KNL PROFILING WITH VTUNE AMPLIFIER XE

Dmitry Prohorov (dmitry.prohorov@intel.com), VTune HPC Lead
Agenda

- Overview
- System configuration
- Analysis Configuration
- Analysis Workflow
- Memory Access analysis
- Micro-arch analysis with General Exploration
- Performance Overview with HPC Performance Characterization (tech preview)
- Basic Hotspots, Concurrency, Locks and Waits
Overview

Explore Performance on Intel® Xeon Phi™ Processor (KNL Self Boot Linux)

- Use VTune Amplifier XE 2016 U4 and further (no NDA package required)
- **Memory Access** analysis
  - Memory access problems by memory hierarchy
  - High Bandwidth Memory analysis
    - Defines if the app is DRAM or MCDRAM bandwidth bound
    - Helps to determine data structures worth to allocate to MCDRAM for DRAM bound apps
- **Micro-architectural issues with General Exploration** analysis
  - Explore how efficiently your code passing through the core pipeline
- **Performance overview with HPC Performance Characterization**
  - Important scalability aspects for OpenMP and hybrid MPI+OpenMP apps
    - CPU utilization
      - Serial vs Parallel time, imbalance, parallel runtime overhead cost, parallel loop parameters
    - Memory access efficiency
    - FPU utilization (upper bound), FLOPS (upper bound), basic loop vectorization info
- **Algorithmic tuning opportunities with Advanced Hotspots**
- Advanced measurements with Custom HW EBS event collection
- And more...
System Configuration
Prerequisites for HW EBS event based collections

• VTune on KNL works with SEP driver (recommended) or upon perf
  • Related to: Advanced Hotspots, Memory Access, General Exploration, HPC Performance Characterization, custom event analysis

• Perf-based collection limitations:
  • Memory Access analysis is enabled with perf starting 2017 Gold
  • To enable uncore event collections, system wide collection set under sudo/root:
    > echo 0 > /proc/sys/kernel/perf_event_paranoid
    Pre-requisite for memory access analysis upon perf
    Switch to system-wide mode with "analyze-system" option for application to launch to avoid perf collection overhead
  • To collect General Exploration or custom event collection with big number of events increase default limit of opened file descriptors:
    In /etc/security/limits.conf increase default number to 100 * <number_of_logic_CPU_cores>:
    <user> hard nofile <100 * number_of_logic_CPU_cores>
    <user> soft nofile <100 * number_of_logic_CPU_cores>

• Default sampling interval on KNL is 10ms
Analysis Configuration (1/4)
How to Run VTune on MPI Applications

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

NB: -r <result_dir> is mandatory option for MPI profiling

• 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

• Enabled for perf-based collection from 2017 Gold

  • Highly recommended to set `/proc/sys/kernel/perf_event_paranoid=0` to allow system wide collection to capture multiple ranks in profiling per node w/o overhead
Selecting 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-hotapots –r result_dir:3,7,11-13” ./my_app
1. Create a VTune project
2. Choose “Arbitrary Targets/Local”
3. Set processor arch and OS
4. Set application name and parameters
5. Check “Use MPI Launcher”

Provide the launcher name, number of ranks, ranks to profile, set result directory
Analysis Configuration (4/4)

MPI Profiling Command Line Generation from GUI

6. Choose analysis type

7. Generate command line
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>
   ```
Memory Access Analysis

Motivation

- **More ranks/threads** to load many-core processor -> **more data** fetching to saturate computations
- Faster reaching DRAM bandwidth limit (~90GB/s) for BW bound apps and it hurts performance
- On-package MCDRAM (400Gb/s+) helps to extend BW limit and increase performance
  - MCDRAM in cache mode is transparent for a user, might bring additional latency if working set does not suit MCDRAM size (16GB)
  - MCDRAM in flat mode is more flexible – user can control what data to allocate to MCDRAM
    - If working set suits 16GB – use “numactl –m 1” to allocate all data to MCDRAM
    - If working set is more than 16GB – use memkind lib to allocate data structures that induce L2 (LLC on KNL) miss traffic

**USE VTUNE TO DEFINE PROPER DATA STRUCTURES FOR MCDRAM ALLOCATION**
Memory Access Analysis

Configuration options:

- **Analyze memory objects:**
  - Enables the instrumentation of memory allocation/de-allocation and mapping hardware events to memory objects
  - May cause additional runtime overhead due to the instrumentation of all system memory allocation/de-allocation API

- **Minimal memory object size to track, in bytes:**
  - Specify a minimal size of memory allocations to analyze. This option helps reduce runtime overhead of the instrumentation

- **“Evaluate max DRAM Bandwidth” is not enabled on KNL, limits hardcoded:**
  - DRAM: 90GB/s
  - MCDRAM 350GB/s
  - Can be adjusted on BW histogram per result if needed

> `amplxe-cl -c memory-access -knob analyze-mem-objects=true -knob mem-object-size-min-thres=1024 -- <app>`
Memory Access Analysis

High Bandwidth Analysis. Step 1

Explore DRAM Bandwidth Bound metric and histogram on summary to see if the app is bandwidth bound.

Significant portion of application time spent in high memory bandwidth utilization.

The app may benefit from MCDRAM.
Memory Access Analysis
High Bandwidth Analysis. Step 2

Investigate the memory allocations inducing bandwidth

- “Bandwidth Domain/Bandwidth Utilization Type/Memory Object/Allocation Stack” grouping with expansion by “DRAM/High” and sorting by L2 Miss Count

Focus on allocations inducing L2 misses
Allocation stack shows the allocation place in user’s code
Memory Access Analysis

Allocate the data structures to MCDRAM using High Bandwidth Memory

E.g.: specifying a custom memory allocator class from memkind lib for stored vector elements

Or using "numactl -m 1" if the whole working set suits 16 GB
Memory Access Analysis
High Bandwidth Analysis. Step 4
Rerun the benchmark

DRAM bandwidth significantly decreased
reducing DRAM memory access stalls
Micro-arch analysis with General Exploration

- Execution pipeline slots distribution by Retiring, Front-End, Back-End, Bad Speculation
- Second level metrics for each aspect of execution pipeline to understand the reason of stalls
Performance Overview with HPC Performance Characterization

Motivation

Show important aspects of application performance in one analysis

- Entry point to assess application efficiency on system resources utilization with definition of the next steps to investigate pathologies with significant performance cost
- Monitor how parameters of a run or code changes impact important different performance aspects to better understand their impact on elapsed time

Customers asking

- I eliminated imbalance with dynamic scheduling but elapsed time of my application became worse, why?
- I vectorized the code but don’t have much benefit, why?
- I moved from pure MPI to MPI + OpenMP but the results are much worse, why?

CPU Utilization, Memory efficiency and FPU utilization aspects are correlated – let’s explore them in one view

- Run HPC Performance Characterization analysis
Performance Overview with HPC Performance Characterization

Motivation

CL

\texttt{>amplxe-cl \textasciitilde collect hpc-performance <options> ./my_app}

\texttt{>mpirun \textasciitilde n 16 \textasciitilde ppn 4 \textasciitilde collect hpc-performance \textasciitilde r result_dir ./my_mpi_app}

GUI

Choose Target and Analysis Type

- **Analysis Target**
  - Basic Hotspots
  - Advanced Hotspots
  - Concurrency
  - Locks and Waits
  - HPC Performance Characterization (preview)
  - Microarchitecture Analysis
  - General Exploration

HPC Performance Characterization (preview)

Preview feature - should we keep it, change it, or drop it? Send us your comments.

Analyze important aspects of your application performance, including CPU utilization with additional details on OpenMP efficiency analysis, memory usage, and FPU utilization with vectorization information. For vectorization optimization data, such as trip counts, data dependencies, and memory access patterns, try Intel Advisor. It identifies the loops that will benefit the most from refined vectorization and gives tips for improvements. The HPC Performance Characterization analysis type is best used for analyzing intensive compute applications. Learn more (F1)
Performance Overview with HPC Performance Characterization
Analysis Structure and Metrics

Two characterization metrics

- Elapsed Time
- GFLOPs Upper Bound*

Three performance aspects

- CPU Utilization
- Memory Bound
- FPU Utilization Upper Bound*

*Calculated based on FLOP HW counters assuming full vector utilization
Performance Overview with HPC Performance Characterization

CPU Utilization

- % of “Effective” 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

Metrics in CPU utilization section

- Average CPU usage
- Additional MPI and OpenMP scalability metrics impacting effective CPU utilization
- CPU utilization histogram
Performance Overview with HPC Performance Characterization

Memory Bound

Since no memory stall measurement on KNL “Memory Bound” high level metric replaced with Backed-Bound with second level based on misses and bandwidth measurement from uncore events:

- **L2Hit Bound**
  - Cost of L1 misses served in L2

- **L2 Miss Bound**
  - Cost of L2 misses

- **MCDRAM Bandwidth Bound**
  - % of app elapsed time consuming high MCDRAM Bandwidth

- **MCDRAM Bandwidth Bound**
  - % of app elapsed time consuming high MCDRAM Bandwidth

- Bandwidth utilization histogram
Performance Overview with HPC Performance Characterization

FPU Utilization Upper Bound

- % of FPU load (100% - FPU is fully loaded, threshold 50%)

Metrics in FPU utilization section

- GLOPs broken down by scalar and packed
- Top 5 loops/functions by FPU usage
  - Dynamically generated issue descriptions on low FPU usage help to define the reason and next steps:
    - Non-vectorized, vectorized with legacy instruction set, memory bound limited loops not benefiting from vectorization etc.
  - Vector instruction set, FP ratio and FLOPs per cycle are available in Grid per rank/loop/region
Performance Overview with HPC Performance Characterization

FPU Utilization

Since on KNL only SIMD scalar and vector instruction count enabled w/o vector utilization info we show FLOP Upper Bound metrics assuming full vector utilization. FMA is counted as single instruction so FLOPs multiplied by 2

- % of FPU load (100% - FPU is fully loaded, threshold 50%)

Metrics in FPU utilization section

- GLOPs broken down by scalar and packed
- Top 5 loops/functions by FPU usage
  - Dynamically generated issue descriptions on low FPU usage help to define the reason and next steps: Non-vectorized, vectorized with legacy instruction set, memory bound limited loops not benefiting from vectorization etc.
  - Vector instruction set and FLOPs, SIMD ratio, Upper Bound per cycle are available in Grid per rank/loop/region

<table>
<thead>
<tr>
<th>Function</th>
<th>CPU Time</th>
<th>FPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop at line 573 in minFE: matvec_std(minFE::CSRMatrix&lt;double, int, long, long, int, long, long, op scatter])</td>
<td>1343.955s</td>
<td>1.4% Vectorized (Remainder)</td>
</tr>
<tr>
<td>Loop at line 573 in minFE: matvec_std(minFE::CSRMatrix&lt;double, int, long, long, int, long, long, op scatter])</td>
<td>932.465s</td>
<td>0.4% Vectorized (Body)</td>
</tr>
<tr>
<td>Loop at line 570 in minFE: matvec_std(minFE::CSRMatrix&lt;double, int, long, long, int, long, long, op scatter])</td>
<td>227.255s</td>
<td>8.8% Scalar (Body)</td>
</tr>
<tr>
<td>native_un automobile</td>
<td>129.108s</td>
<td>0.2%</td>
</tr>
<tr>
<td>Loop at line 152 in minFE: wapshy_std(minFE::Vector&lt;double, int, long, long, op scatter])</td>
<td>113.482s</td>
<td>0.1% Scalar (Body)</td>
</tr>
<tr>
<td>Others</td>
<td>458.008s</td>
<td>N/A</td>
</tr>
</tbody>
</table>

*FMA is applied to non-removable metrics.
Basic Hotspots, Concurrency, Locks and Waits

• Basic Hotspots, Concurrency, Locks and Waits use instrumentation and might bring significant overhead on runs with big number of threads

• Recommendation:
  
  • Use Advanced Hotspots instead of Basic Hotspots
    • Switch on stacks in Advanced Hotspots configuration if absolutely needed
      >amplxe-cl –collect advanced-hotspots –knob collection-detail=stack-sampling ./my_app
  
  • Use HPC Performance Characterization instead of Concurrency and Locks and Waits to assess CPU utilization of throughput applications
    • Only if you have different kinds of locks in one lexical OpenMP region and want to understand each lock impact – use Locks and Waits and be ready to significant collection overhead
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