PERFORMANCE TUNING USING INTEL® ADVISOR AND VTUNE™ AMPLIFIER

Carlos Rosales-Fernandez

February 20th 2019

Agenda

Tools Covered

- VTune™ Amplifier Application Performance Snapshot
- Vtune™ Amplifier
- Intel® Advisor

Examples

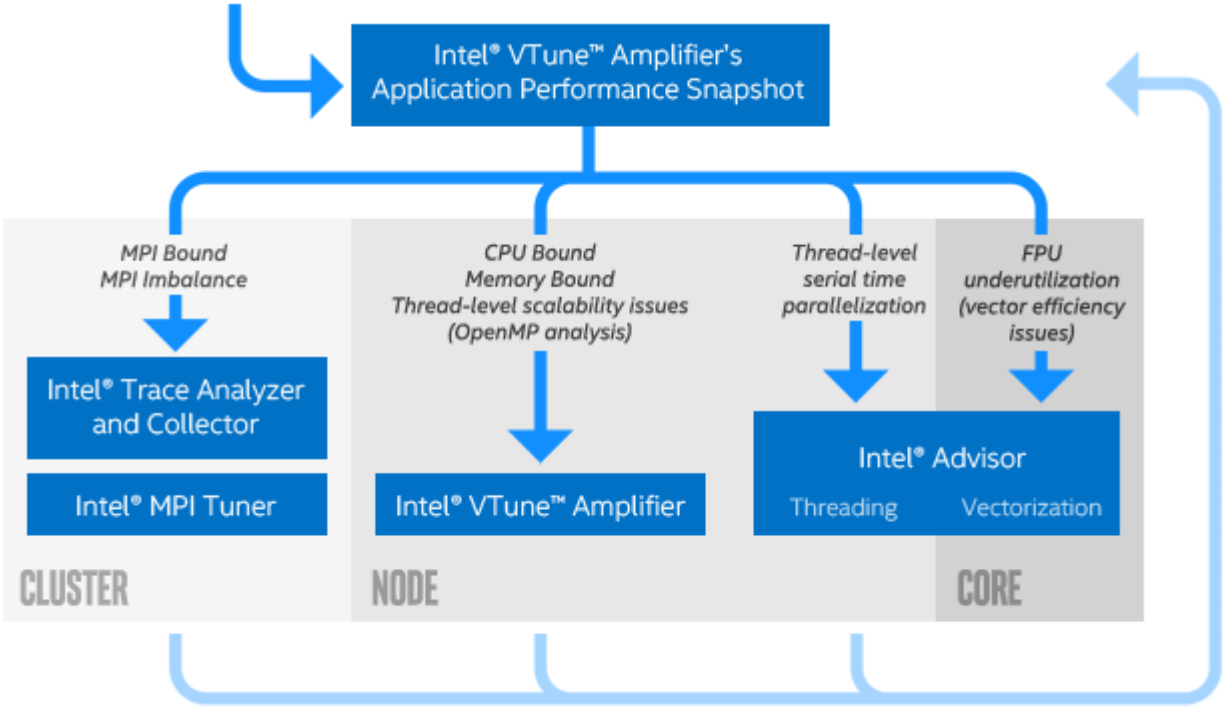
- 3D stencil (C)
- Matrix covariance (Python)

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)

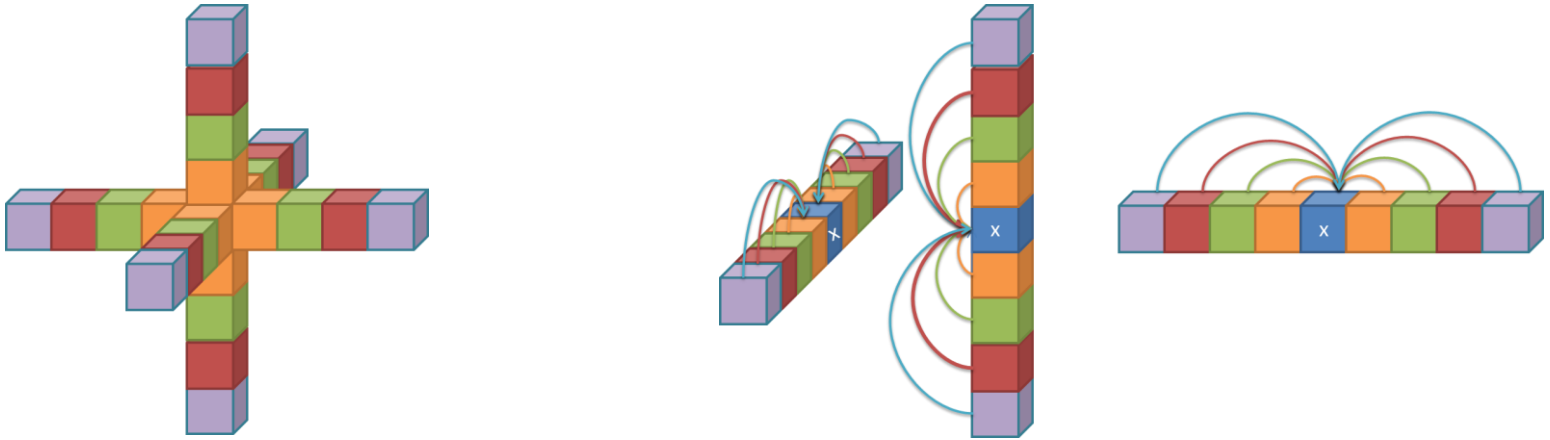
Tuning Workflow



INTRO

Iso3DFD - A Simple 3D Stencil

- For computing $P_{t+1}(x,y,z)$, we need to use all neighbors in the 3 dimensions of $P_t(x,y,z)$.
- The stencil looks like a 3D cross in Iso3DFD

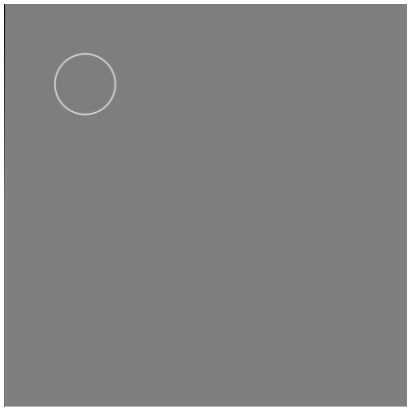
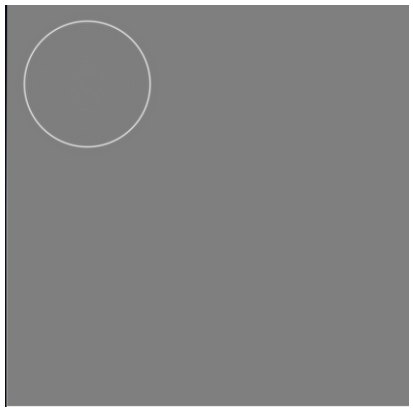
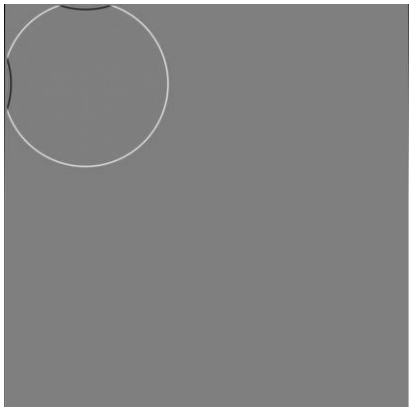
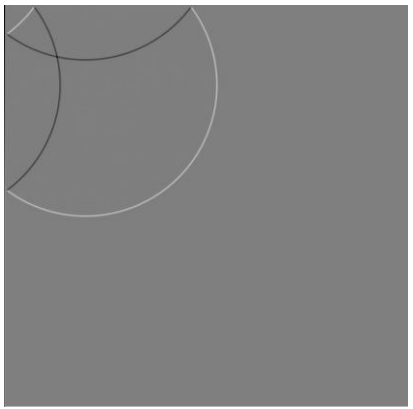


Iso3DFD 2D CUT

This is a 2D cut of our 3D volume

We create a perturbation and look at the pressure for 4 different time steps

We see that there is no boundary condition

Naïve Wave Propagation Kernel Implementation

```
for( int ix=0; ix<nx; ix++ ) {
  for( int iy=0; iy<ny; iy++ ) {
    for( int iz=0; iz<nz; iz++ ) {
     int offset = iz*nx*ny + iy*nx + ix;
     float value = 0.0;
     value += ptr_prev[offset]*coeff[0];
     for( int ir=1; ir<=8; ir++ ) {
       value += coeff[ir] * (ptr_prev[offset + ir] + ptr_prev[offset - ir]);
       value += coeff[ir] * (ptr_prev[offset + ir*nx] + ptr_prev[offset - ir*nx]);
       value += coeff[ir] * (ptr_prev[offset + ir*nx*ny] + ptr_prev[offset - ir*nx*ny]);
     }
     ptr_next[offset] = 2.0f* ptr_prev[offset] - ptr_next[offset] + value*ptr_vel[offset];
    }}}
```

- 3D Finite Difference
- Acoustic isotropic, pressure only scheme
- 16th order in space 2nd order in time
- No proper boundary conditions for this example

Hardware

For the rest of the presentation we are using the following system configuration

- Intel® Xeon Phi™ Processor 7250 @ 1.4 GHz
- Each processor as 68 cores
- Each core has 4 hyperthreads
- 16 GB on-chip MCDRAM
- 96 GB DDR4
- Memory configured in cache-quadrant mode

This is only slightly different from the 7230 processors in Theta, and the instructions have been customized to work there without modification.

GETTING A BASELINE FOR ISO3DFD

Baseline Version

We start with a naïve implementation that includes several performance impacting mistakes. Among other things this version should show:

- No parallelization
- No vectorization
- Sub-optimal memory accesses

Let's pretend we know nothing about this code and simply follow the tuning workflow to answer the following questions:

- Is the code performing well overall?
- How much room for improvement is there?
- Where is the principal bottleneck?

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

Application Performance Snapshot 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 LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/intel/vtune_amplifier/lib64
$ export PMI_NO_FORK=1
```

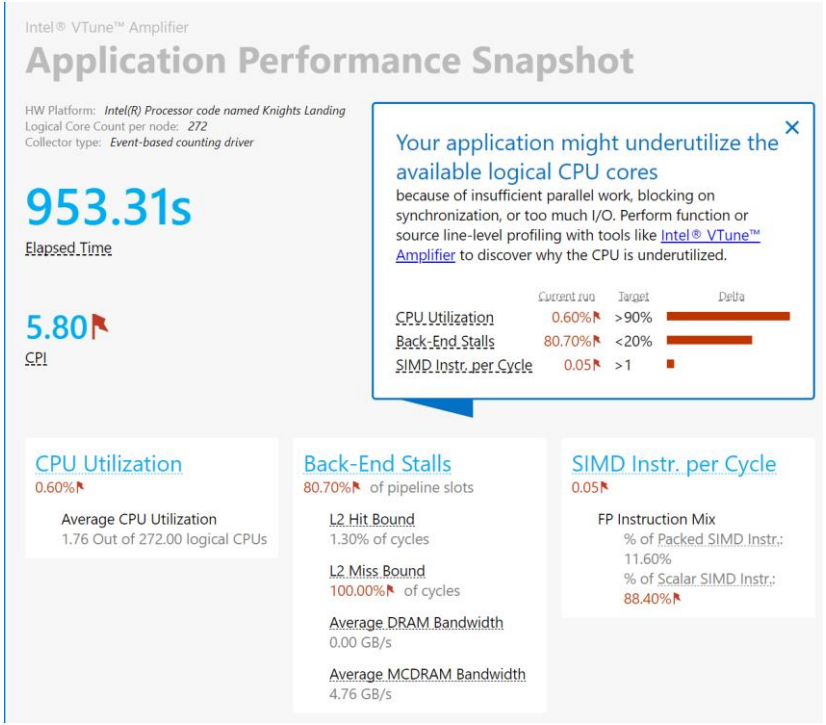
Launch your job in interactive or batch mode:

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

Produce text and html reports:

```
$ aps -report=./aps_result_YYYMMDD
```

Overall Performance for Baseline Version



- Subpar performance in all areas
- Most severe issues related to CPU utilization
- First, let's figure out if optimizing this code is worth our time

Additional information would be available if this version was OpenMP* or MPI enabled

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

Cache-Aware Roofline

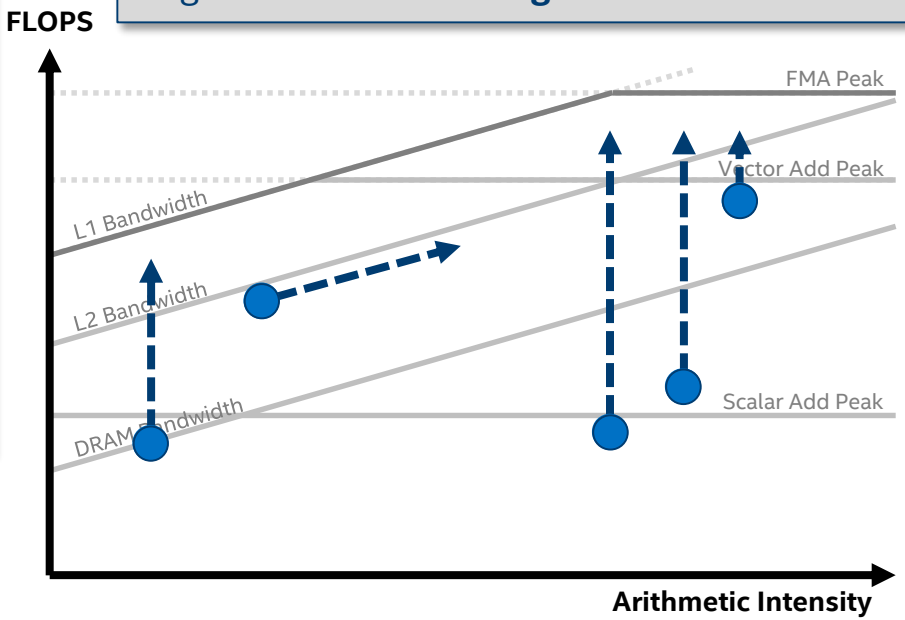
Next Steps

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.

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

```
aprun -n 1 -N 1 advixe-cl -c roofline --project-dir ./adv -- ./exe [args]
```

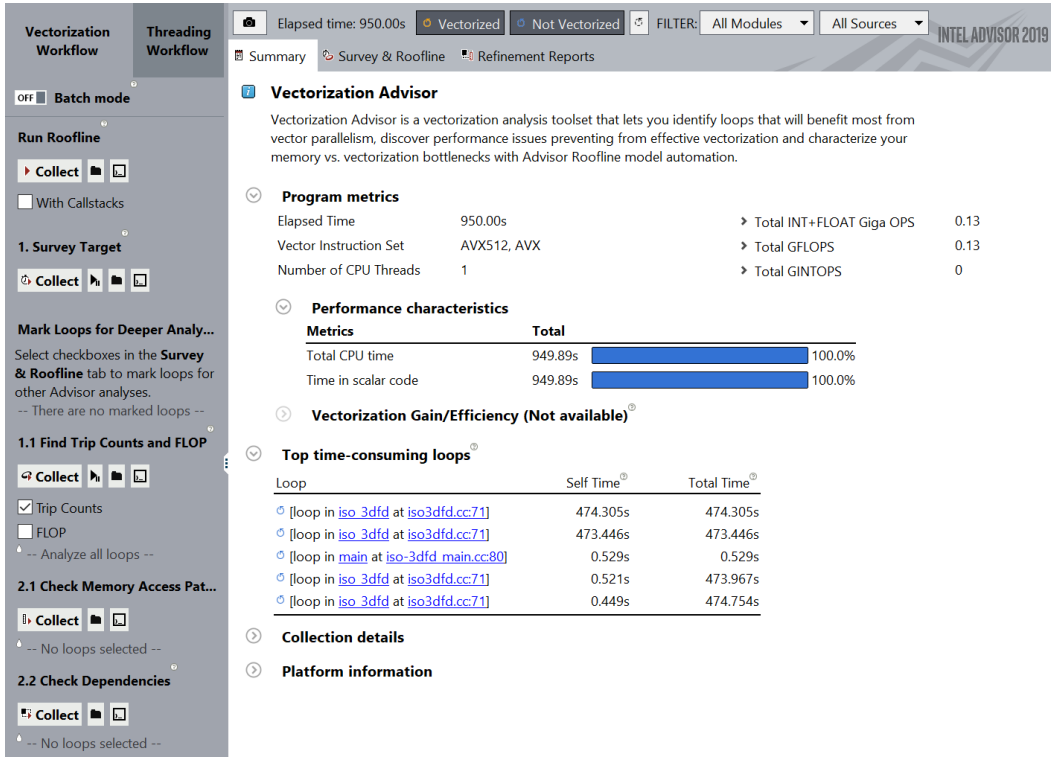
→ Invoke Intel® Advisor

Intel® Advisor Summary

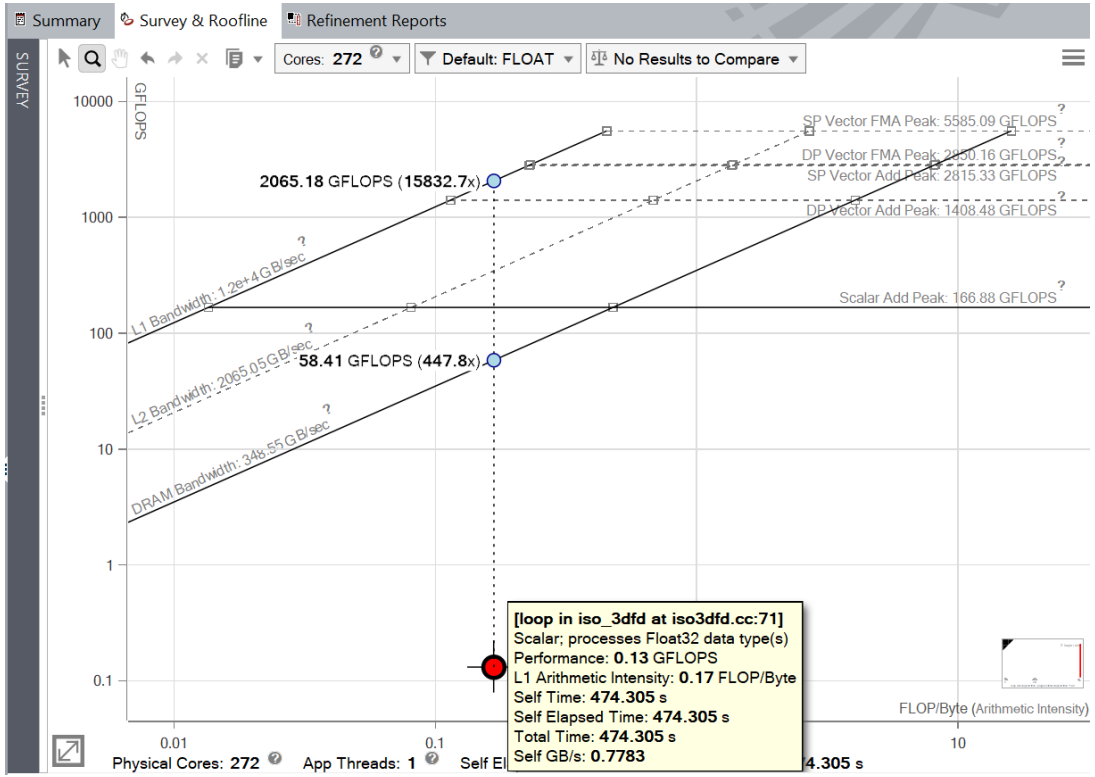
Collect roofline data

Check for data access inefficiencies

Verify that vectorization is possible



Roofline and Room for Improvement



Performance is far from any hardware limit

- Below DRAM BW
- Below FP scalar add peak

It seems worthwhile to invest some time in optimizing this code

Intel® VTune™ Amplifier

VTune Amplifier is a full system profiler

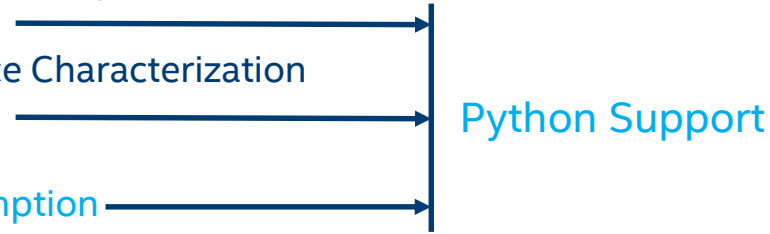
- Accurate
- Low overhead
- Comprehensive (microarchitecture, memory, IO, treading, …)
- Highly customizable interface
- Direct access to source code and assembly

Analyzing code access to shared resources is critical to achieve good performance on modern systems

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
…	



The HPC Performance Characterization Analysis

Threading: CPU Utilization

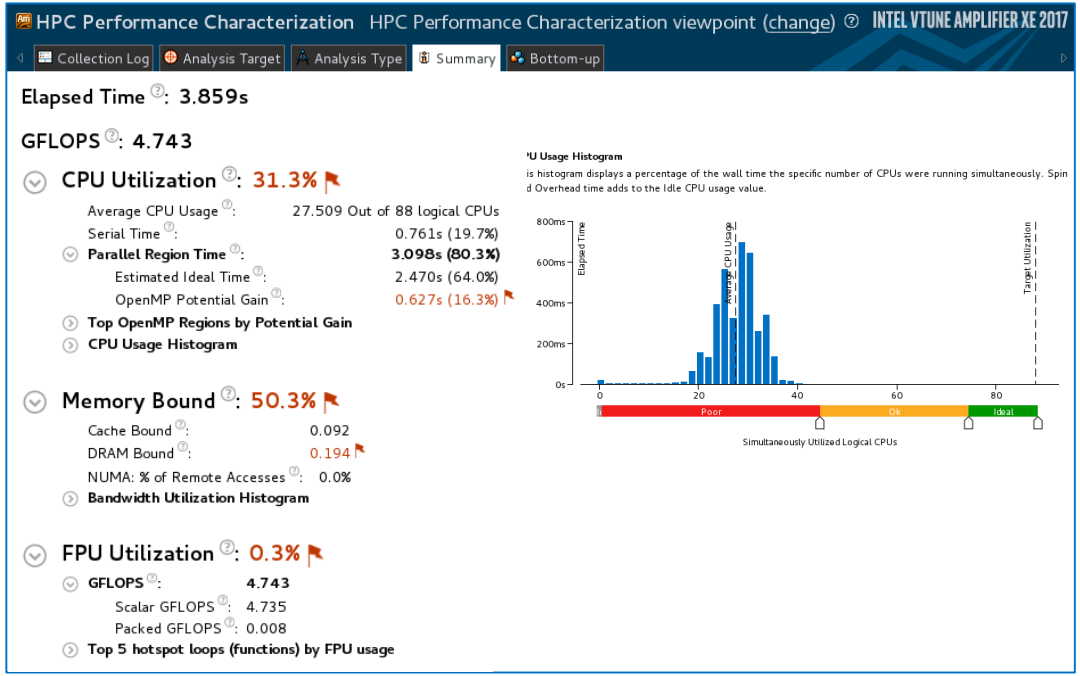
- Serial vs. Parallel time
- Top OpenMP regions by potential gain
- Tip: Use hotspot OpenMP region analysis for more detail

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

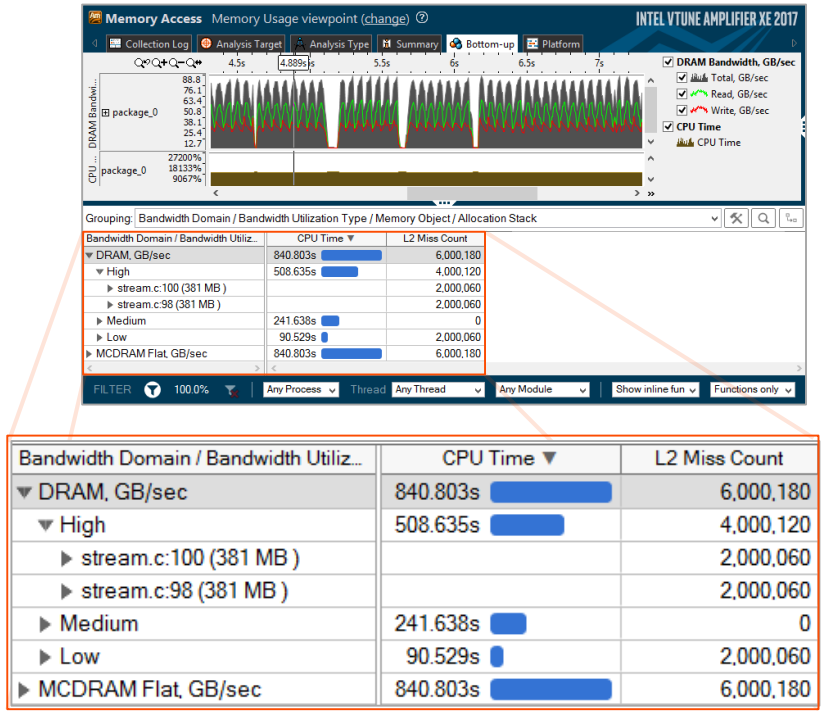
- Attribute cache misses to data structures (not just the code causing the miss)
- Support for custom memory allocators

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

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)

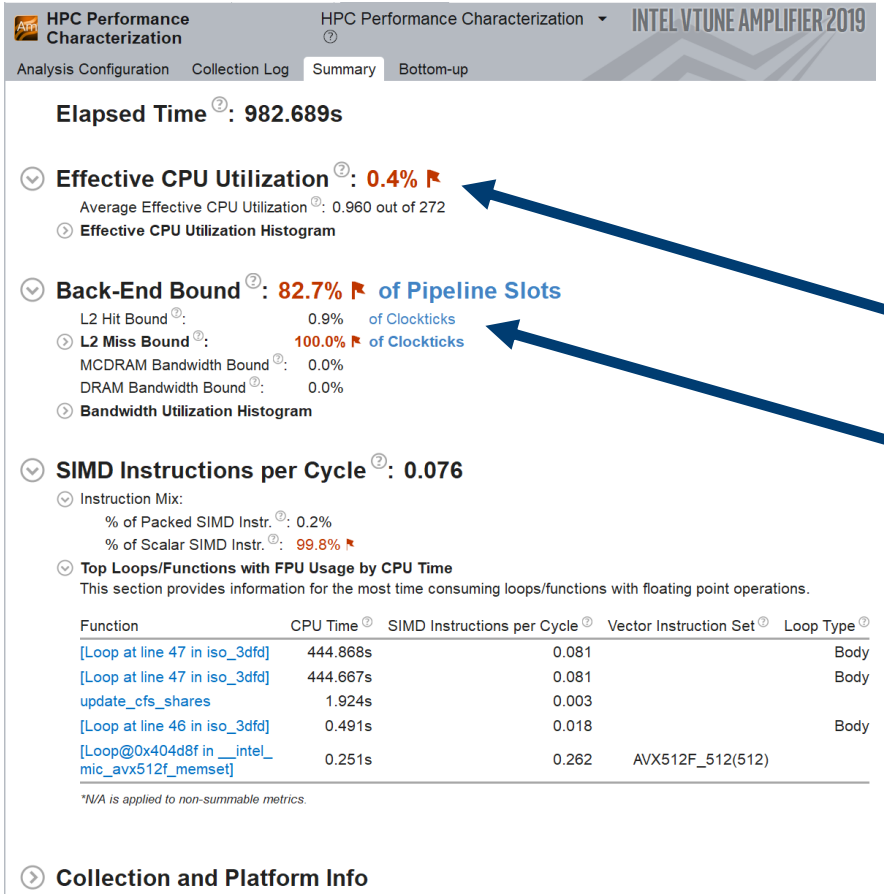
```
export LD_LIBRARY_PATH=/opt/intel/vtune_amplifier/lib64:$LD_LIBRARY_PATH
source /opt/intel/vtune_amplifier/amplxe-vars.sh
export PMI_NO_FORK=1
export OMP_NUM_THREADS=64; export OMP_PROC_BIND=spread; export OMP_PLACES=cores
```

Environment setup

Invoke VTune™ Amplifier

```
aprun -n 1 -N 1 -cc depth -d 256 -j 4 amplxe-cl -c hpc-performance -r ./vtune_hpc -- ./exe [args]
```

HPC Performance Baseline Results



Things look bad:

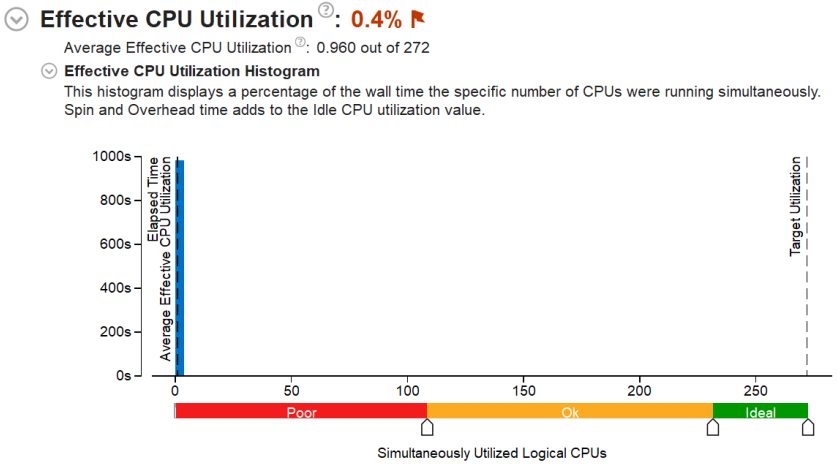
- Less than 1% CPU utilization
- Over 82% of pipeline slots stalled on memory access

No vectorization, but at this point this is a secondary factor

CPU Utilization Details

Note the default CPU utilization histogram has very aggressive targets

- Not all codes will take advantage of 4 threads / core
- We can reasonably expect good scalability to at least 1 thread / core
- Clearly we need threading to make better use of this system



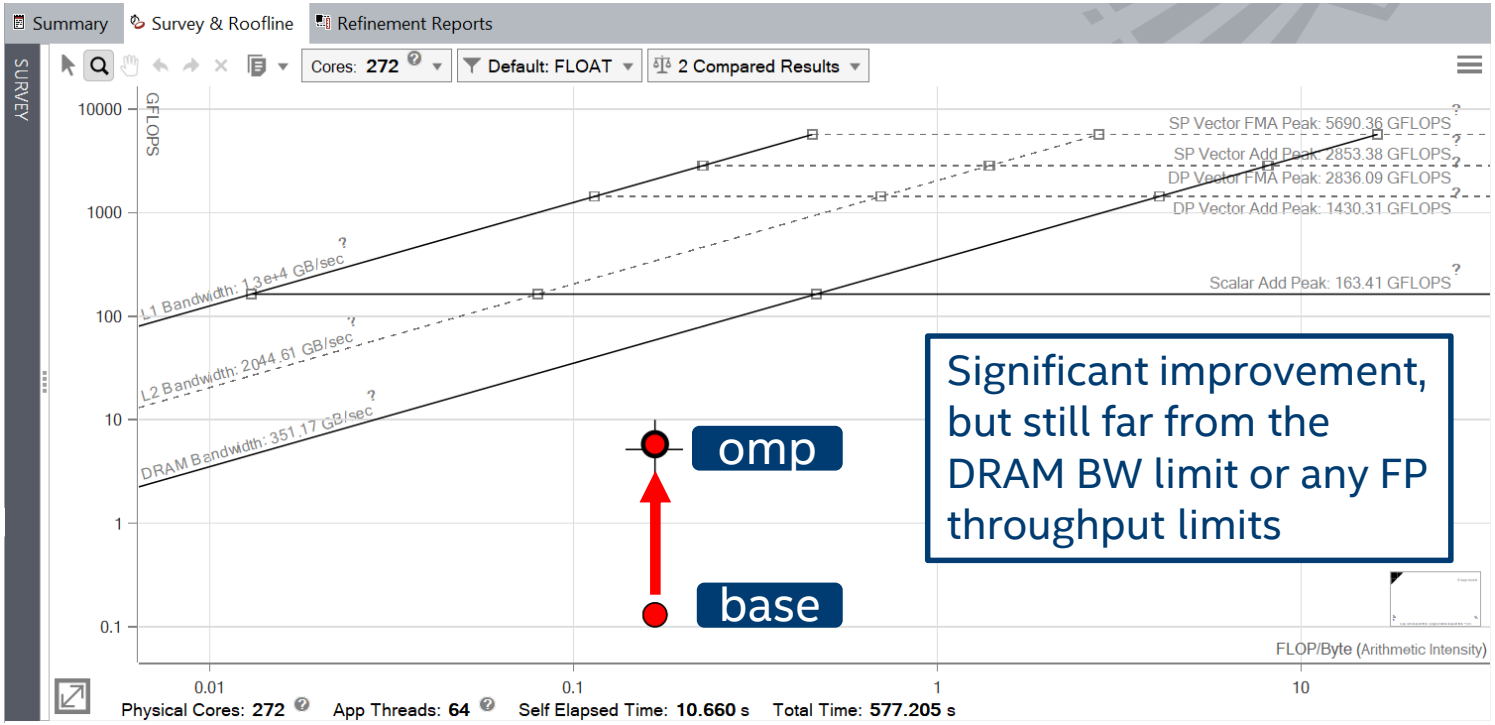
ADDING THREADING WITH OPENMP*

Parallelization with OpenMP*

```
#pragma omp parallel for
for( int ix=0; ix<nx; ix++ ) {
  for( int iy=0; iy<ny; iy++ ) {
    for( int iz=0; iz<nz; iz++ ) {
     int offset = iz*nx*ny + iy*nx + ix;
     float value = 0.0;
     value += ptr_prev[offset]*coeff[0];
     for( int ir=1; ir<=8; ir++ ) {
       value += coeff[ir] * (ptr_prev[offset + ir] + ptr_prev[offset - ir]);
       value += coeff[ir] * (ptr_prev[offset + ir*nx] + ptr_prev[offset - ir*nx]);
       value += coeff[ir] * (ptr_prev[offset + ir*nx*ny] + ptr_prev[offset - ir*nx*ny]);
     }
     ptr_next[offset] = 2.0f* ptr_prev[offset] - ptr_next[offset] + value*ptr_vel[offset];
    }}}
```

Adding OpenMP* in outer loop to have sufficient work per thread

Roofline Comparison



Elapsed Time: 22.546s

Effective CPU Utilization: 18.8%

Average Effective CPU Utilization: 51.229 out of 272

Serial Time (outside parallel regions): 1.517s (6.7%)

Parallel Region Time: 21.029s (93.3%)

Estimated Ideal Time: 17.777s (78.8%)

OpenMP Potential Gain: 3.252s (14.4%)

Top OpenMP Regions by Potential Gain

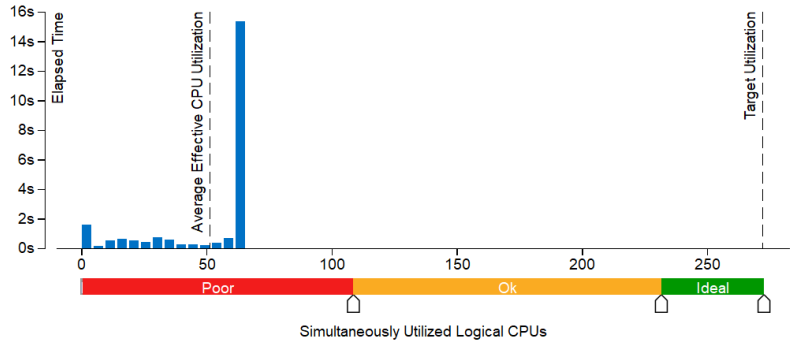
This section lists OpenMP regions with the highest potential for performance improvement. The Potential Gain metric shows the elapsed time that could be saved if the region was optimized to have no load imbalance assuming no runtime overhead.

OpenMP Region	OpenMP Potential Gain	(%)	OpenMP Region Time
iso_3dfdompparallel:64@unknown:45:60	3.252s	14.4%	21.029s

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

Effective CPU Utilization 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 utilization value.



New HPC Performance Results (CPU)

Large runtime improvement (~43x)

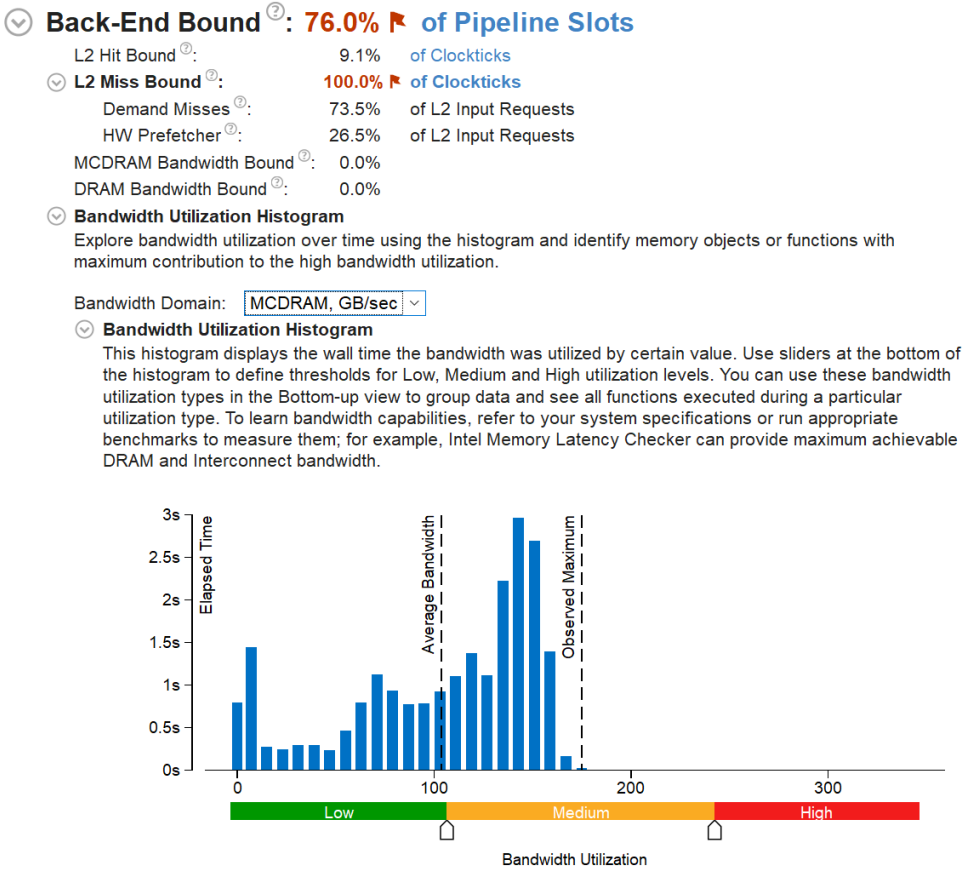
Threading is not perfect

- Some amount of serial time
- 14% imbalance in parallel region

CPU utilization is 51 / 272

- Really 51 / 64 (~80%) since we used only 64 OpenMP* threads
- More threads lead to additional memory conflicts and only minor performance improvements in this case (not shown).

New HPC Performance Results (Memory)



Over 75% of execution pipeline slots spent in data access stalls

Performance bound by misses on L2 cache access

- Forces expensive memory accesses
- Note breakdown between demand and prefetcher request origin

Technically not bandwidth bound, but clearly serious issues with data access.

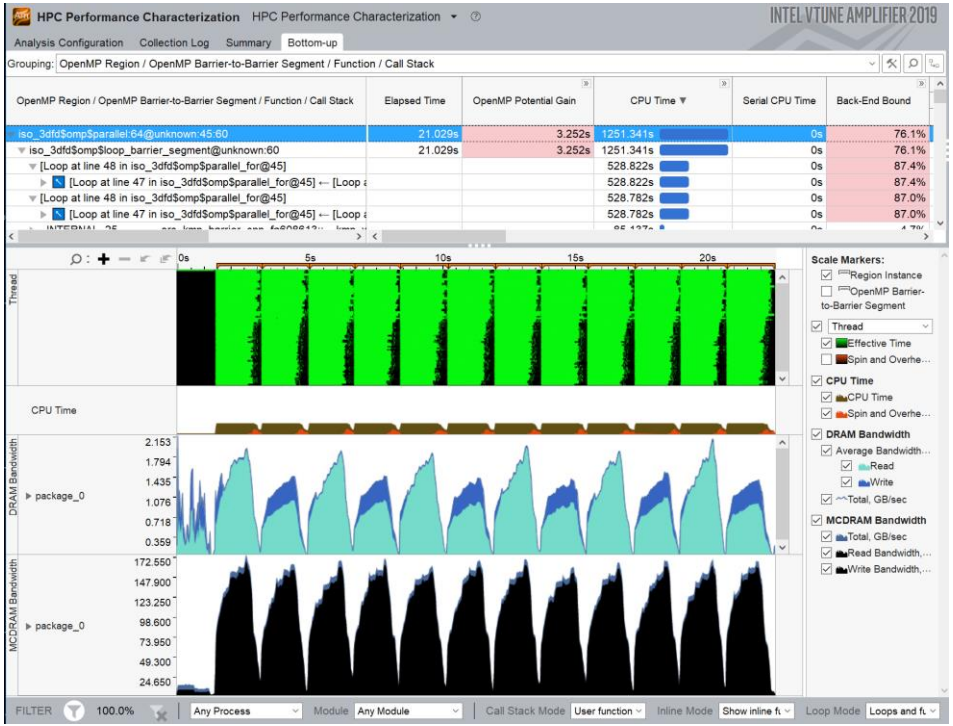
BW domain options are DDR and MCDRAM (most useful for flat mode configurations)

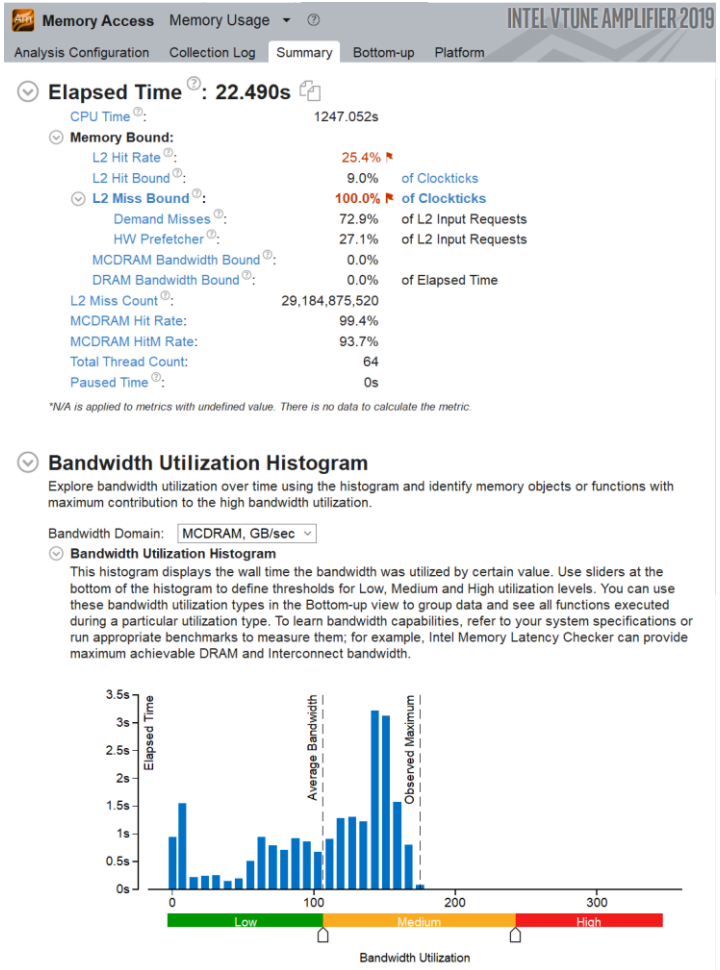
Bottom-Up view

Lots of information

- Read/write bandwidth timeline for DRAM and MCDRAM
- Thread activity timeline, including spin and overhead
- Double clicking on the top time consuming line opens the source code view

In multi-socket systems the UPI traffic timeline is also shown





The memory-access Analysis

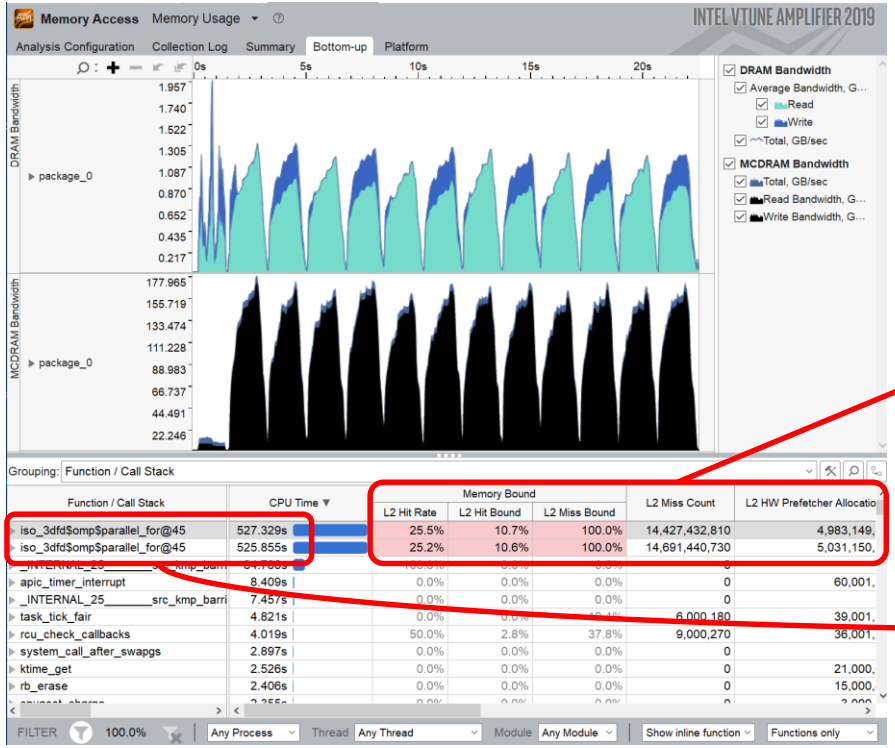
Summary completely focused on data access

- L2 hit rate and miss counts
- MCDRAM hit and miss rates
- Estimated % of time spent bound by L2 access, MCDRAM access, DDR access
- No CPU utilization data

Just invoke

```
[…] amplxe-cl -c memory-access […]
```

Bottom-Up Details Using *memory-access*



Similar view to hpc-performance but focused on memory bw utilization timelines

Specific counts now available for both demand and HW prefetcher initiated accesses

Double click the most expensive section to go to the detailed source view

Investigating Memory Access Issues

While you could look at the main loop VTune™ Amplifier has identified and probably figure out what is gong on in this case we have a way to pinpoint the cause of this issues directly.

- A Memory Access Pattern (MAP) analysis in Intel® Advisor will help here
- MAP is a refinement analysis, so you must have collected roofline data (survey and trip counts) prior to the map analysis, and use the same project directory:

```
[…] advixe-cl -c map --project-dir=./adv […]
```

- This kind of detailed analysis can take a significant amount of time. Try reducing the number of iterations to minimize the execution time

Memory Access Pattern Results

All of this are inefficient

Summary | Survey & Roofline | Refinement Reports

Site Location	Loop-Carried Dependencies	Strides Distribution	Access Pattern	Footprint Estimate	
				Max. Per-Instruction Addr. Range	First Instance Site Footprint
[loop in iso_3dfd at iso3dfd.cc:41]	No Information Available	100% / 0% / 0%	All Unit Strides	8B	708B
[loop in iso_3dfd at iso3dfd.cc:47]	No Information Available	100% / 0% / 0%	All Unit Strides	534MB	2GB
[loop in iso_3dfd at iso3dfd.cc:47]	No Information Available	100% / 0% / 0%	All Unit Strides	534MB	2GB
[loop in iso_3dfd at iso3dfd.cc:48]	No Information Available	60% / 40% / 0%	Mixed Strides	534MB	2GB
[loop in iso_3dfd at iso3dfd.cc:48]	No Information Available	60% / 40% / 0%	Mixed Strides	534MB	2GB
[loop in iso_3dfd at iso3dfd.cc:50]	No Information Available	45% / 55% / 0%	Mixed Strides	534MB	2GB
[loop in iso_3dfd at iso3dfd.cc:50]	No Information Available	45% / 55% / 0%	Mixed Strides	534MB	2GB

Memory Access Patterns Report | Dependencies Report

Wrong loop order or indexing??

ID		Stride	Type	Source			Max. Per-Instruction Addr. Range	Modules
P12		143520	Constant stride	iso3dfd.cc:		2010	534MB	iso3dfd; libiomp5.so
P16		143520	Constant stride	iso3dfd.cc:		2010	534MB	iso3dfd; libiomp5.so
P20		143520	Constant stride	iso3dfd.cc:		2010	534MB	iso3dfd; libiomp5.so
P24		143520	Constant stride	iso3dfd.cc:57		block 0x2b8be12b2010	534MB	iso3dfd; libiomp5.so
P28		143520	Constant stride	iso3dfd.cc:59		block 0x2b8bb4af6010, block 0x2b8c0da6e010	534MB	iso3dfd; libiomp5.so
P32		143520	Constant stride	iso3dfd.cc:59		block 0x2b8bb4af6010	534MB	iso3dfd; libiomp5.so

FIXING BASIC MEMORY ACCESS

Incorrect Loop Ordering ☹

```
#pragma omp parallel for
for( int iz=0; iz<nz; iz++ ) {
  for( int iy=0; iy<ny; iy++ ) {
    for( int ix=0; ix<nx; ix++ ) {
     int offset = iz*nx*ny + iy*nx + ix;
     float value = 0.0;
     value += ptr_prev[offset]*coeff[0];
     for( int ir=1; ir<=8; ir++ ) {
       value += coeff[ir] * (ptr_prev[offset + ir] + ptr_prev[offset - ir]);
       value += coeff[ir] * (ptr_prev[offset + ir*nx] + ptr_prev[offset - ir*nx]);
       value += coeff[ir] * (ptr_prev[offset + ir*nx*ny] + ptr_prev[offset - ir*nx*ny]);
     }
     ptr_next[offset] = 2.0f* ptr_prev[offset] - ptr_next[offset] + value*ptr_vel[offset];
    }}}
```

Loop exchange needed to minimize jumping around in the memory space.

Order shown here is correct.

Updated *hpc-performance* Analysis

Elapsed Time: 4.059s

Effective CPU Utilization: 14.1%
- Average Effective CPU Utilization: 38.439 out of 272
- **Serial Time (outside parallel regions)**: 1.517s (37.4%)
  - **Top Serial Hotspots (outside parallel regions)**
- **Parallel Region Time**: 2.542s (62.6%)
- **Effective CPU Utilization Histogram**

Back-End Bound: 43.9% of Pipeline Slots

L2 Hit Bound:	21.1%	of Clockticks
L2 Miss Bound:	0.4%	of Clockticks
MCDRAM Bandwidth Bound:	0.0%	
DRAM Bandwidth Bound:	0.0%	

- **Bandwidth Utilization Histogram**

SIMD Instructions per Cycle: 0.457
- Instruction Mix:
  - % of Packed SIMD Instr.: 0.2%
  - % of Scalar SIMD Instr.: 99.8%
- **Top Loops/Functions with FPU Usage by CPU Time**

This section provides information for the most time consuming loops/functions with floating point operations.

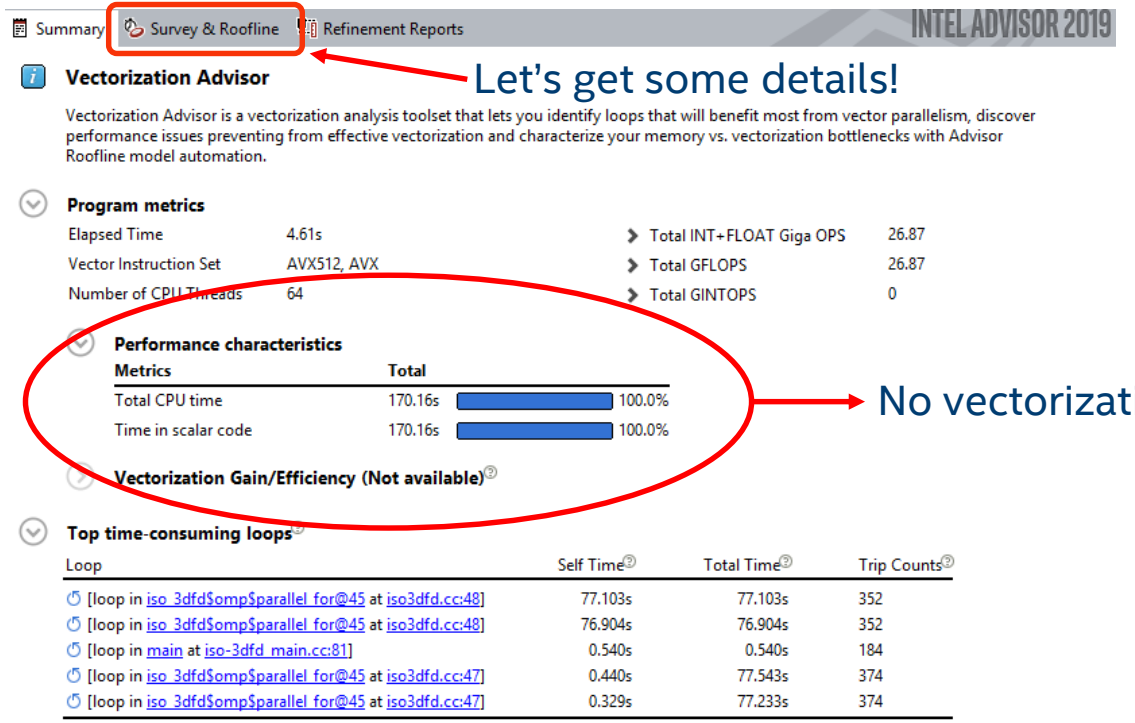
Function	CPU Time	SIMD Instructions per Cycle	Vector Instruction Set	Loop Type
[Loop at line 48 in iso_3dfdompparallel_for@45]	71.285s	0.535		Body
[Loop at line 48 in iso_3dfdompparallel_for@45]	71.014s	0.522		Body
[Loop at line 47 in iso_3dfdompparallel_for@45]	0.381s	0.013		Body
[Loop at line 47 in iso_3dfdompparallel_for@45]	0.321s	0.011		Body
clear_page_c_e	0.301s	0.408		
[Others]	0.200s	0.138	AVX512F_512(512)	

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

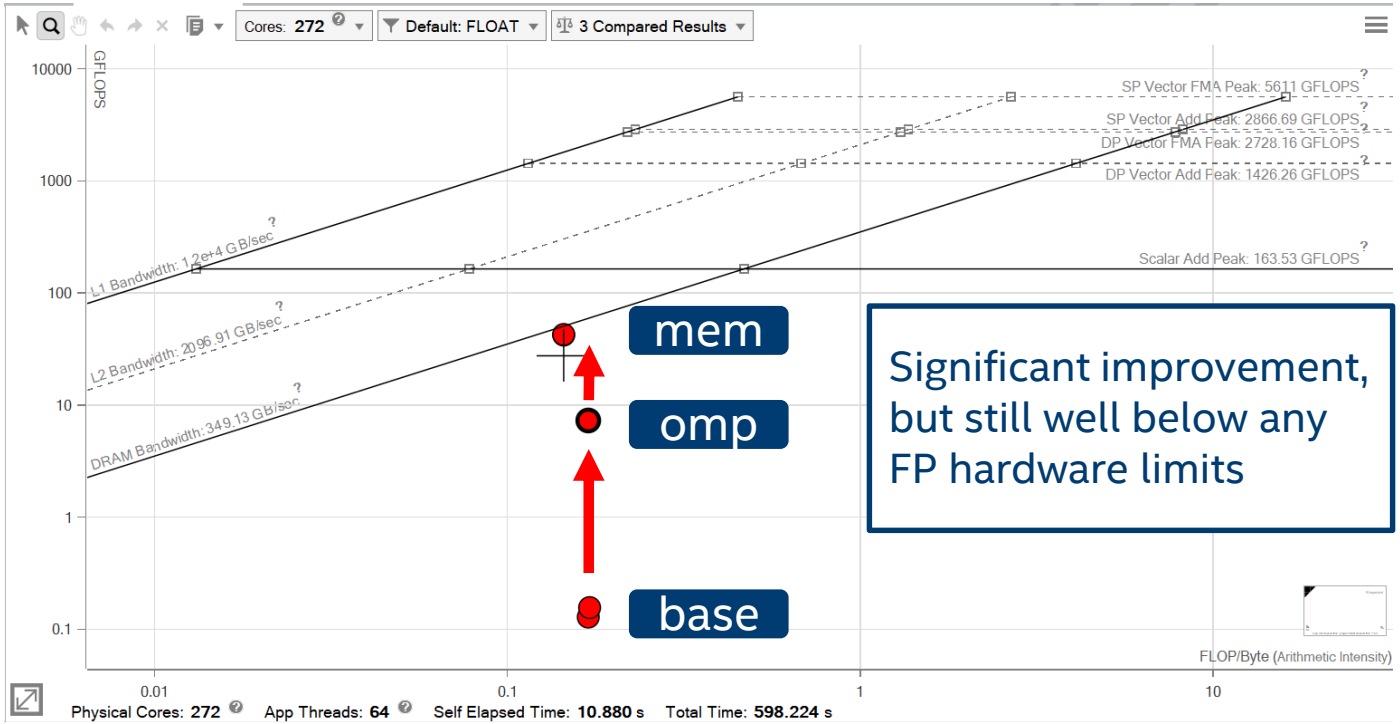
Significant speedup once again (~6x)

- No longer bound by L2 misses
- Serial time starts to become a concern (37.4%)
- Bottleneck now moves to CPU performance again
- Looks like main loop is not vectorized at all

Investigate Further - Roofline

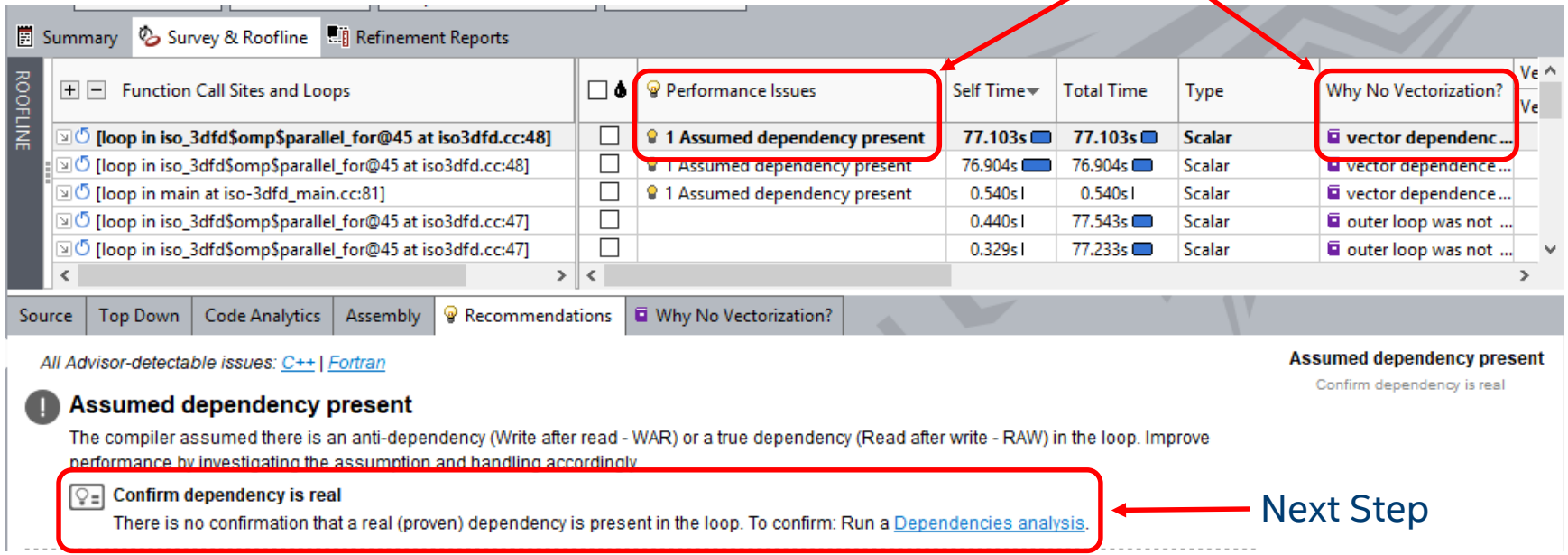


Roofline Changes



Survey Information

Problem is clearly described



Next Step

dependencies Analysis

This kind of detailed analysis can take a significant amount of time.

Reduce the number of iterations and problem size to minimize execution time.

Once completed a new section will appear in the Summary tab

dependencies is a refinement analysis, so you must have collected roofline data (survey and trip counts) prior to the map analysis, and use the same project directory:

```
[…] advixe-cl -c dependencies --project-dir=./adv […]
```

Refinement analysis data

These loops were analyzed for memory access patterns and dependencies:

Site Location	Dependencies	Strides Distribution
[loop in main at iso-3dfd_main.cc:58]	No information available	0% / 100% / 0%
[loop in main at iso-3dfd_main.cc:59]	No information available	0% / 100% / 0%
[loop in main at iso-3dfd_main.cc:60]	No information available	0% / 100% / 0%
[loop in iso_3dfd at iso3dfd.cc:41]	No information available	No strides found
[loop in iso_3dfd at iso3dfd.cc:47]	No information available	65% / 35% / 0%
[loop in iso_3dfd at iso3dfd.cc:47]	No information available	65% / 35% / 0%
[loop in iso_3dfd at iso3dfd.cc:48]	No dependencies found	63% / 38% / 0%
[loop in iso_3dfd at iso3dfd.cc:48]	No dependencies found	63% / 38% / 0%
[loop in iso_3dfd at iso3dfd.cc:54]	RAW:1	100% / 0% / 0%
[loop in iso_3dfd at iso3dfd.cc:54]	RAW:1	100% / 0% / 0%

Problem!

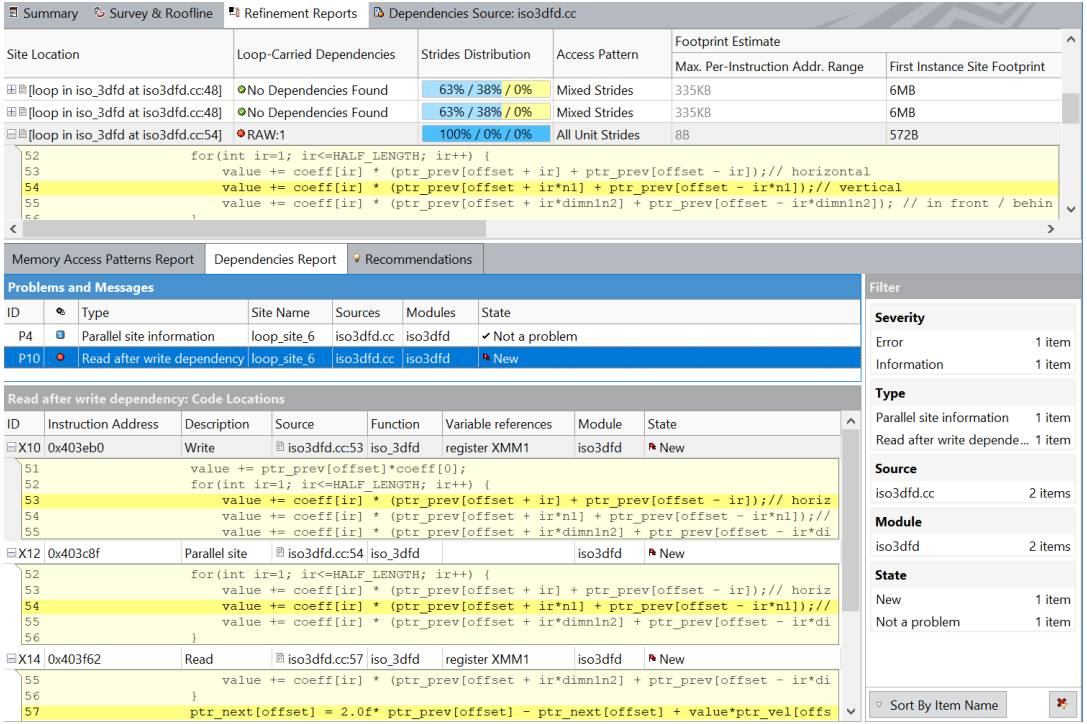
dependencies Analysis Details

Lots of details on report

- Issue is at the core of the compute loop
- Compiler was right - this is a true dependency
- Type of issue is Read-After-Write (RAW)
- Read and write locations shown

Now that the issue is clear we can fix it simply by using an array variable for “value”

Let’s do this and see if the section can be vectorized



FIXING VECTORIZATION

Enabling Vectorization

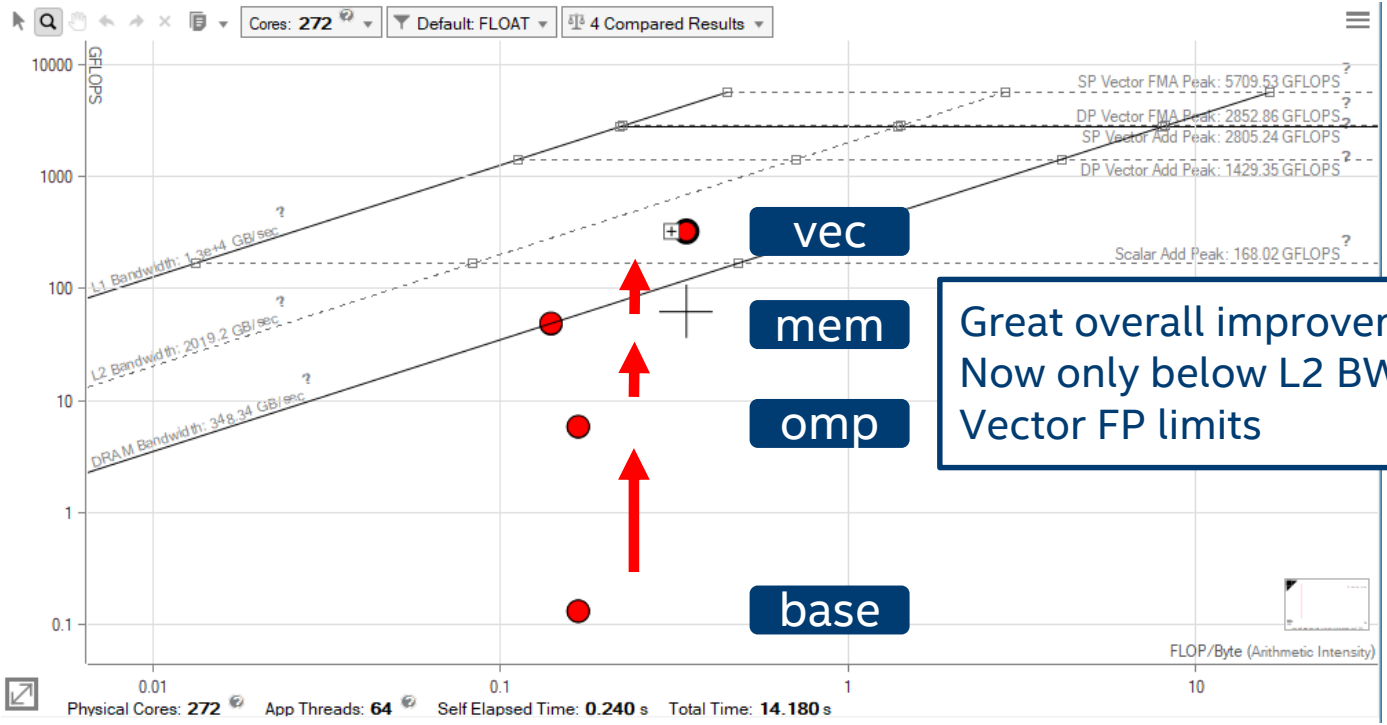
```
#pragma omp parallel for
for( int iz=0; iz<nz; iz++ ) {
  for( int iy=0; iy<ny; iy++ ) {
    #pragma omp simd
    for( int ix=0; ix<nx; ix++ ) {
     int offset = iz*nx*ny + iy*nx + ix;
     value[ix] += ptr_prev[offset]*coeff[0];
    #pragma unroll(8)
     for( int ir=1; ir<=8; ir++ ) {
       value[ix] += coeff[ir] * (ptr_prev[offset + ir] + ptr_prev[offset - ir]);
       ...
```

Innermost loop is short and may be fully unrolled

Scalar *value* can be turned into an array of the same length of the loop we target for vectorization

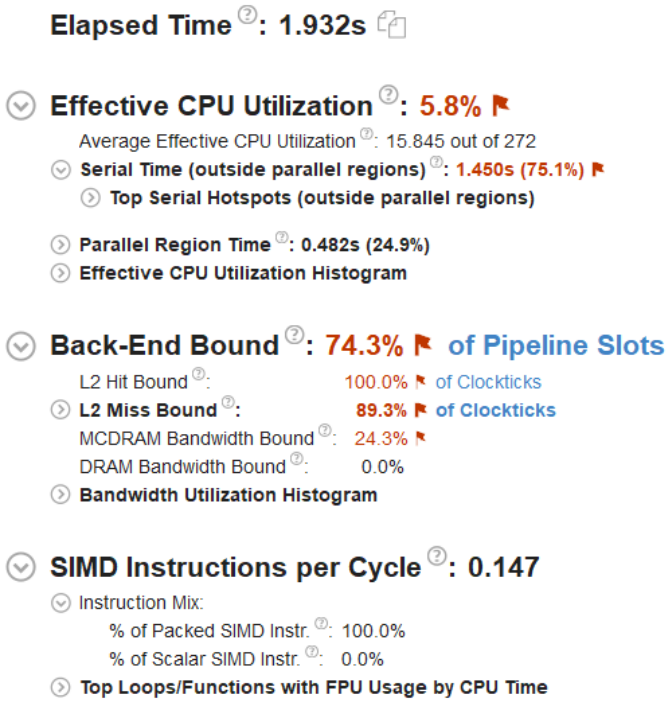
Loop in x can now be vectorized

Roofline Changes



Great overall improvement!
Now only below L2 BW and Vector FP limits

What Next?



Since we started this tuning session the application performance has been improved by 508x

Speedup after vectorization is 2x

But I have 512bits = 16 SP FP ops, what is going on?

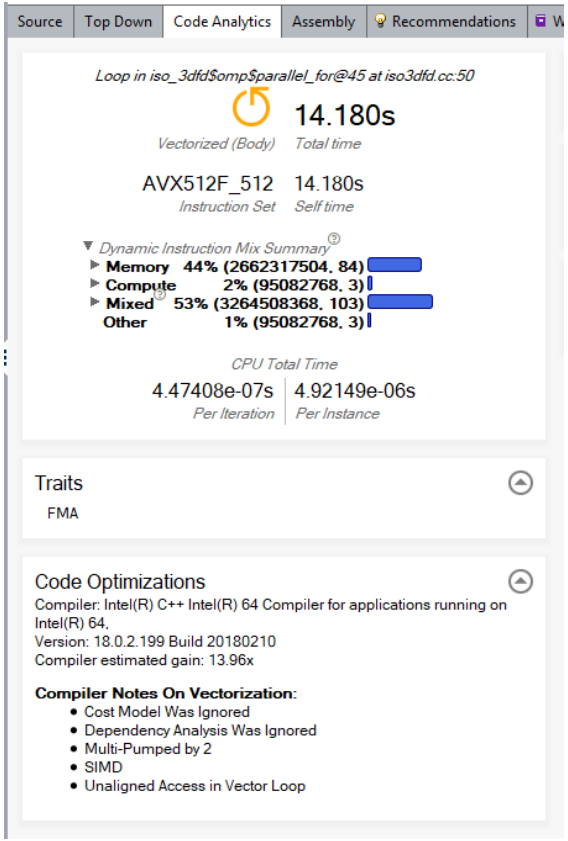
Serial time dominates execution, and overall characteristics have changed

Further Optimization Opportunities

At this point we can continue to use the same techniques to optimize the code, but we should increase the workload size or skip the startup serial section.

The “Code Analytics” tab in Intel® Advisor is useful to investigate vectorization efficiency in detail

In this case an estimated 14/16 efficiency is achieved, so there is room for improvement - note the “Unaligned access in vector Loop”



FOCUS ON PYTHON

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
```

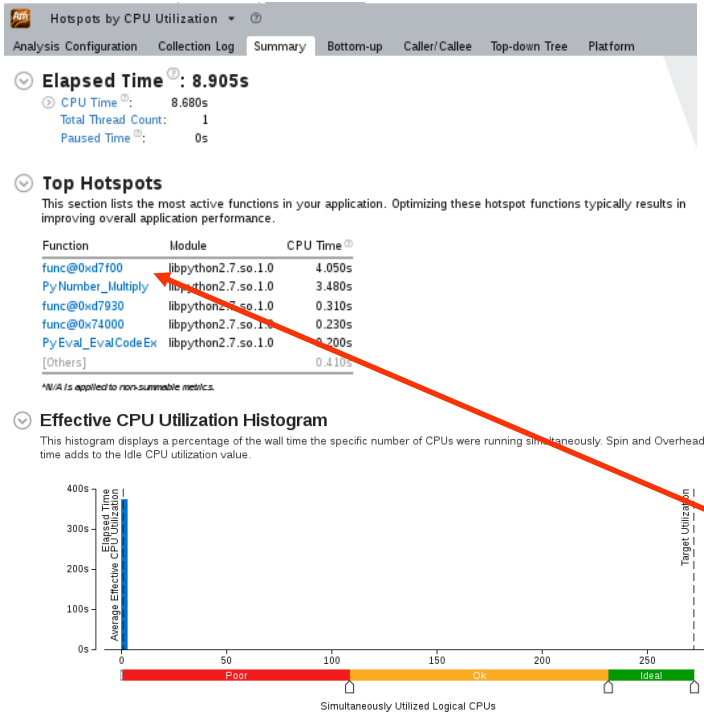
Covariance Matrix

$$cov = \begin{bmatrix} \sum \frac{x_1{}^2}{N} & \sum \frac{x_1 x_2}{N} & \cdots & \sum \frac{x_1 x_n}{N} \\ \sum \frac{x_2 x_1}{N} & \sum \frac{x_2{}^2}{N} & \cdots & \sum \frac{x_2 x_n}{N} \\ \cdots & \cdots & \cdots & \cdots \\ \sum \frac{x_n x_1}{N} & \sum \frac{x_n x_2}{N} & \cdots & \sum \frac{x_n{}^2}{N} \end{bmatrix}$$

The covariance matrix represents the mathematical generalization of variance to multiple dimensions

- Feature variances along the diagonal and element-wise covariance along the off-diagonal
- Typically each element is normalized by the number of examples in the dataset

Covariance Matrix Example



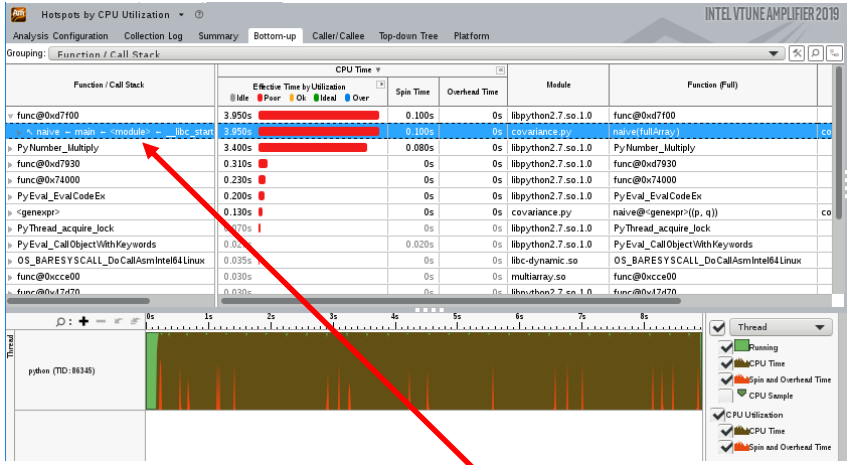
Naïve implementation of the calculation of a covariance matrix

Summary shows:

- Single thread execution
- Top five time consuming functions

Click on top function to go to Bottom-up view

Bottom-up View and Source Code

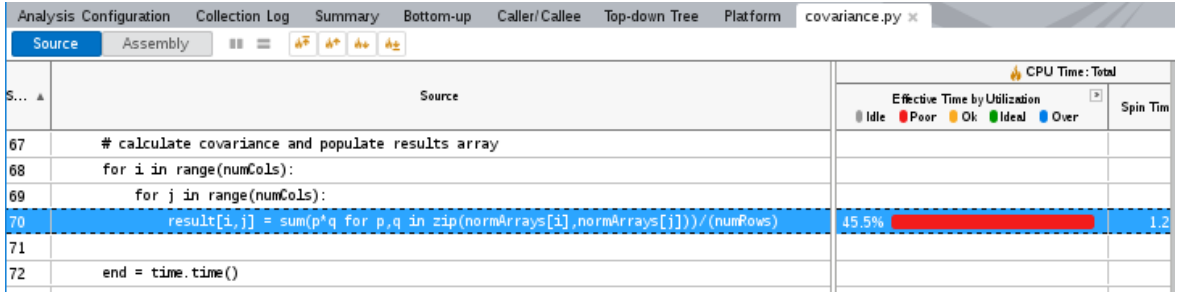


In Bottom-Up we can see the most time consuming sections of the code listed, as well as the CPU utilization graph.

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

Double click to see relevant code

Bottom-up View and Source Code



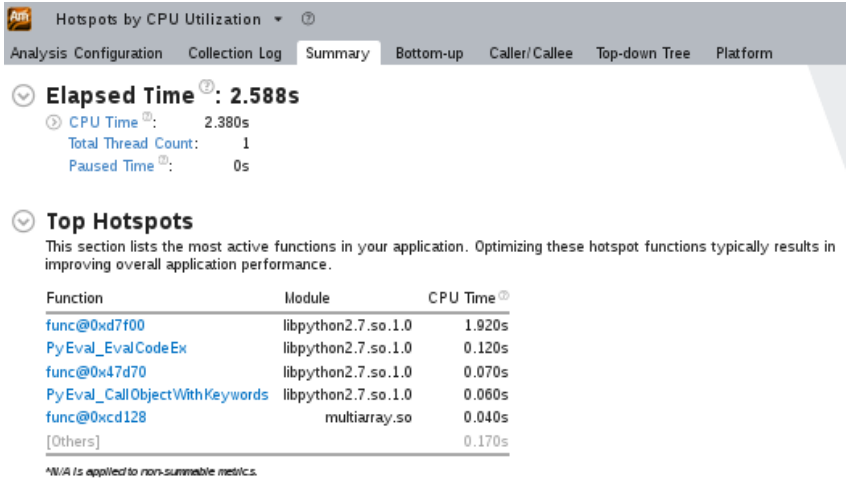
Inefficient array operation found quickly.

We could use numpy to improve on this.

sum(p*q for p,q in zip(normArrays[i],normArrays[j]))/(numRows)

sum(np.multiply(normArrays[i],normArrays[j]))/(numRows)

New Implementation



We gain about 3.4x speedup with this change

But we are still running sequentially so we should next attempt to parallelize the code

But at this point you get the idea…

Intel® Advisor and Vtune™ Amplifier's APS will also work with Python, but possibly miss some of the information available for traditional C/Fortran codes.

OTHER THINGS TO KNOW

Useful Options on Theta

If finalization is slow you can use -finalization-mode=deferred and simply finalize on a login node or a different 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
- https://software.intel.com/en-us/vtune-amplifier-help-analyzing-statically-linked-binaries-on-linux-targets

Legal Disclaimer & Optimization Notice <

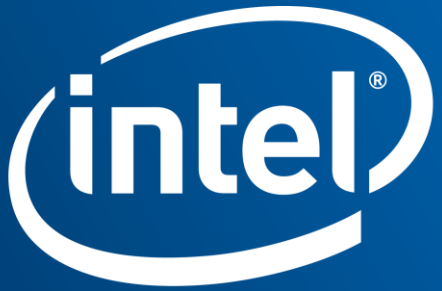
Performance results are based on testing as of 02/10/2019 and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure.

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.

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.

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