Software

PROFILING YOUR APPLICATION WITH INTEL® VTUNE™ AMPLIFIER AND INTEL® ADVISOR

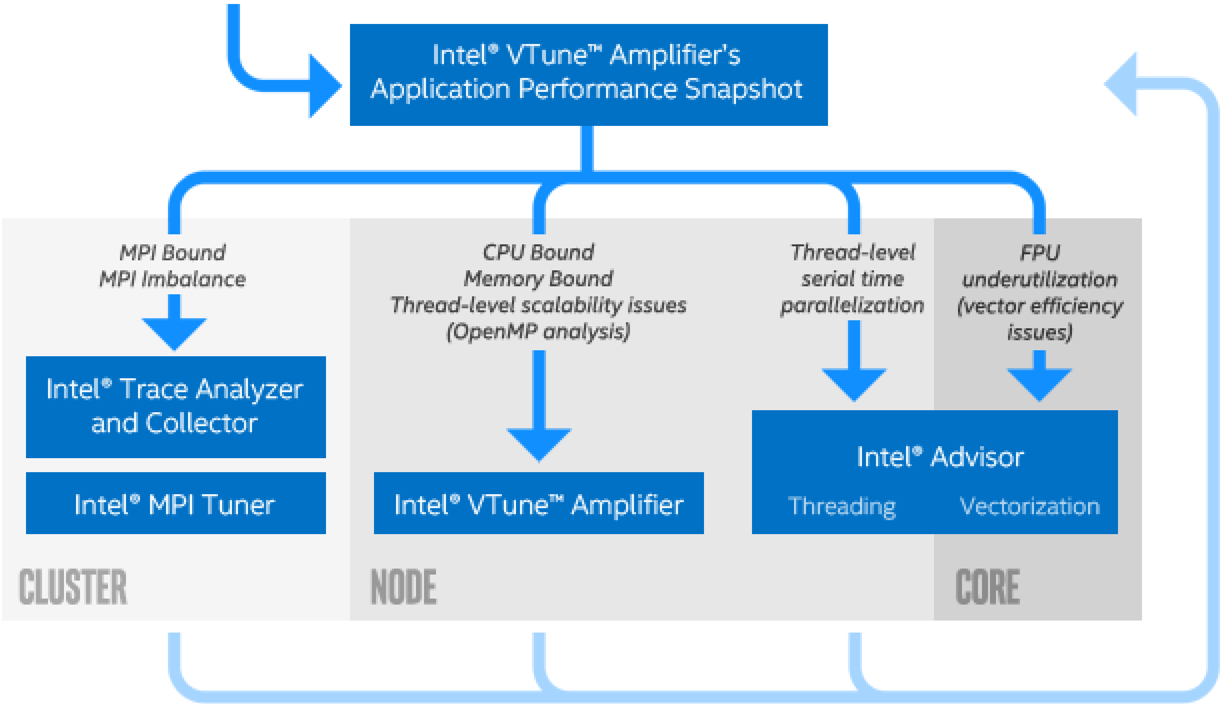
Carlos Rosales-Fernandez

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)

Tuning Workflow



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

INTEL® ADVISOR

Vectorization and Threading

Intel® Advisor

Modern HPC processors explore different level of parallelism:

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

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

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.

Some things of 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
```

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
```

Cache-Aware Roofline

Next Steps

If Under the Vector Add Peak

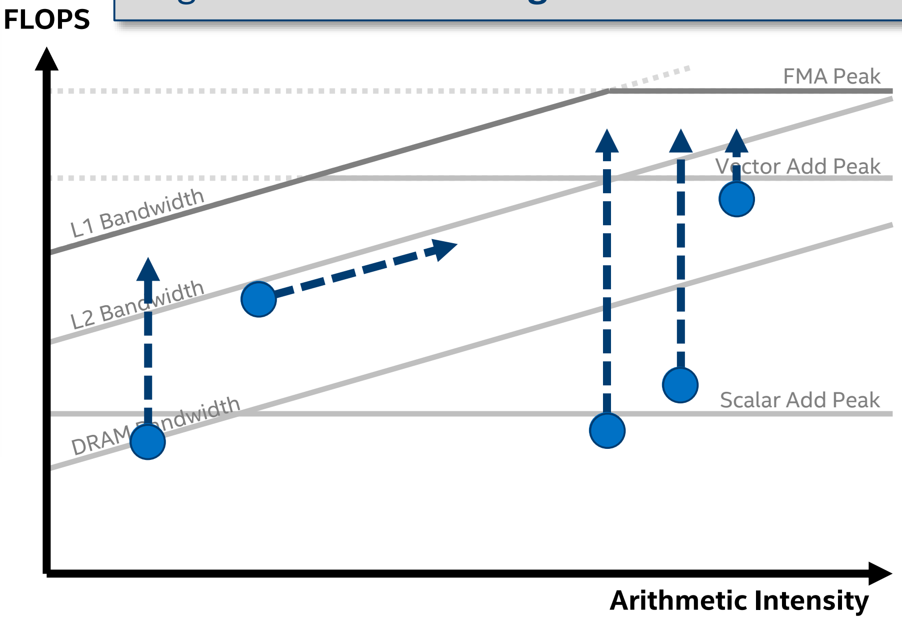
Check "Traits" in the Survey to see if FMAs are used. If not, try altering your code or compiler flags to **induce FMA usage.**

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 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 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.

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:

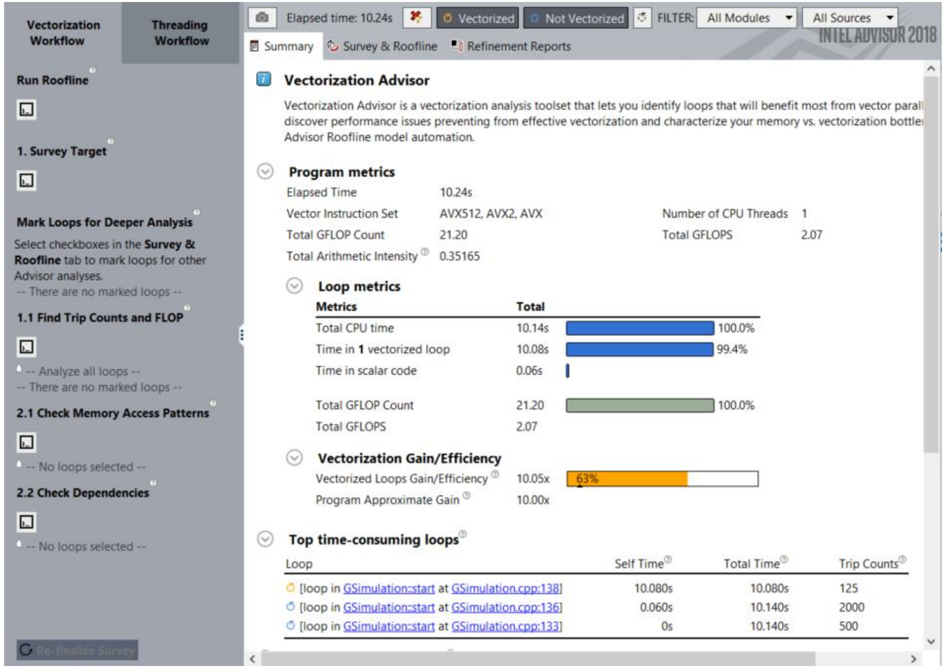
```
export LD_LIBRARY_PATH=/opt/intel/advisor/lib64:$LD_LIBRARY_PATH
source /opt/intel/advisor/advixe-vars.sh
export PMI_NO_FORK=1
aprun -n 1 -N 1 advixe-cl --collect survey --project-dir ./adv_res --search-dir src:=./ \
                        --search-dir bin:=./ -- ./nbody.x
aprun -n 1 -N 1 advixe-cl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

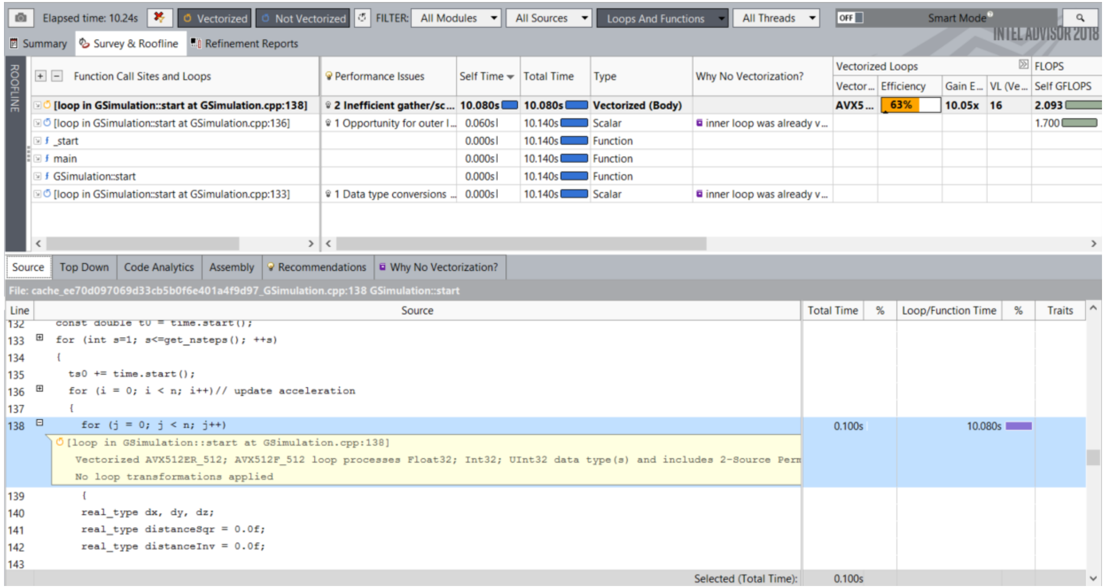


GUI left panel provides access to further tests

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)



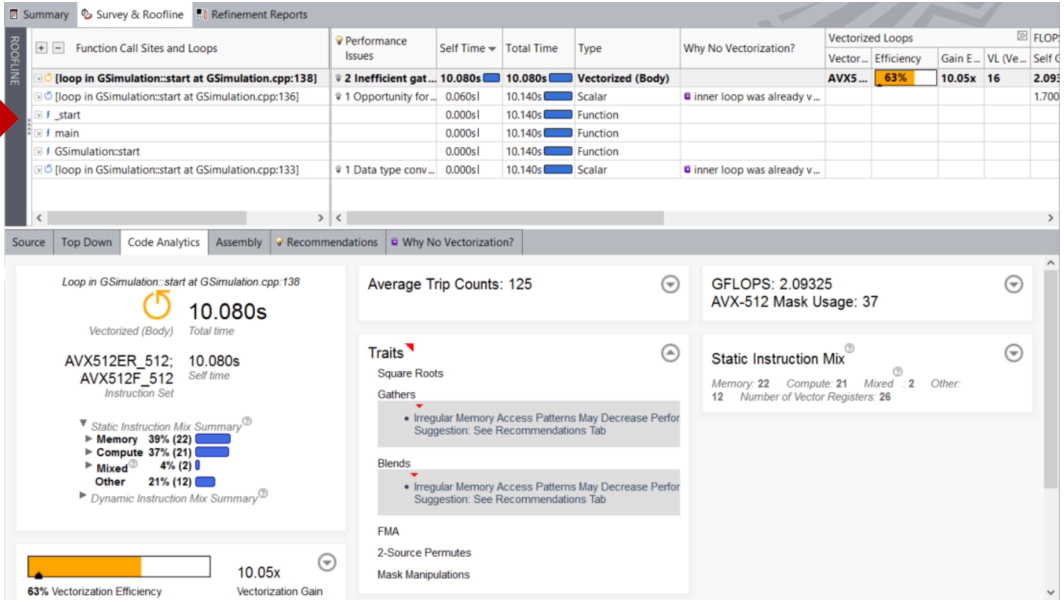
Inline information regarding loop characteristics

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

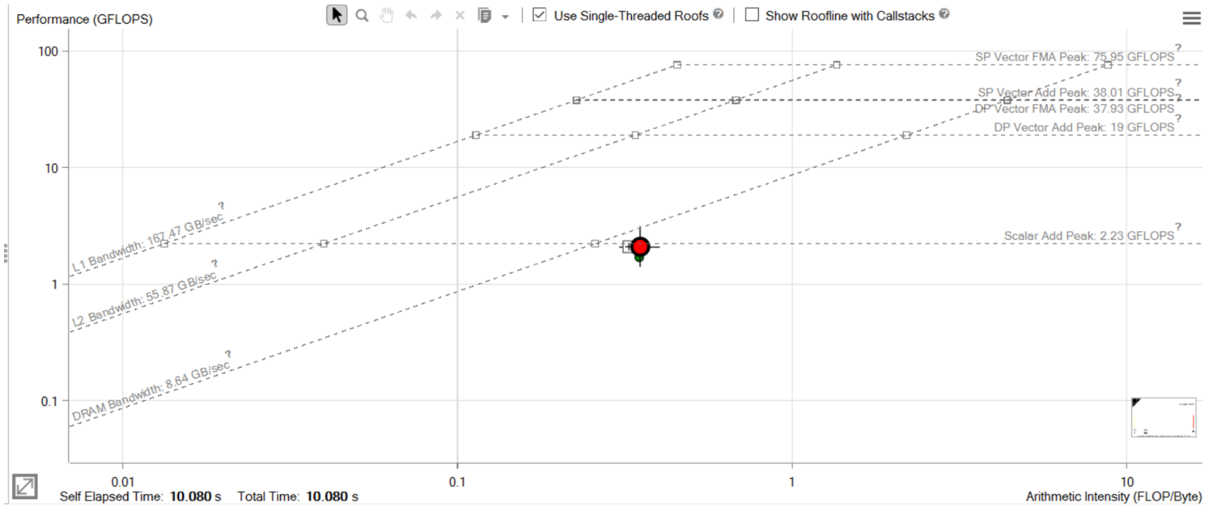
Survey Report (Code Analytics)

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

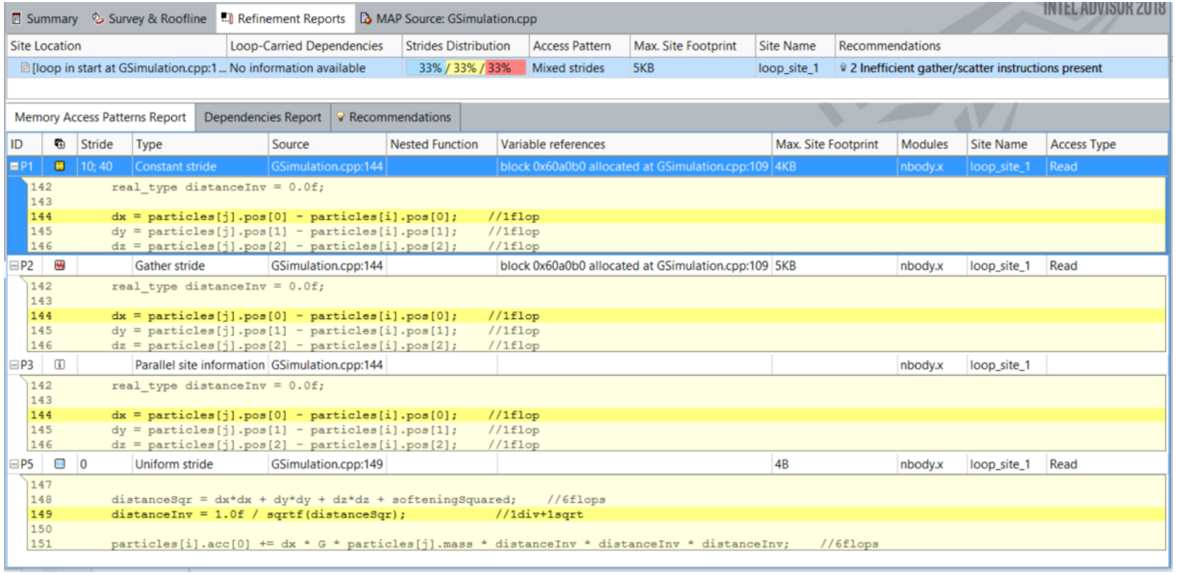
Code vectorized, but performance on par with scalar add peak?

- 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

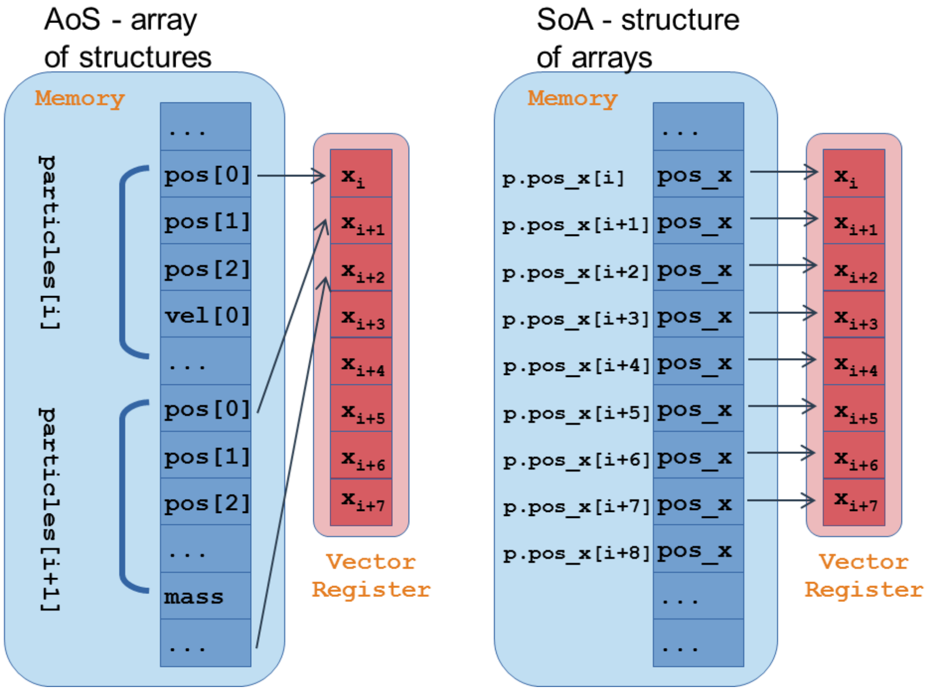
Re-structuring the code into a Structure Of Arrays (SOA) may lead to unit stride access and more effective vectorization

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;
};
```



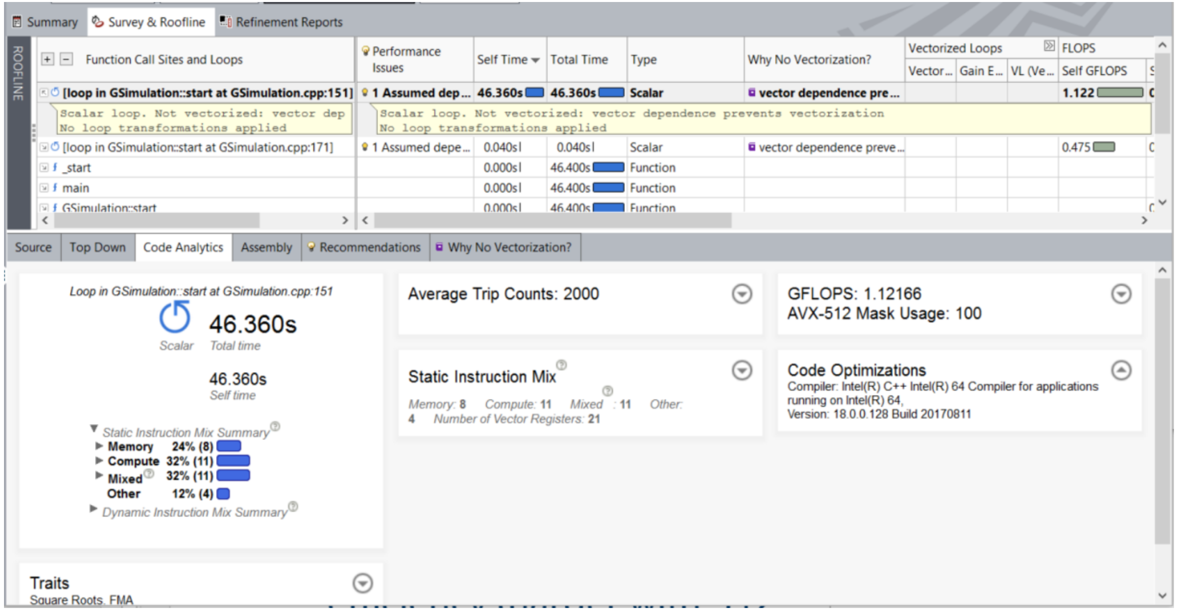
Performance After Data Structure Change

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:

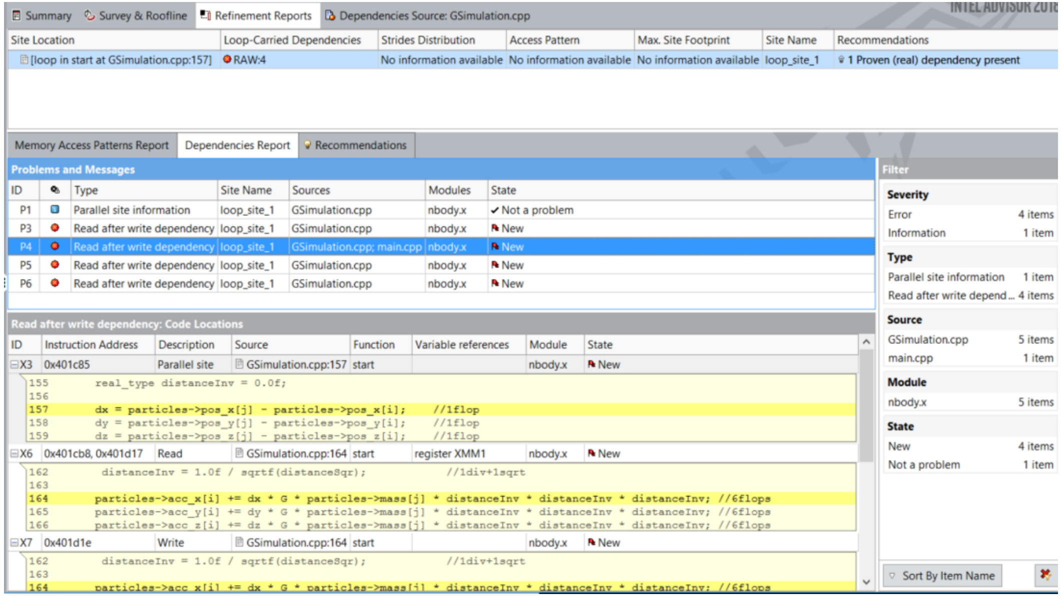
- Performance is lower
- Main loop is no longer vectorized
- Assumed vector dependence prevents automatic vectorization



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:

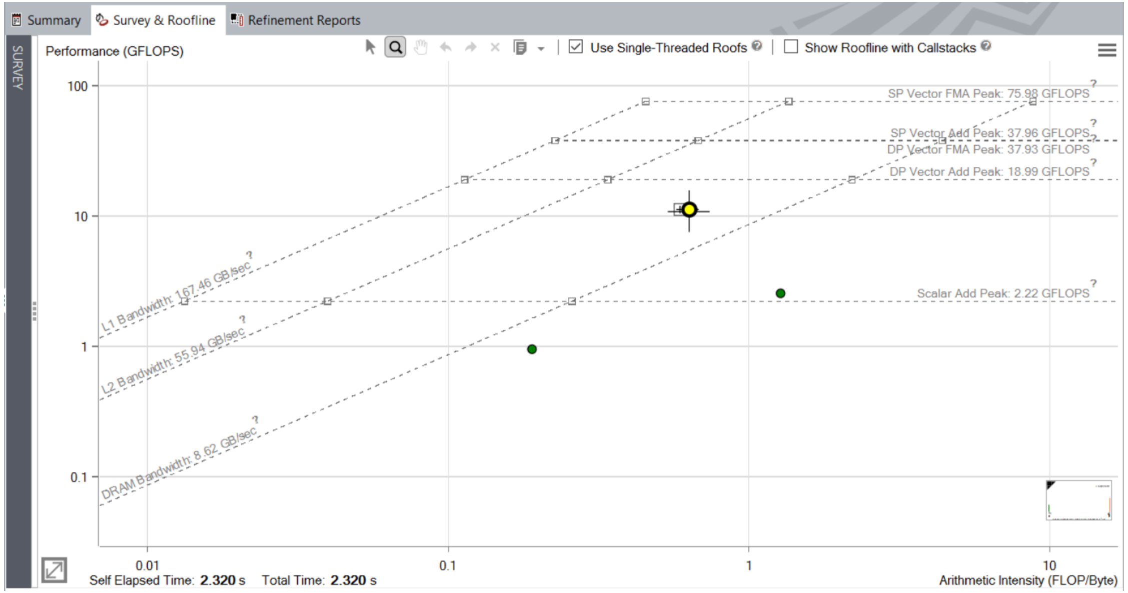
```
#pragma omp simd reduction(+:sumx)
for (k = 0;k < size2; k++)
{
    sumx += x[k]*b[k];
}
```

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



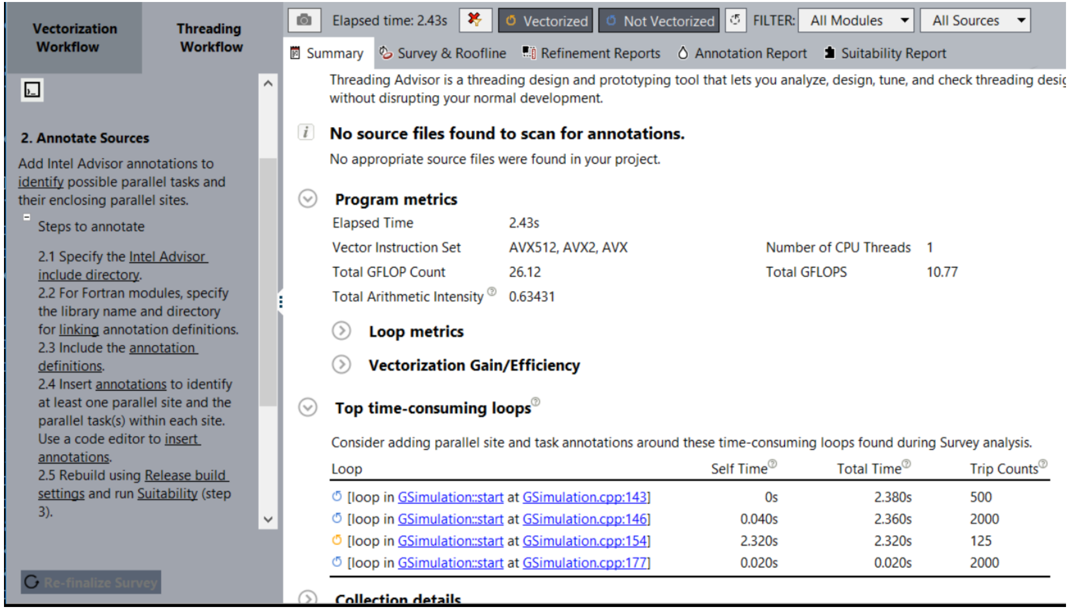
New memory access pattern plus vectorization produces much improved performance!

What next?

Performance per core may be improved, but it is capped at ~5x current value.

Let's explore threading with a suitability analysis.

- Recompile including annotation definitions
- Add headers to file
- Annotate suggested loops
- Run suitability collection



Annotating the code

Add annotations as shown on the left sample

Complex sites may be analyzed in more detail using task sections if needed

- ANNOTATE_SITE_BEGIN / ANNOTATE_SITE_END
- ANNOTATE_TASK_BEGIN / ANNOTATE_TASK_END

Recompile including annotation definitions:

```
-I/opt/intel/advisor/include
```

Collect suitability data

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

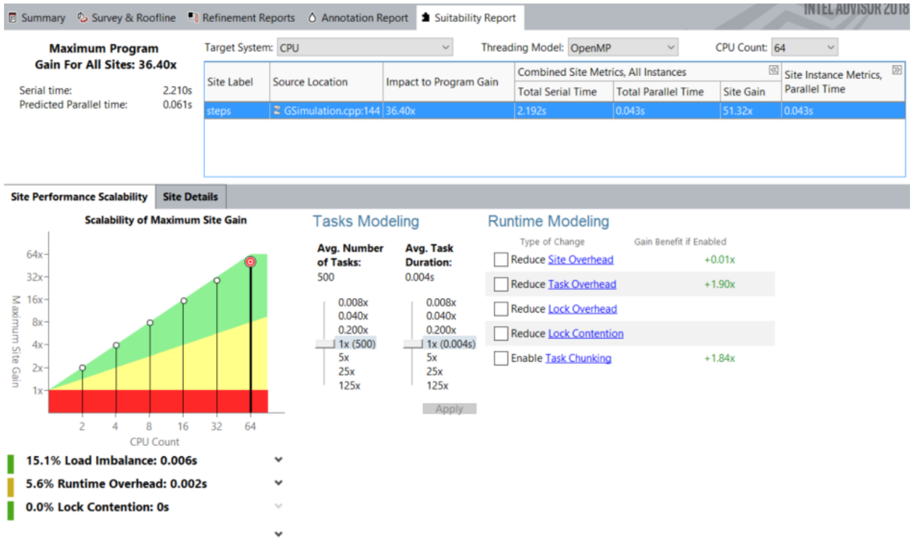
```
 #include "advisor-annotate.h"
...

ANNOTATE_SITE_BEGIN(steps)
for (int s=1; s<=get_nsteps(); ++s)
{
  ...

  ANNOTATE_TASK_BEGIN(particles)
  for (i = 0; i < n; i++)
  {
    ...

   }
   ANNOTATE_TASK_END(particles)
}
ANNOTATE_SITE_END(steps)
```

Suitability report

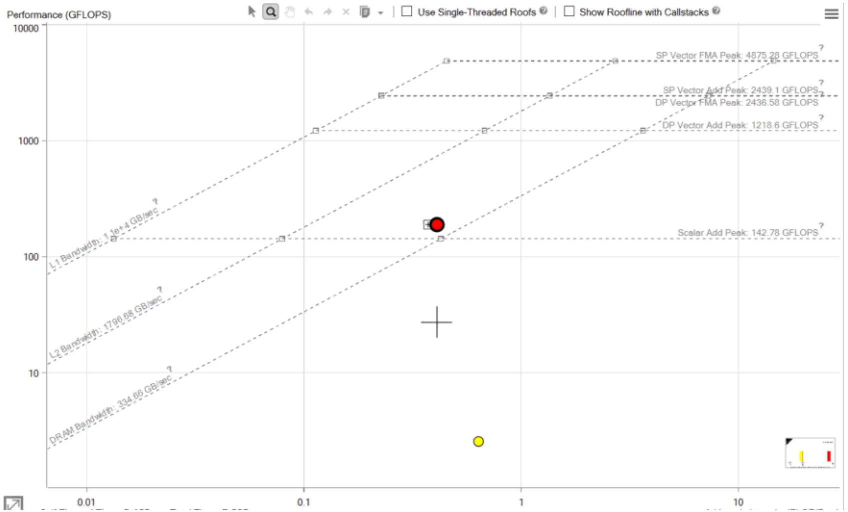
Good speedup expected, but far from ideal (~56% efficiency).

Modeling shows that increasing the task length would improve efficiency.

Next step: add omp parallel region to code and re-rest



Roofline for Threaded Version




```
for (int s=1; s<=get_nsteps(); ++s)
{
  ts0 += time.start();

#pragma omp parallel for
  for (i = 0; i < n; i++)     // update acceleration
  {
    ...
    real_type ax_i = particles->acc_x[i];
    real_type ay_i = particles->acc_y[i];
    real_type az_i = particles->acc_z[i];

#pragma omp simd reduction(+:ax_i,ay_i,az_i)
    for (j = 0; j < n; j++)
    {
      real_type dx, dy, dz;
      real_type distanceSqr = 0.0f;
      real_type distanceInv = 0.0f;

      dx = particles->pos_x[j] - particles->pos_x[i];
```

Now using regular roofline, instead of single-threaded

Still room for improvement, but at this point we need additional detail regarding shared resource utilization

Intel® VTune™ Amplifier

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
hpc-performance	HPC Performance Characterization
locksandwaits	Locks and Waits
memory-access	Memory Access
memory-consumption	Memory Consumption
system-overview	System Overview
…	

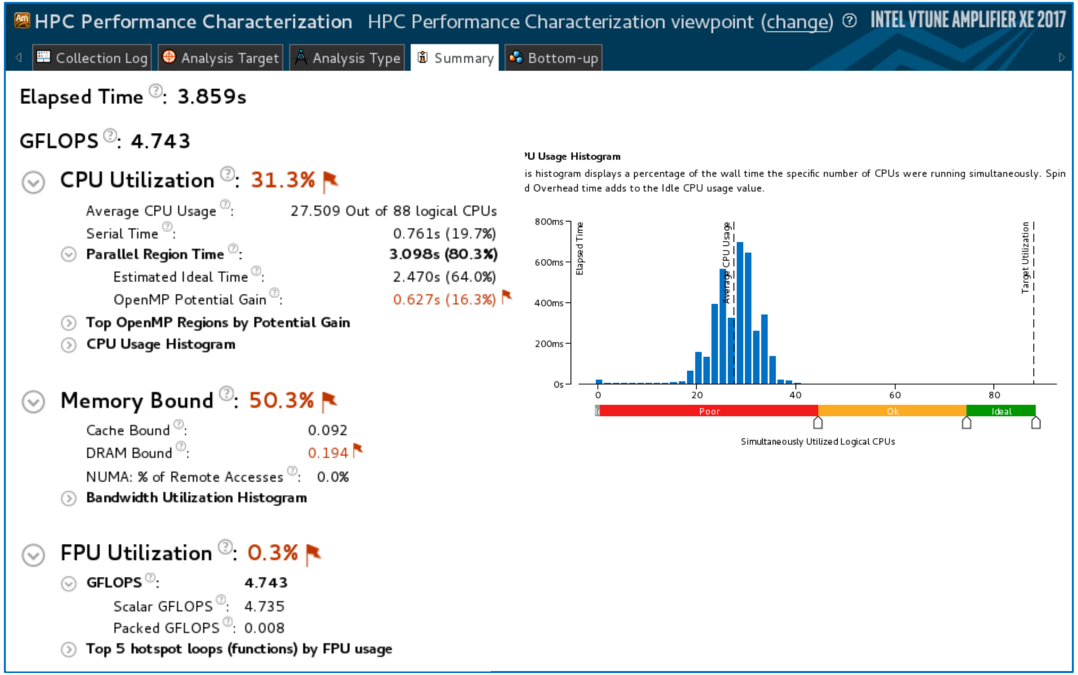
Python Support

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

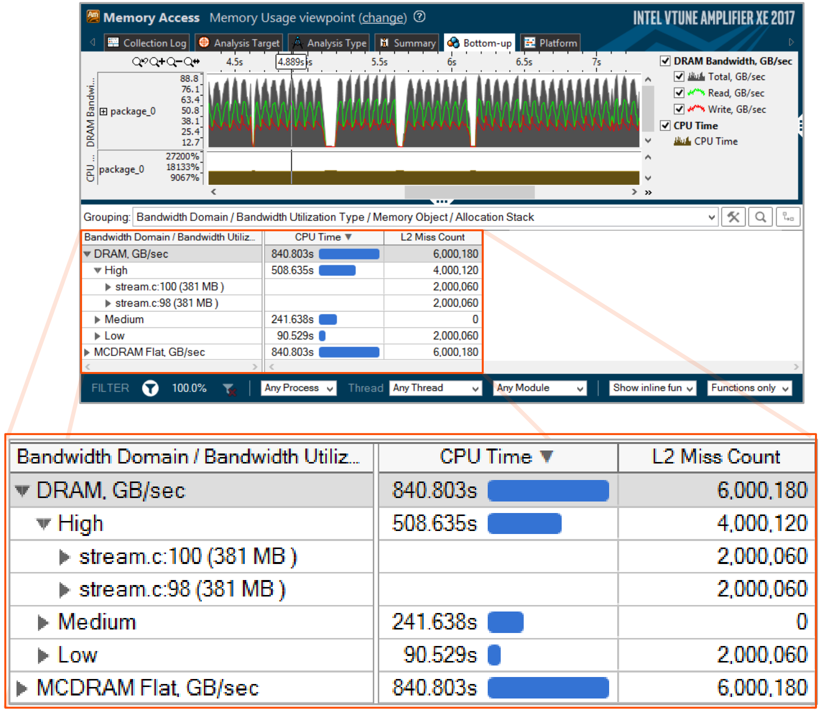
- Attribute cache misses to data structures (not just the code causing the miss)
- Support for custom memory allocators

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



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
```

Sample Script

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

Basic scheduler info (the usual)

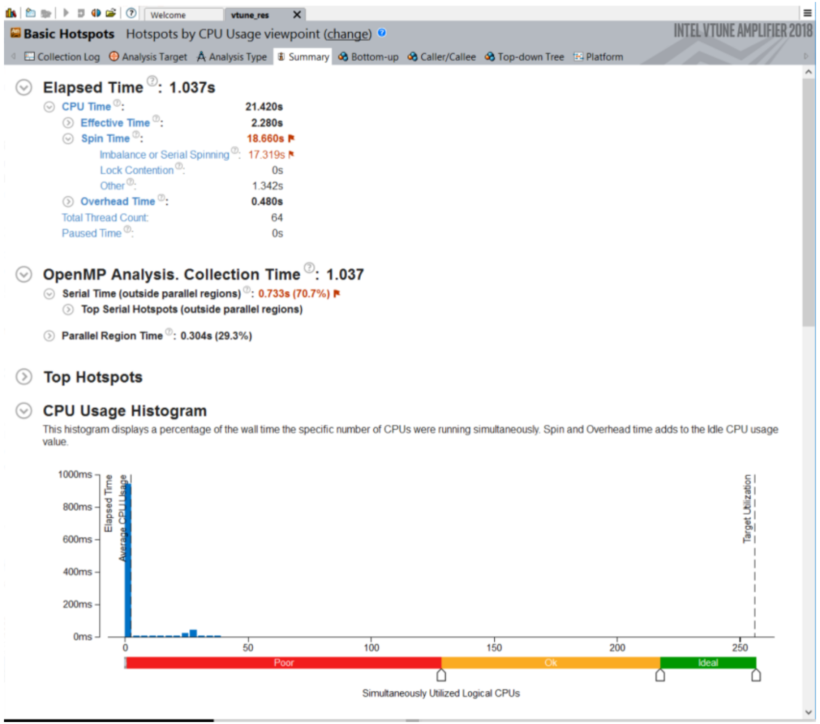
```
export LD_LIBRARY_PATH=/opt/intel/vtune_amplifier/lib64:$LD_LIBRARY_PATH
source /opt/intel/vtune_amplifier/amplxe-vars.sh
export PMI_NO_FORK=1
export OMP_NUM_THREADS=64; export OMP_PROC_BIND=spread; export OMP_PLACES=cores
```

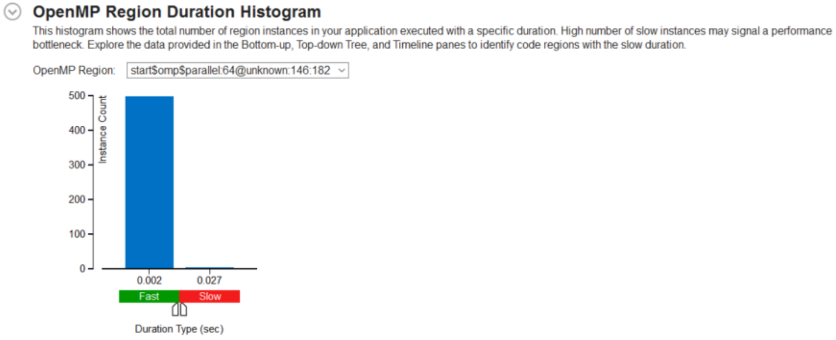
Environment setup

Invoke VTune™ Amplifier

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

Hotspots analysis for nbody demo (ver7: threaded)



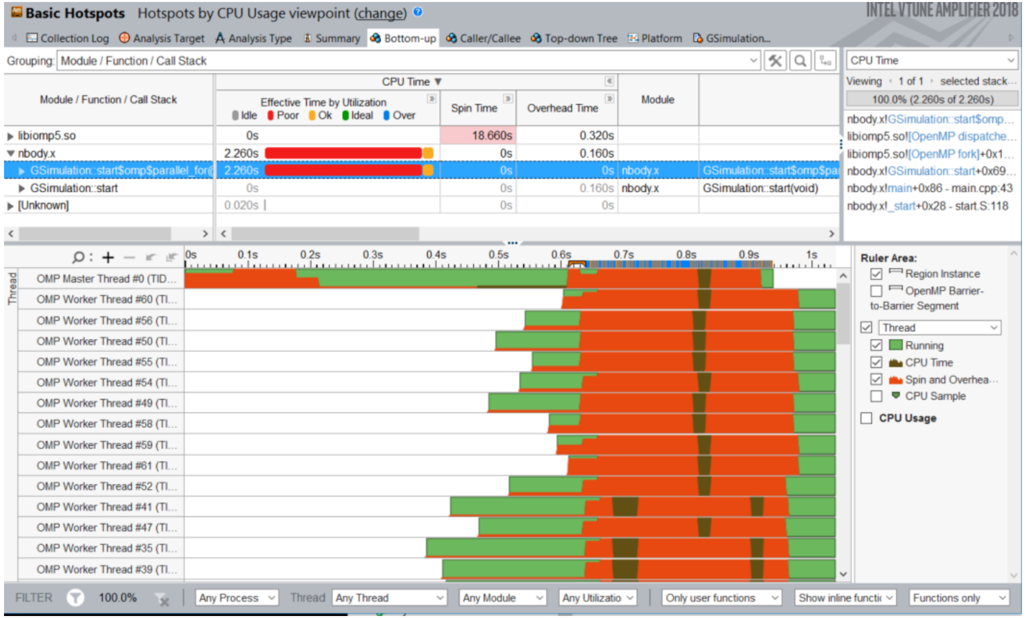


Lots of spin time indicate issues with load balance and synchronization

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 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.

Python

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

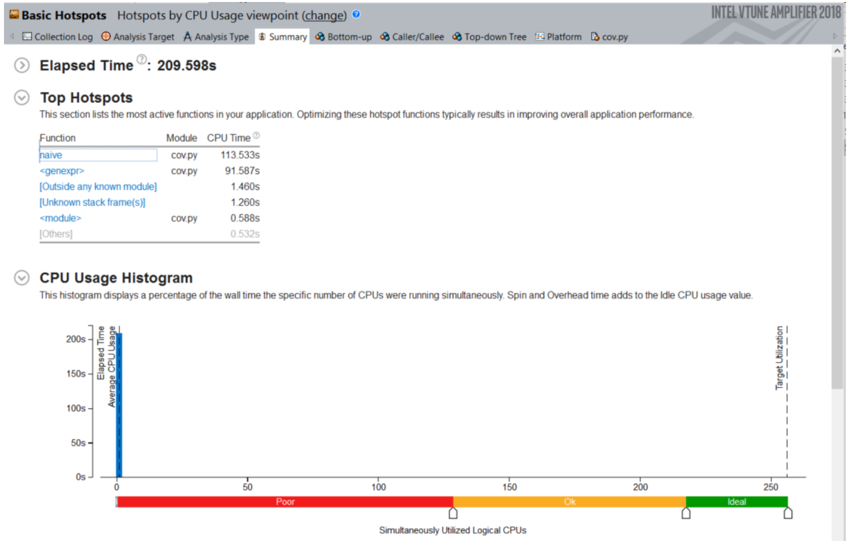
- The "application" should be the full path to the python interpreter used
- The python code should be passed as "arguments" to the "application"

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
```



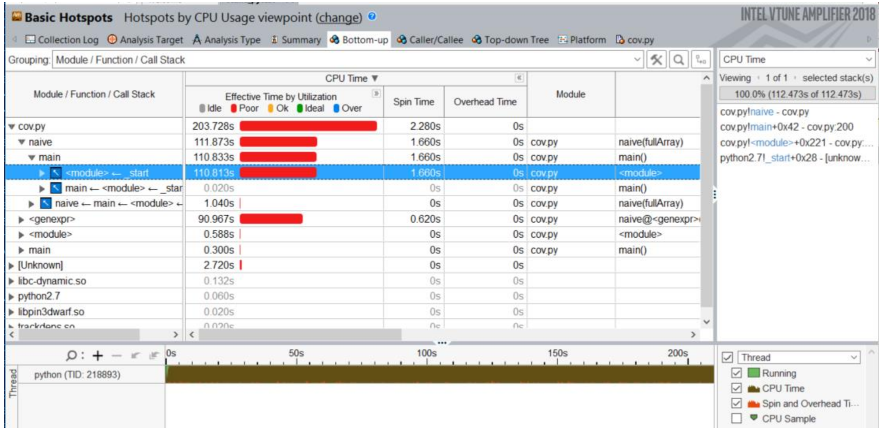
Naïve implementation of the calculation of a covariance matrix

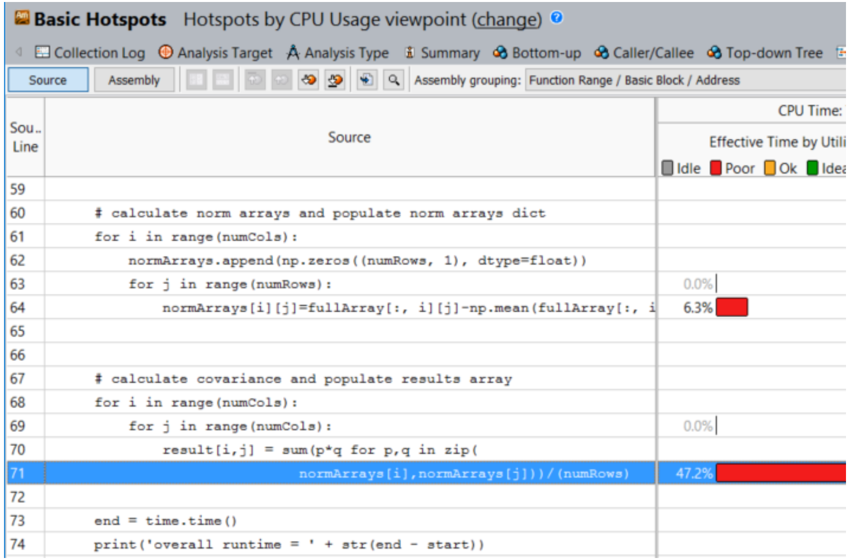
Summary shows:

- 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
```

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[…]
```

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

Software