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

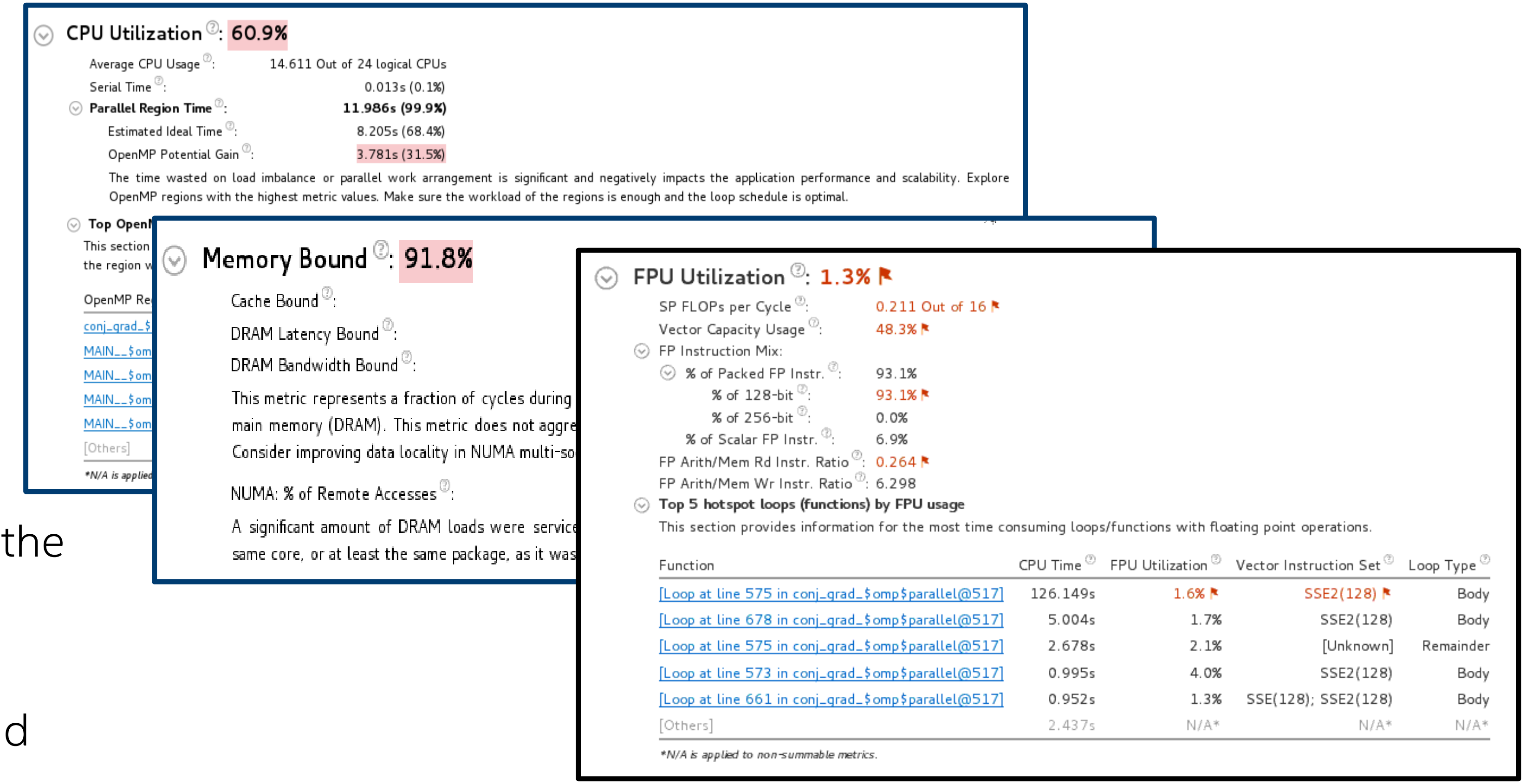
WE KNOW OUR ENEMY, HOW DO WE DEFEAT IT?

SEGMENT SPECIFIC PERFORMANCE ANALYSIS: HPC CHARACTERIZATION

THREE METRICS CLASSES

Three Metric 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



CPU Utilization: 60.9%

Average CPU Usage: 14.611 Out of 24 logical CPUs

Serial Time: 0.013s (0.1%)

Parallel Region Time: 11.986s (99.9%)

Estimated Ideal Time: 8.205s (68.4%)

OpenMP Potential Gain: 3.781s (31.5%)

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.

Top Open

This section

the region w

OpenMP Re

conj_grad_$

MAIN__$om

MAIN__$om

MAIN__$om

MAIN__$om

[Others]

*N/A is applied

Memory Bound: 91.8%

Cache Bound:

DRAM Latency Bound:

DRAM Bandwidth Bound:

This metric represents a fraction of cycles during main memory (DRAM). This metric does not aggre Consider improving data locality in NUMA multi-so

NUMA: % of Remote Accesses:

A significant amount of DRAM loads were service same core, or at least the same package, as it was

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.

Function	CPU Time	FPU Utilization	Vector Instruction Set	Loop Type
[Loop at line 575 in conj_grad_ompparallel@517]	126.149s	1.6%	SSE2(128)	Body
[Loop at line 678 in conj_grad_ompparallel@517]	5.004s	1.7%	SSE2(128)	Body
[Loop at line 575 in conj_grad_ompparallel@517]	2.678s	2.1%	[Unknown]	Remainder
[Loop at line 573 in conj_grad_ompparallel@517]	0.995s	4.0%	SSE2(128)	Body
[Loop at line 661 in conj_grad_ompparallel@517]	0.952s	1.3%	SSE(128); SSE2(128)	Body
[Others]	2.437s	N/A*	N/A*	N/A*

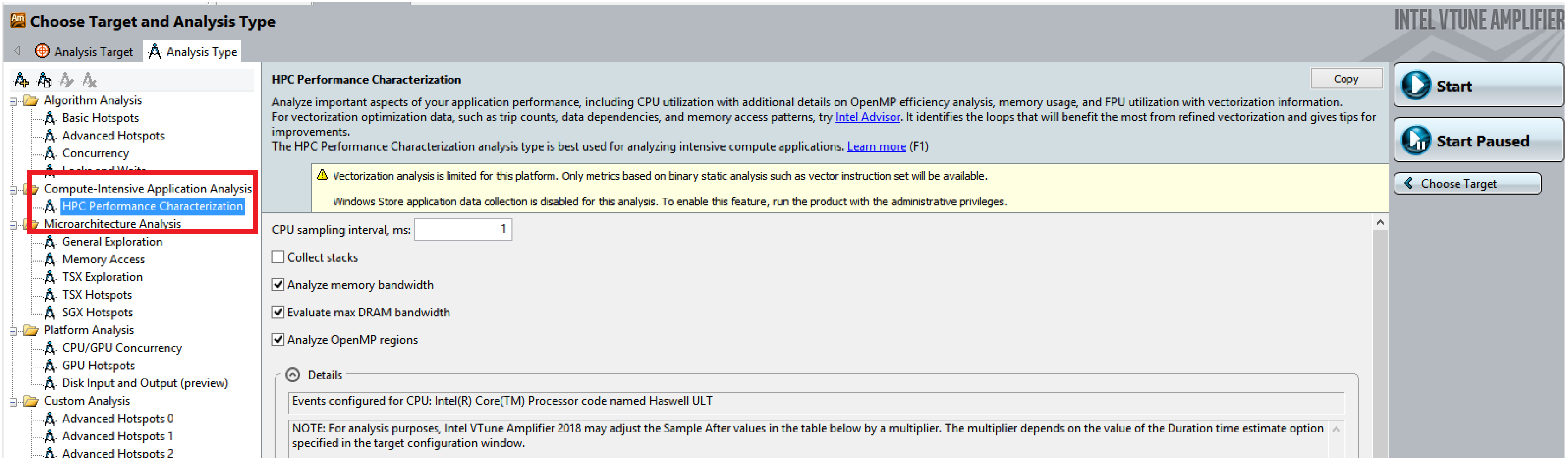
*N/A is applied to non-summable metrics.

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



HPC Performance Characterization

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)

⚠ Vectorization analysis is limited for this platform. Only metrics based on binary static analysis such as vector instruction set will be available.

Windows Store application data collection is disabled for this analysis. To enable this feature, run the product with the administrative privileges.

CPU sampling interval, ms: 1

□ Collect stacks

☑ Analyze memory bandwidth

☑ Evaluate max DRAM bandwidth

☑ Analyze OpenMP regions

Details

Events configured for CPU: Intel(R) Core(TM) Processor code named Haswell ULT

NOTE: For analysis purposes, Intel VTune Amplifier 2018 may adjust the Sample After values in the table below by a multiplier. The multiplier depends on the value of the Duration time estimate option specified in the target configuration window.

- Or Easy command line collection
  - *>amplxe-cl **–collect hpc-performance** –data-limit=0 –r result_dir ./my_app*

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

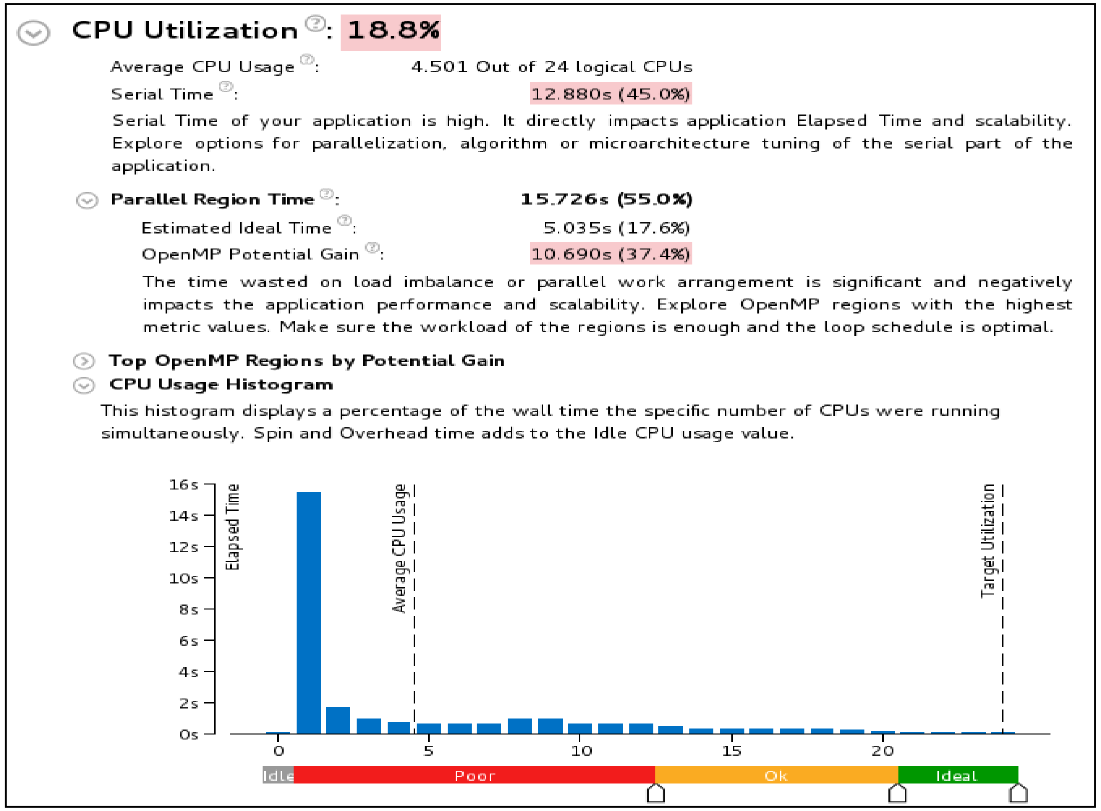
HPC CHARACTERIZATION: CPU UTILIZATION

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



CPU Utilization: 18.8%

Average CPU Usage: 4.501 Out of 24 logical CPUs

Serial Time: 12.880s (45.0%)

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.

Parallel Region Time: 15.726s (55.0%)

Estimated Ideal Time: 5.035s (17.6%)

OpenMP Potential Gain: 10.690s (37.4%)

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.

Top OpenMP Regions by Potential Gain

CPU Usage Histogram

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU usage value.

WHEN CORES SIT IDLE, PERFORMANCE IS LOST.

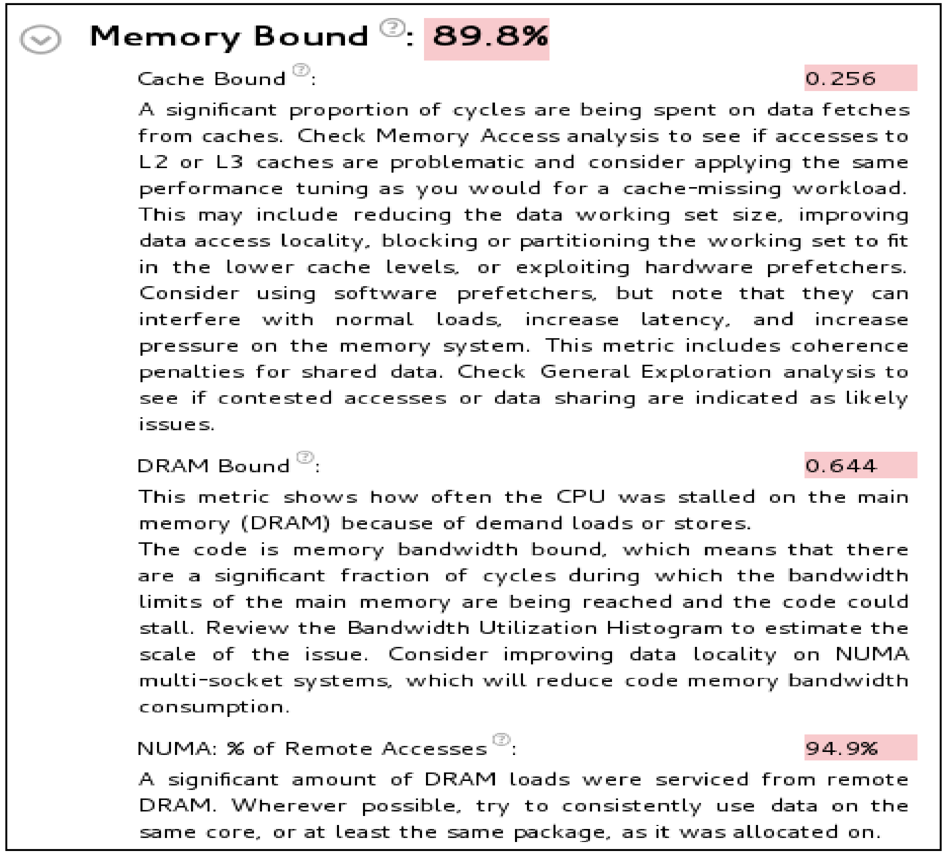
HPC CHARACTERIZATION: MEMORY BOUND

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



Memory Bound: 89.8%

Cache Bound: 0.256

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: 0.644

This metric shows how often the CPU was 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: 94.9%

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.

MEMORY IS OFTEN THE BOTTLENECK. FIND AND RELIEVE THE PRESSURE.

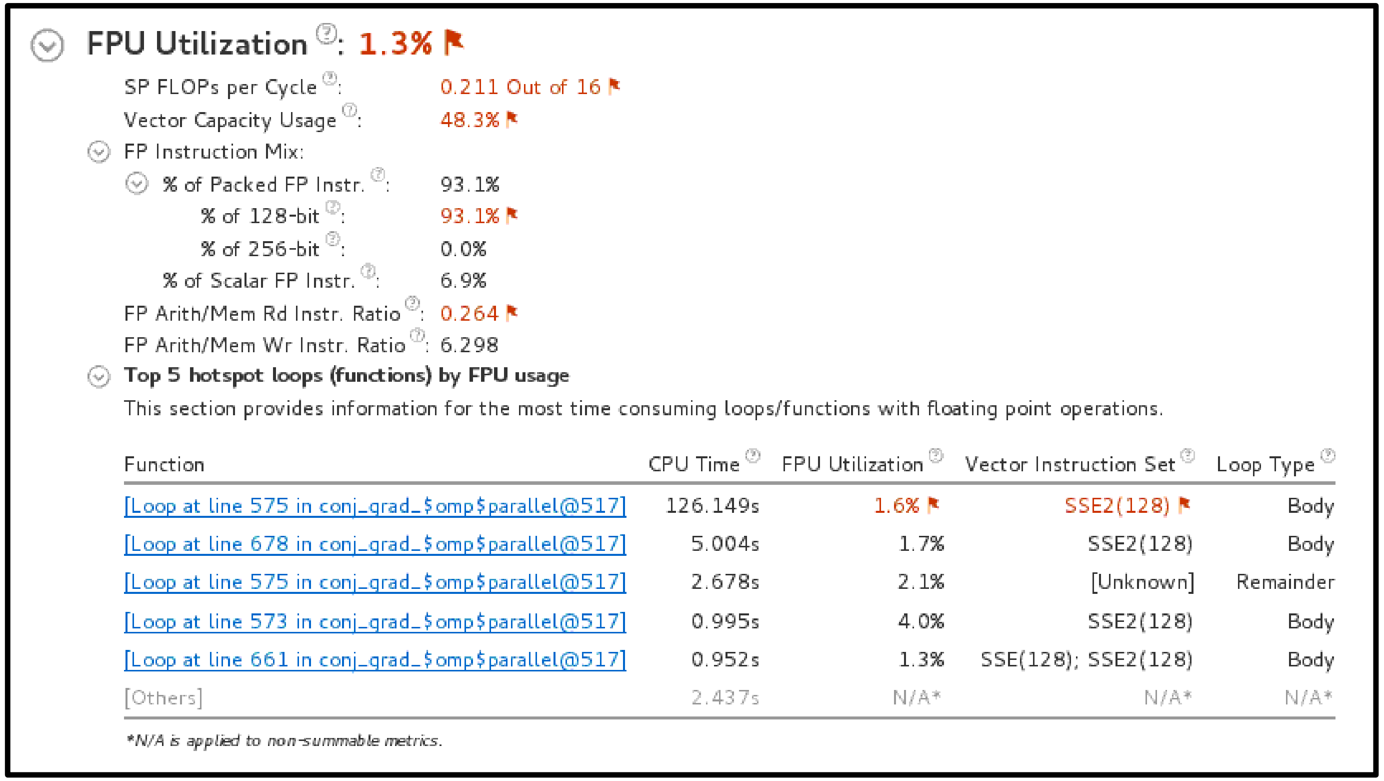
HPC CHARACTERIZATION: FPU UTILIZATION

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.

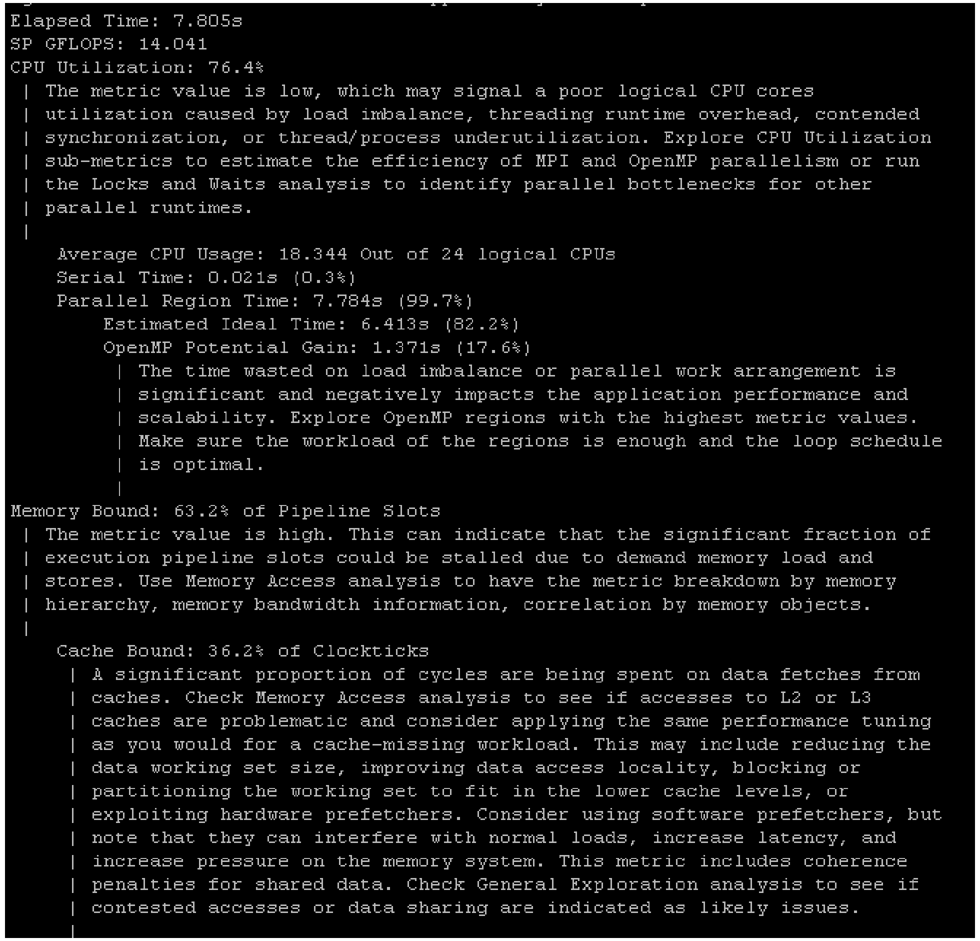
Function	CPU Time	FPU Utilization	Vector Instruction Set	Loop Type
[Loop at line 575 in conj_grad_ompparallel@517]	126.149s	1.6%	SSE2(128)	Body
[Loop at line 678 in conj_grad_ompparallel@517]	5.004s	1.7%	SSE2(128)	Body
[Loop at line 575 in conj_grad_ompparallel@517]	2.678s	2.1%	[Unknown]	Remainder
[Loop at line 573 in conj_grad_ompparallel@517]	0.995s	4.0%	SSE2(128)	Body
[Loop at line 661 in conj_grad_ompparallel@517]	0.952s	1.3%	SSE(128); SSE2(128)	Body
[Others]	2.437s	N/A*	N/A*	N/A*

*N/A is applied to non-summable metrics.

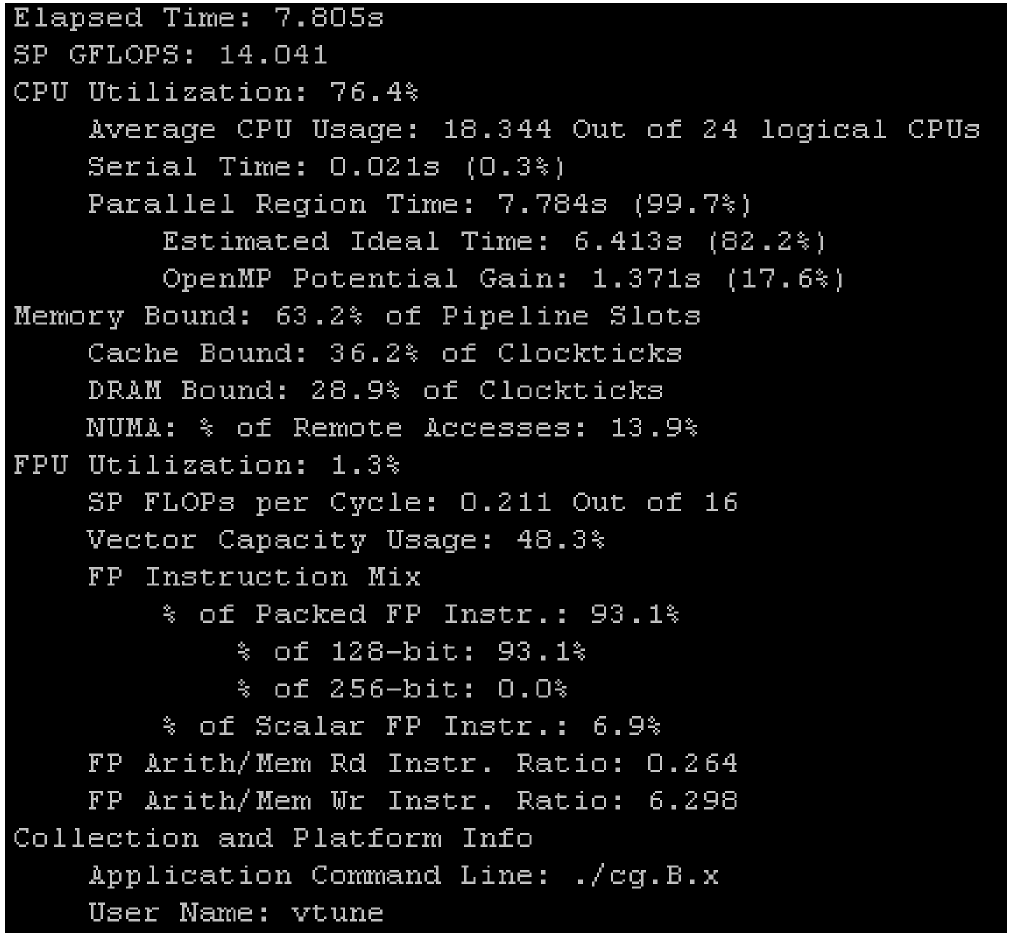
HARDWARE IS BECOMING MORE VECTORIZED, SO SHOULD YOU!

HPC CHARACTERIZATION: COMMAND LINE REPORTS

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



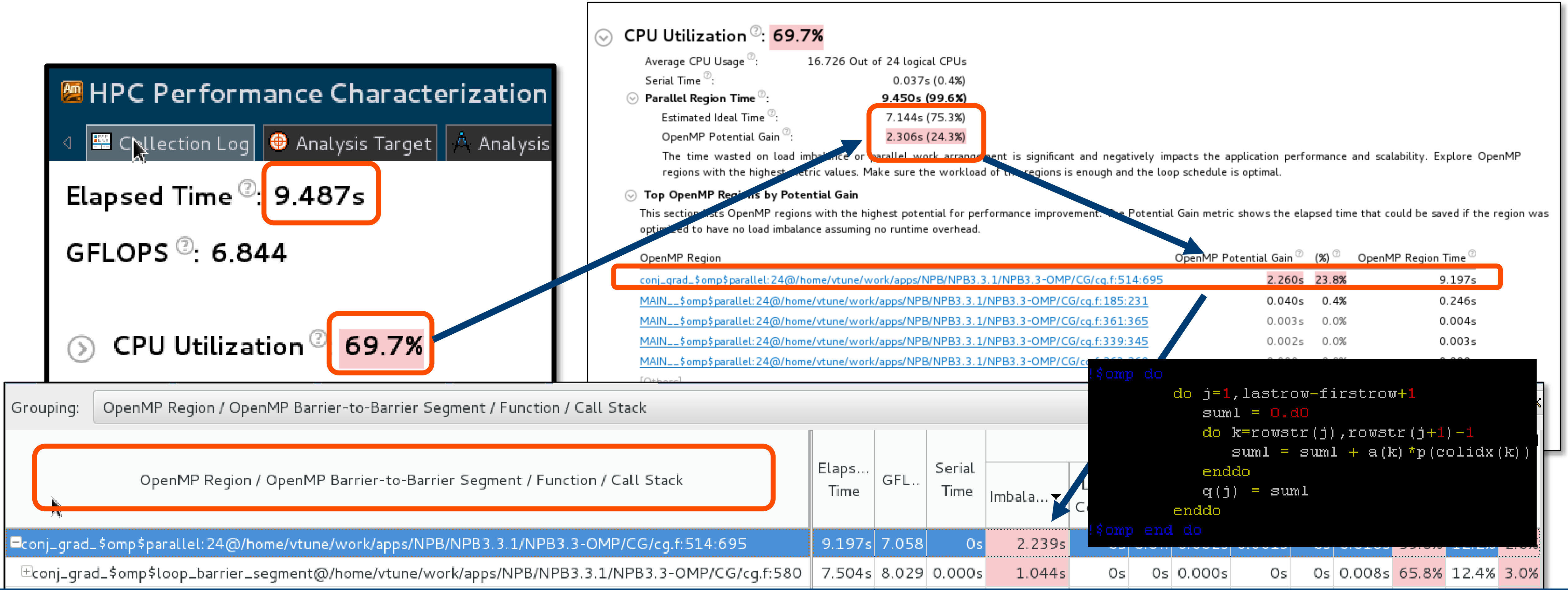
```
Elapsed Time: 7.805s
SP GFLOPS: 14.041
CPU Utilization: 76.4%
 | The metric value is low, which may signal a poor logical CPU cores
 | utilization caused by load imbalance, threading runtime overhead, contended
 | synchronization, or thread/process underutilization. Explore CPU Utilization
 | sub-metrics 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.
     |
```



```
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
```

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%

CPU Utilization: 69.7%

Average CPU Usage: 16.726 Out of 24 logical CPUs

Serial Time: 0.037s (0.4%)

Parallel Region Time: 9.450s (99.6%)

Estimated Ideal Time: 7.144s (75.3%)

OpenMP Potential Gain: 2.306s (24.3%)

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.

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	OpenMP Potential Gain	(%)	OpenMP Region Time
conj_grad_ompparallel:24@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:514:695	2.260s	23.8%	9.197s
MAIN__ompparallel:24@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:185:231	0.040s	0.4%	0.246s
MAIN__ompparallel:24@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:361:365	0.003s	0.0%	0.004s
MAIN__ompparallel:24@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:339:345	0.002s	0.0%	0.003s
MAIN__ompparallel:24@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/c[illegible]	[illegible]	[illegible]	[illegible]
[Others]			

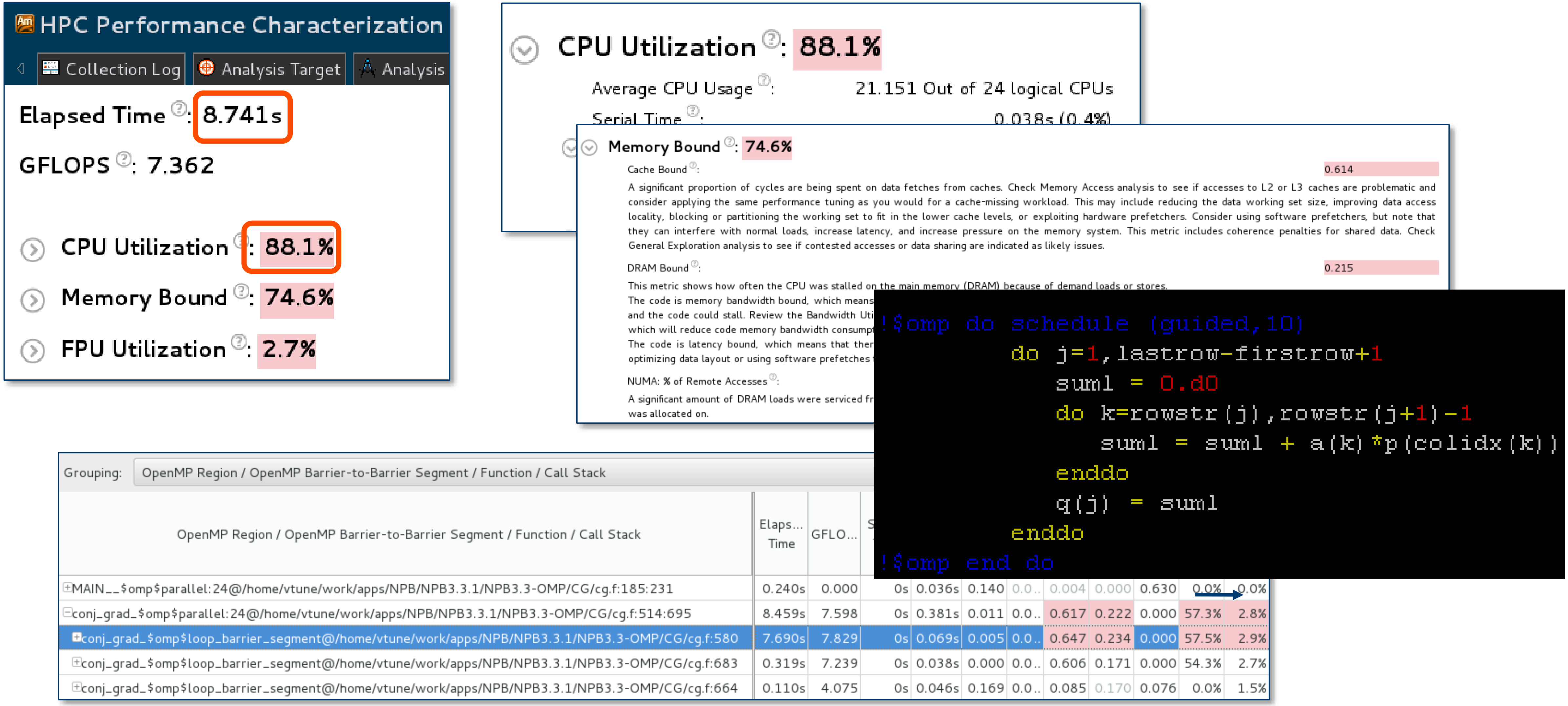
Grouping: OpenMP Region / OpenMP Barrier-to-Barrier Segment / Function / Call Stack

OpenMP Region / OpenMP Barrier-to-Barrier Segment / Function / Call Stack	Elaps... Time	GFL..	Serial Time	Imbala...	C								
⊟conj_grad_ompparallel:24@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:514:695	9.197s	7.058	0s	2.239s	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
⊞conj_grad_omploop_barrier_segment@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:580	7.504s	8.029	0.000s	1.044s	0s	0s	0.000s	0s	0s	0.008s	65.8%	12.4%	3.0%

```
!$omp do
         do j=1,lastrow-firstrow+1
            suml = 0.d0
            do k=rowstr(j),rowstr(j+1)-1
               suml = suml + a(k)*p(colidx(k))
            enddo
            q(j) = suml
         enddo
!$omp end do
```

APPLY DYNAMIC SCHEDULING TO AVOID IMBALANCE

PERFORMANCE EXAMPLES – GUIDED SCHEDULING (CHUNK 10)



Elapsed Time: 8.741s

GFLOPS: 7.362

CPU Utilization: 88.1%

Memory Bound: 74.6%

FPU Utilization: 2.7%

CPU Utilization: 88.1%

Average CPU Usage: 21.151 Out of 24 logical CPUs

Serial Time: 0.038s (0.4%)

Memory Bound: 74.6%

Cache Bound: 0.614

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: 0.215

This metric shows how often the CPU was stalled on the main memory (DRAM) because of demand loads or stores.
The code is memory bandwidth bound, which means
and the code could stall. Review the Bandwidth Uti
which will reduce code memory bandwidth consump
The code is latency bound, which means that ther
optimizing data layout or using software prefetches

NUMA: % of Remote Accesses:

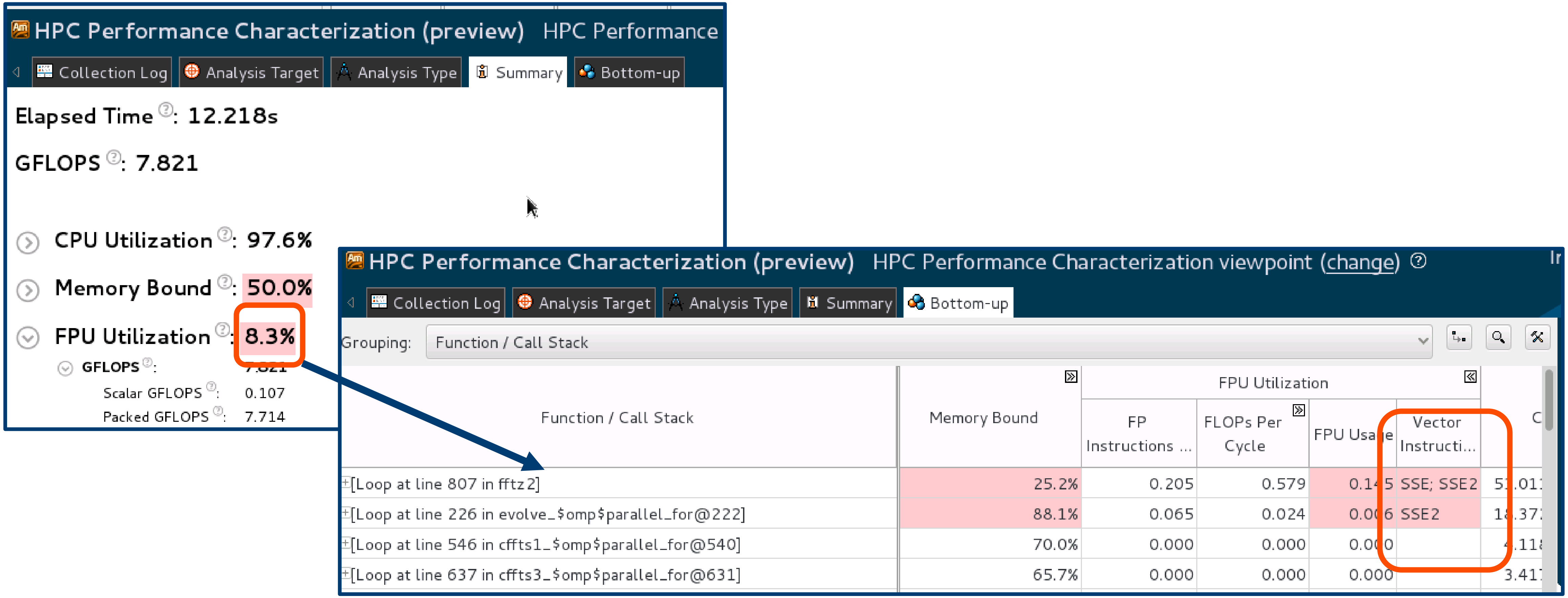
A significant amount of DRAM loads were serviced fr
was allocated on.

```
!$omp do schedule (guided,10)
         do j=1,lastrow-firstrow+1
            sum1 = 0.d0
            do k=rowstr(j),rowstr(j+1)-1
               sum1 = sum1 + a(k)*p(colidx(k))
            enddo
            q(j) = sum1
         enddo
!$omp end do
```

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

OpenMP Region / OpenMP Barrier-to-Barrier Segment / Function / Call Stack	Elaps... Time	GFLO...	S								
⊞MAIN__ompparallel:24@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:185:231	0.240s	0.000	0s	0.036s	0.140	0.0..	0.004	0.000	0.630	0.0%	0.0%
⊟conj_grad_ompparallel:24@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:514:695	8.459s	7.598	0s	0.381s	0.011	0.0..	0.617	0.222	0.000	57.3%	2.8%
⊞conj_grad_omploop_barrier_segment@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:580	7.690s	7.829	0s	0.069s	0.005	0.0..	0.647	0.234	0.000	57.5%	2.9%
⊞conj_grad_omploop_barrier_segment@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:683	0.319s	7.239	0s	0.038s	0.000	0.0..	0.606	0.171	0.000	54.3%	2.7%
⊞conj_grad_omploop_barrier_segment@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:664	0.110s	4.075	0s	0.046s	0.169	0.0..	0.085	0.170	0.076	0.0%	1.5%

PERFORMANCE EXAMPLES – FLOATING POINT UTILIZATION



HPC Performance Characterization (preview) HPC Performance

Collection Log | Analysis Target | Analysis Type | Summary | Bottom-up

Elapsed Time: 12.218s

GFLOPS: 7.821

CPU Utilization: 97.6%

Memory Bound: 50.0%

FPU Utilization: 8.3%

GFLOPS: 7.821

Scalar GFLOPS: 0.107

Packed GFLOPS: 7.714

HPC Performance Characterization (preview) HPC Performance Characterization viewpoint (change)

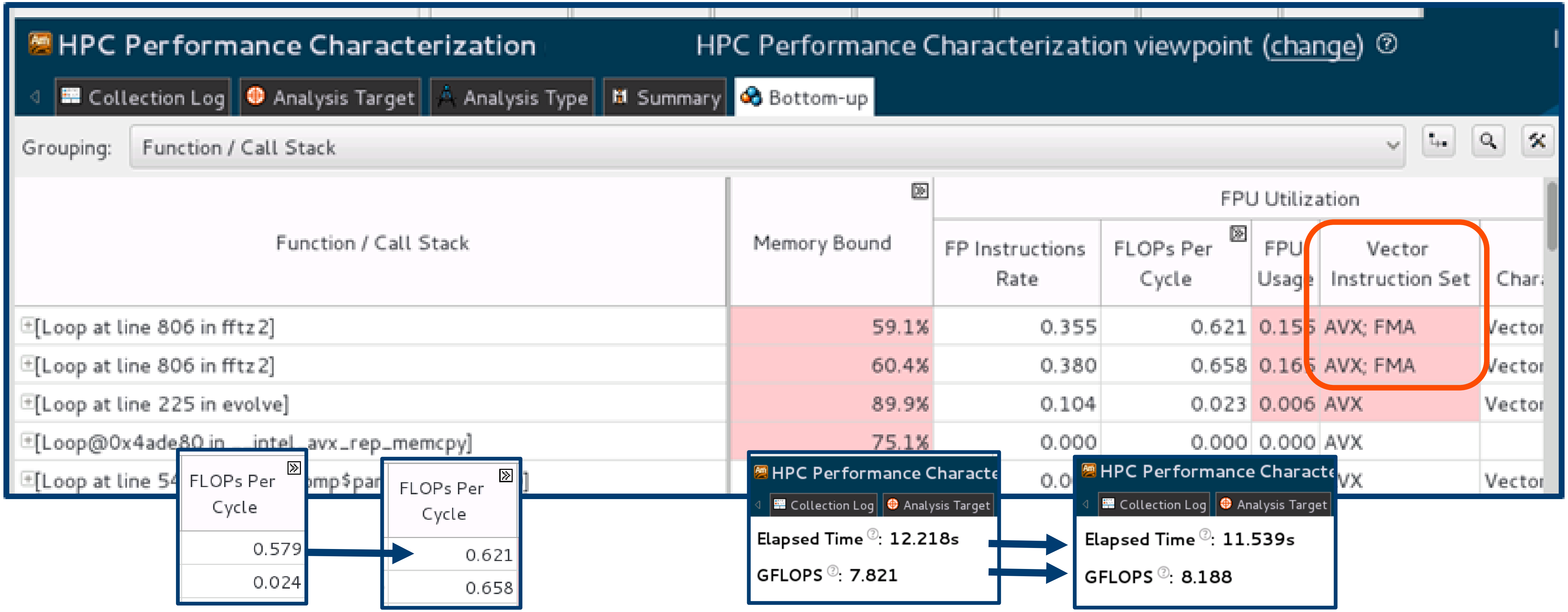
Collection Log | Analysis Target | Analysis Type | Summary | Bottom-up

Grouping: Function / Call Stack

Function / Call Stack	Memory Bound	FPU Utilization: FP Instructions ...	FPU Utilization: FLOPs Per Cycle	FPU Utilization: FPU Usage	FPU Utilization: Vector Instructi...	C
[Loop at line 807 in fftz2]	25.2%	0.205	0.579	0.145	SSE; SSE2	51.01
[Loop at line 226 in evolve_ompparallel_for@222]	88.1%	0.065	0.024	0.006	SSE2	18.37
[Loop at line 546 in cffts1_ompparallel_for@540]	70.0%	0.000	0.000	0.000		4.118
[Loop at line 637 in cffts3_ompparallel_for@631]	65.7%	0.000	0.000	0.000		3.417

OUTDATED VECTORIZATION INSTRUCTIONS – UPDATE COMPILER SETTINGS

PERFORMANCE EXAMPLES – FLOATING POINT UTILIZATION


HPC Performance Characterization
HPC Performance Characterization viewpoint (change)
Collection Log
Analysis Target
Analysis Type
Summary
Bottom-up

Grouping: Function / Call Stack

Function / Call Stack	Memory Bound	FPU Utilization				
		FP Instructions Rate	FLOPs Per Cycle	FPU Usage	Vector Instruction Set	Char
[Loop at line 806 in fftz2]	59.1%	0.355	0.621	0.15	AVX; FMA	Vector
[Loop at line 806 in fftz2]	60.4%	0.380	0.658	0.16	AVX; FMA	Vector
[Loop at line 225 in evolve]	89.9%	0.104	0.023	0.006	AVX	Vector
[Loop@0x4ade80 in __intel_avx_rep_memcpy]	75.1%	0.000	0.000	0.000	AVX	
[Loop at line 54 omp$par]		0.0			VX	Vector

FLOPs Per Cycle
0.579
0.024

→

FLOPs Per Cycle
0.621
0.658

HPC Performance Characte
Collection Log | Analysis Target

Elapsed Time: 12.218s
GFLOPS: 7.821

→

HPC Performance Characte
Collection Log | Analysis Target

Elapsed Time: 11.539s
GFLOPS: 8.188

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*…

Register for Beta at: http://intel.ly/intel-parallel-studio-xe-2018-beta

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

intel®

Software