
HPC CHARACTERIZATION: MEMORY BOUND

Memory Bound
- % of potential execution pipeline slots lost because of fetching memory (threshold 80%)
- Metrics based on PMU counters

Metrics in Memory Bound section
- Cache Bound: Stalls while requests are pending that eventually come from cache
- DRAM Bound: Stalls while requests are pending that eventually come from DRAM
- Bandwidth bound: lots of pending requests per cycle based on offcore counters
- Latency bound: very few pending requests per cycle based on offcore counters
- NUMA: % of remote accesses

- Cache Bound: 89.8%

- DRAM Bound: 0.644

Memory is often the bottleneck. Find and relieve the pressure.
HPC CHARACTERIZATION: FPU UTILIZATION

FPU utilization

- % of FPU load (100% - FPU is fully loaded, threshold 50%)
- Calculation based on PMU events representing scalar and packed single and double precision SIMD instructions

Metrics in FPU utilization section

- FLOPs broken down by scalar and packed
- Instruction Mix
- Top 5 loops/functions by FPU usage
  ▪ Detected with static binary analysis
- Vectorized vs. Non-vectorized, ISA, and characterization detected by static analysis

**FPU Utilization**

- **1.3%**
  - SP FLOPs per Cycle: 0.211 Out of 16%
  - Vector Capacity Usage: 48.3%
  - FP Instruction Mix:
    - % of Packed FP Instr: 93.1%
    - % of 128-bit: 93.1%
    - % of 256-bit: 0.0%
    - % of Scalar FP Instr: 6.9%
  - FP Arith/Mem Rd Instr Ratio: 0.264
  - FP Arith/Mem Wr Instr Ratio: 6.298

**Top 5 hotspot loops (functions) by FPU usage**

This section provides information for the most time consuming loops/functions with floating point operations.

<table>
<thead>
<tr>
<th>Function</th>
<th>CPU Time</th>
<th>FPU Utilization (%)</th>
<th>Vector Instruction Set</th>
<th>Loop Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Loop at line 575 in conjgrad)emfparallel@517</td>
<td>126.149s</td>
<td>1.4%</td>
<td>SSE2(128)</td>
<td>Body</td>
</tr>
<tr>
<td>(Loop at line 678 in conjgrad)emfparallel@517</td>
<td>5.004s</td>
<td>1.7%</td>
<td>SSE2(128)</td>
<td>Body</td>
</tr>
<tr>
<td>(Loop at line 575 in conjgrad)emfparallel@0517</td>
<td>2.678s</td>
<td>2.1%</td>
<td>[Unknown]</td>
<td>Remainder</td>
</tr>
<tr>
<td>(Loop at line 573 in conjgrad)emfparallel@0517</td>
<td>0.995s</td>
<td>4.0%</td>
<td>SSE2(128)</td>
<td>Body</td>
</tr>
<tr>
<td>(Loop at line 964 in conjgrad)emfparallel@0517</td>
<td>0.952s</td>
<td>1.3%</td>
<td>SSE(128); SSE2(128)</td>
<td>Body</td>
</tr>
<tr>
<td>[Others]</td>
<td>2.437s</td>
<td>N/A*</td>
<td>N/A*</td>
<td>N/A*</td>
</tr>
</tbody>
</table>

*N/A* is applied to non-measurable metrics.
HPC CHARACTERIZATION: COMMAND LINE REPORTS

- Generated after collection is done or with “-R summary” option of amplxe-cl
- Matches GUI metrics hierarchy

---

**Elapsed Time:** 7.805s
**SP GFLOPS:** 14.041

**CPU Utilization:** 76.4%

- CPU Usage: 18.344 Out of 24 logical CPUs
- Serial Time: 0.021s (0.3%)
- Parallel Region Time: 7.784s (99.7%)
- Estimated Ideal Time: 6.411s (60.8%)
- OpenMP Potential Gain: 1.373s (17.4%) (The time saved on load imbalance or parallel work arrangement is significant and negatively impacts the application performance and scalability. Explore OpenMP regions with the highest metric values. Make sure the workload of the regions is enough and the loop schedule is optimal.)

**Memory Bound:** 63.2% of Pipeline Slots

- The metric value is high. This can indicate that the significant fraction of execution pipeline slots could be stalled due to demand memory load and store. Use memory access analysis to have the metric breakdown by memory hierarchy, memory bandwidth, instruction, correlation by memory objects.

**Cache Bound:** 36.4% of Clockticks

- A significant proportion of cycles are being spent on data that are not cached. Check memory access analysis to see if accesses to L1 or L3 are problematic and consider applying the same performance tuning as you could for a code-moving workload. This may include reducing the data working set size, improving data access locality, blocking or partitioning the working set to fit in the lower cache levels, or exploiting hardware prefetchers. Consider using software prefetchers, but note that they can interfere with normal loads, increase latency, and increase pressure on the memory system. This metric includes coherence penalties for shared data. Check general exploration analysis to see if contented accesses or data sharing are indicated as likely issues.

---

© 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.
PERFORMANCE EXAMPLES - STATIC SCHEDULING

Elapsed Time: 9.487s
GFLOPS: 6.844

CPU Utilization: 69.7%

Performance examples:

Apply Dynamic scheduling to avoid imbalance

```
do i=1,ni
  do j=1,ni
    suml = 0.0
    do k=1,1
      suml = suml + a(k) * b(k)
    enddo
    q(j) = suml
  enddo
endo
```
Performance Examples – Guided Scheduling (Chunk 10)

CPU Utilization: 88.1%
Average CPU Usage: 21.151 of 24 logical CPUs
Serial Time: 0.038s (0.4%)

Memory Bound: 74.6%

FPU Utilization: 2.7%

Grouping: OpenMP Region / OpenMP Barrier-to-Boundary Segment / Function / Call Stack

```
$omp do schedule (guided,10) do j=1,lastrow-firstrow-1
    sum1 = 0.0d0
    do k=rowstr(j),rowstr(j)+1-1
        sum1 = sum1 + a(k)*p(colidx(k))
    enddo
    q(j) = sum1
enddo
$omp end do
```
Performance Examples - Floating Point Utilization

- Elapsed Time: 12.218s
- GFLOPS: 7.821
- CPU Utilization: 97.6%
- Memory Bound: 50.0%
- FPU Utilization: 8.3%

Outdated Vectorization Instructions - Update Compiler Settings

© 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.
PERFORMANCE EXAMPLES – FLOATING POINT UTILIZATION

**Improves FLOPS and Time – Small Increases are HPC Fundamentals**
ADDITIONAL NOTES

- The power of the methodology is in collecting all 3 metrics at once because they impact each other. For example:
  - CPU Utilization is high but it’s all OpenMP overhead
  - FPU Utilization may be low – but the real cause is a memory bandwidth bottleneck
  - Don’t lose the forest for the trees
- Wall-clock time is usually the “real” indicator of performance
- SMT (Hyper-Threading) on/off should always be considered as it makes things tricky
  - Helps with memory-bound applications more than compute-bound
  - Competition for L1 cache
SUMMARY

- Performance analysis and tuning continues to be an expert-level task
  - HPC Characterization is attempting to shift this
- Focusing segment-specific metrics simplifies and quickens the process
  - CPU Utilization, Memory Bottlenecks, FP Utilization
- This characterization uses a wide array of hardware and software capabilities
  - PMU Counters, un-core events, instrumented OpenMP, compiler diagnostics, static analysis
- The metrics are more than a sum of their parts
  - Each metric may affect or shed light on another issue
LEGAL DISCLAIMER AND 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 © 2017, 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
CREATE FASTER HPC AND CLOUD SOFTWARE

WHAT’S NEW IN INTEL® PARALLEL STUDIO XE 2018 BETA

Modernize Code for Performance, Portability and Scalability on the Latest Intel® Platforms

- Use fast Intel® AVX-512 instructions on Intel® Xeon® and Xeon Phi™ processors.
- Parallelize and vectorize C++ STL easily using Parallel STL*.
- Intel® Advisor - Roofline finds high impact, but under optimized loops
- Intel® Distribution for Python* - Faster Python* applications
- Stay up-to-date with the latest standards and IDE:
  - C++2017 draft parallelizes and vectorizes C++ easily using Parallel STL*
  - Full Fortran* 2008, Fortran 2015 draft
  - OpenMP* 5.0 draft, Microsoft Visual Studio* 2017
- Support for Intel® Omni-Path Architecture

Flexibility for Your Needs

- Application Snapshot - Quick answers: Does my hybrid code need optimization?
- Intel® VTune™ Amplifier – Profile private clouds with Docker* and Mesos* containers, Java* daemons

And much more*...


© 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.
* See Release Notes for the full list with further updates and new features.