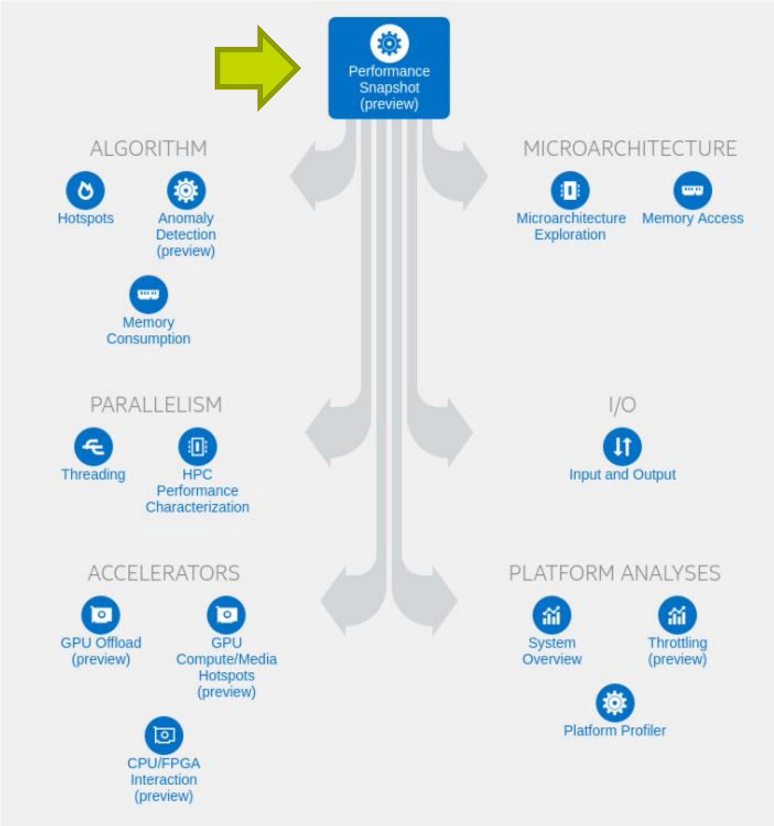
INTEL® VTUNE™ PROFILER
A QUICK INTRO FOR HPC DEVELOPERS

Vladimir Tsymbal
Rev. 09/2020

Agenda

- Intel® VTune™ Profiler 2020 analysis types overview
- Basic analysis: Performance Snapshot and Hotspots
- HPC Performance Characterization analysis for hybrid MPI + OpenMP apps
- How to run VTune Profiler on MPI applications
- Memory Access analysis
- Accelerators (GPU and FPGA) analysis

Intel® VTune™ Profiler 2020 Analysis Types



Start with

- Command line
- Within a GUI

Which type of analysis

- Performance Snapshot for initial assessment
- With little experience you know which one to use

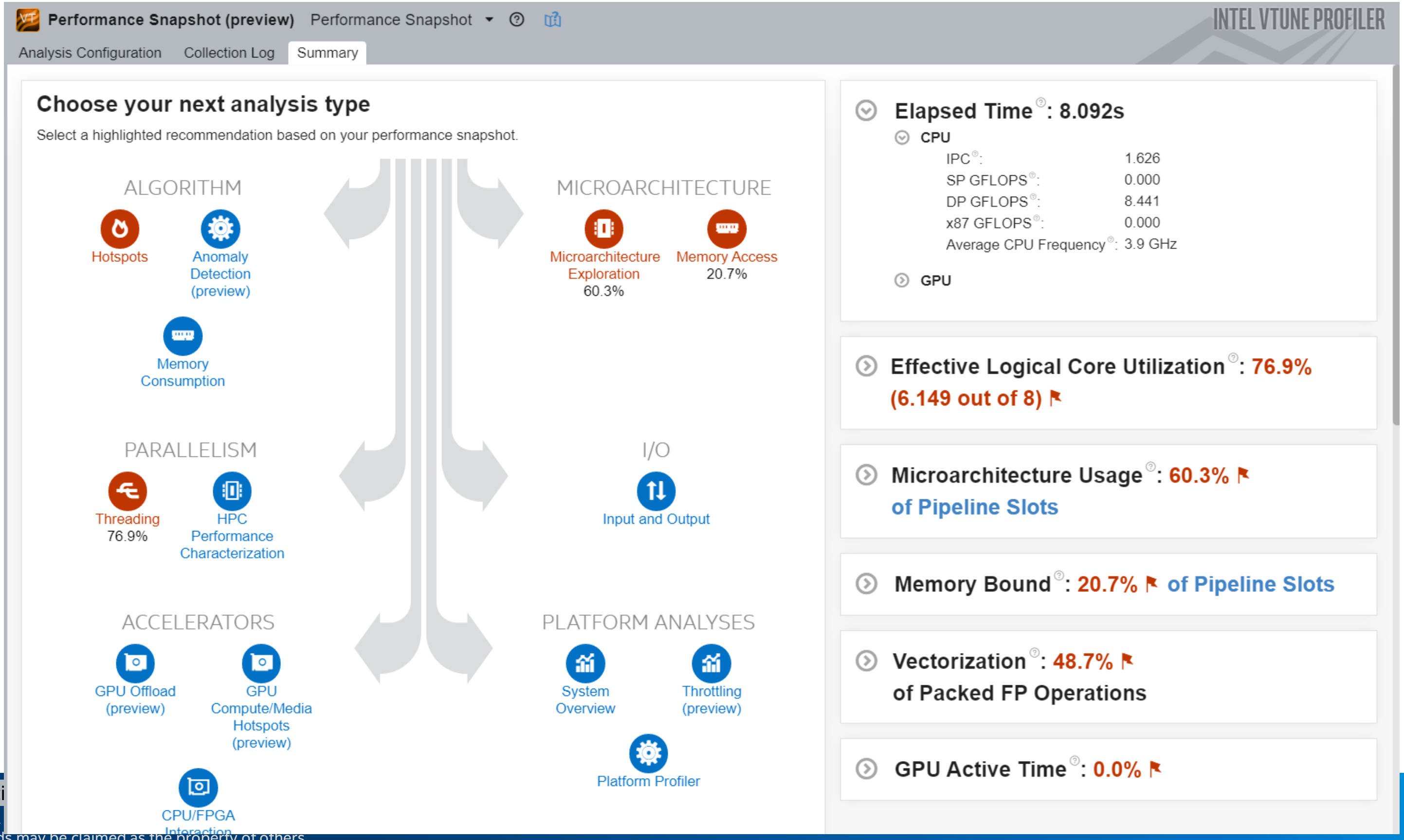
Simplest command line

- **vtune –collect performance-snapshot ./my_app**
- Almost every GUI functionality is available in CL

Launching analysis in a cluster environment

- The same CL, but wrapped into a mpi launcher commands

VTune Performance Snapshot



Performance Snapshot report in CL

vtune -collect performance-snapshot ./my_app

```
Memory Bound: 77.5% 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
| stores. Use Memory Access analysis to have the metric breakdown by memory
| hierarchy, memory bandwidth information, correlation by memory objects.
|
    L1 Bound: 0.0% of Clockticks
    L2 Bound: 0.0% of Clockticks
    L3 Bound: 60.2% of Clockticks
     | This metric shows how often CPU was stalled on L3 cache, or contended
     | with a sibling Core. Avoiding cache misses (L2 misses/L3 hits) improves
     | the latency and increases performance.
     |
    DRAM Bound: 13.1% of Clockticks
     | This metric shows how often CPU was stalled on the main memory (DRAM).
     | Caching typically improves the latency and increases performance.
     |
        DRAM Bandwidth Bound: 0.0% of Elapsed Time
    Store Bound: 0.0% of Clockticks
    NUMA: % of Remote Accesses: 20.9%
     | A significant amount of DRAM loads were serviced from remote DRAM.
     | Wherever possible, try to consistently use data on the same core, or at
     | least the same package, as it was allocated on.
```

HPC Performance Characterization Analysis

Show important aspects of application performance in one analysis

- Entry point to assess application efficiency with definition of the next steps to investigate events with significant performance cost

Threading: CPU Utilization

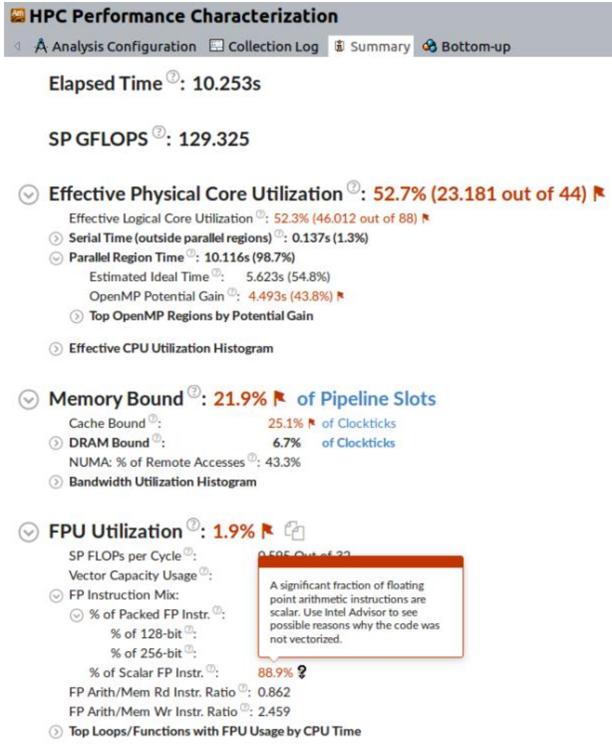
- Serial vs. Parallel time
- Top OpenMP regions by potential gain
- Tip: Use Hotspot OpenMP region analysis for more detail

Memory Access Efficiency

- Stalls by memory hierarchy
- Bandwidth utilization
- Tip: Use Memory Access analysis

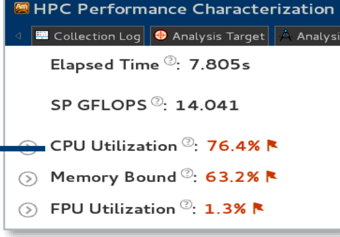
Vectorization: FPU Utilization

- FLOPS † estimates from sampling
- Tip: Use Intel Advisor for precise metrics and vectorization optimization




```
>vtune –collect hpc-performance –data-limit=0 –r result_dir ./my_app
```

Performance Aspects: CPU Utilization (1/4)

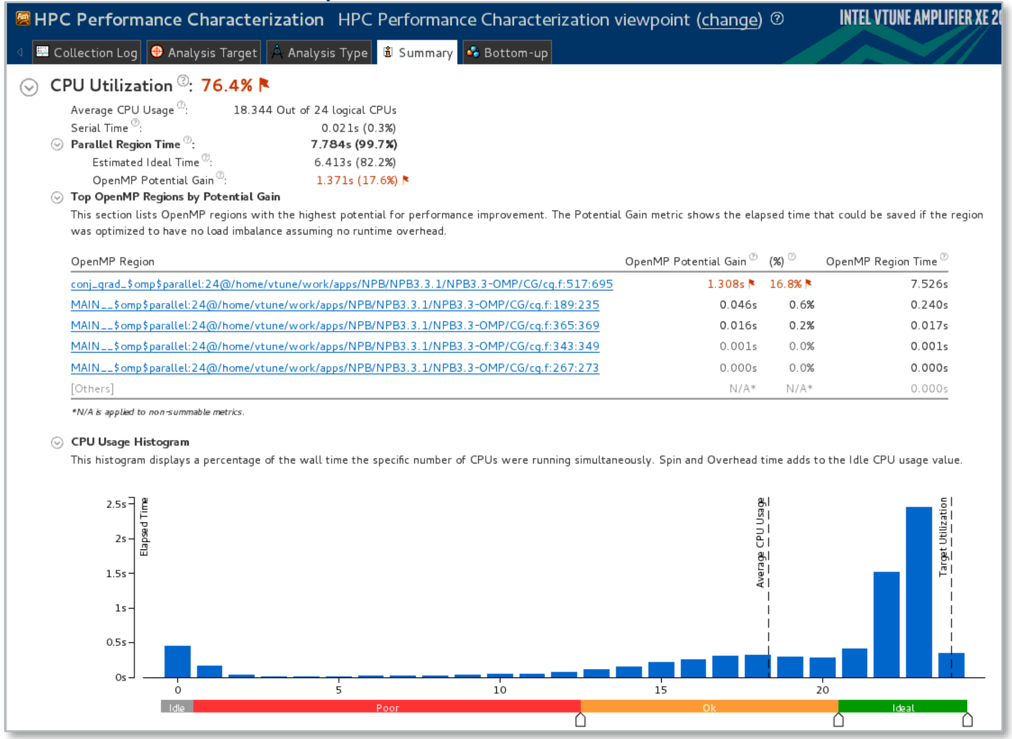


CPU Utilization

- % 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
- Intel OpenMP scalability metrics impacting effective CPU utilization
- CPU utilization histogram



Performance Aspects: CPU Utilization (2/4)

VTune Profiler OpenMP* Analysis: answering on customers' questions about performance in the same language a program was written in

Is serial time of my application significant to prevent scaling?

Serial Time (outside any parallel region): 4.020s (27.7%)

Serial Time of your application is high. It directly impacts application Elapsed Time and scalability. Explore options for parallelization, algorithm or microarchitecture tuning of the serial part of the application.

How efficient is my parallelization towards ideal parallel execution?

Parallel Region Time: 10.469s (72.3%)

Estimated Ideal Time: 7.115s (49.1%)

How much theoretical gain I can get if invest in tuning?

Potential Gain: 3.354s (23.1%)

The time wasted 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..

Top OpenMP Regions by Potential Gain

This section lists OpenMP regions with the highest potential for performance improvement. The Potential Gain metric shows the elapsed time that could be saved if the region was optimized to have no load imbalance assuming no runtime overhead.

OpenMP Region	Potential Gain	(%)	Elapsed Time
conj_grad_ompparallel:24@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:514:695	3.294s	22.7%	10.208s
MAIN__ompparallel:24@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:185:231	0.059s	0.4%	0.260s
MAIN__ompparallel:24@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:339:345	0.001s	0.0%	0.001s
MAIN__ompparallel:24@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:361:365	0.001s	0.0%	0.001s
MAIN__ompparallel:24@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:263:269	0.000s	0.0%	0.000s
[Others]	0.000s	0.0%	0.000s

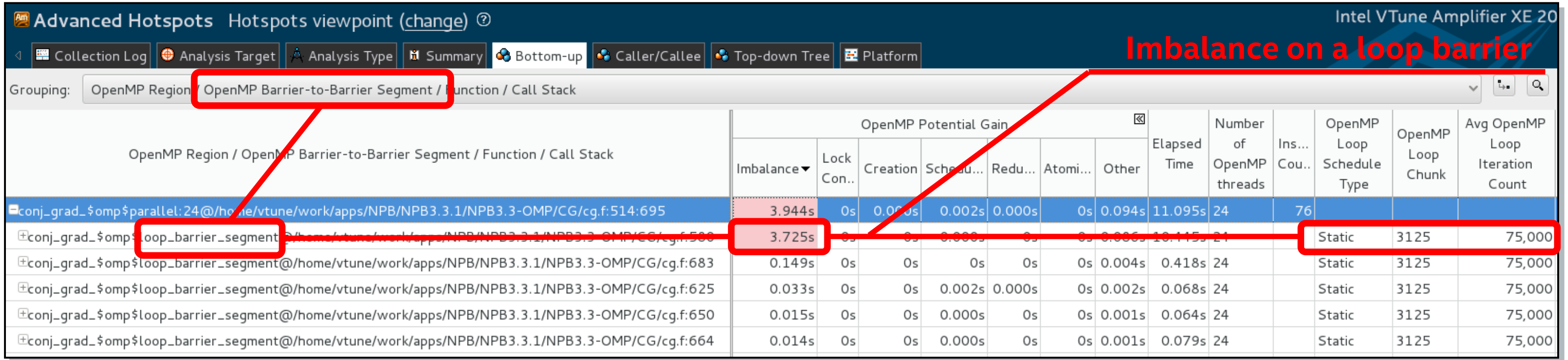
What regions are more perspective to invest?

Links to grid view for more details on inefficiency

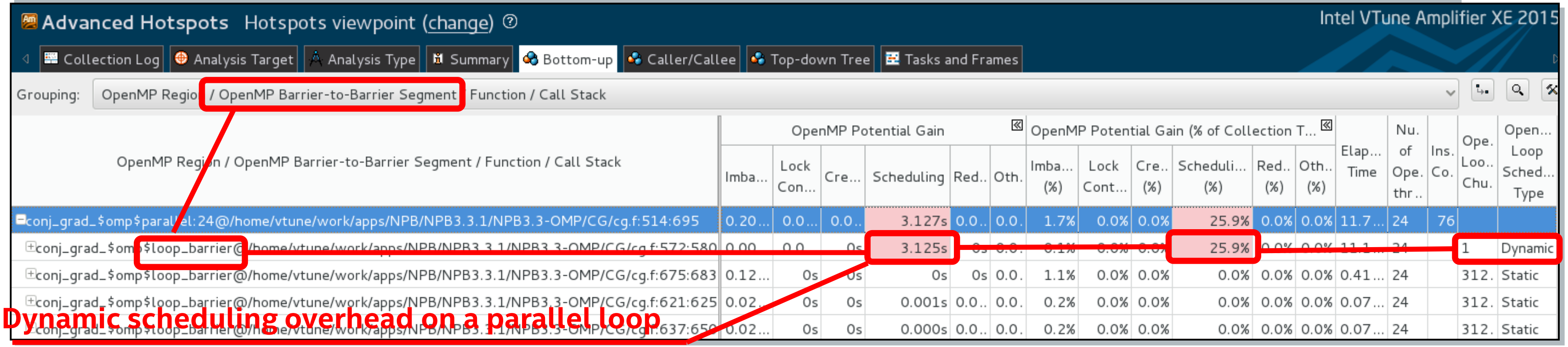
Performance Aspects: CPU Utilization (3/4)

Details in grid view: expansion by constructs with a barrier inside the region

Imbalance distribution by loop, single, reduction, user, join barriers



OpenMP Region / OpenMP Barrier-to-Barrier Segment / Function / Call Stack	OpenMP Potential Gain: Imbalance	Lock Con..	Creation	Schedu...	Redu...	Atomi...	Other	Elapsed Time	Number of OpenMP threads	Ins... Cou..	OpenMP Loop Schedule Type	OpenMP Loop Chunk	Avg OpenMP Loop Iteration Count
conj_grad_ompparallel:24@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:514:695	3.944s	0s	0.000s	0.002s	0.000s	0s	0.094s	11.095s	24	76			
conj_grad_omploop_barrier_segment@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:580	3.725s	0s	0s	0.000s	0s	0s	0.086s	10.445s	24		Static	3125	75,000
conj_grad_omploop_barrier_segment@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:683	0.149s	0s	0s	0s	0s	0s	0.004s	0.418s	24		Static	3125	75,000
conj_grad_omploop_barrier_segment@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:625	0.033s	0s	0s	0.002s	0.000s	0s	0.002s	0.068s	24		Static	3125	75,000
conj_grad_omploop_barrier_segment@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:650	0.015s	0s	0s	0.000s	0s	0s	0.001s	0.064s	24		Static	3125	75,000
conj_grad_omploop_barrier_segment@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:664	0.014s	0s	0s	0.000s	0s	0s	0.001s	0.079s	24		Static	3125	75,000

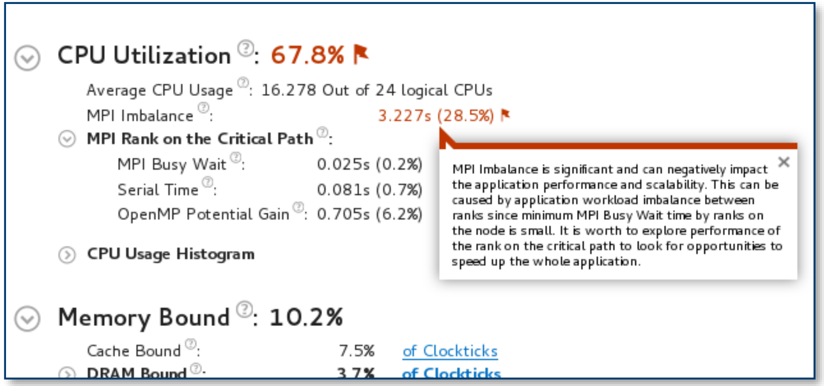


OpenMP Region / OpenMP Barrier-to-Barrier Segment / Function / Call Stack	OpenMP Potential Gain: Imba...	Lock Con...	Cre...	Scheduling	Red..	Oth.	OpenMP Potential Gain (% of Collection T...: Imba... (%)	Lock Cont...	Cre.. (%)	Scheduli... (%)	Red.. (%)	Oth.. (%)	Elap... Time	Nu. of Ope. thr..	Ins. Co.	Ope. Loo.. Chu.	Open... Loop Sched... Type
conj_grad_ompparallel:24@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:514:695	0.20...	0.0...	0.0..	3.127s	0.0..	0.0.	1.7%	0.0%	0.0%	25.9%	0.0%	0.0%	11.7...	24	76		
conj_grad_omploop_barrier@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:572:580	0.00...	0.0...	0s	3.125s	0s	0.0.	0.1%	0.0%	0.0%	25.9%	0.0%	0.0%	11.1...	24		1	Dynamic
conj_grad_omploop_barrier@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:675:683	0.12...	0s	0s	0s	0s	0.0.	1.1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.41...	24		312.	Static
conj_grad_omploop_barrier@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:621:625	0.02...	0s	0s	0.001s	0.0..	0.0.	0.2%	0.0%	0.0%	0.0%	0.0%	0.0%	0.07...	24		312.	Static
conj_grad_omploop_barrier@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:637:650	0.02...	0s	0s	0.000s	0.0..	0.0.	0.2%	0.0%	0.0%	0.0%	0.0%	0.0%	0.07...	24		312.	Static

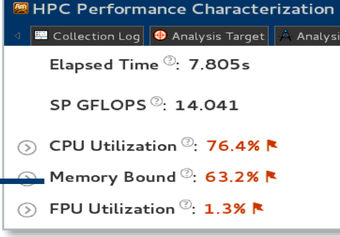
Performance Aspects: CPU Utilization (4/4)

Specifics for hybrid MPI + OpenMP apps

- MPI Imbalance metric for the node as average MPI busy wait time by rank (based on MPI progress sampling for MPICH-based MPIs (Intel MPI, CRAY MPI, .._)
  - Serial (outside any parallel region) and OpenMP Potenital Gain time for the MPI rank on critical path for the node (with minimal MPI busy wait per the node)
  - Worth to explore details on serial hotspots and OpenMP inefficiencies for the MPI rank on critical path in grid view



Performance Aspects: Memory Bound

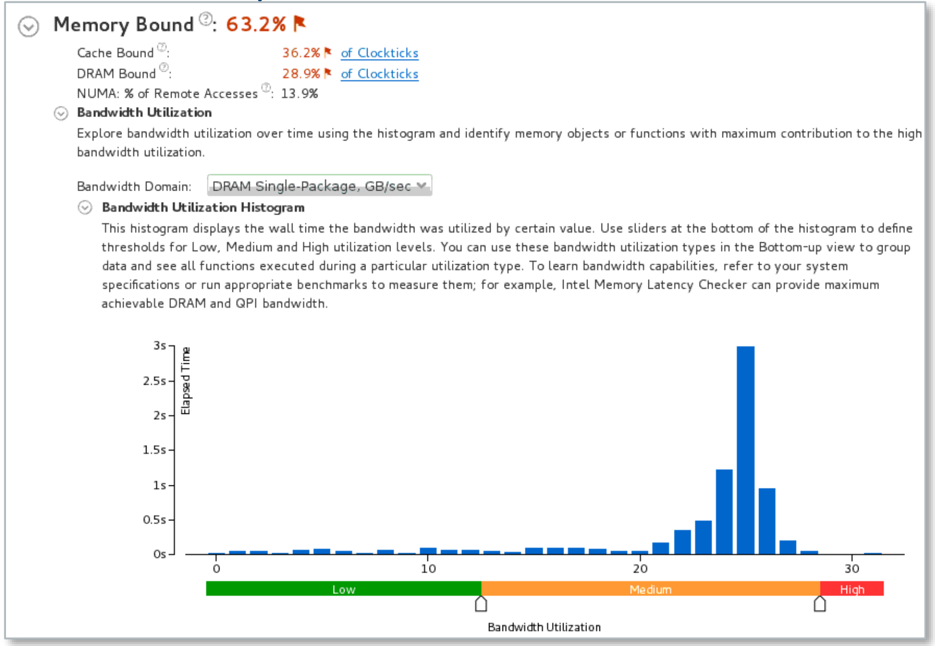


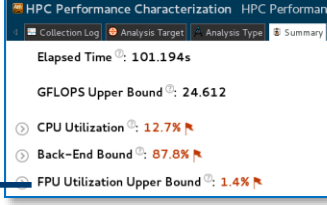
Memory Bound

- % of potential execution pipeline slots lost because of fetching memory from different levels of hierarchy (threshold 20%)

Metrics in Memory Bound section

- Cache bound
- DRAM bound
  - Issue description specifies if the code is bandwidth or latency bound with proper advice of how to fix
  - % of app elapsed time consuming high DRAM Bandwidth
- NUMA: % of remote accesses
  - Important to explore if the code is bandwidth bound
- Bandwidth utilization histogram





Performance Aspects: FPU Utilization

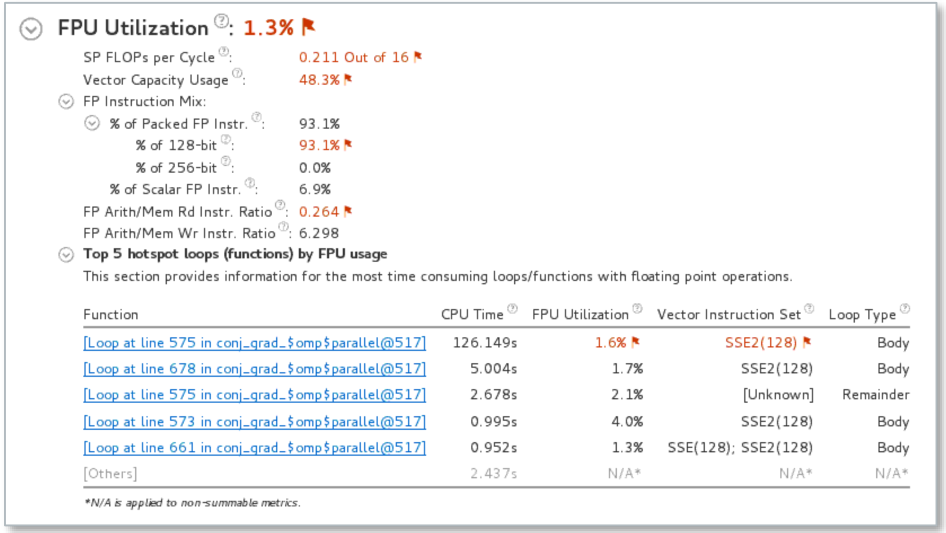
FPU utilization

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

Metrics in FPU utilization section

- SP FLOPs per Cycle (vector code generation and execution efficiency)
- Vector Capacity Usage and FP Instruction Mix, FPArith/Mem ratios (vector code generation efficiency)
- 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.



Performance Aspects: Process/Thread Affinity

Process/Thread Affinity Collection and Reporting

- Visualization of affinity masks collected at the end of application
- Command Line Report with threads by cores affinity
- Experimental GUI report with affinity shown along with thread execution on CPU and remote accesses

How to

>vtune –c h-p –knob collect-affinity=true –r my_result ./my_app

my_result_11.10.125.99.54.html

CL report:

>vtune –R affinity –r my_result

my_result_11.10.125.99.54.html

GUI report (tech preview):

>vtune –R affinity -format=html –r my_result

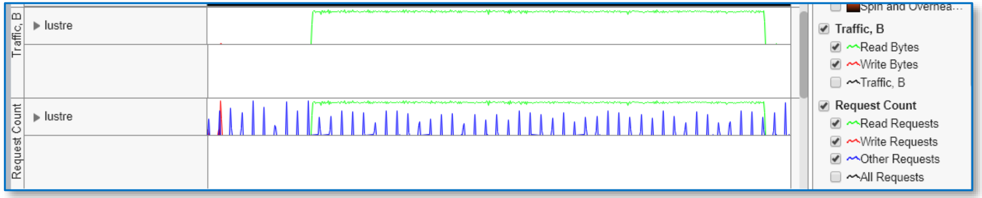
r003hpc.html

Performance Aspects: Lustre Parallel File I/O

- Key Metrics:
  - I/O Wait Time, I/O Bandwidth, Packet Rate, Packet Size
- Availability:
  - Average on summary
  - Overtime on timeline with ability to
    filter by time for context summary
- Details by Parallel File System Shares
- To enable set “AMPLXE_EXPERIMENTAL=lustre”, add “-knob parallel-fs-collection=true” to hpc-performance command line

Lustre File System IO Wait Time: 64.569s

Bandwidth, B/s:	6,011,995.982
Package Rate, pack/s:	25.750
Average Package Size, KB:	233



Parallel File System detailed info:

Parallel File System	Read Bytes	Read Requests	Read Wait Time	Write Bytes	Write Requests	Write Wait Time	All Requests	Request Wait Time
/lfs/lfs09	6,161,204,090	26,379	48.513s	381,814	12	0.011s	27,086	49.799s
/lfs/lfs12	0	0	0s	0	0	0s	6,058	11.961s
/lfs/lfs13	0	0	0s	0	0	0s	963	2.020s
/lfs/lfs11	0	0	0s	0	0	0s	635	0.789s

N/A is applied to non-summable metrics.

HPC Performance Characterization – Command Line Reporting

- Generated after collection is done or with “-R summary” option of vtune
- Mimicking GUI metrics hierarchy

```
Elapsed Time: 7.805s
SP GFLOPS: 14.041
CPU Utilization: 76.4%
 | The metric value is low, which may signal a poor logical CPU cores
 | utilization caused by load imbalance, threading runtime overhead, contended
 | synchronization, or thread/process underutilization. Explore CPU Utilization
 | sub-metrics to estimate the efficiency of MPI and OpenMP parallelism or run
 | the Locks and Waits analysis to identify parallel bottlenecks for other
 | parallel runtimes.
 |
    Average 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: 6.413s (82.2%)
        OpenMP Potential Gain: 1.371s (17.6%)
         | The time wasted 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
 | stores. Use Memory Access analysis to have the metric breakdown by memory
 | hierarchy, memory bandwidth information, correlation by memory objects.
 |
    Cache Bound: 36.2% of Clockticks
     | A significant proportion of cycles are being spent on data fetches from
     | caches. Check Memory Access analysis to see if accesses to L2 or L3
     | caches are problematic and consider applying the same performance tuning
     | as you would for a cache-missing 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
     | contested accesses or data sharing are indicated as likely issues.
     |
    DRAM Bound: 28.9% of Clockticks
     | The metric value is high. This indicates that a significant fraction of
     | cycles could be stalled on the main memory (DRAM) because of demand loads
     | or stores.
     |
     | The code is memory bandwidth bound, which means that there are a
     | significant fraction of cycles during which the bandwidth limits of the
     | main memory are being reached and the code could stall. Review the
     | Bandwidth Utilization Histogram to estimate the scale of the issue.
     | Consider improving data locality on NUMA multi-socket systems, which will
     | reduce code memory bandwidth consumption.
     |
    NUMA: % of Remote Accesses: 13.9%
```

With issue descriptions that can be suppressed by “–report-knob show-issues=false” option

```
Elapsed Time: 7.805s
SP GFLOPS: 14.041
CPU Utilization: 76.4%
    Average 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: 6.413s (82.2%)
        OpenMP Potential Gain: 1.371s (17.6%)
Memory Bound: 63.2% of Pipeline Slots
    Cache Bound: 36.2% of Clockticks
    DRAM Bound: 28.9% of Clockticks
    NUMA: % of Remote Accesses: 13.9%
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
Collection and Platform Info
    Application Command Line: ./cg.B.x
    User Name: vtune
```

How to Run VTune on MPI Applications

Single or multiple node application launch:

<mpi_launcher> – n N <vtune_command_line> ./app_to_run

- >aprun –n 48 -N 16 vtune –collect hotspots **–trace-mpi** –r result_dir ./my_mpi_app
- >mpirun –n 48 -ppn 16 vtune –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
  - **–trace-mpi** option is mandatory to enable per-node result directories for MPIs not using PMI_RANK (e.g. CRAY MPI)
- Works for software and driver-based/driverless collectors
  - Highly recommended to set /proc/sys/kernel/perf_event_paranoid to 0 for perf-based collections to avoid collection overhead on modern multi-/many-core systems

Selective Rank profiling of MPI applications with VTune

To reduce data collected by VTune on big scale runs use selective rank profiling using MPMD (Multiple Program Multiple Data) runs:

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 to simplify the launch configuration

Example: profile ranks 2, 3, 10-15 from 0-15:

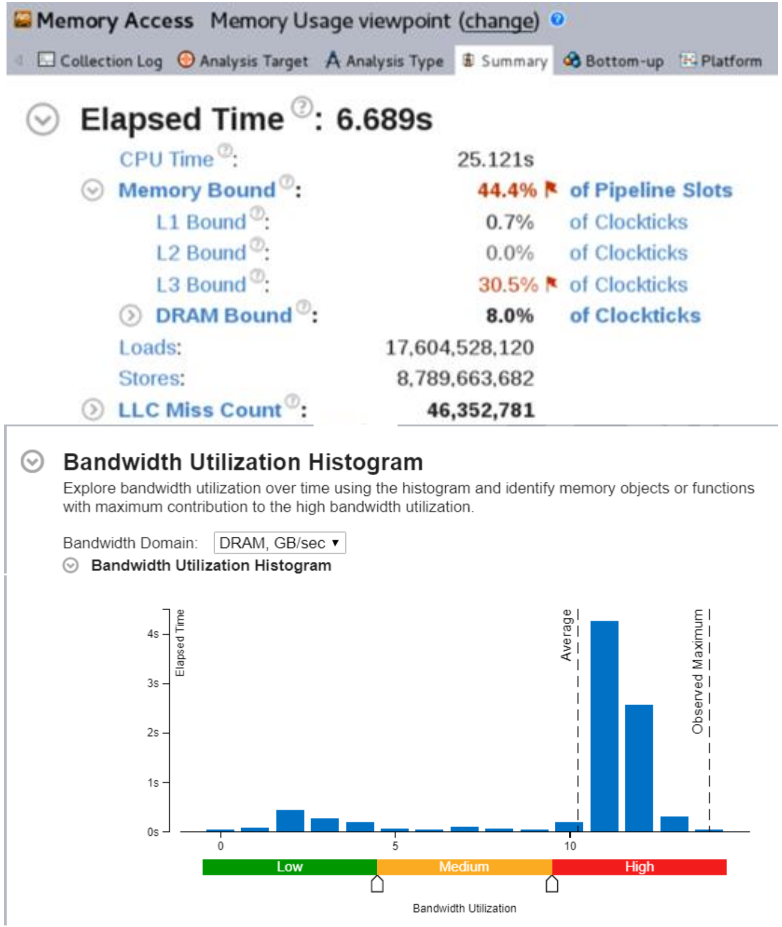
>mpirun –gtool “vtune –collect hotspots –r result_dir:2,3,10-15” ./my_app

Memory Access Analysis

Specialized analysis type “Memory Access” to focus on memory-related issues:

- Performance problems by memory hierarchy (e.g. L1-, L2-, LLC-, DRAM-bound)
- Bandwidth-limited accesses
  - Covers DRAM, QPI, and PMEM bandwidth
- NUMA problems
- Performance metrics by memory objects (data structures)

>vtune –collect ***memory-access*** *./my_app*



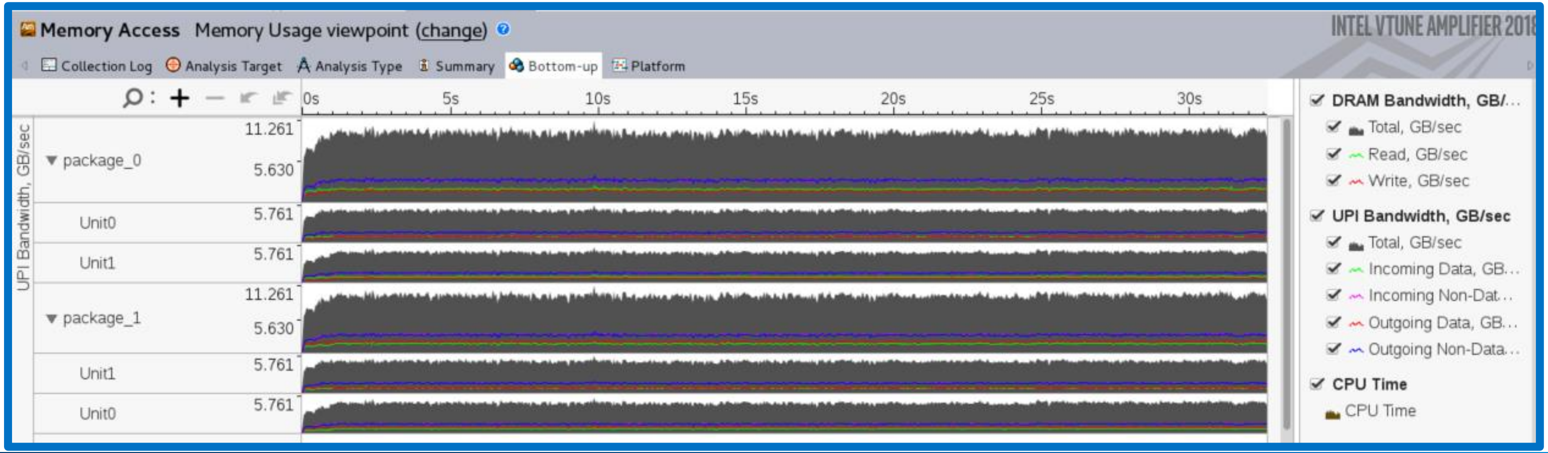
Memory Bandwidth Analysis

DRAM and PMEM bandwidth

- Can be expanded per package/channel
- Split to Read/Write
- Automatic DRAM pick measurement on Xeon

QPI bandwidth per-package

- Can be expanded per-link
- Split to Incoming/Outgoing and Data/Non-Data

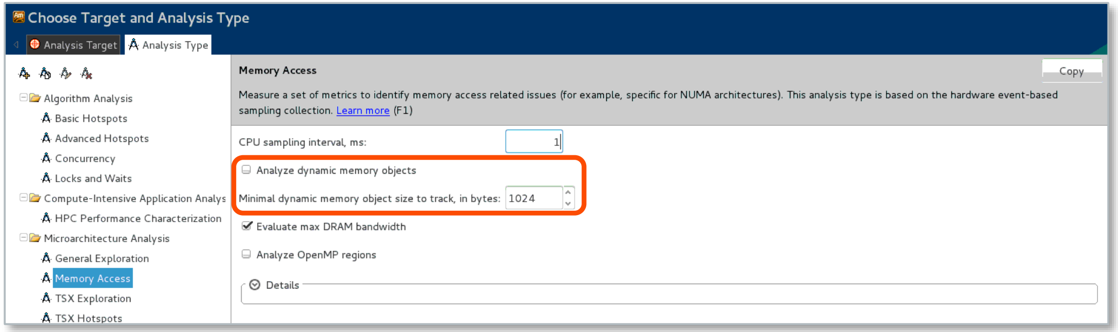


View Performance Metrics by Memory Objects

Memory Access analysis configuration options for profiling memory objects:

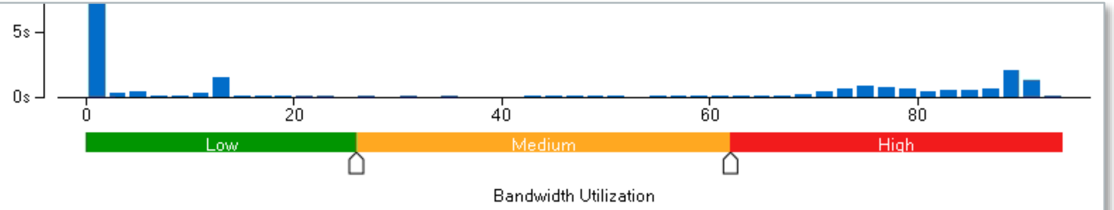
- **Analyze dynamic memory objects:** enables the instrumentation of memory allocation/de-allocation and map hardware events to memory objects
- **Minimal memory object size to track, in bytes:** specify a minimal size of memory allocations to analyze. This option helps reduce runtime overhead of the instrumentation. Default is 1K.

*>vtune -c memory-access -knob **analyze-mem-objects=true -knob mem-object-size-min-thres=2048** -- <app>*



High Memory Bandwidth Analysis by Memory Objects

If memory object instrumentation is ON Bandwidth Utilization histogram is enriched with the table of Memory Objects by LLC Miss Count



Top Memory Objects with High Bandwidth Utilization

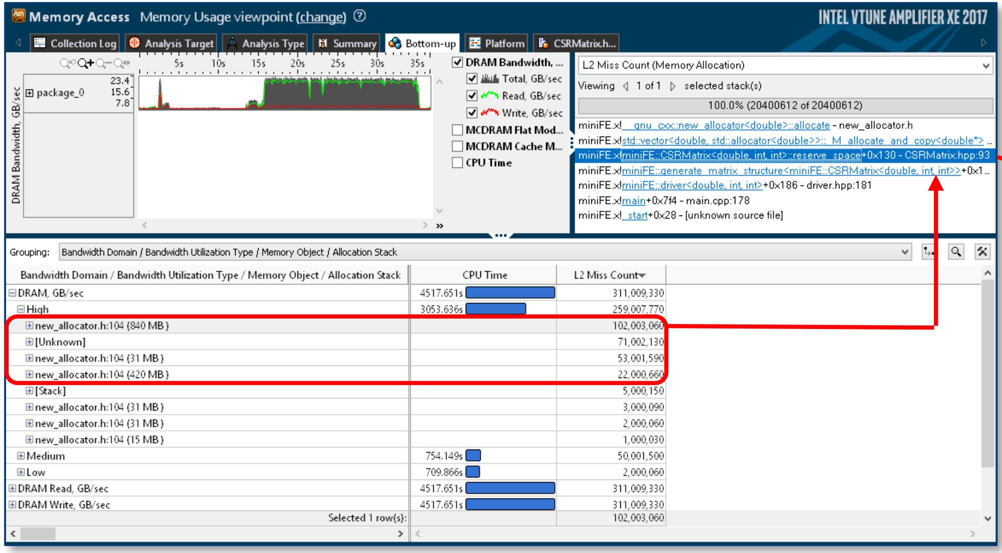
This section shows top memory objects, sorted by LLC Misses, that were accessed when bandwidth utilization was high for the domain selected in the histogram area.

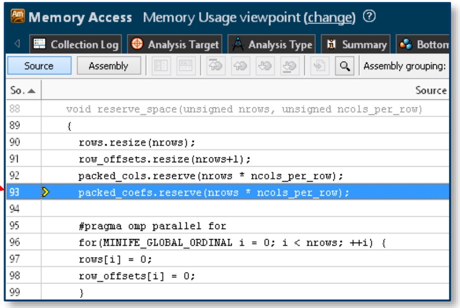
Memory Object	LLC Miss Count
new_allocator.h:104 (61 MB)	42.6%
new_allocator.h:104 (1 GB)	23.3%
new_allocator.h:104 (836 MB)	12.4%
[Unknown]	10.1%
new_allocator.h:104 (61 MB)	6.2%
[Others]	2.3%

High Memory Bandwidth Analysis by Memory Objects

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 Count

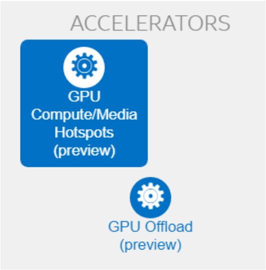




Focus on allocations inducing L2 misses

Allocation stack shows the allocation place in user’s code

GPU Application Analysis

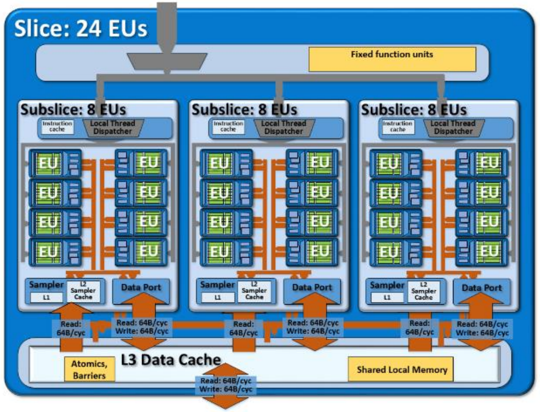


GPU Compute/Media Hotspots

- Visibility into both host and GPU sides
- HW-events based performance tuning methodology
- Provides overtime and aggregated views

GPU In-kernel Profiling

- GPU source/instruction level profiling
- SW instrumentation
- Two modes: basic Block latency and Memory access latency

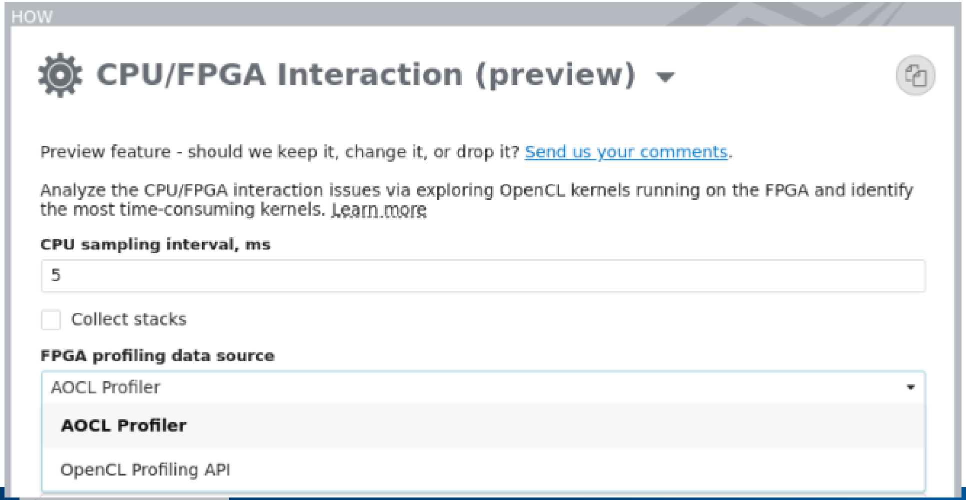


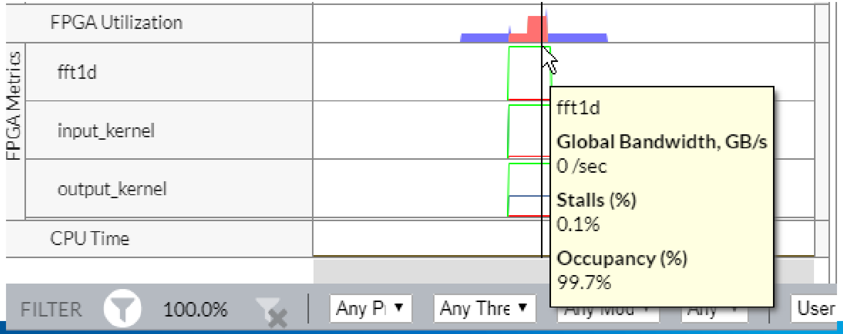
Identify GPU occupancy and which kernel to profile.
Tune a kernel on a fine-grain level.

CPU/FPGA Interaction Analysis

Now process data sources collected with AOCL Profiler (new mode) in addition to OpenCL Profiling API (legacy mode).

Extended with FPGA device-side metrics, like Stalls, Global Bandwidth and Occupancy, and mapping FPGA kernel performance data to the source code





intel®

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.

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