Software

Profiling your application with Intel® Vtune™ Amplifier and Intel® Advisor

Paulius Velesko

Tuning at Multiple Hardware Levels

Exploiting all features of modern processors requires good use of the available resources

- Core
  - Vectorization is critical with 512bit FMA vector units (32 DP ops/cycle)
  - Targeting the current ISA is fundamental to fully exploit vectorization
- Socket
  - Using all cores in a processor requires parallelization (MPI, OMP, …)
  - Up to 64 Physical cores and 256 logical processors per socket on Theta!
- Node
  - Minimize remote memory access (control memory affinity)
  - Minimize resource sharing (tune local memory access, disk IO and network traffic)

Intel® Compiler Reports

FREE* performance metrics

Compile with -qopt-report=5

- Which loops were vectorized
  - Vector Length
  - Estimated Gain
  - Alignment
  - Scatter/Gather
- Prefetching
- Issues preventing vectorization
- Inline reports
- Interprocedural optimizations
- Register Spills/Fills

```
LOOP BEGIN at ../src/timestep.F(4835,13)
   remark #15389: vectorization support: reference nbd_(i) has unaligned access   [ ../src/timestep.F(4836,16) ]
   remark #15381: vectorization support: unaligned access used inside loop body
   remark #15335: loop was not vectorized: vectorization possible but seems inefficient. Use vector always directive or -vec-threshold0 to override
   remark #15329: vectorization support: irregularly indexed store was emulated for the variable <coefd_(nbd_(i))>, part of index is read from memory
   remark #15305: vectorization support: vector length 2
   remark #15399: vectorization support: unroll factor set to 4
   remark #15309: vectorization support: normalized vectorization overhead 0.139
   remark #15450: unmasked unaligned unit stride loads: 1
   remark #15463: unmasked indexed (or scatter) stores: 1
   remark #15475: --- begin vector cost summary ---
   remark #15476: scalar cost: 4
   remark #15477: vector cost: 4.500
   remark #15478: estimated potential speedup: 0.880
   remark #15488: --- end vector cost summary ---
   remark #25439: unrolled with remainder by 2
LOOP END
```

Intel® Application Performance Snapshot

Bird's eye view

VTune™ Amplifier's Application Performance Snapshot

High-level overview of application performance

- Identify primary optimization areas
- Recommend next steps in analysis
- Extremely easy to use
- Informative, actionable data in clean HTML report
- Detailed reports available via command line
- Low overhead, high scalability

Usage on Theta

Launch all profiling jobs from **/projects** rather than **/home**

No module available, so setup the environment manually:

```
$ source /opt/intel/vtune_amplifier/apsvars.sh
$ export PMI_NO_FORK=1
```

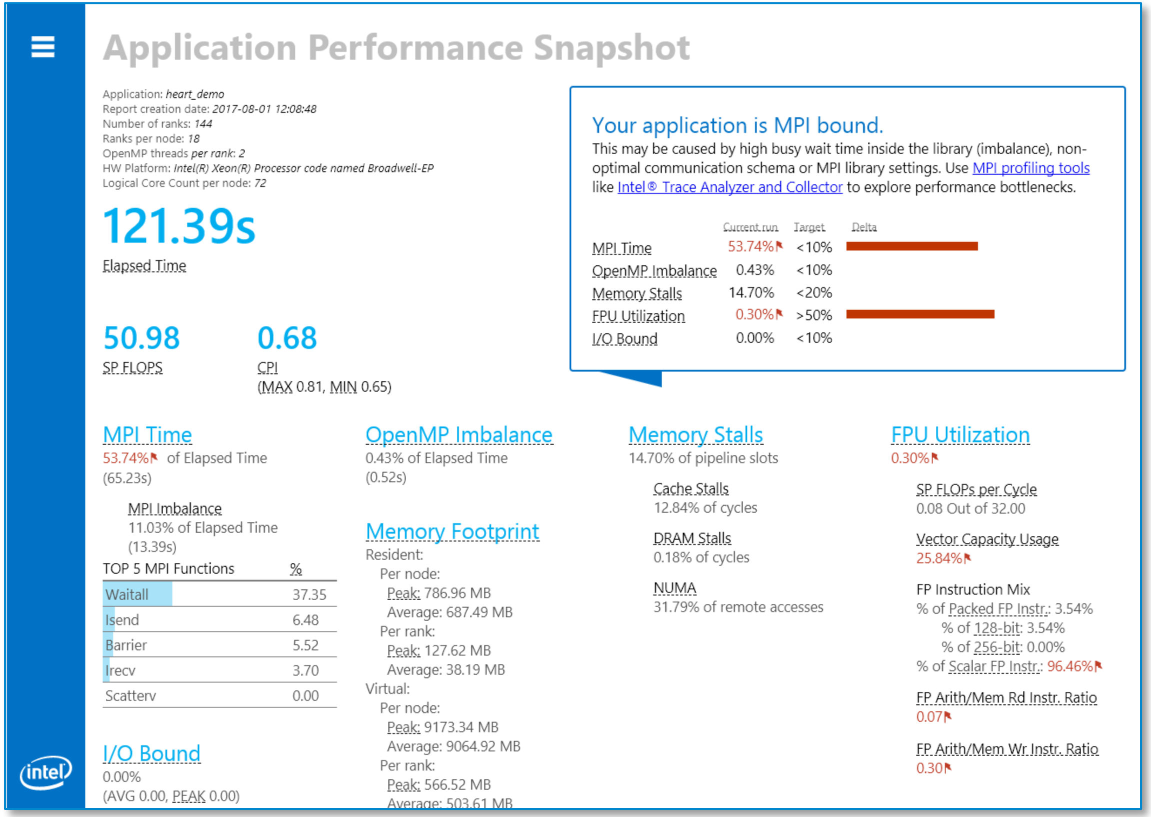
Launch your job in interactive or batch mode:

```
$ aprun -N <ppn> -n <totRanks> [affinity opts] aps ./exe
```

Produce text and html reports:

```
$ aps -report=./aps_result_ ….
```

APS HTML Report



Application Performance Snapshot

Application: *heart_demo*
Report creation date: *2017-08-01 12:08:48*
Number of ranks: *144*
Ranks per node: *18*
OpenMP threads *per rank: 2*
HW Platform: *Intel(R) Xeon(R) Processor code named Broadwell-EP*
Logical Core Count per node: *72*

121.39s
Elapsed Time

50.98
SP FLOPS

0.68
CPI
(MAX 0.81, MIN 0.65)

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.

	Current run	Target	Delta
MPI Time	53.74%	<10%	
OpenMP Imbalance	0.43%	<10%	
Memory Stalls	14.70%	<20%	
FPU Utilization	0.30%	>50%	
I/O Bound	0.00%	<10%	

MPI Time

53.74% of Elapsed Time
(65.23s)

MPI Imbalance
11.03% of Elapsed Time
(13.39s)

TOP 5 MPI Functions	%
Waitall	37.35
Isend	6.48
Barrier	5.52
Irecv	3.70
Scatterv	0.00

I/O Bound

0.00%
(AVG 0.00, PEAK 0.00)

OpenMP Imbalance

0.43% of Elapsed Time
(0.52s)

Memory Footprint

Resident:
Per node:
Peak: 786.96 MB
Average: 687.49 MB
Per rank:
Peak: 127.62 MB
Average: 38.19 MB
Virtual:
Per node:
Peak: 9173.34 MB
Average: 9064.92 MB
Per rank:
Peak: 566.52 MB
Average: 503.61 MB

Memory Stalls

14.70% of pipeline slots

Cache Stalls
12.84% of cycles

DRAM Stalls
0.18% of cycles

NUMA
31.79% of remote accesses

FPU Utilization

0.30%

SP FLOPs per Cycle
0.08 Out of 32.00

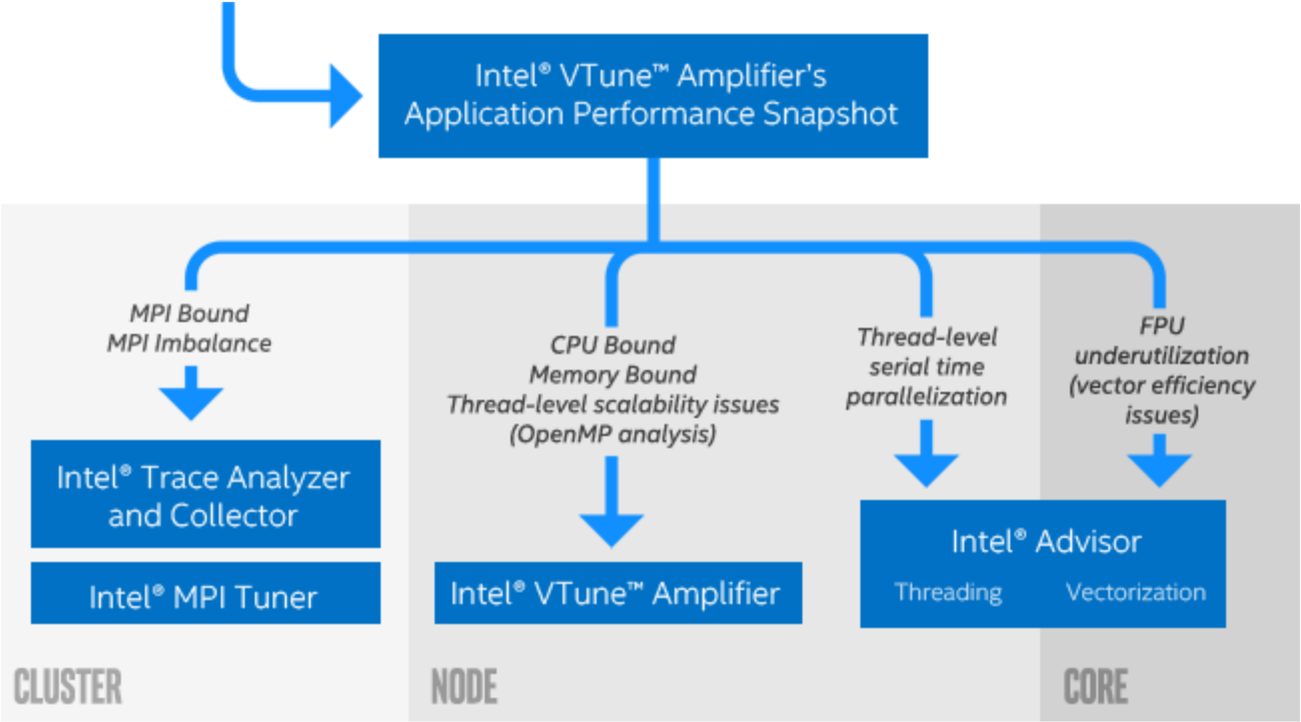
Vector Capacity Usage
25.84%

FP Instruction Mix
% of Packed FP Instr.: 3.54%
% of 128-bit: 3.54%
% of 256-bit: 0.00%
% of Scalar FP Instr.: 96.46%

FP Arith/Mem Rd Instr. Ratio
0.07

FP Arith/Mem Wr Instr. Ratio
0.30

Tuning Workflow



Intel® Advisor

Vectorization and Threading

Intel® Advisor

Modern HPC processors explore different level of parallelism:

- within a core: vectorization (Theta: 8 DP elements, 16 SP elements)
- between the cores: multi-threading (Theta: 64 cores, 256 threads)

Adapting applications to take advantage of such high parallelism is quite demanding and requires code modernization

The Intel® Advisor is a software tool for vectorization and thread prototyping

The tool guides the software developer to resolve issues during the vectorization process

Typical Vectorization Optimization Workflow

There is no need to recompile or relink the application, but the use of -g is recommended.

1. Collect survey and tripcounts data
   - Investigate **application** place within roofline model
   - Determine vectorization efficiency and opportunities for improvement
2. Collect memory access pattern data
   - Determine data structure optimization needs
3. Collect dependencies
   - Differentiate between real and assumed issues blocking vectorization

Cache-Aware Roofline

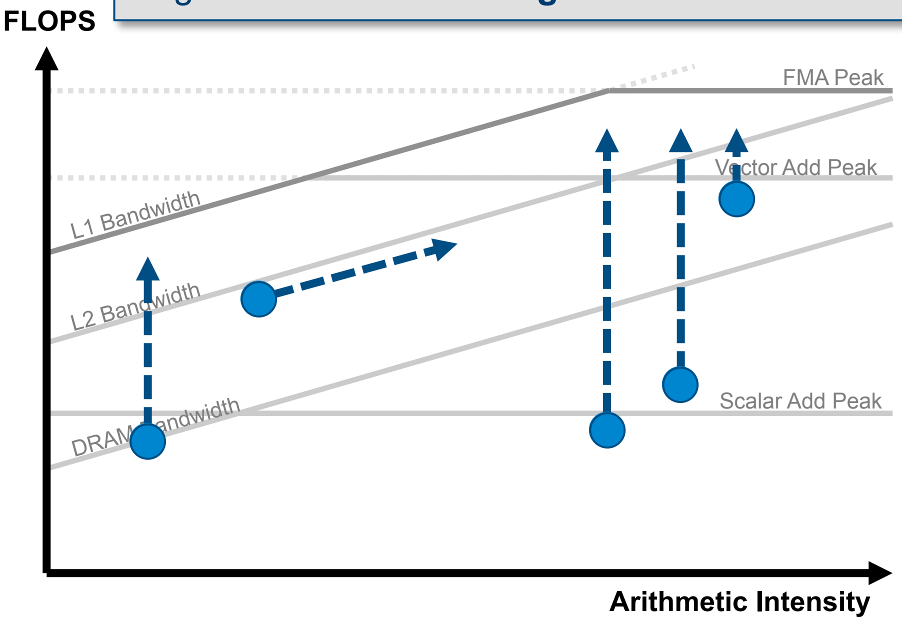
Next Steps

If under or near a memory roof…

- Try a MAP analysis. Make any appropriate **cache optimizations.**
- If cache optimization is impossible, try **reworking the algorithm to have a higher AI.**

If Under the Vector Add Peak

Check “Traits” in the Survey to see if FMAs are used. If not, try altering your code or compiler flags to **induce FMA usage.**



If just above the Scalar Add Peak

Check **vectorization efficiency** in the Survey. Follow the recommendations to improve it if it’s low.

If under the Scalar Add Peak…

Check the Survey Report to see if the loop vectorized. If not, try to **get it to vectorize** if possible. This may involve running Dependencies to see if it’s safe to force it.

Using Intel® Advisor on Theta

Two options to setup collections: GUI (advixe-gui) or command line (advixe-cl).

I will focus on the command line since it is better suited for batch execution, but the GUI provides the same capabilities in a user-friendly interface.

I recommend taking a snapshot of the results and analyzing in a local machine (Linux, Windows, Mac) to avoid issues with lag.

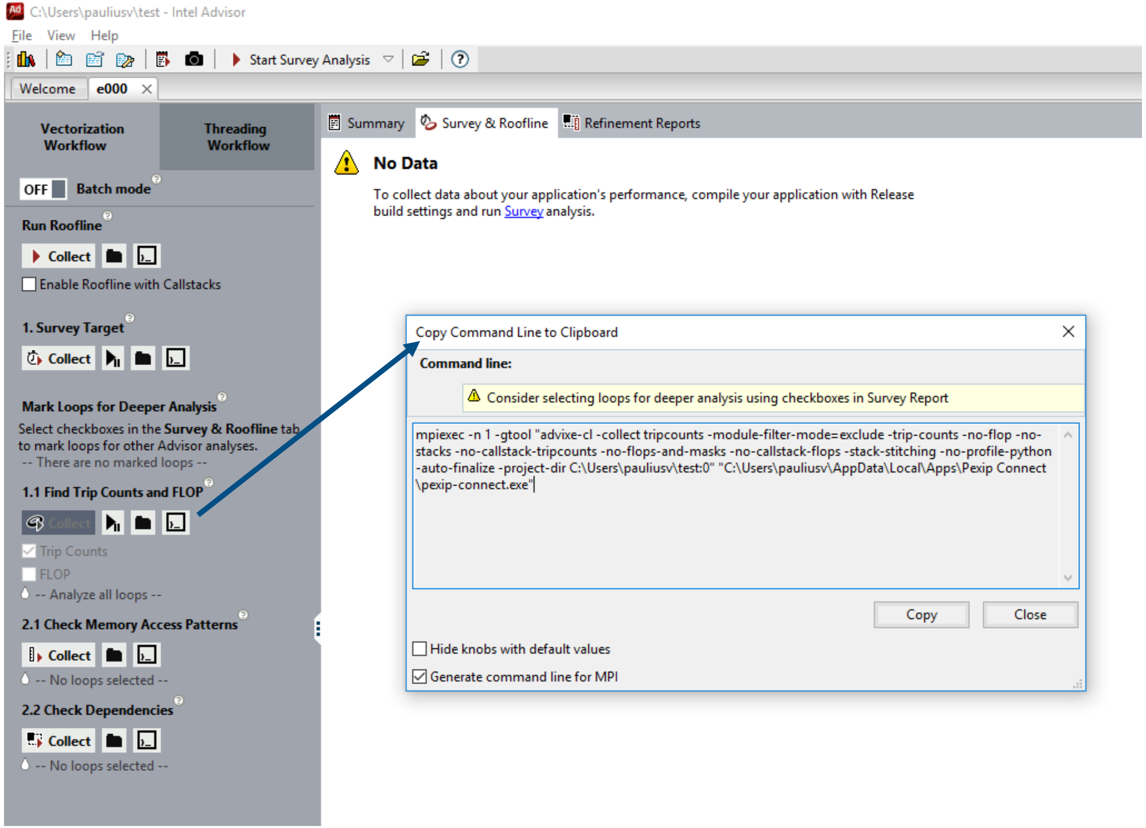
```
advixe-cl --snapshot --cache-sources --cache-binaries ./advixe_res_dir
```

Some things to note:

- Use /projects rather than /home for profiling jobs
- Set your environment:

```
$ source /opt/intel/advisor/advixe-vars.sh
$ export LD_LIBRARY_PATH=/opt/intel/advisor/lib64:$LD_LIBRARY_PATH
$ export PMI_NO_FORK=1
```

Using Intel® Advisor on Theta



Sample Script

```
#!/bin/bash

#COBALT -t 30

#COBALT -n 1

#COBALT -q debug-cache-quad

#COBALT -A <project>
```

→Basic scheduler info (the usual)

```
export LD_LIBRARY_PATH=/opt/intel/advisor/lib64:$LD_LIBRARY_PATH

source /opt/intel/advisor/advixe-vars.sh

export PMI_NO_FORK=1
```

→Environment setup

Two separate collections

```
aprun -n 1 -N 1 advixe-cl -c survey --project-dir ./adv_res --search-dir src:=./ --search-dir bin:=./ --./exe

aprun -n 1 -N 1 advixe-cl -c tripcounts -flops-and-masks --project-dir ./adv_res \
                          --search-dir src:=./ --search-dir bin:=./ -- ./exe
```

Nbody demonstration

The naïve code that could

Nbody gravity simulation

https://github.com/fbaru-dev/nbody-demo (Dr. Fabio Baruffa)

Let's consider a distribution of point masses m_1,…,m_n located at r_1,…,r_n.

We want to calculate the position of the particles after a certain time interval using the Newton law of gravity.

```
struct Particle
{
 public:
   Particle() { init();}
   void init()
   {
     pos[0] = 0.; pos[1] = 0.; pos[2] = 0.;
     vel[0] = 0.; vel[1] = 0.; vel[2] = 0.;
     acc[0] = 0.; acc[1] = 0.; acc[2] = 0.;
     mass   = 0.;
   }
   real_type pos[3];
   real_type vel[3];
   real_type acc[3];
   real_type mass;
};
```

```
for (i = 0; i < n; i++){          // update acceleration
   for (j = 0; j < n; j++){
     real_type distance, dx, dy, dz;
     real_type distanceSqr = 0.0;
     real_type distanceInv = 0.0;

     dx = particles[j].pos[0] - particles[i].pos[0];
     …

     distanceSqr = dx*dx + dy*dy + dz*dz + softeningSquared;
     distanceInv = 1.0 / sqrt(distanceSqr);

     particles[i].acc[0] += dx * G * particles[j].mass *
                            distanceInv * distanceInv * distanceInv;
     particles[i].acc[1] += …
     particles[i].acc[2] += …
```

Collect Roofline Data

Starting with version 2 of the code we collect both survey and tripcounts data:

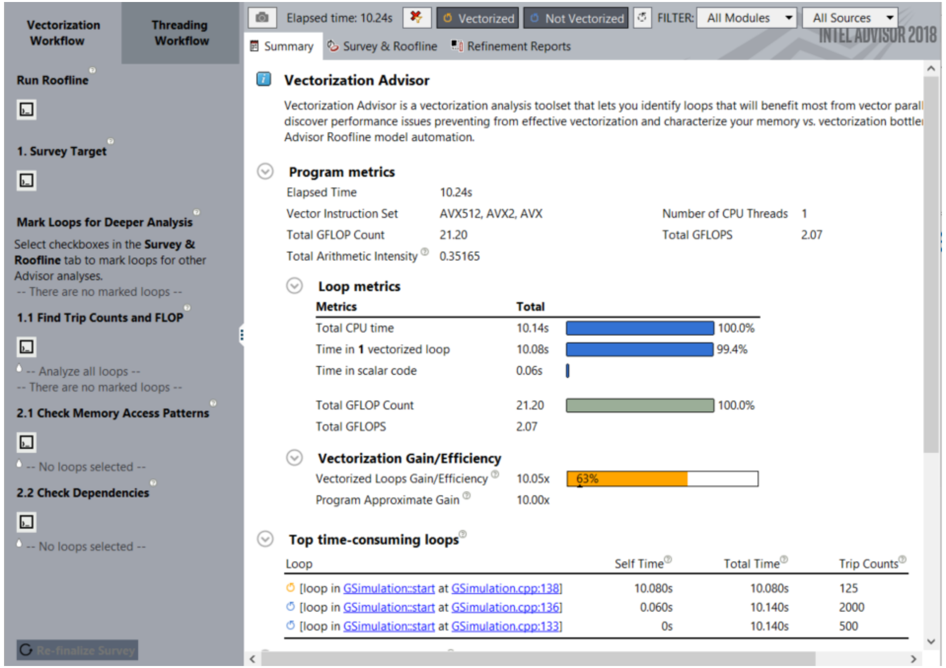
```
export LD_LIBRARY_PATH=/opt/intel/advisor/lib64:$LD_LIBRARY_PATH
source /opt/intel/advisor/advixe-vars.sh
export PMI_NO_FORK=1
aprun -n 1 -N 1 advixe-cl --collect roofline --project-dir ./adv_res --search-dir src:=./ \
                        --search-dir bin:=./ -- ./nbody.x
```

And generate a portable snapshot to analyze anywhere:

```
advixe-cl --snapshot --project-dir ./adv_res --pack --cache-sources \
--cache-binaries --search-dir src:=./ --search-dir bin:=./ -- nbody_naive
```

If finalization is too slow on compute add -no-auto-finalize to collection line.

Summary Report

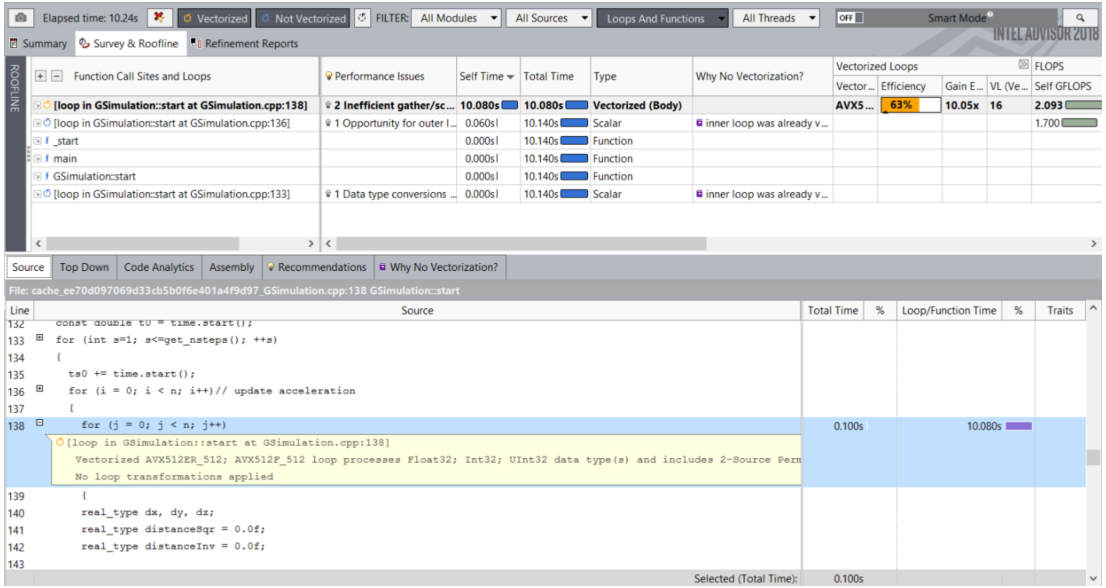


GUI left panel provides access to further tests

Summary provides overall performance characteristics

- Lists instruction set(s) used
- Top time consuming loops are listed individually
- Loops are annotated as vectorized and non-vectorized
- Vectorization efficiency is based on used ISA, in this case Intel® Advanced Vector Extensions 512 (AVX512)

Survey Report (Source)



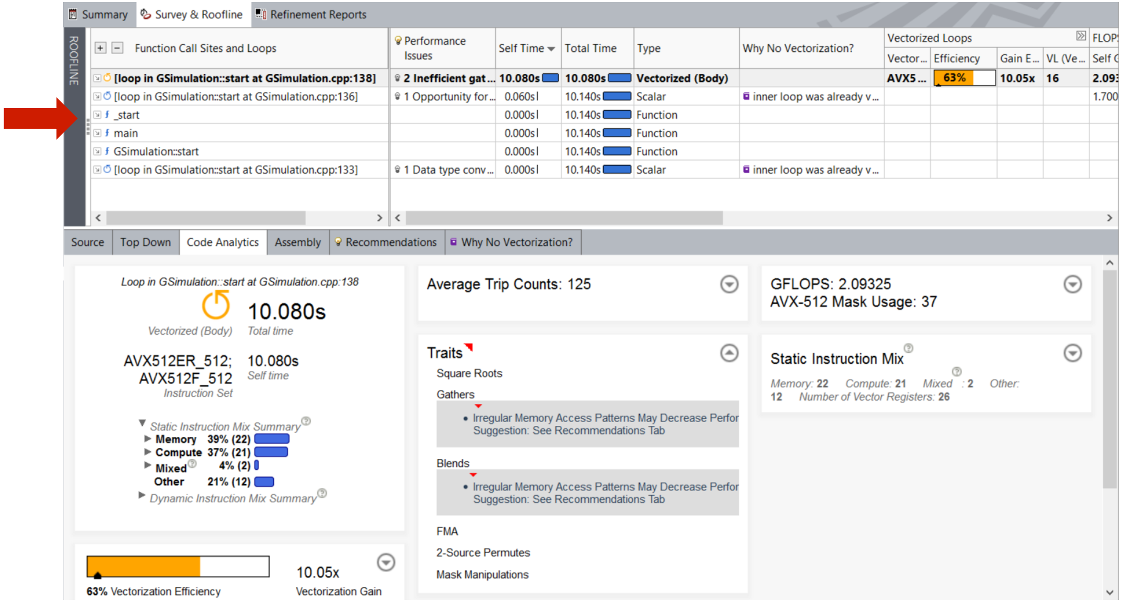
Inline information regarding loop characteristics

- ISA used
- Types processed
- Compiler transformations applied
- Vector length used
- …

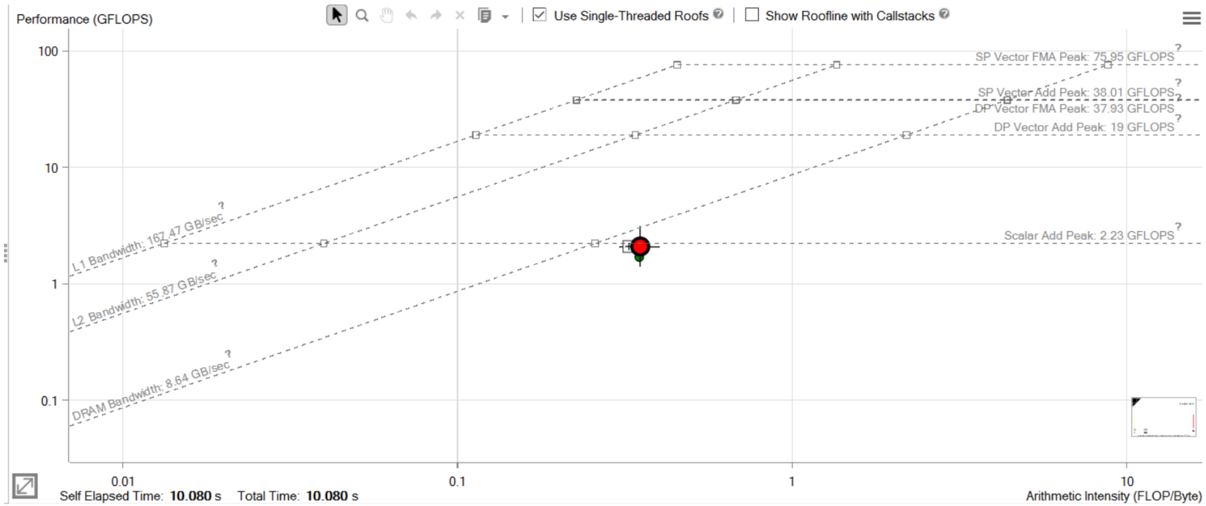
Survey Report (Code Analytics)

Detailed loop information

- Instruction mix
- ISA used, including subgroups
- Loop traits
  - FMA
  - Square root
  - Gathers / Blends point to memory issues and vector inefficiencies



CARM Analysis



Using single threaded roof

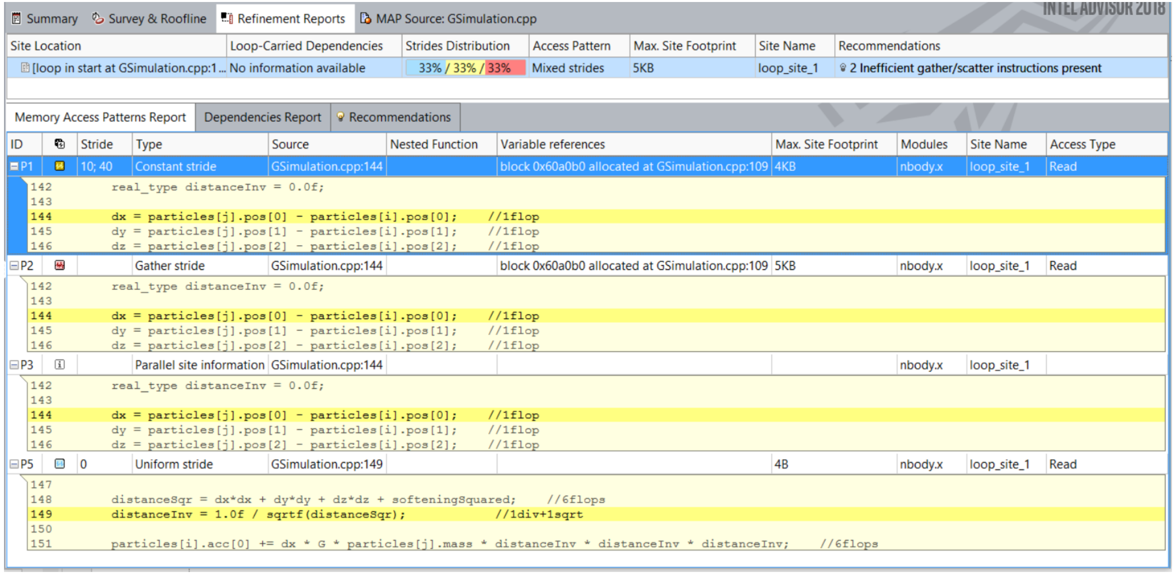
Code vectorized, but performance on par with scalar add peak?

- Irregular memory access patterns force gather operations.
- Overhead of setting up vector operations reduces efficiency.

Next step is clear: perform a Memory Access Pattern analysis

Memory Access Pattern Analysis (Refinement)

```
aprun -n 1 -N 1 advixe-cl --collect map --project-dir ./adv_res \
                --search-dir src:=./ --search-dir bin:=./ -- ./nbody.x
```



Storage of particles is in an Array Of Structures (AOS) style

This leads to regular, but non-unit strides in memory access

- 33% unit
- 33% uniform, non-unit
- 33% non-uniform

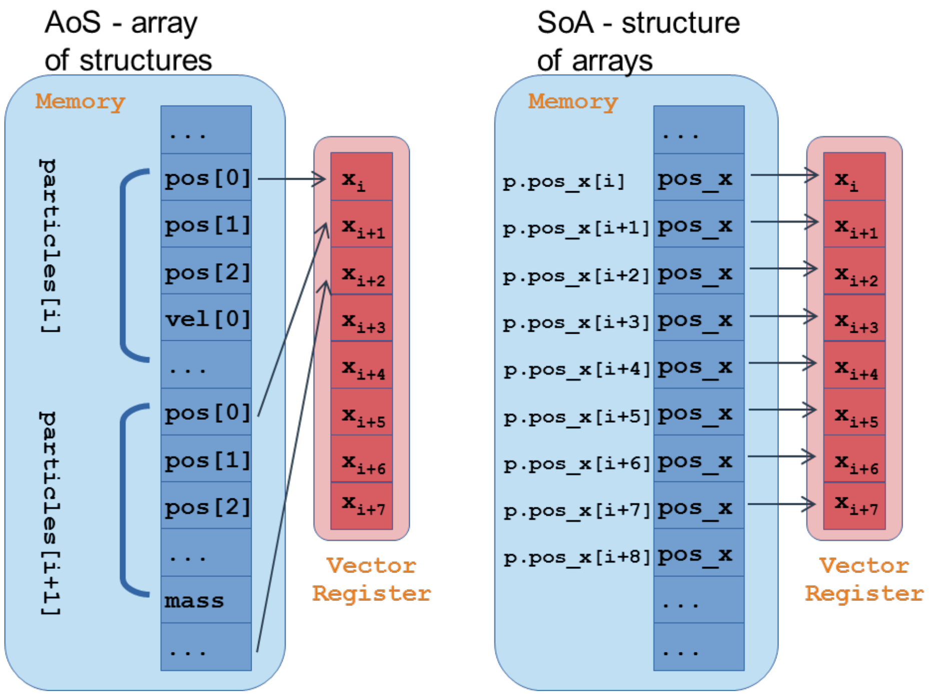
Re-structuring the code into a Structure Of Arrays (SOA) may lead to unit stride access and more effective vectorization

Vectorization: gather/scatter operation

The compiler might generate gather/scatter instructions for loops automatically vectorized where memory locations are not contiguous

```
struct Particle
{
 public:
   ...
   real_type pos[3];
   real_type vel[3];
   real_type acc[3];
   real_type mass;
};
```

```
struct ParticleSoA
{
 public:
   ...
   real_type *pos_x,*pos_y,*pos_z;
   real_type *vel_x,*vel_y,*vel_z;
   real_type *acc_x,*acc_y;*acc_z
   real_type *mass;
};
```



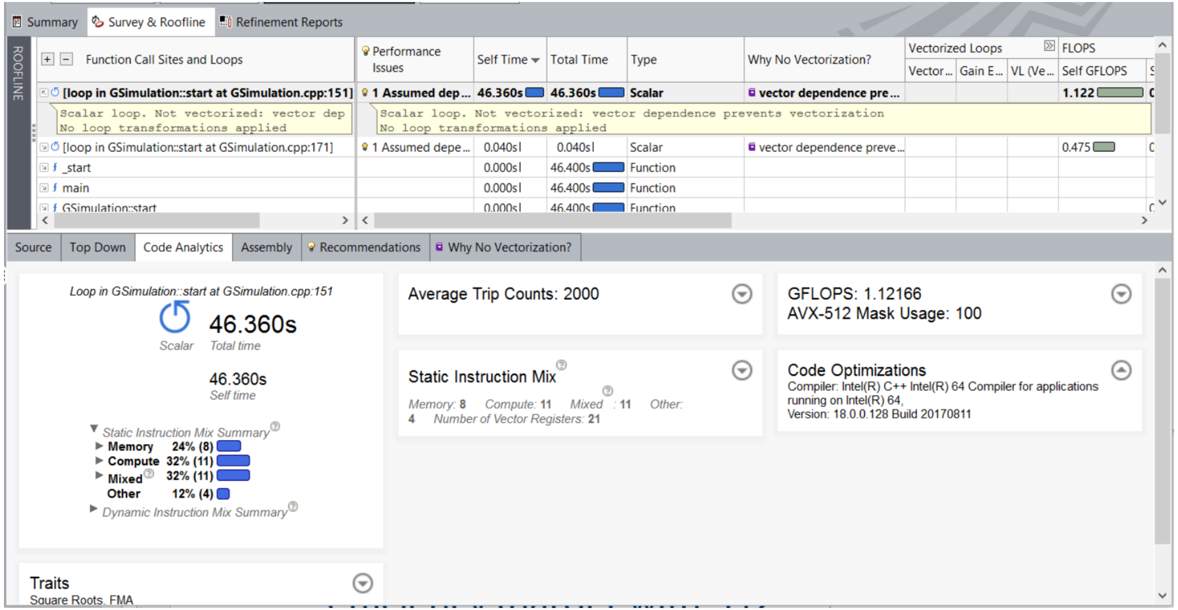
Performance After Data Structure Change

In this new version (version 3 in github sample) we introduce the following change:

- Change particle data structures from AOS to SOA

Note changes in report:

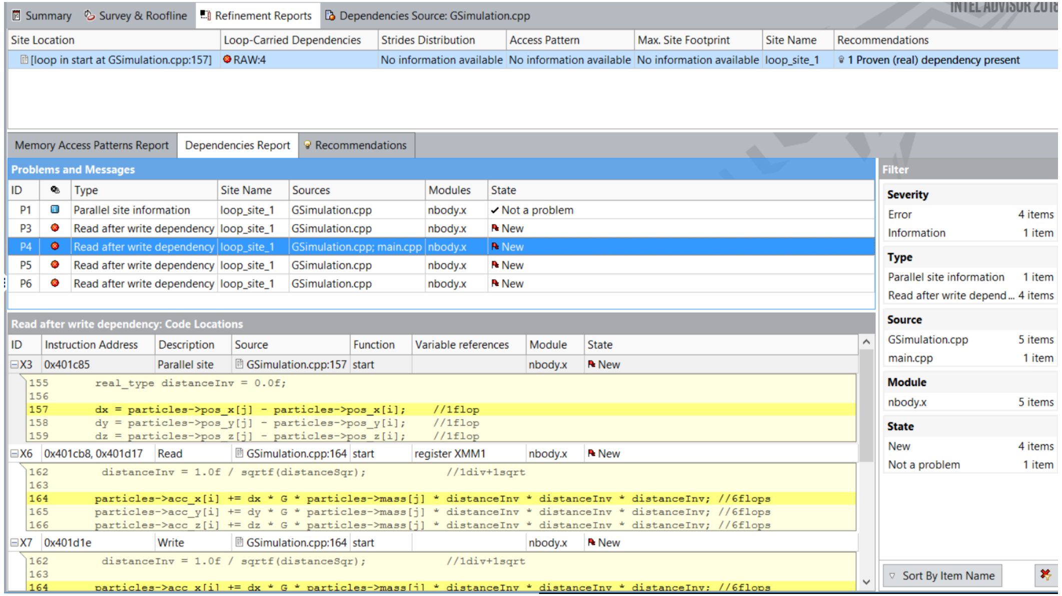
- Performance is lower
- Main loop is no longer vectorized
- Assumed vector dependence prevents automatic vectorization



Next step is clear: perform a Dependencies analysis

Dependencies Analysis (Refinement)

```
aprun -n 1 -N 1 advixe-cl --collect dependencies --project-dir ./adv_res \
                --search-dir src:=./ --search-dir bin:=./ -- ./nbody.x
```



Dependencies analysis has high overhead:

- Run on reduced workload

Advisor Findings:

- RAW dependency
- Multiple reduction-type dependencies

Recommendations

Memory Access Patterns Report | Dependencies Report | Recommendations

All Advisor-detectable issues: *C++* | *Fortran*

Recommendation: Resolve dependency

The Dependencies analysis shows there is a real (proven) dependency in the loop. To fix: Do one of the following:

- If there is an anti-dependency, enable vectorization using the directive `#pragma omp simd safelen(length)` , where `length` is smaller than the distance between dependent iterations in anti-dependency. For example:

```
#pragma omp simd safelen(4)
for (i = 0; i < n - 4; i += 4)
{
    a[i + 4] = a[i] * c;
}
```

- If there is a reduction pattern dependency in the loop, enable vectorization using the directive `#pragma omp simd reduction(operator:list)` . For example:

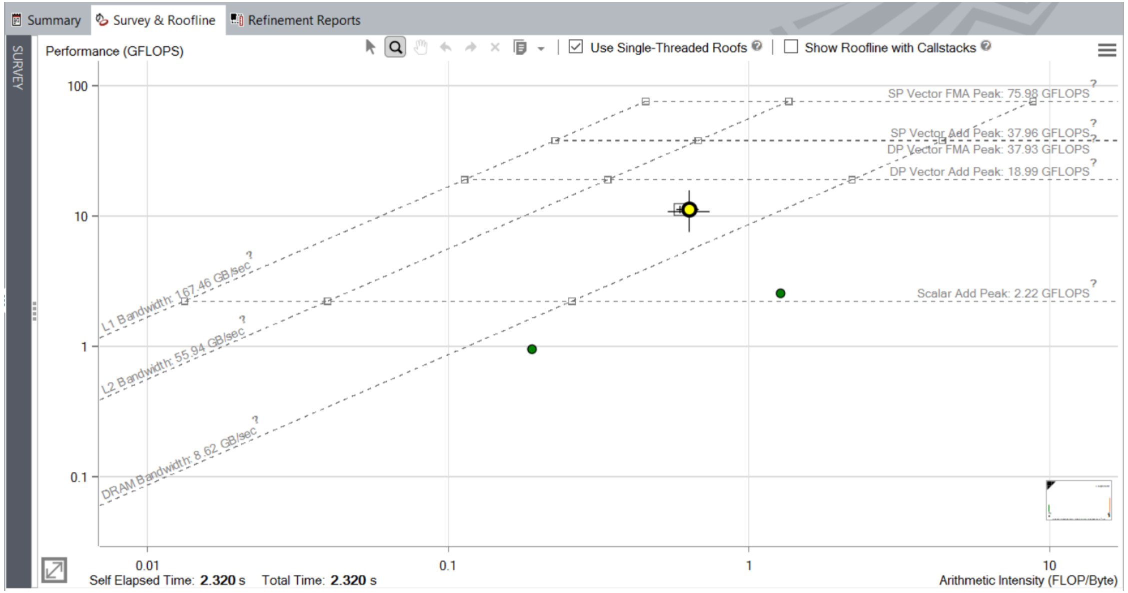
```
#pragma omp simd reduction(+:sumx)
for (k = 0;k < size2; k++)
{
    sumx += x[k]*b[k];
}
```

ISSUE: PROVEN (REAL) DEPENDENCY PRESENT

The compiler assumed there is an anti-dependency (Write after read - WAR) or true dependency (Read after write - RAW) in the loop. Improve performance by investigating the assumption and handling accordingly.

Resolve dependency

Performance After Resolved Dependencies



New memory access pattern plus vectorization produces much improved performance!
What's next?

Intel® VTUNE™ Amplifier

Core-level hardware metrics

Intel® VTune™ Amplifier

VTune Amplifier is a full system profiler

- Accurate
- Low overhead
- Comprehensive (microarchitecture, memory, IO, treading, …)
- Highly customizable interface
- Direct access to source code and assembly

Analyzing code access to shared resources is critical to achieve good performance on multicore and manycore systems

VTune Amplifier takes over where Intel® Advisor left

Predefined Collections

Many available analysis types:

▪	advanced-hotspots	Advanced Hotspots
▪	concurrency	Concurrency
▪	disk-io	Disk Input and Output
▪	general-exploration	General microarchitecture exploration
▪	gpu-hotspots	GPU Hotspots
▪	gpu-profiling	GPU In-kernel Profiling
▪	hotspots	Basic Hotspots → Python Support
▪	hpc-performance	HPC Performance Characterization
▪	locksandwaits	Locks and Waits → Python Support
▪	memory-access	Memory Access
▪	memory-consumption	Memory Consumption → Python Support
▪	system-overview	System Overview
▪	…	

Python Support

The HPC Performance Characterization Analysis

Threading: CPU Utilization

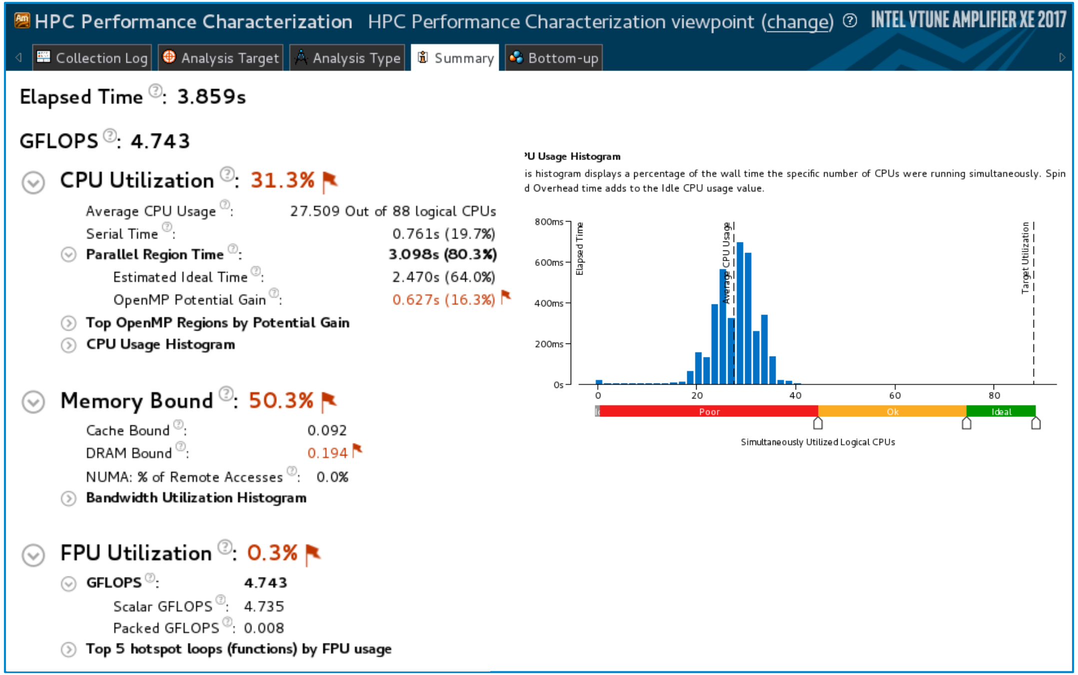
- Serial vs. Parallel time
- Top OpenMP regions by potential gain
- Tip: Use hotspot OpenMP region analysis for more detail

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.

Memory Access Analysis

Tune data structures for performance

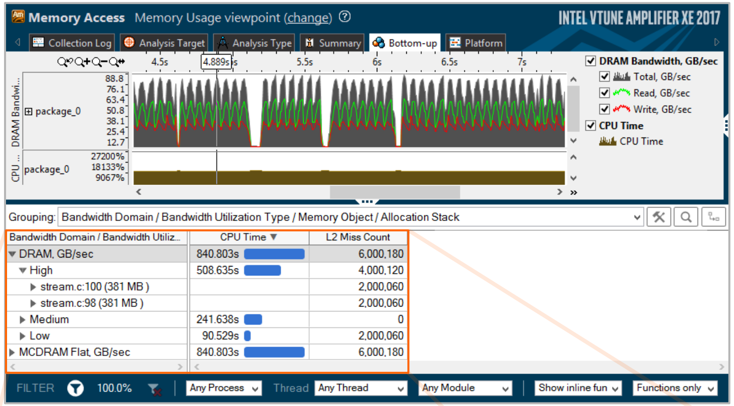
- Attribute cache misses to data structures (not just the code causing the miss)
- Support for custom memory allocators

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



Bandwidth Domain / Bandwidth Utiliz...	CPU Time ▼	L2 Miss Count
▼ DRAM, GB/sec	840.803s	6,000,180
▼ High	508.635s	4,000,120
▶ stream.c:100 (381 MB)		2,000,060
▶ stream.c:98 (381 MB)		2,000,060
▶ Medium	241.638s	0
▶ Low	90.529s	2,000,060
▶ MCDRAM Flat, GB/sec	840.803s	6,000,180

Using Intel® VTune™ Amplifier on Theta

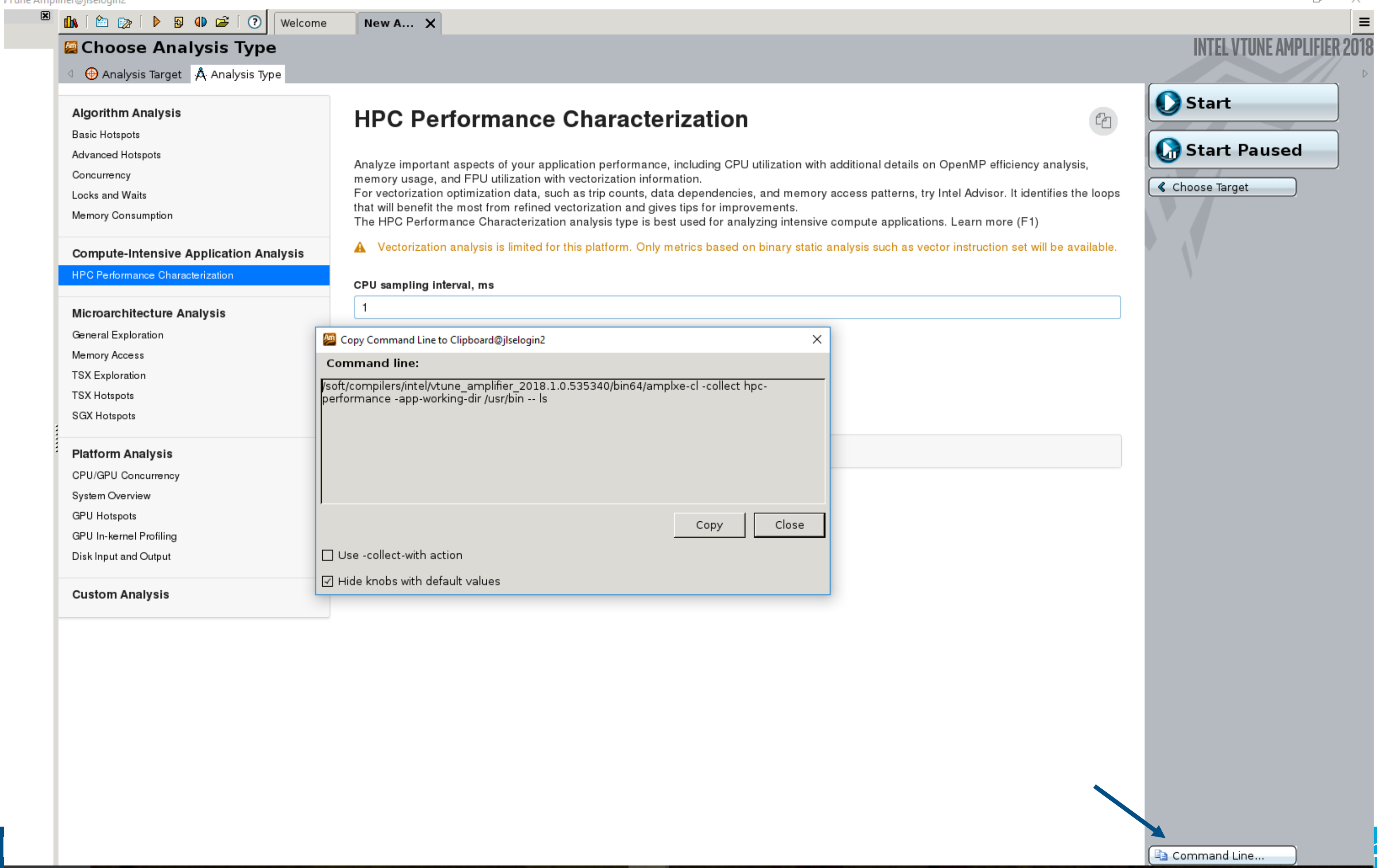
Two options to setup collections: GUI (amplxe-gui) or command line (amplxe-cl).

I will focus on the command line since it is better suited for batch execution, but the GUI provides the same capabilities in a user-friendly interface.

Some things of note:

- Use /projects rather than /home for profiling jobs
- Set your environment:

```
$ source /opt/intel/vtune_amplifier/amplxe-vars.sh
$ export LD_LIBRARY_PATH=/opt/intel/vtune_amplifier/lib64:$LD_LIBRARY_PATH
$ export PMI_NO_FORK=1
```


Welcome
New A...
Choose Analysis Type
INTEL VTUNE AMPLIFIER 2018
Analysis Target
Analysis Type
Algorithm Analysis
Basic Hotspots
Advanced Hotspots
Concurrency
Locks and Waits
Memory Consumption
Compute-Intensive Application Analysis
HPC Performance Characterization
Microarchitecture Analysis
General Exploration
Memory Access
TSX Exploration
TSX Hotspots
SGX Hotspots
Platform Analysis
CPU/GPU Concurrency
System Overview
GPU Hotspots
GPU In-kernel Profiling
Disk Input and Output
Custom Analysis
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.
CPU sampling interval, ms
1
Copy Command Line to Clipboard@jlselogin2
Command line:
/soft/compilers/intel/vtune_amplifier_2018.1.0.535340/bin64/amplxe-cl -collect hpc-performance -app-working-dir /usr/bin -- ls
Copy
Close
Use -collect-with action
Hide knobs with default values
Start
Start Paused
Choose Target
Command Line...

Sample Script

```
#!/bin/bash
#COBALT -t 30
#COBALT -n 1
#COBALT -q debug-cache-quad
#COBALT -A <project>
```

→Basic scheduler info (the usual)

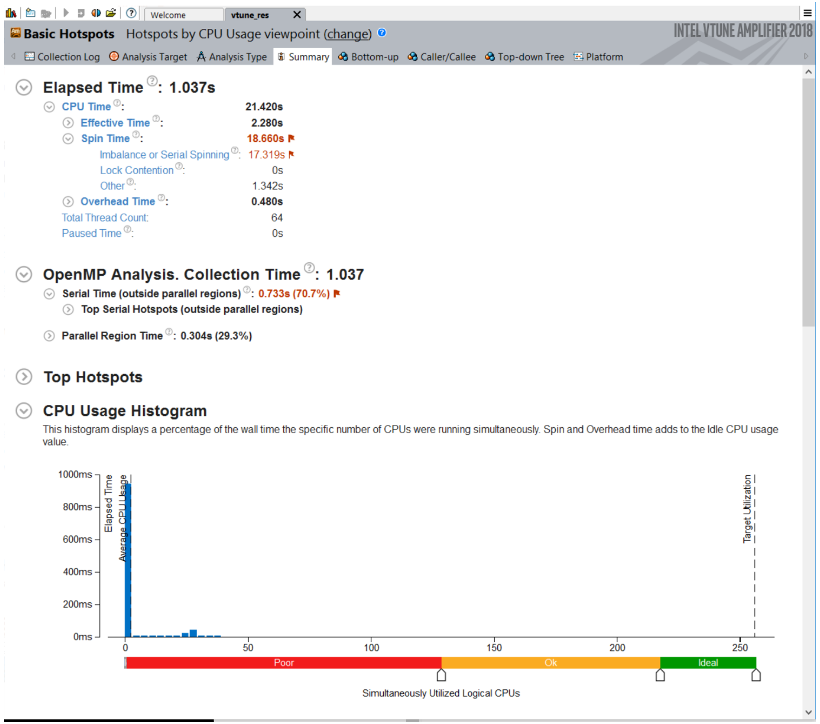
```
export LD_LIBRARY_PATH=/opt/intel/vtune_amplifier/lib64:$LD_LIBRARY_PATH
source /opt/intel/vtune_amplifier/amplxe-vars.sh
export PMI_NO_FORK=1
export OMP_NUM_THREADS=64; export OMP_PROC_BIND=spread; export OMP_PLACES=cores
```

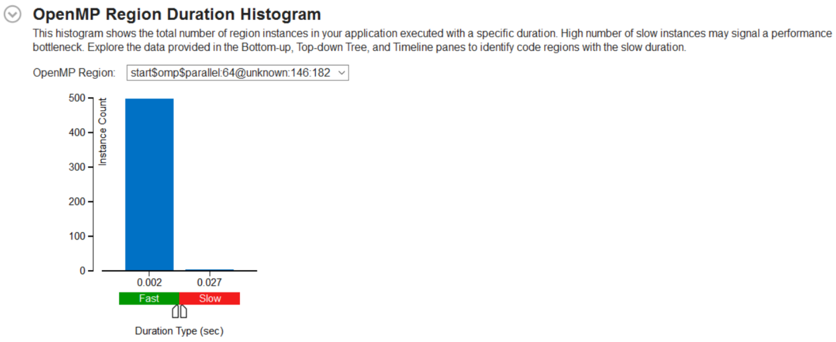
→Environment setup

Invoke VTune™ Amplifier

```
aprun -n 1 -N 1 -cc depth -d 256 -j 4 amplxe-cl -c hotspots -knob analyze-openmp=true \
                                                 -r ./adv_res -- ./exe
```

Hotspots analysis for nbody demo (ver7: threaded)



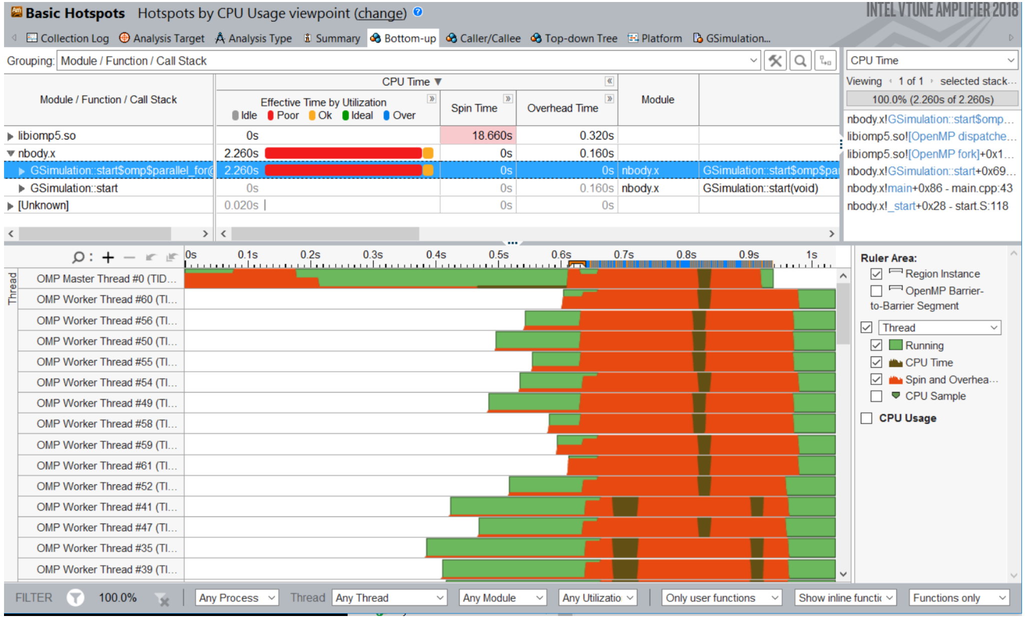


Lots of spin time indicate issues with load balance and synchronization

Given the short OpenMP region duration it is likely we do not have sufficient work per thread

Let's look a the timeline for each thread to understand things better…

Bottom-up Hotspots view



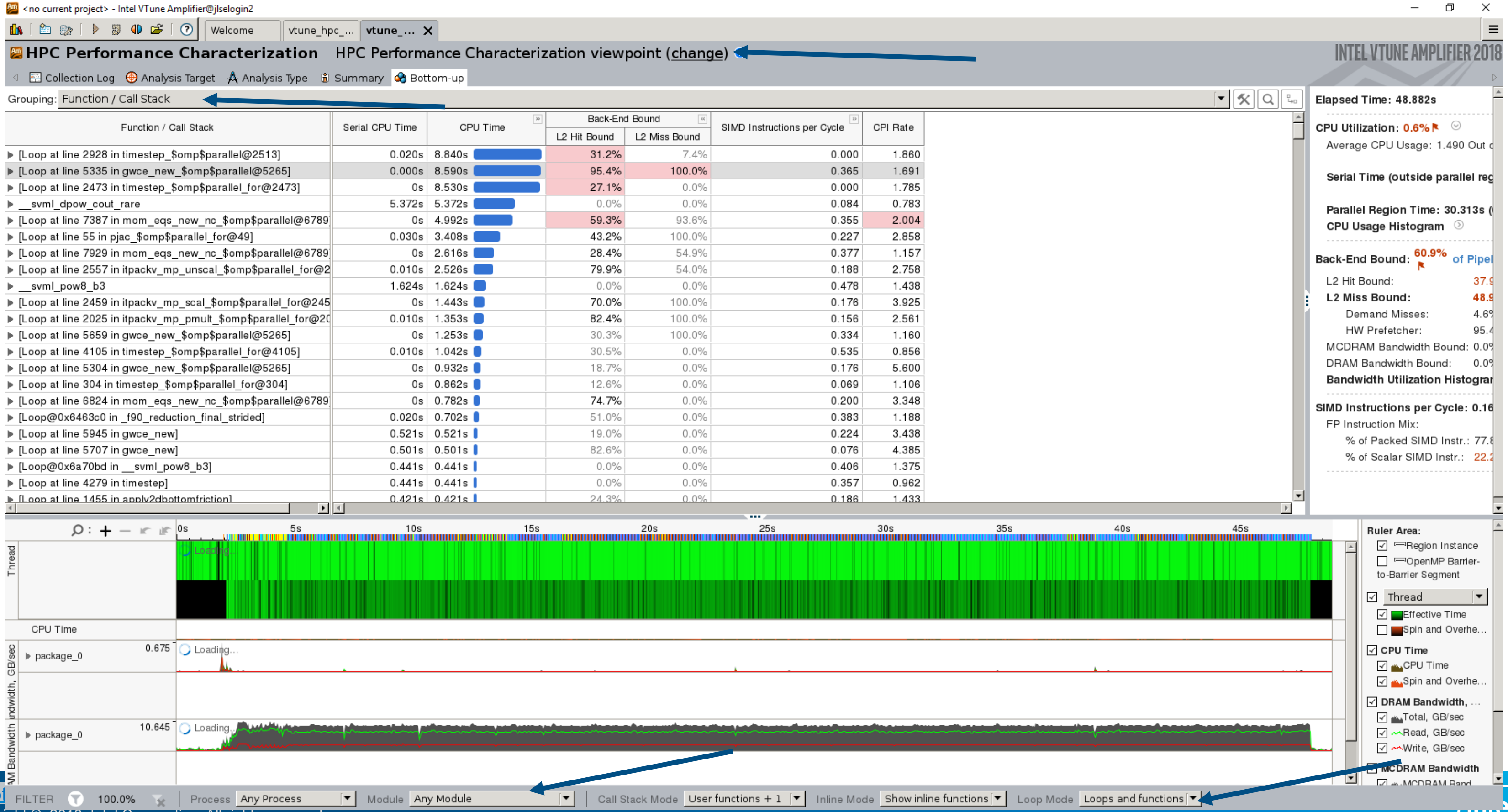
There is not enough work per thread in this particular example.

Double click on line to access source and assembly.

Notice the filtering options at the bottom, which allow customization of this view.

Next steps would include additional analysis to continue the optimization process.

Bottom-up HPC Characterization View



Bottom-up HPC Characterization View - Thread

<no current project> - Intel VTune Amplifier@jlselogin2

Welcome | vtune_hpc_... | vtune_...

HPC Performance Characterization HPC Performance Characterization viewpoint (change)

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

Grouping: OpenMP Region / Thread / Function / Call Stack

OpenMP Region / Thread / Function / Call Stack	Elapsed Time	Serial CPU Time	OpenMP Potential Gain: Imbalance	OpenMP Potential Gain: Lock Contention	OpenMP Potential Gain: Creation	OpenMP Potential Gain: Scheduling	OpenMP Potential Gain: Reduction	OpenMP Potential Gain: Atomics	CPU Time	Back-End Bound: L2 Hit Bound	Back-End Bound: L2 Miss Bound	SIMD Instructions per
[Serial - outside parallel regions]	18.568s	18.202s							19.024s	14.5%	30.9%	
gwce_new_ompparallel:2@unknown:5265:5673	6.316s	0s	0.259s	0.005s	0s	0.025s	0s	0s	11.887s	73.0%	95.4%	
OMP Master Thread #0 (TID: 183872)		0s							6.204s	70.3%	64.1%	
OMP Worker Thread #1 (TID: 184145)		0s							5.683s	76.0%	100.0%	
mom_eqs_new_nc_ompparallel:2@unknown:6789:7951	5.170s	0s	0.184s	0.020s	0s	0.045s	0.005s	0s	9.903s	46.3%	61.3%	
timestep_ompparallel:2@unknown:2513:2966	4.814s	0.000s	0.166s	0s	0s	0s	0s	0s	9.201s	29.6%	7.0%	
timestep_ompparallel:2@unknown:2473:2502	4.863s	0s	0.161s	0s	0.005s	0.030s	0s	0s	9.151s	25.4%	0.0%	
pjac_ompparallel:2@unknown:49:59	2.490s	0s	0.117s	0.005s	0.010s	0.035s	0s	0s	4.671s	37.2%	100.0%	
itpackv_mp_unscal_ompparallel:2@unknown:2552:2561	1.537s	0s	0.112s	0s	0.010s	0.010s	0s	0s	2.847s	75.0%	50.7%	
timestep_ompparallel:2@unknown:4105:4134	0.960s	0s	0.061s	0s	0.010s	0s	0s	0s	1.664s	22.3%	0.0%	
itpackv_mp_scal_ompparallel:2@unknown:2454:2463	0.884s	0s	0.047s	0.005s	0s	0.005s	0s	0s	1.644s	59.1%	100.0%	
itpackv_mp_pmult_ompparallel:2@unknown:2019:2029	0.843s	0s	0.056s	0s	0s	0.015s	0s	0s	1.503s	68.7%	100.0%	
timestep_ompparallel:2@unknown:304:345	0.774s	0s	0.075s	0s	0s	0.005s	0s	0s	1.153s	13.6%	0.0%	
itpackv_mp_sdot_ompparallel:2@unknown:2950:2954	0.870s	0.000s	0.274s	0.005s	0.025s	0.040s	0.005s	0s	1.093s	22.3%	50.2%	
mom_eqs_new_nc_ompparallel:2@unknown:7971:8045	0.392s	0s	0.037s	0s	0.005s	0.020s	0s	0s	0.692s	86.0%	100.0%	
itpackv_mp_unscal_ompparallel:2@unknown:2506:2540	0.203s	0.000s	0.019s	0s	0.005s	0.010s	0s	0s	0.371s	29.1%	0.0%	
itpackv_mp_itjcg_ompparallel:2@unknown:570:575	0.174s	0s	0.030s	0s	0s	0s	0s	0s	0.291s	100.0%	100.0%	
gwce_new_ompparallel:2@unknown:4639:4762	0.023s	0s	0.001s	0s	0s	0s	0.005s	0s	0.050s	0.0%	0.0%	

Bottom-up General Exploration

<no current project> - Intel VTune Amplifier@jlselogin2

Welcome | vtune_hpc_... | vtune_hpc_... | vtune_...

General Exploration General Exploration viewpoint (change)

INTEL VTUNE AMPLIFIER 2018

Collection Log | Analysis Target | Analysis Type | Summary | Bottom-up | Event Count | Platform

Grouping: Source Function / Function / Call Stack

	Front-End Bound					Back-End Bound										
						Memory Latency										
Source Function / Function / Call Stack	ITLB Overhead	BACLEARS	MS Entry	ICache Line Fetch	Bad Speculation	L1 Hit Rate	L2 Hit Rate	L2 Hit Bound	L2 Miss Bound	UTLB Overhead	Sl…	Sl…	C…	Page Walk	Split Loads	Split Stor
[Outside any loop]	4.5%	5.1%	1.0%	31.8%	2.3%	93.0%	93.7%	14.5%	13.3%	3.2%	0…	11…	0.…	7.6%	2.3%	0.7
[Loop at line 865 in _INTERNAL_25________src_kmp_barrier_cpp_ce635	0.0%	0.0%	6.7%	0.1%	0.0%	100.0%	0.0%	0.0%	2.9%	0.0%	0…	0.…	0.…	0.0%	0.0%	0.0
[Loop at line 5247 in gwce_new_ompparallel_for@5247]	0.0%	0.0%	0.0%	3.4%	0.7%	73.8%	96.0%	100.0%	86.1%	49.4%	1…	4.…	0.…	10.9%	0.0%	0.0
[Loop at line 56 in pjac_ompparallel_for@49]	0.0%	0.0%	0.0%	1.3%	0.4%	83.3%	100.0%	56.7%	0.0%	2.5%	1…	9.…	0.…	0.0%	25.9%	0.0
[Loop at line 7339 in mom_eqs_new_nc_ompparallel_for@7339]	0.0%	0.0%	0.0%	4.3%	0.9%	50.0%	96.3%	100.0%	97.9%	40.9%	0…	1.…	0.…	9.4%	0.0%	0.0
[Loop@0x6727bd in _f90_reduction_final_strided]	0.0%	0.0%	0.0%	2.0%	0.0%	94.4%	100.0%	34.2%	0.0%	6.0%	1…	23…	0.…	1.0%	0.0%	0.0
[Loop at line 2482 in timestep_ompparallel_for@2480]	0.0%	0.0%	0.0%	1.0%	0.0%	100.0%	100.0%	17.8%	0.0%	37.7%	0…	0.…	0.…	1.0%	0.0%	0.0
[Loop at line 2915 in timestep]	0.0%	0.0%	0.0%	0.0%	0.0%	100.0%	0.0%	0.0%	0.0%	14.0%	0…	0.…	0.…	0.0%	0.0%	0.0
[Loop at line 4082 in timestep]	0.0%	6.0%	0.0%	1.2%	7.2%	100.0%	100.0%	100.0%	0.0%	0.0%	1…	7.…	0.…	4.8%	0.0%	0.0
[Loop at line 55 in pjac_ompparallel_for@49]	0.0%	0.0%	0.0%	3.2%	0.0%	82.4%	100.0%	81.6%	0.0%	0.0%	1…	6.…	0.…	3.2%	29.4%	0.0
[Loop@0x6727a0 in _f90_reduction_final_strided]	0.0%	0.0%	0.0%	0.0%	0.0%	88.9%	100.0%	100.0%	0.0%	0.0%	1…	4.…	0.…	2.0%	0.0%	0.0
[Loop at line 1840 in __kmp_fork_call]	0.0%	0.0%	0.0%	43.2%	0.0%	100.0%	100.0%	38.6%	0.0%	0.0%	0…	0.…	0.…	6.8%	0.0%	0.0
[Loop at line 1949 in __kmp_join_barrier]	2.8%	0.0%	0.0%	45.1%	8.5%	80.0%	0.0%	0.0%	0.0%	0.0%	0…	0.…	0.…	14.1%	0.0%	0.0
[Loop at line 772 in __kmpc_for_static_init_4]	8.3%	20.8%	12.5%	58.3%	4.2%	100.0%	0.0%	0.0%	0.0%	0.0%	1…		0.…	20.8%	0.0%	0.0
[Loop@0x6c0f46 in cvtas_x_to_a]	0.0%	44.4%	0.0%	44.4%	0.0%	100.0%	0.0%	0.0%	0.0%	0.0%	0…	0.…	0.…	4.4%	0.0%	0.0
[Loop@0x6808e1 in _f90_reduction_init_array]	0.0%	0.0%	0.0%	0.0%	0.0%	100.0%	0.0%	0.0%	0.0%	0.0%			0.…	0.0%	0.0%	0.0
func@0x13a00	0.0%	27.8%	0.0%	5.6%	0.0%	100.0%	0.0%	0.0%	0.0%	0.0%	1…		0.…	5.6%	0.0%	0.0
[Loop at line 957 in chgcon]	0.0%	0.0%	0.0%	40.0%	0.0%	100.0%	0.0%	0.0%	0.0%	0.0%			0.…	5.7%	0.0%	0.0
[Loop at line 1130 in _INTERNAL_22________src_kmp_lock_cpp_7c9c5b	11.4%	0.0%	0.0%	11.4%	0.0%	100.0%	0.0%	0.0%	0.0%	0.0%	0…	0.…	0.…	5.7%	0.0%	0.0
[Loop at line 2122 in prbndx]	0.0%	0.0%	0.0%	0.0%	0.0%	100.0%	0.0%	0.0%	0.0%	0.0%	1…		0.…	0.0%	28.6%	0.0
[Loop@0x380e0 in getenv]	0.0%	31.3%	0.0%	0.0%	12.5%	0.0%	100.0%	100.0%	0.0%	0.0%	0…	0.…	0.…	0.0%	0.0%	0.0

Python

Profiling Python is straightforward in VTune™ Amplifier, as long as one does the following:

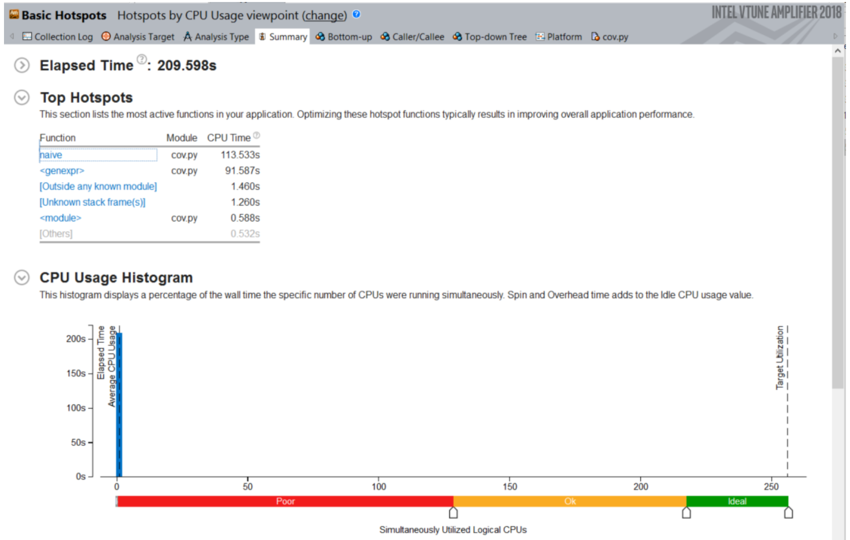
- The “application” should be the full path to the python interpreter used
- The python code should be passed as “arguments” to the “application”

In Theta this would look like this:

```
aprun -n 1 -N 1 amplxe-cl -c hotspots -r res_dir \
                -- /usr/bin/python3 mycode.py myarguments
```

Simple Python Example on Theta

```
aprun -n 1 -N 1 amplxe-cl -c hotspots -r vt_pytest \
                          -- /usr/bin/python ./cov.py naive 100 1000
```



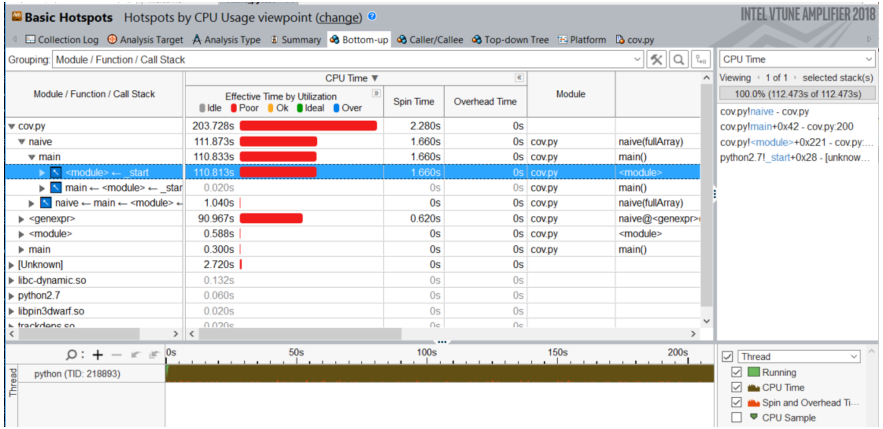
Naïve implementation of the calculation of a covariance matrix

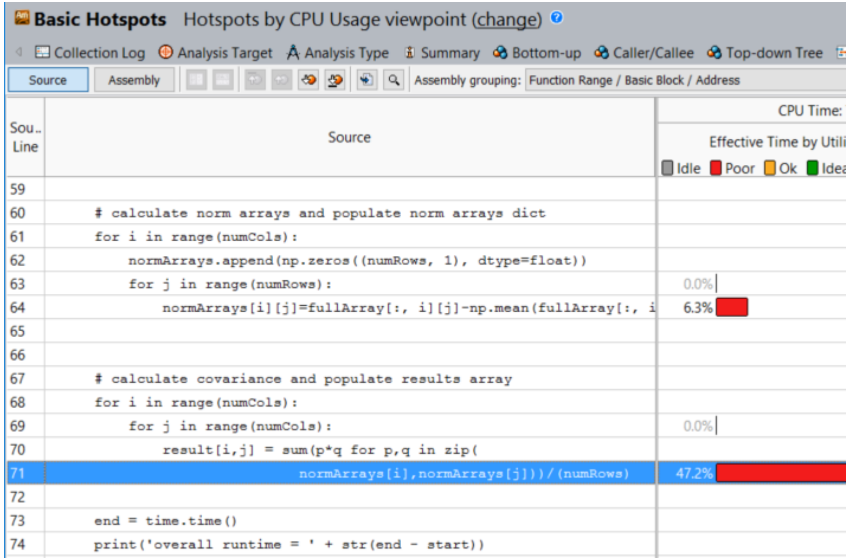
Summary shows:

- Single thread execution
- Top function is "naive"

Click on top function to go to Bottom-up view

Bottom-up View and Source Code





Inefficient array multiplication found quickly
We could use numpy to improve on this

Note that for mixed Python/C code a Top-Down view can often be helpful to drill down into the C kernels

Useful Options on Theta

If finalization is slow you can use -finalization-mode=deferred and simply finalize on a login node or a differenet machine

If the collection stops because too much data has been collected you can override that with the -data-limit=0 option (unlimited) or to a number (in MB)

Use the -trace-mpi option to allow VTune Amplifier to assign execution to the correct task when not using the Intel® MPI Library.

Reduce results size by limiting your collection to a single node using an mpmd style execution:

```
aprun -n X1 -N Y amplxe-cl -c hpc-performance -r resdir -- ./exe : \
      -n X2 -N Y ./exe
```

Resources

Product Pages

- https://software.intel.com/sites/products/snapshots/application-snapshot
- https://software.intel.com/en-us/advisor
- https://software.intel.com/en-us/intel-vtune-amplifier-xe

Detailed Articles

- https://software.intel.com/en-us/articles/intel-advisor-on-cray-systems
- https://software.intel.com/en-us/articles/using-intel-advisor-and-vtune-amplifier-with-mpi
- https://software.intel.com/en-us/articles/profiling-python-with-intel-vtune-amplifier-a-covariance-demonstration

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. 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

Software

EMON Collection

General Exploration analysis may be performed using EMON

- Reduced size of collected data
- Overall program data, no link to actual source (only summary)
- Useful for initial analysis of production and large scale runs
- Currently available as experimental feature

```
export AMPLXE_EXPERIMENTAL=emon
aprun […] amplxe-cl -c general-exploration -knob summary-mode=true[…]
```

VTune Cheat Sheet

```
amplxe-cl -c hpc-performance -flags -- ./executable
```

- `--result-dir=./vtune_output_dir`
- `--search-dir src:=../src --search-dir bin:=./`
- `-knob enable-stack-collection=true -knob collect-memory-bandwidth=false`
- `-knob analyze-openmp=true`
- `-finalization-mode=deferred`
- `-data-limit=125` ← `in mb`
- `-trace-mpi`

https://software.intel.com/en-us/vtune-amplifier-help-amplxe-cl-command-syntax

Advisor Cheat Sheet

```
advixe-cl –c roofline/depencies/map –flags -- ./executable
```

- `--project-dir=./advixe_output_dir`
- `--search-dir src:=../src --search-dir bin:=./`
- `-no-auto-finalize`
- `--interval 1`
- `-data-limit=125` ← in mb

https://software.intel.com/en-us/advisor-help-lin-command-line-interface-reference

Profiling a single rank (for a 4 node, 256 rank job)

```
mpirun –n 1 \
amplxe-cl –c hotspots \
-- ./exe \
: -n 255 ./exe
```

Intel (JLSE/BEBOP)

```
profile1.sh:
#!/bin/bash
export PE_RANK=$ALPS_APP_PE
export PMI_NO_FORK=1
if [ "$PE_RANK" == 0];then
    $1 -- $2
else
    $2
fi
aprun -n 256 –N 64 profile1.sh
"amplxe-cl –c hotspots" "exe"
```

Intel + Cray(Theta)