



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)
 - Cache use needed to feed vector units
- Socket
 - Using all cores in a processor requires parallelization (MPI, OMP, ...)
 - Using coherent, shared socket caches
- Node
 - Minimize remote memory access (control memory affinity)
 - Minimize resource sharing (tune local memory access, disk IO and network traffic)

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



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
```

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



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

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Usage on Theta

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

Load the APS module:

```
$ module swap intel/18.0.0.128 intel/19.0.3.199
```

Launch your job in interactive or batch mode:

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

Produce text and html reports:

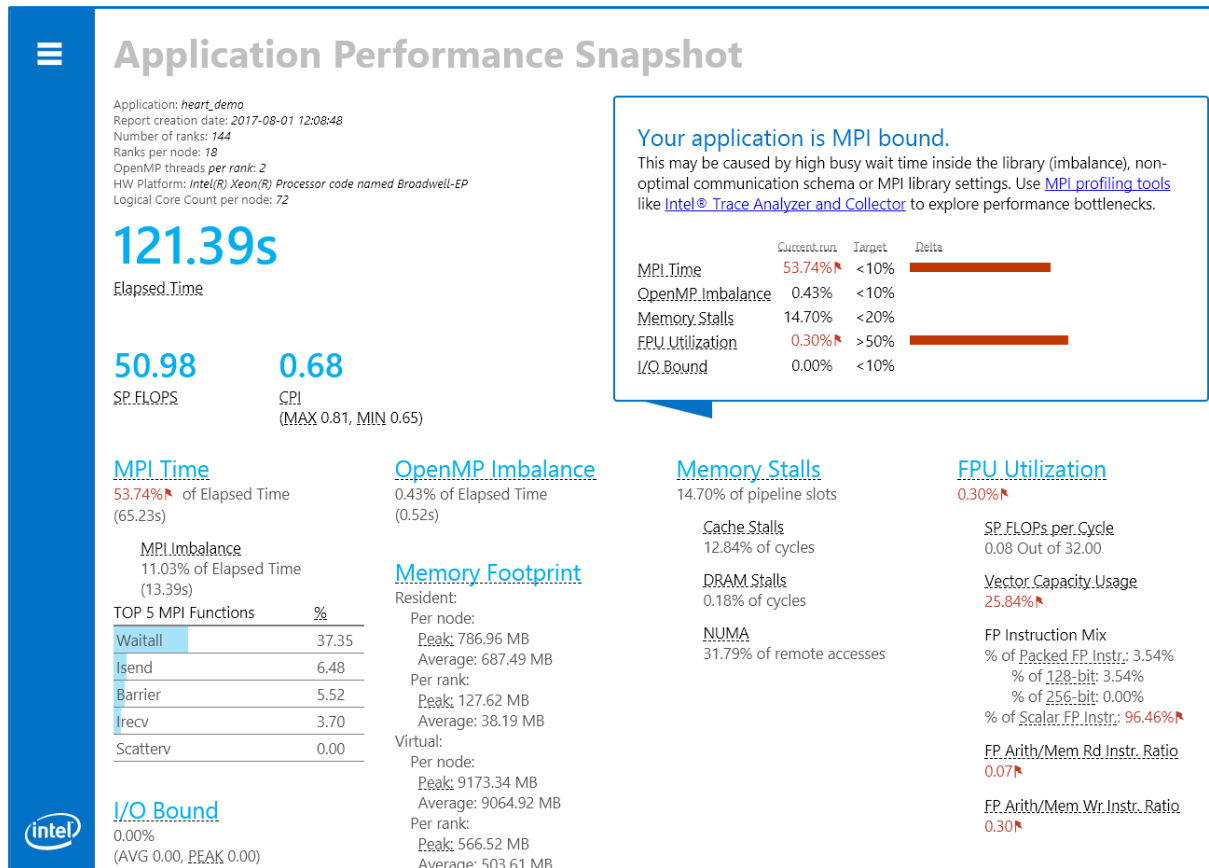
```
$ aps -report=./aps_result_ ... ./
```

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



APS HTML Report

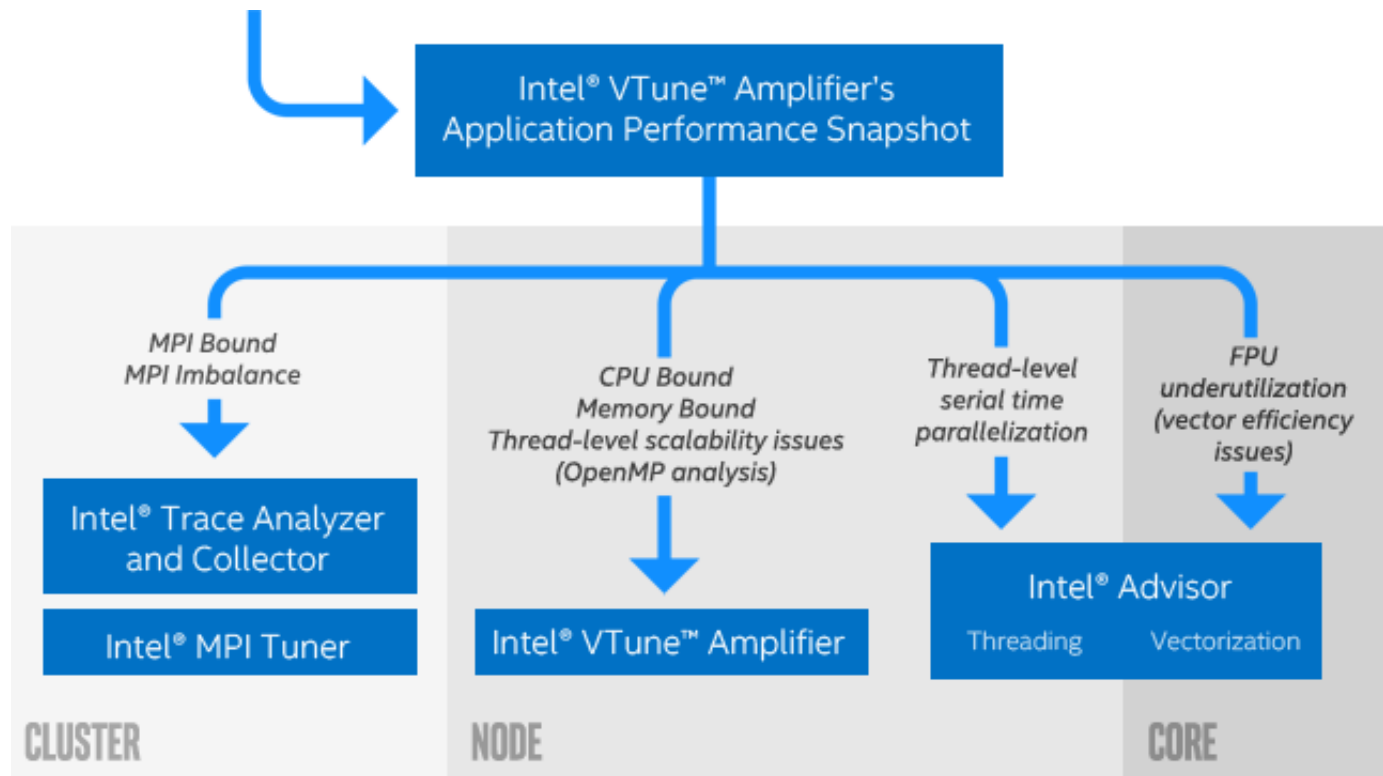


Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
 *Other names and brands may be claimed as the property of others.



Tuning Workflow



Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Intel[®] Advisor

Vectorization and Static Analysis

<https://www.alcf.anl.gov/user-guides/advisor-xc40>

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



Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



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 (roofline)
 - 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

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

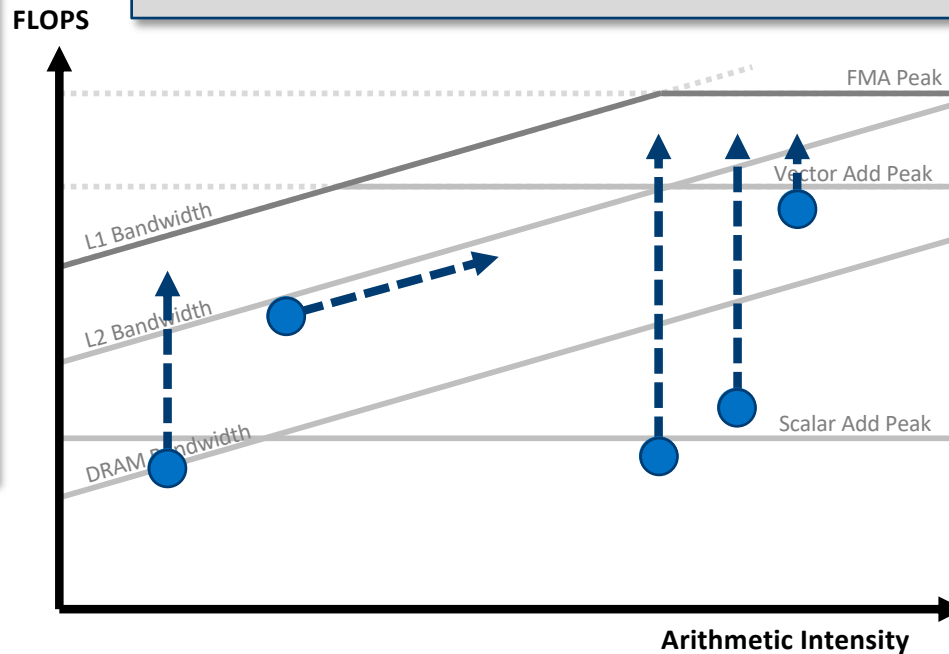


Cache-Aware Roofline

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.

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



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
- Compile with **-g** and **-dynamic**
- Set your environment:

```
$ module swap intel/18.0.0.128 intel/19.0.3.199
```

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Use -h Option!

advixe-cl -h collect

Examples:

1) Survey the application to determine hotspots.

```
advixe-cl --collect survey --project-dir ./advi --search-dir src:r=./src  
-- ./bin/myApplication
```

2) Collect memory access patterns data with specified loops for analysis.

```
advixe-cl --collect map --mark-up-list=5,10,12 --project-dir ./advi  
--search-dir src:r=./src -- ./bin/myApplication
```

3) Collect survey data on 4 nodes of MPI cluster into the shared ./advi project directory.

```
mpirun -n 4 advixe-cl --project-dir ./advi --collect survey  
-- <PATH>/mpi-sample/1_mpi_sample_serial
```

4) Collect dependencies data for all loops that are both innermost and hold above 2% of the total CPU time.

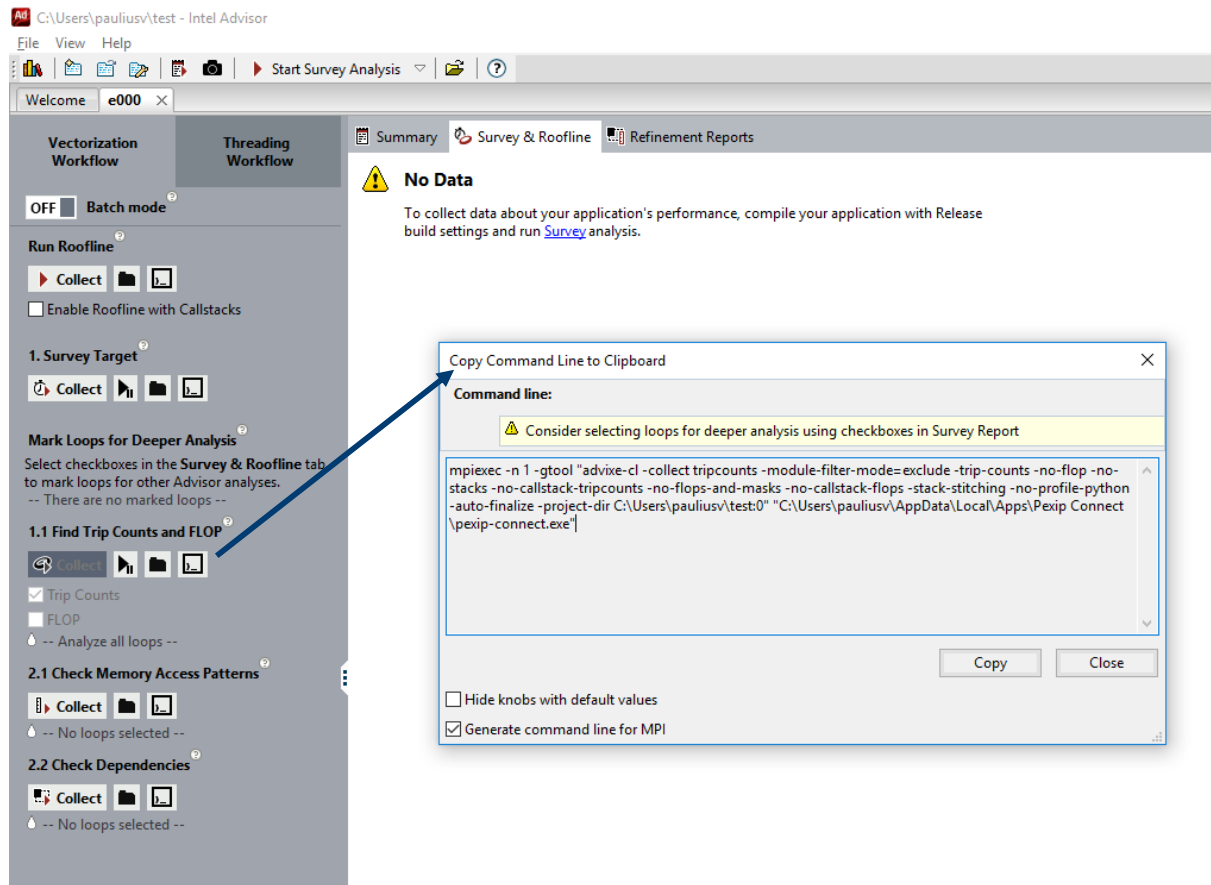
```
advixe-cl --collect dependencies --project-dir ./advi --loops="loop-height=0,total-time>2"  
-- ./bin/myApplication
```

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Using Intel® Advisor on Theta



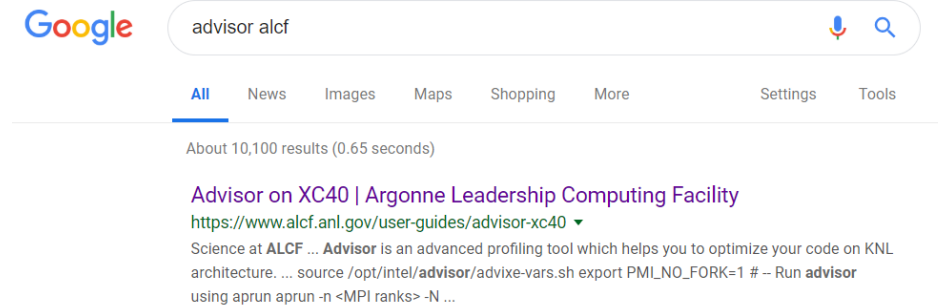
Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



advixe.qsub Script

- Copy and customize the script from `/soft/perftools/intel/advisor/advixe.qsub`
- All-in-one script for profiling
 - Job size - ranks, threads, hyperthreads, affinity
 - Attach to a single, multiple or all ranks
 - Binary as `arg#1`, input as `arg#2`
 - `qsub advixe.qsub ./your_exe ./inputs/inp`
 - Binary and source search directory locations
 - Timestamp + binary name + input name as result directory
 - Save cobalt job files to result directory



Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Advisor Collections

Every advisor study depends on results collected from “survey”

If you try to run tripcounts/map/dependencies without having completed survey the collection *will fail*

- Either
 - Collect survey and any additional analyses in one qsub submission
 - Collect survey, replace `#{RESDIR}` with generated directory name, qsub additional analyses
 - Write your own script

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Nbody demonstration

The naïve code that could

<https://github.com/pvelesko/nbody-demo>

Nbody gravity simulation

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

git clone <https://github.com/pvelesko/nbody-demo.git> ./ ; cd ./nbody-demo/ver2; make

```
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] += ...
```

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Collect Roofline Data

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

```
cp /soft/perftools/intel/advisor/advixe.qsub ./
<modify advixe.qsub as needed>
qsub ./advixe.qsub ./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.

You will have to **finalize manually**:

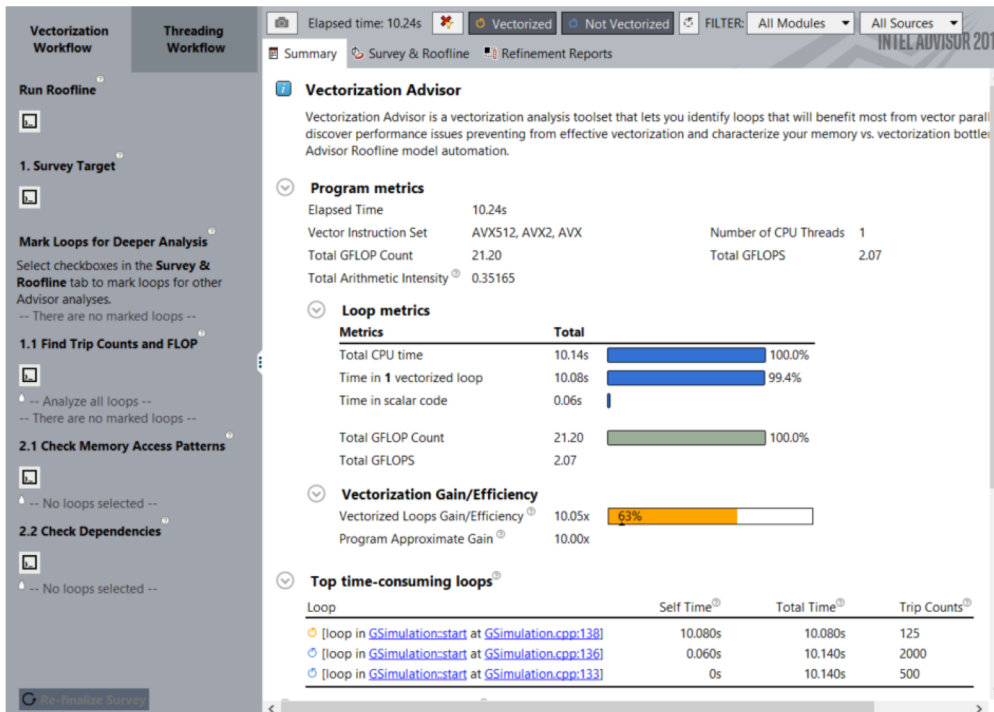
```
advixe-cl -report survey --refinalize-survey --project_dir ./result_dir
```

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



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)

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Survey Report (Source)

Elapsed time: 10.24s | Vectorized | Not Vectorized | FILTER: All Modules | All Sources | Loops And Functions | All Threads | OFF | Smart Mode

Summary | Survey & Roofline | Refinement Reports

Function Call Sites and Loops	Performance Issues	Self Time	Total Time	Type	Why No Vectorization?	Vectorized Loops	FLOPS
						Vector... Efficiency Gain E... VL (Ve... Self GFLOPS	
[loop in GSimulation::start at GSimulation.cpp:138]	2 Inefficient gather/sc...	10.080s	10.080s	Vectorized (Body)		AVX5... 63% 10.05x 16	2.093
[loop in GSimulation::start at GSimulation.cpp:136]	1 Opportunity for outer l...	0.060s	10.140s	Scalar	inner loop was already v...		1.700
f_start		0.000s	10.140s	Function			
f_main		0.000s	10.140s	Function			
GSimulation::start		0.000s	10.140s	Function			
[loop in GSimulation::start at GSimulation.cpp:133]	1 Data type conversions	0.000s	10.140s	Scalar	inner loop was already v...		

Source | Top Down | Code Analytics | Assembly | Recommendations | Why No Vectorization?

File: cache_e770d097069d33cb5b0f6e401a4f9d97_GSimulation.cpp:138 GSimulation::start

Line	Source	Total Time	%	Loop/function Time	%	Traits
132	const double t0 = time.start();					
133	for (int a=1; a<=get_nsteps(); ++a)					
134	{					
135	t0 += time.start();					
136	for (i = 0; i < n; i++) // update acceleration					
137	{					
138	for (j = 0; j < n; j++)	0.100s		10.080s		
	[loop in GSimulation::start at GSimulation.cpp:138]					
	Vectorized AVX512ER_512; AVX512F_512 loop processes Float32; Int32; UInt32 data type(s) and includes 2-Source Perm					
	No loop transformations applied					
139	{					
140	real_type dx, dy, dz;					
141	real_type distanceBqr = 0.0f;					
142	real_type distanceInv = 0.0f;					
143						

Selected (Total Time): 0.100s

Inline information regarding loop characteristics

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

Optimization Notice

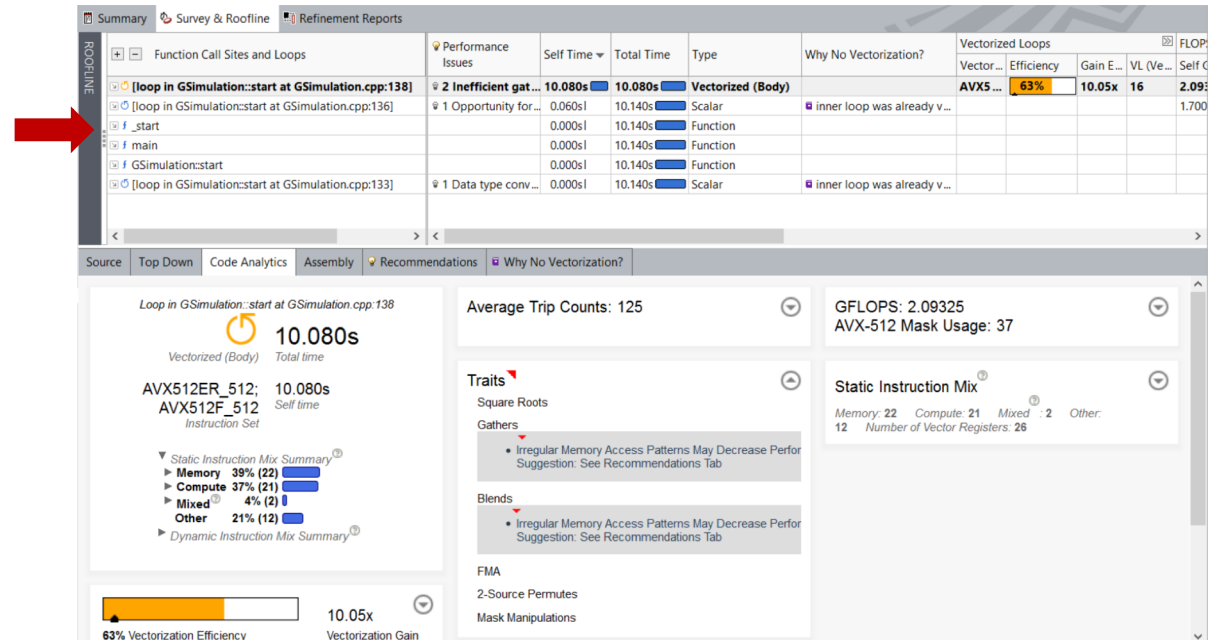
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



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

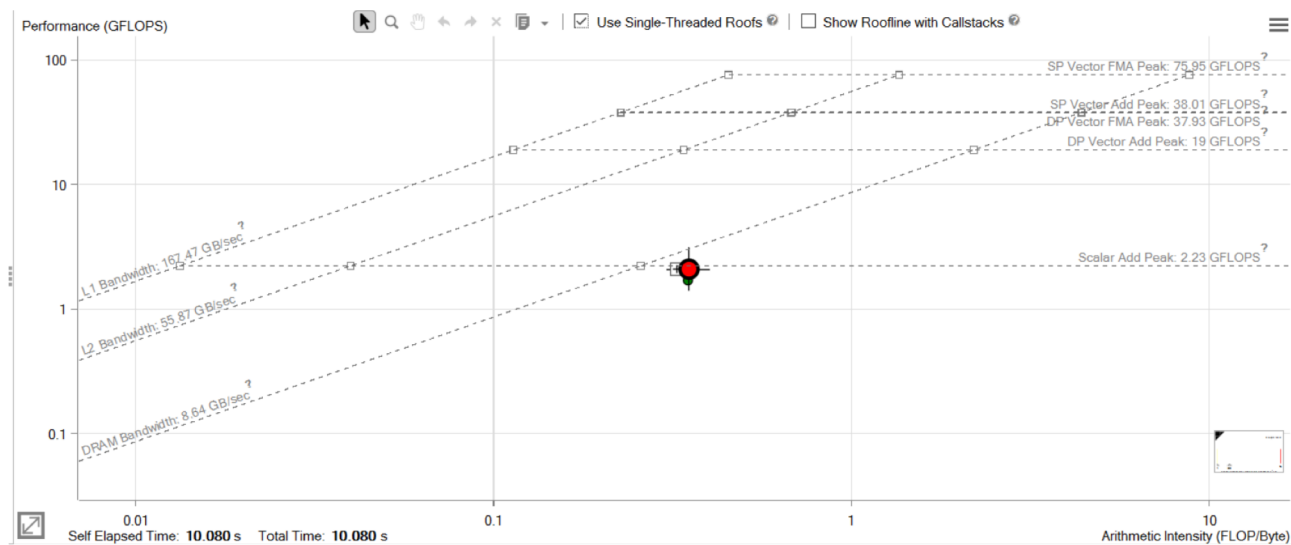


Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



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

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Memory Access Pattern Analysis (Refinement)

Modify `advixe.qsub` to collect "survey" followed by "map"

```
aprun -n <...> ./profile1.sh "advixe-cl -c map <...>"
```

ID	Stride	Type	Source	Nested Function	Variable references	Max. Site Footprint	Modules	Site Name	Access Type
P1	10; 40	Constant stride	GSimulation.cpp:144		block 0x60a0b0 allocated at GSimulation.cpp:109	4KB	nbody.x	loop_site_1	Read
P2		Gather stride	GSimulation.cpp:144		block 0x60a0b0 allocated at GSimulation.cpp:109	5KB	nbody.x	loop_site_1	Read
P3		Parallel site information	GSimulation.cpp:144				nbody.x	loop_site_1	
P5	0	Uniform stride	GSimulation.cpp:149			4B	nbody.x	loop_site_1	Read

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

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

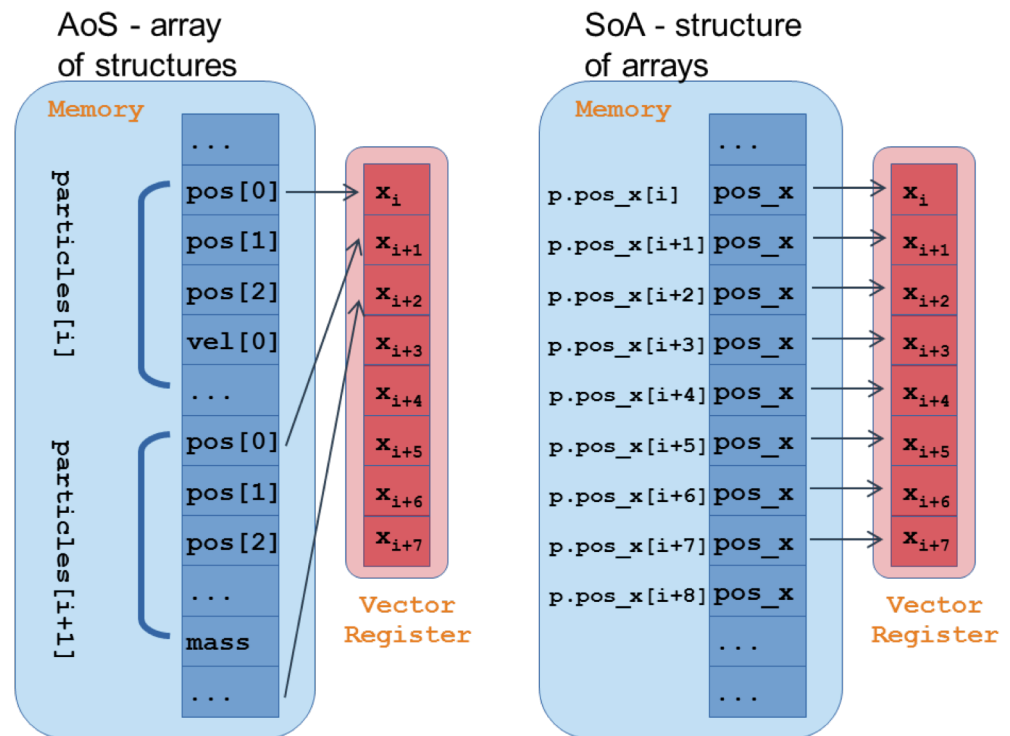


Vectorization: gather/scatter operation

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;
};
```



Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



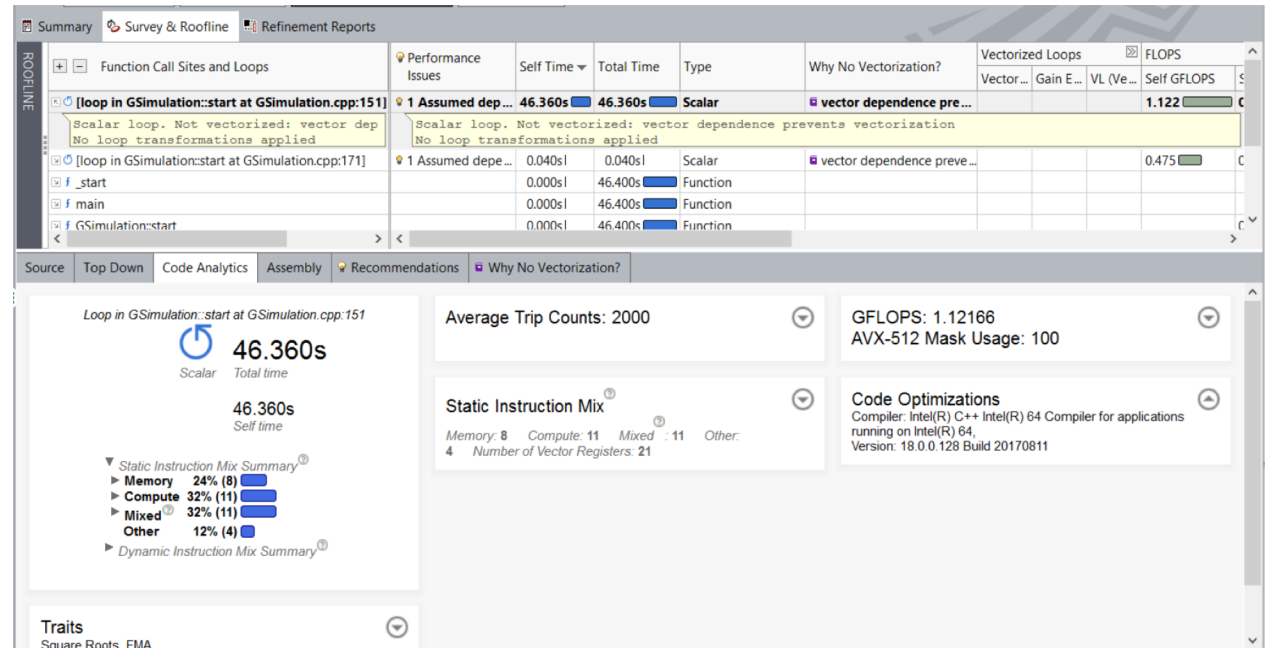
Performance After Data Structure Change

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

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Dependencies Analysis (Refinement)

Modify `advixe.qsub` to collect "survey" followed by "dependencies"

```
aprun -n <...> ./profile1.sh "advixe-cl -c dependencies <...>
```

```
qsub advixe.qsub ./ver3/nbody.x
```

The screenshot displays the Intel Advisor ZUI interface. At the top, the 'Dependencies Source: GSimulation.cpp' is shown. Below this, a table lists various metrics for a loop in `GSimulation.cpp:157`, including 'Loop-Carried Dependencies', 'Strides Distribution', 'Access Pattern', 'Max. Site Footprint', 'Site Name', and 'Recommendations'. A 'RAW:4' dependency is noted.

The 'Problems and Messages' section contains a table with the following data:

ID	Type	Site Name	Sources	Modules	State
P1	Parallel site information	loop_site_1	GSimulation.cpp	nbody.x	✓ Not a problem
P3	Read after write dependency	loop_site_1	GSimulation.cpp	nbody.x	🔴 New
P4	Read after write dependency	loop_site_1	GSimulation.cpp; main.cpp	nbody.x	🔴 New
P5	Read after write dependency	loop_site_1	GSimulation.cpp	nbody.x	🔴 New
P6	Read after write dependency	loop_site_1	GSimulation.cpp	nbody.x	🔴 New

The 'Read after write dependency: Code Locations' section shows the following code snippets:

```

X3 0x401c85 Parallel site GSimulation.cpp:157 start nbody.x 🔴 New
155 real_type distanceInv = 0.0f;
156
157 dx = particles->pos_x[i] - particles->pos_x[i]; //1flop
158 dy = particles->pos_y[i] - particles->pos_y[i]; //1flop
159 dz = particles->pos_z[i] - particles->pos_z[i]; //1flop

X6 0x401cb8, 0x401d17 Read GSimulation.cpp:164 start register XMM1 nbody.x 🔴 New
162 distanceInv = 1.0f / sqrtf(distanceSqr); //1div+1sqrt
163
164 particles->acc_x[i] += dx * G * particles->mass[j] * distanceInv * distanceInv * distanceInv; //6flops
165 particles->acc_y[i] += dy * G * particles->mass[j] * distanceInv * distanceInv * distanceInv; //6flops
166 particles->acc_z[i] += dz * G * particles->mass[j] * distanceInv * distanceInv * distanceInv; //6flops

X7 0x401d1e Write GSimulation.cpp:164 start nbody.x 🔴 New
162 distanceInv = 1.0f / sqrtf(distanceSqr); //1div+1sqrt
163
164 particles->acc_x[i] += dx * G * particles->mass[j] * distanceInv * distanceInv * distanceInv; //6flops
    
```

Dependencies analysis has high overhead:

- Run on reduced workload

Advisor Findings:

- RAW dependency
- Multiple reduction-type dependencies

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



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.

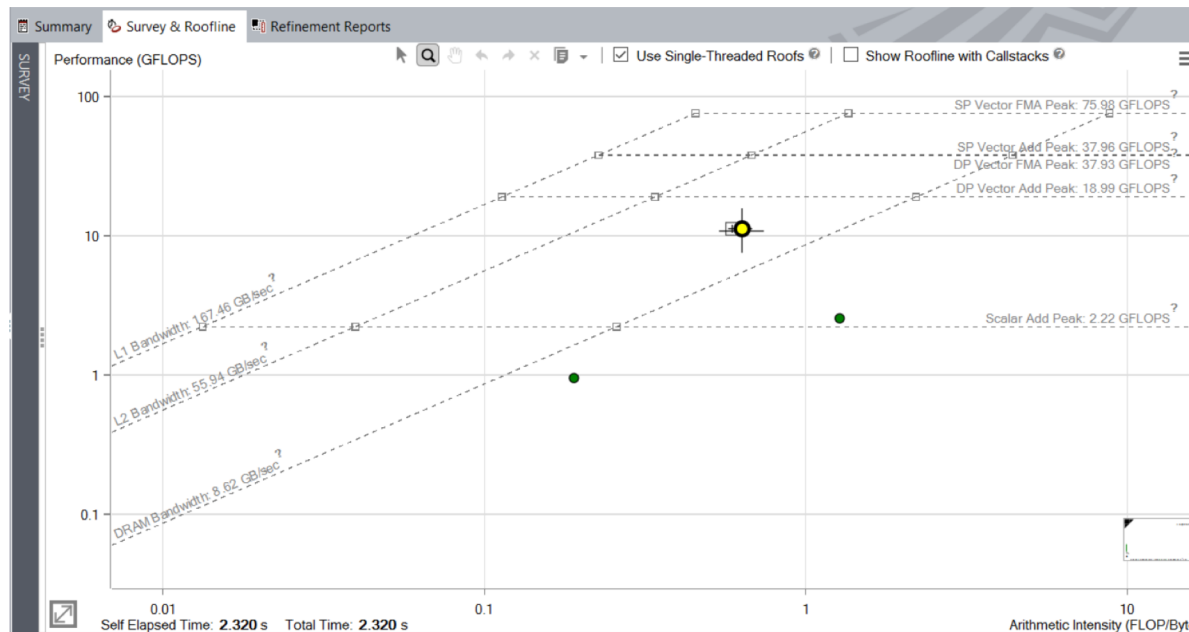
 [Resolve dependency](#)

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Performance After Resolved Dependencies



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

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Advisor Roofline – How much further can we go?

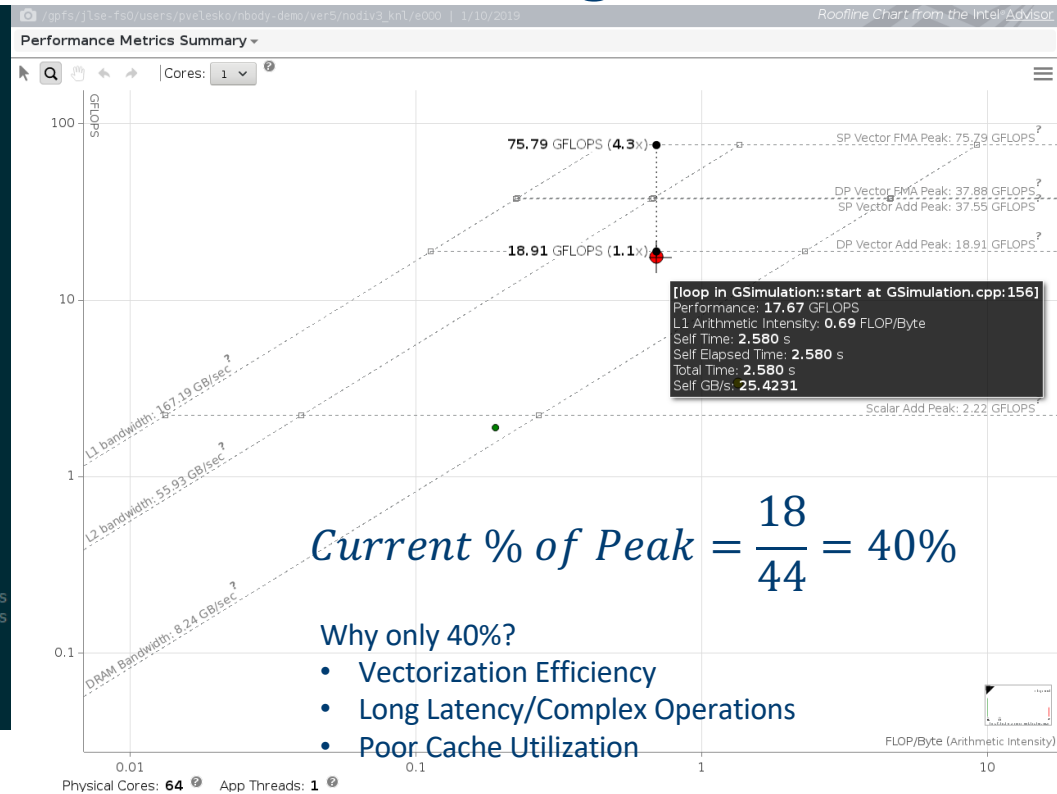
```

for (i = 0; i < n; i++)// update acceleration
{
#ifdef ASALIGN
    __assume_aligned(particles->pos_x, alignment);
    __assume_aligned(particles->pos_y, alignment);
    __assume_aligned(particles->pos_z, alignment);
    __assume_aligned(particles->acc_x, alignment);
    __assume_aligned(particles->acc_y, alignment);
    __assume_aligned(particles->acc_z, alignment);
    __assume_aligned(particles->mass, alignment);
#endif
    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 simdlen(16) 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]; //1flop
        dy = particles->pos_y[j] - particles->pos_y[i]; //1flop
        dz = particles->pos_z[j] - particles->pos_z[i]; //1flop

        distanceSqr = dx*dx + dy*dy + dz*dz + softeningSquared; //6flops
        distanceInv = 1.0f / sqrtf(distanceSqr); //1div+1sqrt

        ax_i+= dx * G * particles->mass[j] * distanceInv * distanceInv * distanceInv; //6flops
        ay_i += dy * G * particles->mass[j] * distanceInv * distanceInv * distanceInv; //6flops
        az_i += dz * G * particles->mass[j] * distanceInv * distanceInv * distanceInv; //6flops
    }
    particles->acc_x[i] = ax_i;
    particles->acc_y[i] = ay_i;
    particles->acc_z[i] = az_i;
}
    
```



$$FMA \text{ Ratio} = \frac{3}{29} = 10\%$$

Peak = SP Vector ADD * (1+ FMA Ratio)

Peak = 40 * (1 + 0.1) = 44 GFLOPS

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
 *Other names and brands may be claimed as the property of others.



Vectorization Efficiency?

The screenshot shows the Intel VTune Performance Analyzer interface. At the top, it displays 'Elapsed time: 5.19s' and two filters: 'Vectorized' (active) and 'Not Vectorized'. Below this, there are tabs for 'Summary', 'Survey & Roofline', and 'Refinement Reports'. The 'Roofline' view is active, showing a table of 'Vectorized Loops'. The table has columns for 'Vec...', 'Efficiency', 'Gai...', and 'VL (...)'. One row is highlighted, showing a loop in GSimulation with an efficiency of ~97%.

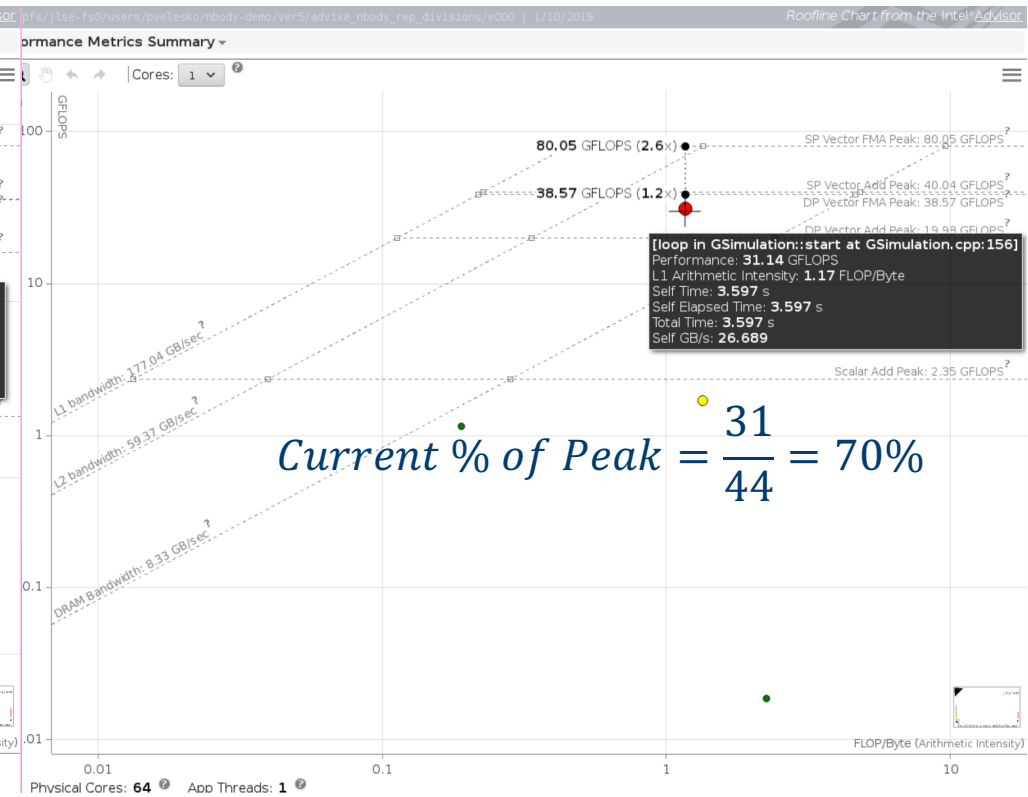
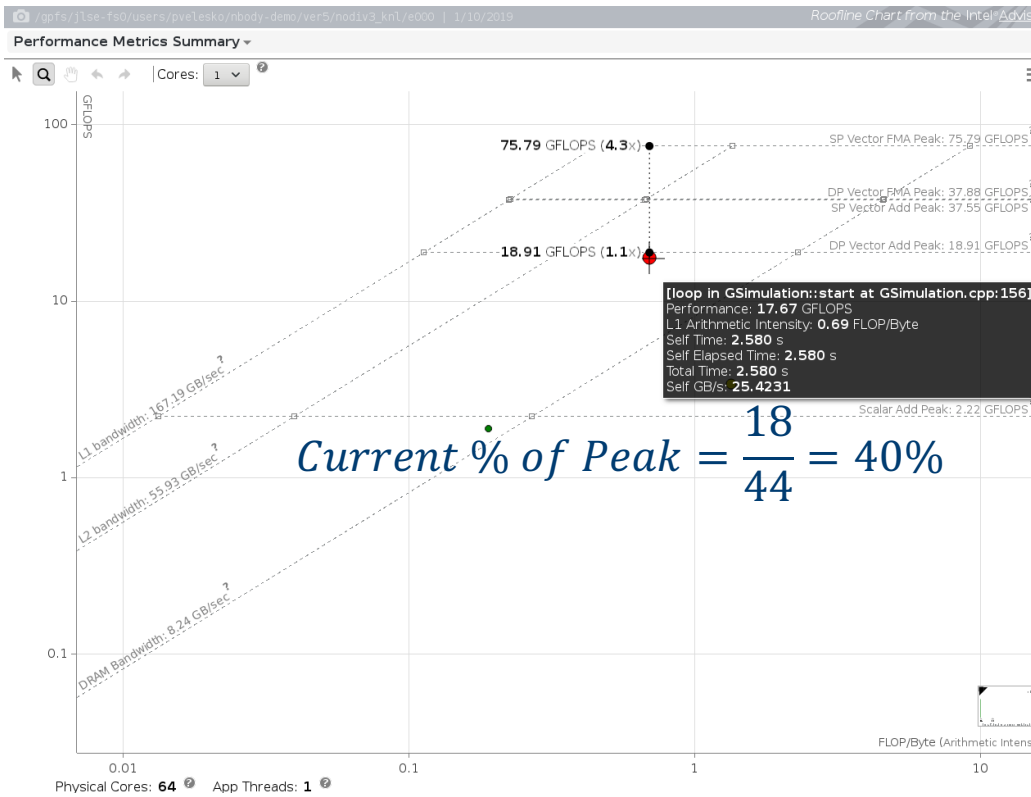
Function Call Sites and Loops		Vectorized Loops			
Vec...	Efficiency	Gai...	VL (...)		
[loop in GSimulation::start at GSim	~97%	15...	16		
[loop in GSimulation::start at GSimulati					

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Complex Operations?



Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
 *Other names and brands may be claimed as the property of others.



Poor Cache Utilization?

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.



Intel® VTUNE™ Amplifier

Core-level hardware metrics

<https://www.alcf.anl.gov/user-guides/vtune-xc40>

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

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Predefined Collections

Many available analysis types:

- uarch-exploration General microarchitecture exploration
- hpc-performance HPC Performance Characterization
- memory-access Memory Access
- disk-io Disk Input and Output
- concurrency Concurrency
- gpu-hotspots GPU Hotspots
- gpu-profiling GPU In-kernel Profiling
- hotspots Basic Hotspots
- locksandwaits Locks and Waits
- memory-consumption Memory Consumption
- system-overview System Overview
- ...

Python Support

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



The HPC Performance Characterization Analysis

Threading: CPU Utilization

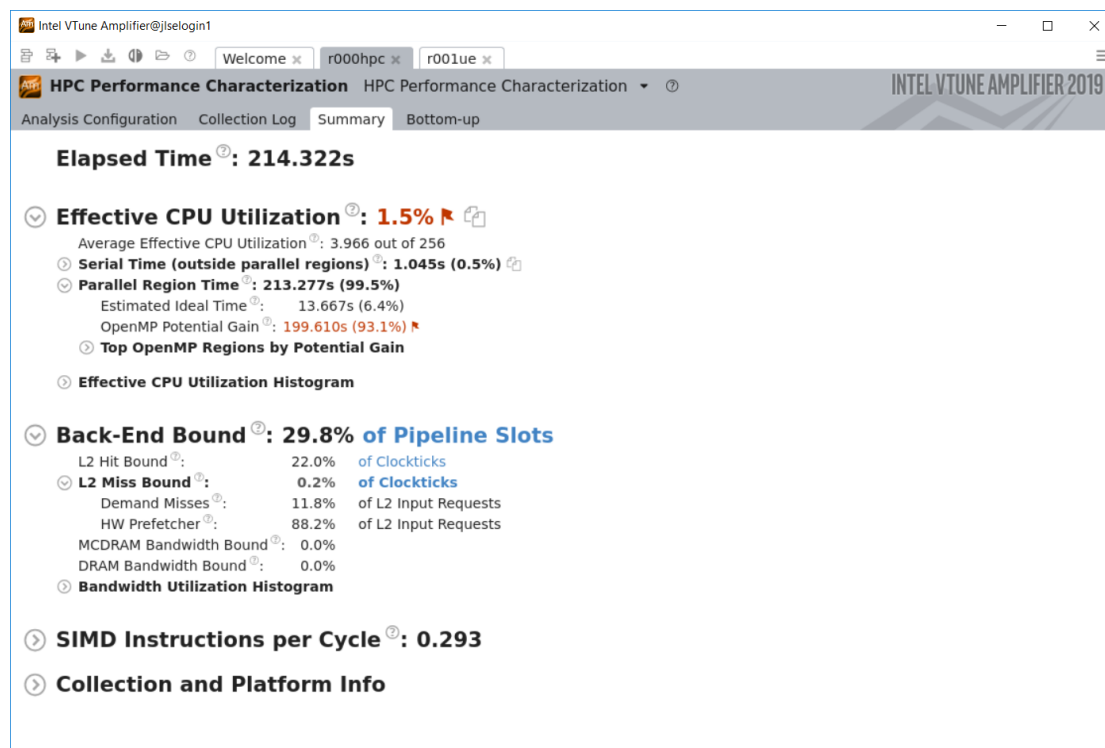
- Serial vs. Parallel time
- Top OpenMP regions by potential gain
- Tip: Use hotspot OpenMP region analysis for more detail

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.

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



uArch Exploration

Core Issues

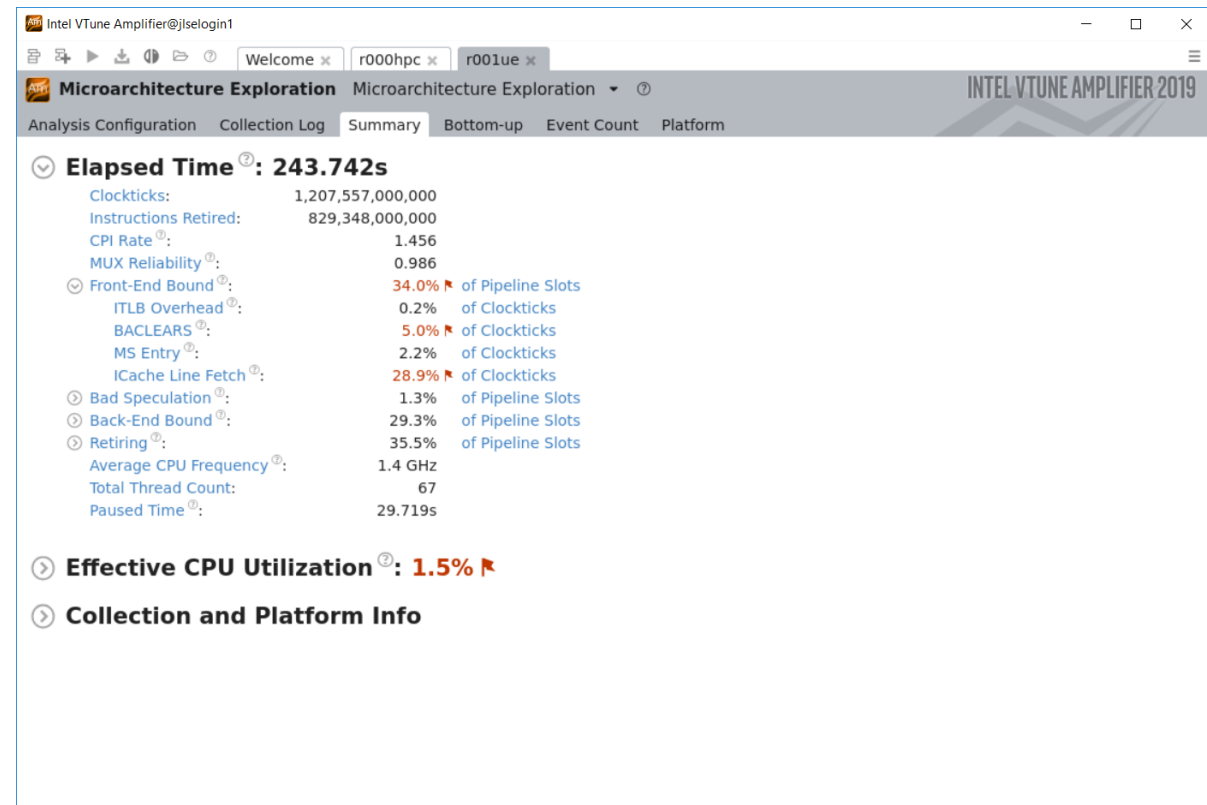
- Branch Misprediction
- CPI Rate

Back-End Bound Issues

- Instruction Cache
- Data Cache
- Split Loads
- TLB Overheads

Vectorization: FPU Utilization

- SIMD Arithmetic Intensity
- 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.

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Memory Access Analysis

Tune data structures for performance

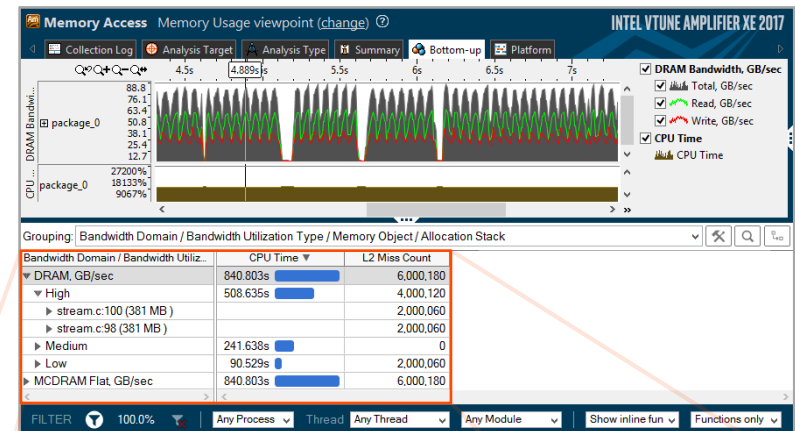
- Attribute cache misses to data structures (not just the code causing the miss)
- Support for custom memory allocators

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

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
 *Other names and brands may be claimed as the property of others.



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
- Compile with `-g` and `-dynamic`
- Set your environment:

```
$ module swap intel/18.0.0.128 intel/19.0.3.199
```

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



INTEL VTUNE AMPLIFIER 2018

Choose Analysis Type

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

Copy Command Line to Clipboard@jlselgin2

Command line:

```
/soft/compilers/intel/ltune_amplifier_2018.1.0.535340/bin64/ampkx-cl -collect hpc-performance -app-working-dir /usr/bin -- ls
```

Use -collect-with action

Hide knobs with default values

Copy Close

Start

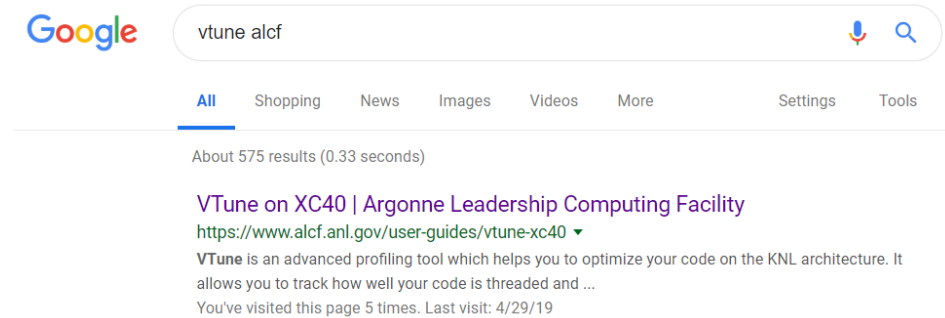
Start Paused

Choose Target

Command Line...

amplx.qsub Script

- Copy and customize the script from `/soft/perftools/intel/vtune/amplx.qsub`
- All-in-one script for profiling
 - Job size - ranks, threads, hyperthreads, affinity
 - Attach to a single, multiple or all ranks
 - Binary as `arg#1`, input as `arg#2`
 - `qsub amplx.qsub ./your_exe ./inputs/inp`
 - Binary and source search directory locations
 - Timestamp + binary name + input name as result directory
 - Save cobalt job files to result directory

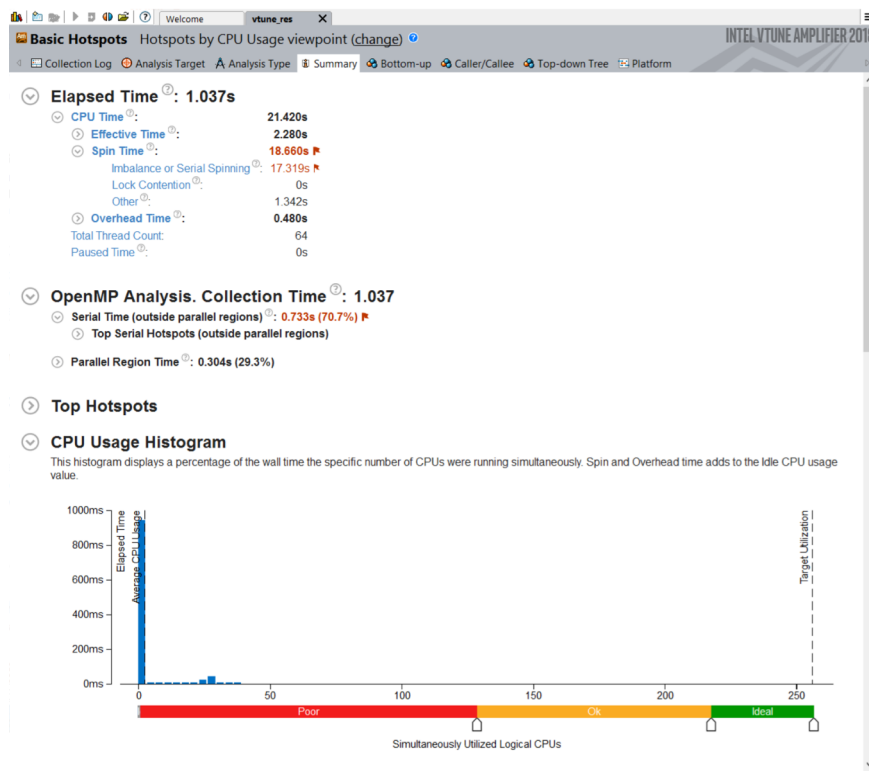


Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



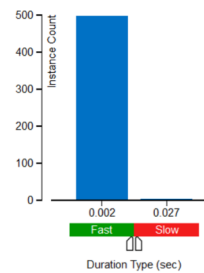
Hotspots analysis for nbody demo (ver7: threaded)



OpenMP Region Duration Histogram

This histogram shows the total number of region instances in your application executed with a specific duration. High number of slow instances may signal a performance bottleneck. Explore the data provided in the Bottom-up, Top-down Tree, and Timeline panes to identify code regions with the slow duration.

OpenMP Region: startSomp\$parallel64@unknown:146:182



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

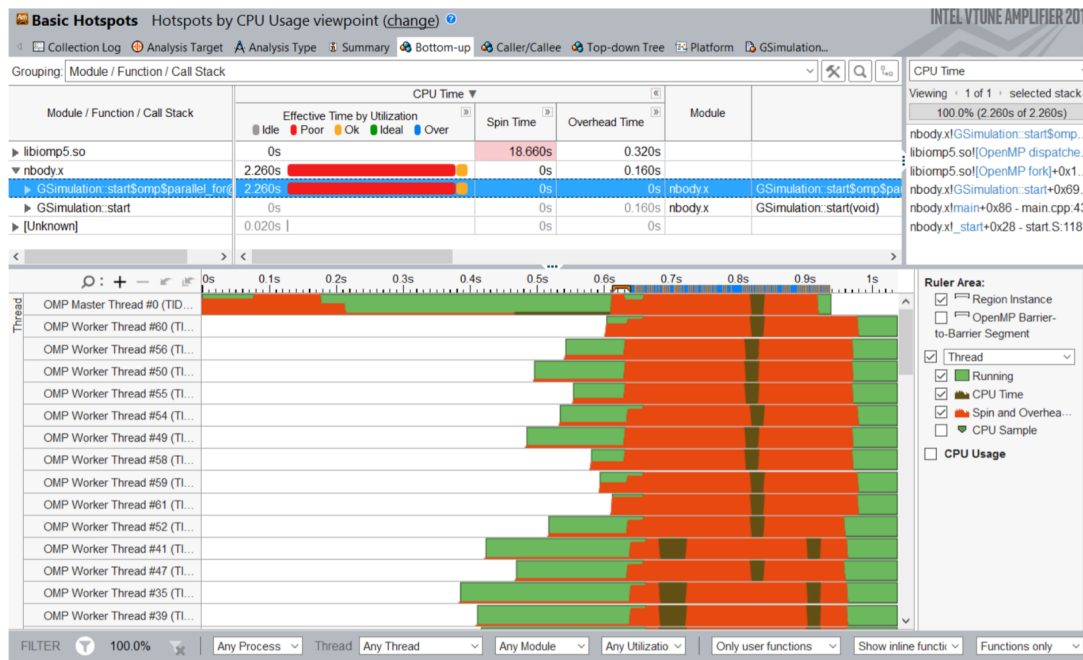
Let's look at the timeline for each thread to understand things better...

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



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.

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Intel VTune Amplifier

Hotspots Hotspots by CPU Utilization

Analysis Configuration Collection Log Summary **Bottom-up** Caller/Callee Top-down Tree Platform

Grouping: Function / Call Stack

Function / Call Stack	CPU Time	Module	Function (Full)	Source File	Start Address
▶ vdpowr_	18.664s	libmkl_intel_lp64.so	vdpowr_		0x695310
▶ aa	10.495s	distress	aa	aux.f90	0x41ec1c
▶ aa	9.674s	distress	aa	aux.f90	0x41ec9a
▶ invariants	9.055s	distress	invariants	aux.f90	0x41d550
▶ __libm_csqrt_ex	7.792s	libimf.so	__libm_csqrt_ex		0xc7a50
▶ spinoru	7.779s	distress	spinoru	aux.f90	0x41e9e0
▶ ktjet	7.137s	distress	ktjet	analysis.f90	0x420ae0
▶ __svml_log8_mask_b3	6.056s	distress	__svml_log8_mask_b3		0x532f50
▶ breit2lab	2.096s	distress	breit2lab	PS.f90	0x4602d0
▶ getljet	1.857s	distress	getljet	analysis.f90	0x421830
▶ me0_qlqgg	1.814s	distress	me0_qlqgg	amplitudes.f90	0x4408d0
▶ __libm_acos_l9	1.688s	libimf.so	__libm_acos_l9		0xedd80
▶ analyzejet	1.658s	distress	analyzejet	analysis.f90	0x422050
▶ ds_ql_s_nnlo_qcd_g	1.605s	distress	ds_ql_s_nnlo_qcd_g	sub.f90	0x4694e0
▶ csart	1.384s	libimf.so	csart		0x1d430

0s 20s 40s 60s 80s 100s 120s 140s

Thread

distress (TID: 55598)

CPU Utilization

Thread
 Running
 CPU Time
 Spin and Overhead ...
 CPU Sample
 CPU Utilization
 CPU Time
 Spin and Overhead ...

FILTER 100.0%
 Any Process Any Thread Any Module Any Utilizati User functions + 1 Show inline funct Functions only

Optimi Copyright © *Other nam

intel 47

Intel VTune Amplifier

Hotspots Hotspots by CPU Utilization

Analysis Configuration Collect Log Summary Bottom-up Caller/Callee Top-down Tree Platform

Grouping: Function / Call Stack

Function / Call Stack	CPU Time	Module	Function (Full)	Source File	Start Address
vdpowr_	13.1%	libmkl_intel_ip64.so	vdpowr_		0x695310
▶ aa	7.4%	distress	aa	aux.f90	0x41ec1c
▶ aa	6.8%	distress	aa	aux.f90	0x41ec9a
▶ invariants	6.4%	distress	invariants	aux.f90	0x41d550
▶ __libm_csqrt_ex	5.5%	libimf.so	__libm_csqrt_ex		0xc7a50
▶ spinoru	5.5%	distress	spinoru	aux.f90	0x41e9e0
▶ ktjet	5.0%	distress	ktjet	analysis.f90	0x420ae0
▶ __svml_log8_mask_b3	4.3%	distress	__svml_log8_mask_b3		0x532f50
▶ breit2lab	1.5%	distress	breit2lab	PS.f90	0x4602d0
▶ gettjet	1.3%	distress	gettjet	analysis.f90	0x421830
▶ me0_qlqgg	1.3%	distress	me0_qlqgg	amplitudes.f90	0x4408d0
▶ __libm_acos_l9	1.2%	libimf.so	__libm_acos_l9		0xedd80
▶ analyzejet	1.2%	distress	analyzejet	analysis.f90	0x422050
▶ ds_ql_s_nnlo_qcd_g	1.1%	distress	ds_ql_s_nnlo_qcd_g	sub.f90	0x4694e0
▶ csart	1.0%	libimf.so	csart		0x1d430

0s 20s 40s 60s 80s 100s 120s 140s

Thread: distress (TID: 55598)

CPU Utilization

Thread: Running CPU Time Spin and Overhead ... CPU Sample

CPU Utilization CPU Time Spin and Overhead ...

Optimi Copyright © *Other nam

FILTER 100.0%

Any Process Any Thread Any Module Any Utilizati User functions + 1 Show inline func Functions only

intel 48

Intel VTune Amplifier

Hotspots Hotspots by CPU Utilization

Analysis Configuration Collection Log Summary Bottom-up Caller/Callee Top-down Tree Platform

Grouping: Source Function / Function / Call Stack

Source Function / Function / Call Stack	CPU Time	Module	Function (Full)	Source File	Start Address
▶ aa	14.2%	aa	aa	aux.f90	0
▶ vdpowr_	13.1%	vdpowr_	vdpowr_		0
▶ invariants	6.4%	invariants	invariants	aux.f90	0
▶ __libm_csqrt_ex	5.5%	__libm_csqrt_ex	__libm_csqrt_ex		0
▶ spinoru	5.5%	spinoru	spinoru	aux.f90	0
▶ ktjet	5.0%	ktjet	ktjet	analysis.f90	0
▶ __svml_log8_mask_b3	4.3%	__svml_log8_mask_b3	__svml_log8_mask_b3		0
▶ subqcd	3.2%	subqcd	subqcd	amplitudes.f90	0
▶ breit2lab	1.6%	breit2lab	breit2lab	PS.f90	0
▶ hamp_qlqlqb_1	1.4%	hamp_qlqlqb_1	hamp_qlqlqb_1	amplitudes.f90	0
▶ getljet	1.3%	getljet	getljet	analysis.f90	0
▶ me0_qlqlgg	1.3%	me0_qlqlgg	me0_qlqlgg	amplitudes.f90	0
▶ __libm_acos_l9	1.2%	__libm_acos_l9	__libm_acos_l9		0
▶ analyzejet	1.2%	analyzejet	analyzejet	analysis.f90	0
▶ hamo_alalaab_2	1.1%	hamo_alalaab_2	hamo_alalaab_2	amolitudes.f90	0

0s 20s 40s 60s 80s 100s 120s 140s

Thread distress (TID: 55598)

CPU Utilization


- Thread
 - Running
 - CPU Time
 - Spin and Overhead ...
 - CPU Sample
- CPU Utilization
 - CPU Time
 - Spin and Overhead ...

Optimi Copyright © *Other nam

FILTER 100.0%

Any Process Any Thread Any Module Any Utilizati

User functions + 1 Show inline func Functions only



49

Intel VTune Amplifier

Hotspots Hotspots by CPU Utilization

Analysis Configuration Collection Log Summary Bottom-up Caller/Callee Top-down Tree Platform aux.f90 x aux.f90 x

Grouping: Source Function / Function / Call Stack

Source Function / Function / Call Stack	CPU Time	Module	Function (Full)	Source File	Start Address
spinoru	27.5%		spinoru	aux.f90	0
▶ invariants	9.0%		invariants	aux.f90	0
▶ getpdfs	8.3%		getpdfs	fitpdf.f90	0
▶ ktjet	6.9%		ktjet	analysis.f90	0
▶ me0_qlqlgg	6.1%		me0_qlqlgg	amplitudes.f90	0
▶ __svml_log8_mask_b3	5.9%		__svml_log8_mask_b3		0
▶ breit2lab	2.5%		breit2lab	PS.f90	0
▶ dli2	2.4%		dli2	lis.f90	0
▶ getljet	1.8%		getljet	analysis.f90	0
▶ analyzejet	1.6%		analyzejet	analysis.f90	0
▶ me0_qlqlqb_f3	1.6%		me0_qlqlqb_f3	amplitudes.f90	0
▶ ds_ql_s_nnlo_qcd_g	1.6%		ds_ql_s_nnlo_qcd_g	sub.f90	0
▶ me0_qlqlqb_f4	1.3%		me0_qlqlqb_f4	amplitudes.f90	0
▶ ps4	1.3%		ps4	PS.f90	0
▶ for constr	1.3%		for constr		0

0s 20s 40s 60s 80s 100s 120s 140s

Thread distress (TID: 55598)

CPU Utilization

Thread

- Running
- CPU Time
- Spin and Overhead ...
- CPU Sample
- CPU Utilization
 - CPU Time
 - Spin and Overhead ...

Optimi Copyright © *Other nam

FILTER 98.9%

Any Process Any Thread [98.9%] distres Any Utilizatic User functions + 1 Hide inline funcic Functions only

intel 50

Intel VTune Amplifier

Hotspots Hotspots by CPU Utilization

Analysis Configuration Collection Log Summary Bottom-up Caller/Callee Top-down Tree Platform aux.f90 x aux.f90 x

Grouping: Source Function / Function / Call Stack

Source Function / Function / Call Stack	CPU Time	Module	Function (Full)	Source File	Start Address
[Loop at line 264 in spinoru]	23.8%		[Loop at line 264 in spinoru]	aux.f90	0
[Loop at line 141 in nnlobeami]	19.3%		[Loop at line 141 in nnlobeami]	beamintegrand.f90	0
[Loop at line 2499 in dxsec_ql_nnlor]	11.1%		[Loop at line 2499 in dxsec_ql_nnlor]	xsec.f90	0
[Loop at line 112 in vegas]	10.6%		[Loop at line 112 in vegas]	vegas.f90	0
[Loop at line 2750 in dxsec_ql_nnlov_a]	3.2%		[Loop at line 2750 in dxsec_ql_nnlov_a]	xsec.f90	0
[Loop at line 60 in ktjet]	3.1%		[Loop at line 60 in ktjet]	analysis.f90	0
[Loop at line 1778 in ds_ql_s_nnlo_qcd_g]	2.9%		[Loop at line 1778 in ds_ql_s_nnlo_qcd_g]	sub.f90	0
[Loop at line 181 in invariants]	2.6%		[Loop at line 181 in invariants]	aux.f90	0
[Loop at line 180 in invariants]	2.1%		[Loop at line 180 in invariants]	aux.f90	0
[Loop at line 2055 in ds_ql_s_nnlo_qcd_f2]	2.0%		[Loop at line 2055 in ds_ql_s_nnlo_qcd_f2]	sub.f90	0
[Loop at line 43 in ktjet]	2.0%		[Loop at line 43 in ktjet]	analysis.f90	0
[Loop at line 1986 in ds_ql_s_nnlo_qcd_f1]	1.8%		[Loop at line 1986 in ds_ql_s_nnlo_qcd_f1]	sub.f90	0
[Loop at line 1882 in ds_ql_s_nnlo_qcd_g]	1.8%		[Loop at line 1882 in ds_ql_s_nnlo_qcd_g]	sub.f90	0
[Loop at line 1846 in ds_ql_s_nnlo_qcd_g]	1.8%		[Loop at line 1846 in ds_ql_s_nnlo_qcd_g]	sub.f90	0
[Loop at line 1812 in ds_ql_s_nnlo_qcd_g]	1.7%		[Loop at line 1812 in ds_ql_s_nnlo_qcd_g]	sub.f90	0

Thread: distress (TID: 55598)

CPU Utilization

Thread: Running CPU Time Spin and Overhead ... CPU Sample

CPU Utilization: CPU Time Spin and Overhead ...

Optimi Copyright © *Other nam

FILTER 98.9%

Any Process Any Thread [98.9%] distres Any Utilizatic User functions + 1 Show inline functi Loops only

INTEL VTUNE AMPLIFIER 2019

51

Intel VTune Amplifier

Hotspots Hotspots by CPU Utilization

Analysis Configuration Collection Log Summary Bottom-up Caller/Callee Top-down Tree Platform aux.f90 x aux.f90 x

Grouping: Call Stack

Function Stack	CPU Time: Total	CPU Time: Self	Module	Function (Full)	Source File	Start Address
▼ Total	100.0%	0s				
▼ [Outside any loop]	99.9%	0.020s		[Outside any loop]		0
▼ [Loop at line 100 in vegas]	99.6%	0s	distress	[Loop at line 100 in vegas]	vegas.f90	0x4162c8
▼ [Loop at line 112 in vegas]	99.6%	1.531s	distress	[Loop at line 112 in vegas]	vegas.f90	0x416641
▼ [Loop at line 112 in vegas]	98.2%	13.427s	distress	[Loop at line 112 in vegas]	vegas.f90	0x4166f1
▼ [Loop at line 2499 in dxsec_ql]	36.9%	15.606s	distress	[Loop at line 2499 in dxsec_ql...]	xsec.f90	0x49ba17
▼ [Loop at line 263 in spinoru]	24.2%	1.422s	distress	[Loop at line 263 in spinoru]	aux.f90	0x41ecd6
[Loop at line 264 in spinoru]	23.2%	32.939s	distress	[Loop at line 264 in spinoru]	aux.f90	0x41edcf
▶ [Loop at line 258 in spinoru]	1.1%	0.498s	distress	[Loop at line 258 in spinoru]	aux.f90	0x41ea94
▶ [Loop at line 260 in spinoru]	0.4%	0.324s	distress	[Loop at line 260 in spinoru]	aux.f90	0x41ec41
▶ [Loop at line 2487 in LHAPD]	0.1%	0.048s	libLHAPDF.so	[Loop at line 2487 in LHAPDF:...]	stl_algo.h	0x669c9
▶ [Loop at line 1169 in LHAPD]	0.1%	0.036s	libLHAPDF.so	[Loop at line 1169 in LHAPDF:...]	stl_tree.h	0x66960
▶ [Loop at line 139 in nnlobeami]	19.1%	0s	distress	[Loop at line 139 in nnlobeami]	beaminteg...	0x4310f9
▶ [Loop at line 43 in ktjet]	6.4%	2.808s	distress	[Loop at line 43 in ktjet]	analysis.f90	0x420c70
▶ [Loop at line 2750 in dxsec_d]	3.8%	4.494s	distress	[Loop at line 2750 in dxsec_d]	xsec.f90	0x49d2b2

Thread: distress (TID: 55598)

CPU Utilization

0s 20s 40s 60s 80s 100s 120s 140s

Thread: Thread

- Running
- CPU Time
- Spin and Overhead ...
- CPU Sample

CPU Utilization

- CPU Time
- Spin and Overhead ...

Optimi Copyright © *Other nam

FILTER 100.0%

Any Process Any Thread Any Module Any Utilizati User functions + 1 Show inline functi Loops only

intel 52

Intel VTune Amplifier

Hotspots Hotspots by CPU Utilization

Analysis Configuration HPC Performance Characterization Bottom-up Caller/Callee Top-down Tree Platform aux.f90 x aux.f90 x

Grouping: Call Stack Hotspots by CPU Utilization

Function	CPU Time: Self	Module	Function (Full)	Source File	Start Address
Total	100.0%				
[Outside any loop]	99.9%		[Outside any loop]		0
[Loop at line 100 in vegas]	99.6%	distress	[Loop at line 100 in vegas]	vegas.f90	0x4162c8
[Loop at line 112 in vegas]	99.6%	distress	[Loop at line 112 in vegas]	vegas.f90	0x416641
[Loop at line 112 in vegas]	98.2%	distress	[Loop at line 112 in vegas]	vegas.f90	0x4166f1
[Loop at line 2499 in dxsec_ql...]	36.9%	distress	[Loop at line 2499 in dxsec_ql...]	xsec.f90	0x49ba17
[Loop at line 263 in spinoru]	24.2%	distress	[Loop at line 263 in spinoru]	aux.f90	0x41ecd6
[Loop at line 264 in spinoru]	23.2%	distress	[Loop at line 264 in spinoru]	aux.f90	0x41edcf
[Loop at line 258 in spinoru]	1.1%	distress	[Loop at line 258 in spinoru]	aux.f90	0x41ea94
[Loop at line 260 in spinoru]	0.4%	distress	[Loop at line 260 in spinoru]	aux.f90	0x41ec41
[Loop at line 2487 in LHAPDF...]	0.1%	libLHAPDF.so	[Loop at line 2487 in LHAPDF...]	stl_algo.h	0x669c9
[Loop at line 1169 in LHAPDF...]	0.1%	libLHAPDF.so	[Loop at line 1169 in LHAPDF...]	stl_tree.h	0x66960
[Loop at line 139 in nnlobeami]	19.1%	distress	[Loop at line 139 in nnlobeami]	beaminteg...	0x4310f9
[Loop at line 43 in ktjet]	6.4%	distress	[Loop at line 43 in ktjet]	analysis.f90	0x420c70
[Loop at line 2750 in dxsec_ql...]	3.8%	distress	[Loop at line 2750 in dxsec_ql...]	xsec.f90	0x49d2b2

Thread: distress (TID: 55598)

CPU Utilization

Thread Legend:

- Thread
- Running
- CPU Time
- Spin and Overhead ...
- CPU Sample
- CPU Utilization
- CPU Time
- Spin and Overhead ...

Optimi Copyright © *Other nam

FILTER 100.0%

Any Process Any Thread Any Module Any Utilizati User functions + 1 Show inline functi Loops only

intel 53

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
```

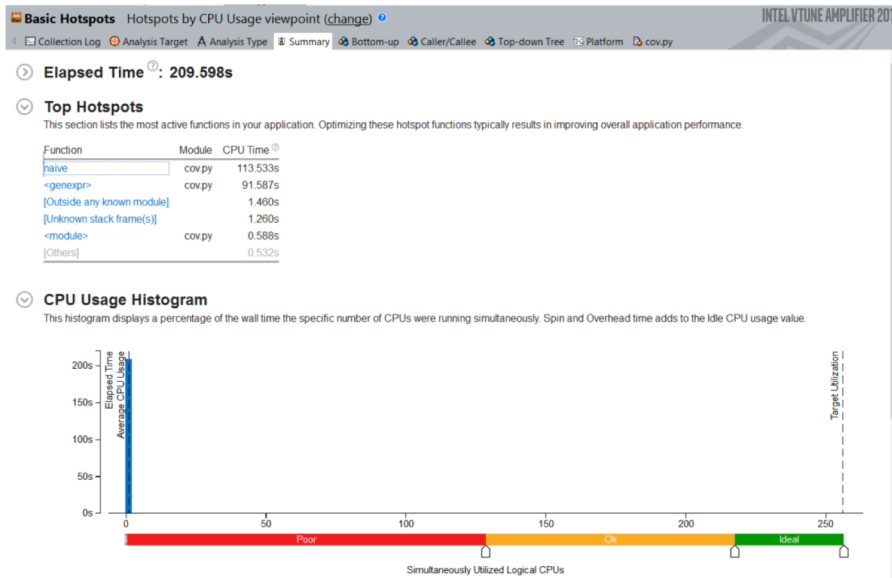
Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



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”

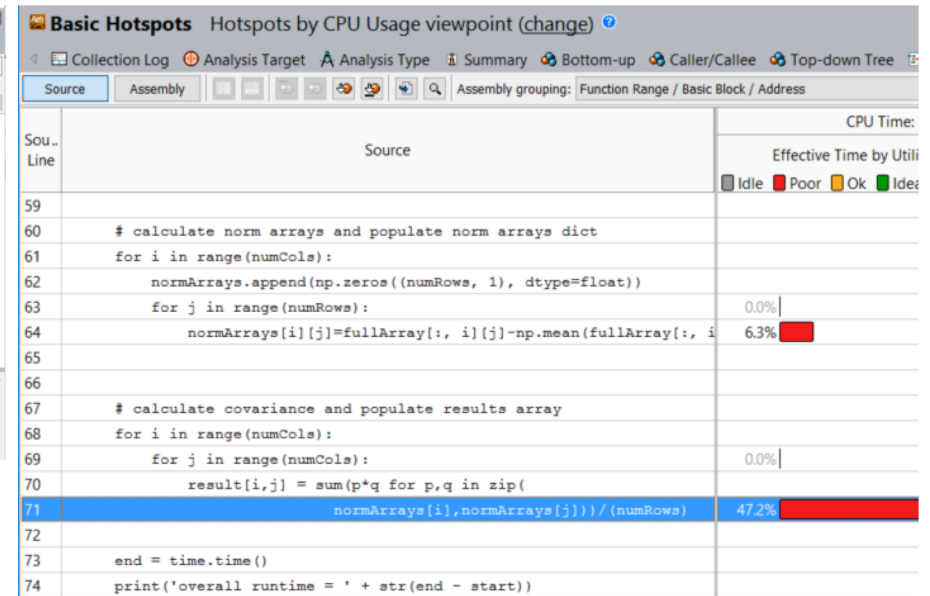
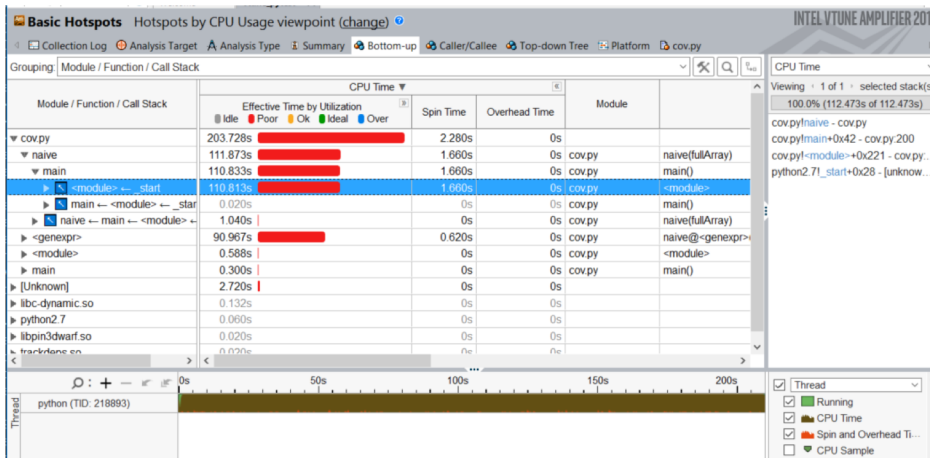
Click on top function to go to Bottom-up view

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



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

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
 *Other names and brands may be claimed as the property of others.



When do I use Vtune vs Advisor?

Vtune

- What's my cache hit ratio?
- Which loop/function is consuming most time overall? (bottom-up)
- Am I stalling often? IPC?
- Am I keeping all the threads busy?
- Am I hitting remote NUMA?
- When do I maximize my BW?

Advisor

- Which vector ISA am I using?
- How is time spent starting from entering my binary? (top-down)
- What is my vectorization efficiency?
- Can I safely force vectorization?
- Inlining? Data type conversions?
- Roofline

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Remember

Compile with `-g` and `-dynamic`

Profile 1 rank - `amplxe.qsub/advixe.qsub`

Advisor for big picture

Vtune for details

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.



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>

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



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.

Copyright © 2018, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804



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 [...]
```

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

