



Software

Profiling your application with Intel[®] Vtune[™] Amplifier

Paulius Velesko

Tuning at Multiple Hardware Levels

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

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

Intel® Compiler Reports

FREE* performance metrics

Compile with -qopt-report=5

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

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

Intel® Application Performance Snapshot

Bird's eye view

VTune™ Amplifier's Application Performance Snapshot

High-level overview of application performance

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

Usage on Theta

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

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

```
$ 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%	<div style="width: 40%; background-color: #C00000; height: 10px;"></div>
OpenMP Imbalance	0.43%	<10%	<div style="width: 5%; background-color: #C00000; height: 10px;"></div>
Memory Stalls	14.70%	<20%	<div style="width: 10%; background-color: #C00000; height: 10px;"></div>
FPU Utilization	0.30% ▲	>50%	<div style="width: 60%; background-color: #C00000; height: 10px;"></div>
I/O Bound	0.00%	<10%	<div style="width: 0%; background-color: #C00000; height: 10px;"></div>

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 ▲

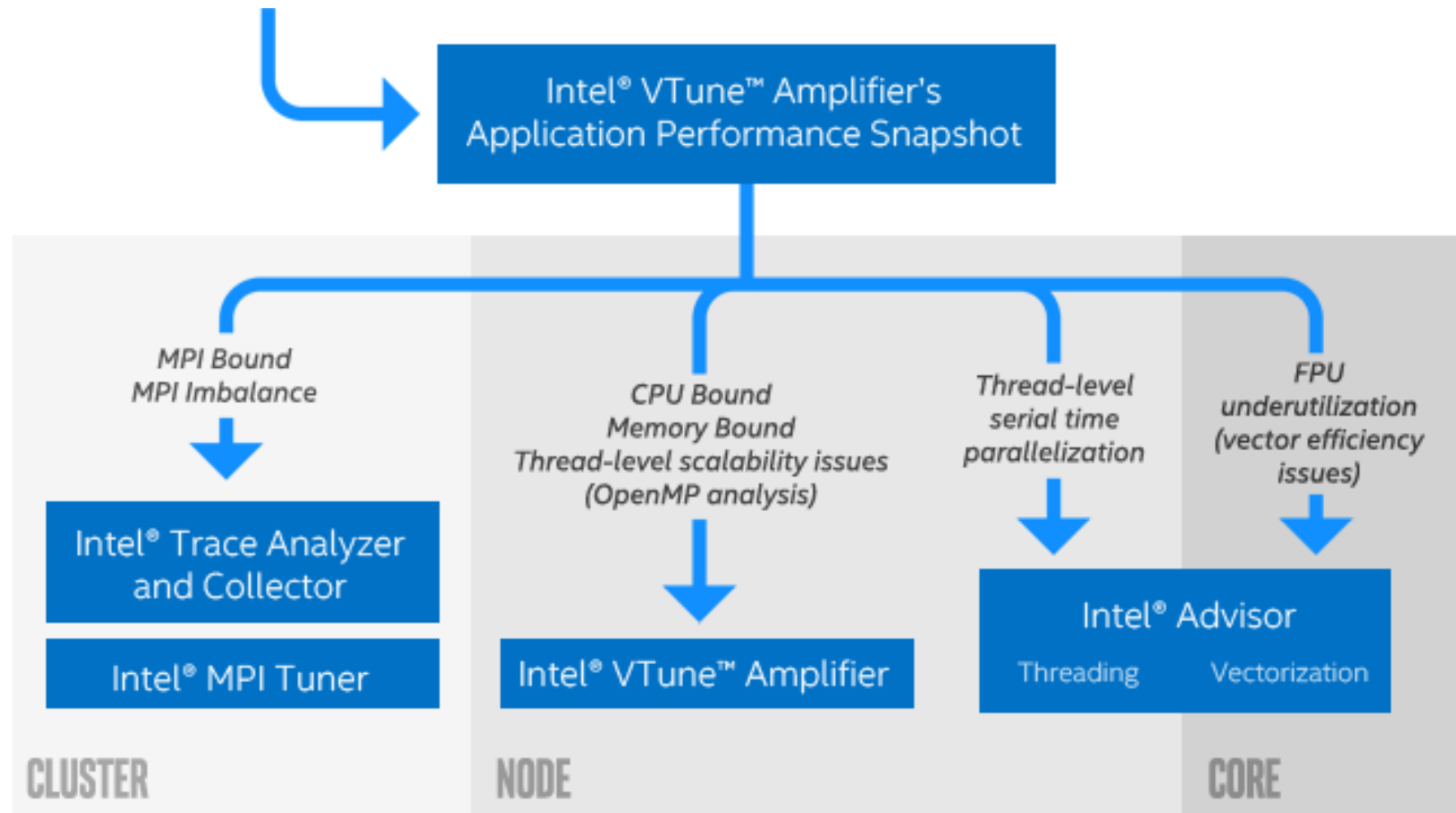
FP Arith/Mem Wr Instr. Ratio
0.30 ▲

Optimization Notice

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

|
8

Tuning Workflow



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
- User-mode driverless sampling
- Event-based sampling

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

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

Getting your application ready for profiling

-g

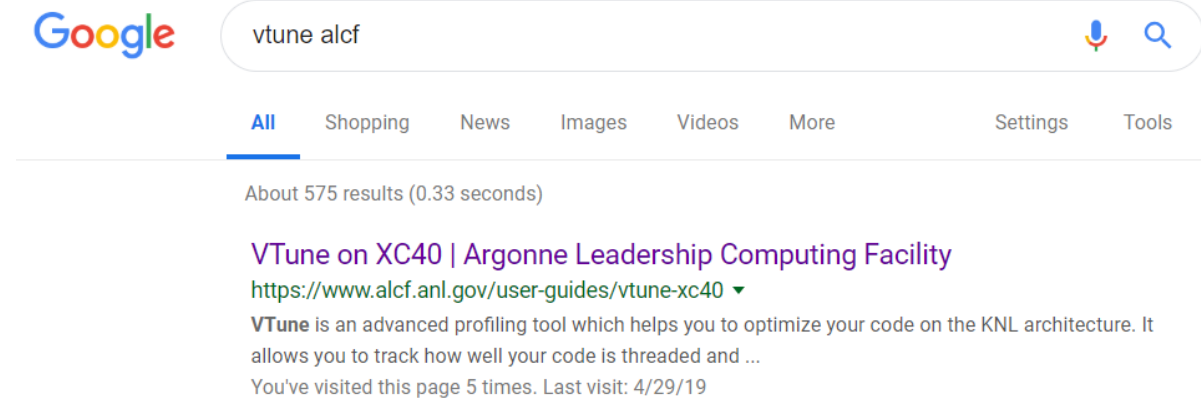
-dynamic

Running on Theta

- Cray systems (such as Theta) use aprun instead of mpirun
 - No SPMD notation
 - `mpirun -n 1 amplxe-cl -c hotspots ./exe : -n <N-1> ./exe`
 - Use `$PE_RANK` in a bash script instead
 - If `$PE_RANK==0` `amplxe-cl -c hotspots ./exe`; else ...
 - `PMI_NO_FORK`
- Darshan profiling
- Dynamic Linking

amplxe.qsub Script

- Copy and customize the script from `/soft/perftools/intel/vtune/amplxe.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 amplxe.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



Google search results for "vtune alcf". The search bar shows "vtune alcf" and the results include a link to "VTune on XC40 | Argonne Leadership Computing Facility" with the URL <https://www.alcf.anl.gov/user-guides/vtune-xc40>. The snippet describes VTune as an advanced profiling tool for KNL architecture.

Optimization Notice

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



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

1

Copy Command Line to Clipboard@jlselgin2

Command line:

```
yssoft/compilers/intel/vtune_amplifier_2018.1.0.535340/bin64/amplxe-cl -collect hpc-performance -app-working-dir /usr/bin -- ls
```

Use -collect-with action

Hide knobs with default values

Copy Close

Start

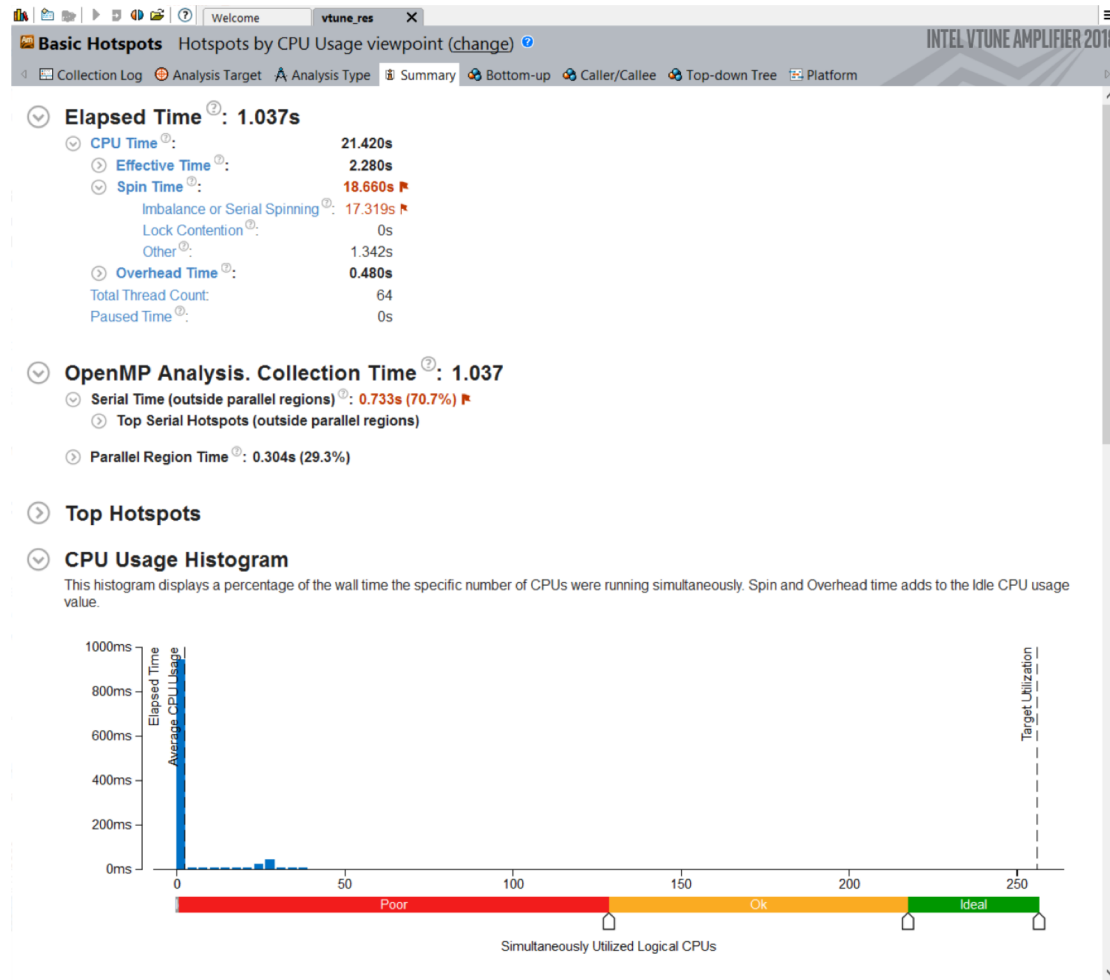
Start Paused

Choose Target

Command Line...

Hotspots analysis for nbody demo (ver7: threaded)

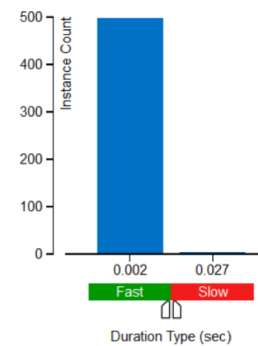
- `qsub amplxe.qsub ./your_exe ./inputs/inp`



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$parallel:64@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

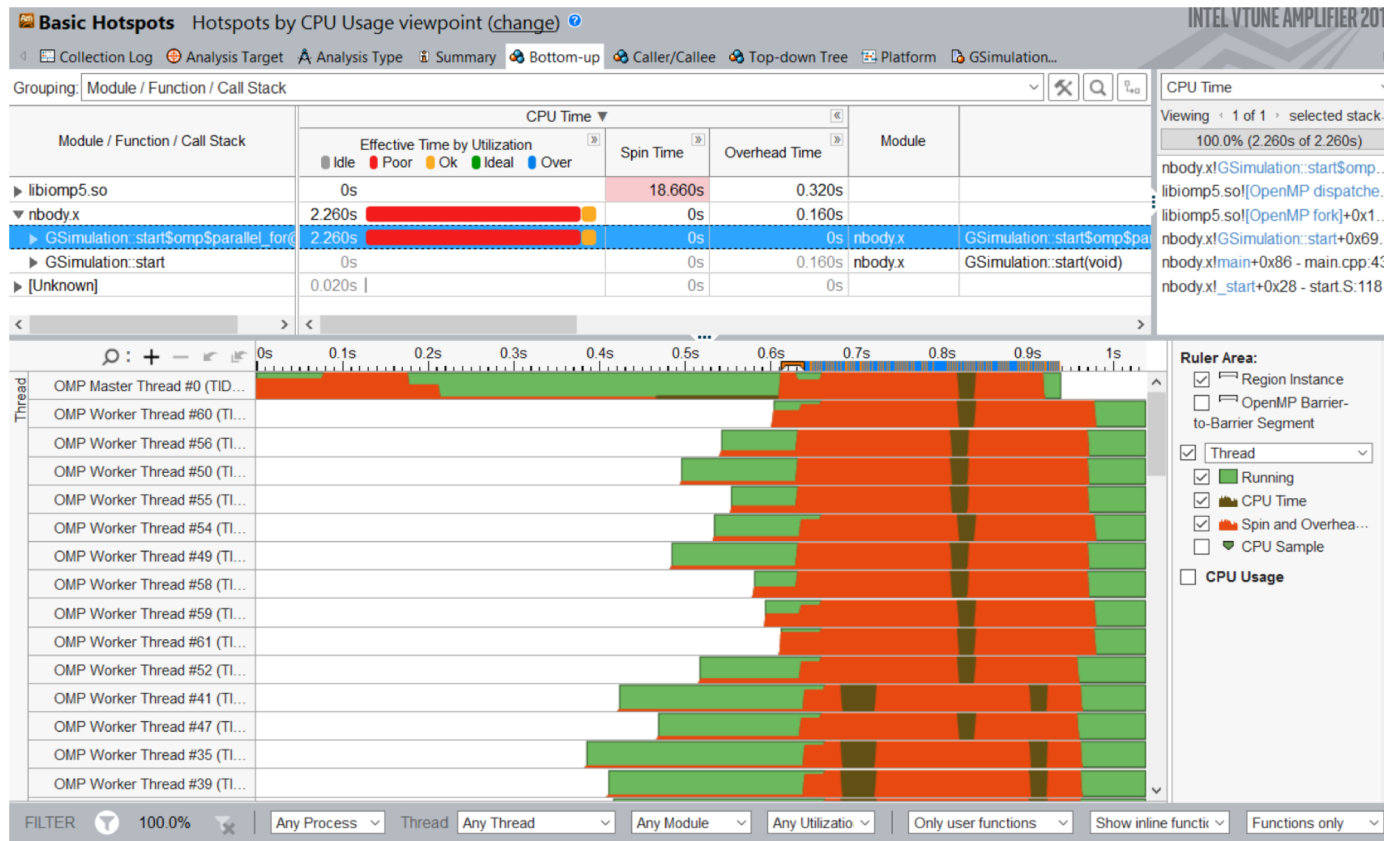
Let's look at the timeline for each thread to understand things better...

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.

Grouping: Function / Call Stack

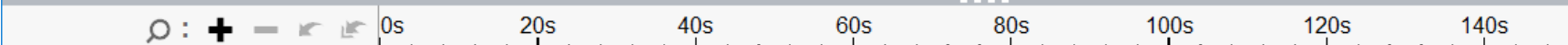
Function / Call Stack	CPU Time	Module	Function (Full)	Source File	Start Address
▶ vdpowr_	18.664s	libmkl_intel_lp64.so	vdpowr_		0x695310
▶ aa	10.495s	distress	aa	aux.f90	0x41ec1c
▶ aa	9.674s	distress	aa	aux.f90	0x41ec9a
▶ invariants	9.055s	distress	invariants	aux.f90	0x41d550
▶ __libm_csqrt_ex	7.792s	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_qlqlgg	1.814s	distress	me0_qlqlgg	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



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

Grouping: Function / Call Stack

Function / Call Stack	CPU Time	Module	Function (Full)	Source File	Start Address
▶ vdpowr_	13.1%	libmkl_intel_lp64.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
▶ getljet	1.3%	distress	getljet	analysis.f90	0x421830
▶ me0_qlqlgg	1.3%	distress	me0_qlqlgg	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

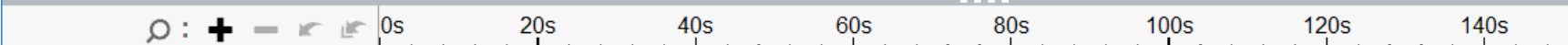


- Thread
 - Running
 - CPU Time
 - Spin and Overhead ...
 - CPU Sample
- CPU Utilization
 - CPU Time
 - Spin and Overhead ...

CPU Utilization

Grouping Source Function / Function / Call Stack

Source Function / Function / Call Stack	CPU Time	Module	Function (Full)	Source File	Start Address
▶ aa	14.2%		aa	aux.f90	0
▶ vdpowr_	13.1%		vdpowr_		0
▶ invariants	6.4%		invariants	aux.f90	0
▶ __libm_csqrt_ex	5.5%		__libm_csqrt_ex		0
▶ spinoru	5.5%		spinoru	aux.f90	0
▶ ktjet	5.0%		ktjet	analysis.f90	0
▶ __svml_log8_mask_b3	4.3%		__svml_log8_mask_b3		0
▶ subqcd	3.2%		subqcd	amplitudes.f90	0
▶ breit2lab	1.6%		breit2lab	PS.f90	0
▶ hamp_qlqlqbb_1	1.4%		hamp_qlqlqbb_1	amplitudes.f90	0
▶ getljet	1.3%		getljet	analysis.f90	0
▶ me0_qlqlgg	1.3%		me0_qlqlgg	amplitudes.f90	0
▶ __libm_acos_l9	1.2%		__libm_acos_l9		0
▶ analyzejet	1.2%		analyzejet	analysis.f90	0
▶ hamp_alalaab_2	1.1%		hamp_alalaab_2	amplitudes.f90	0

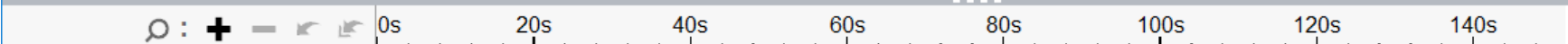


- Thread
 - Running
 - CPU Time
 - Spin and Overhead ...
 - CPU Sample
- CPU Utilization
 - CPU Time
 - Spin and Overhead ...



Grouping: Source Function / Function / Call Stack

Source Function / Function / Call Stack	CPU Time	Module	Function (Full)	Source File	Start Address
▶ spinoru	27.5%	spinoru	spinoru	aux.f90	0
▶ invariants	9.0%	invariants	invariants	aux.f90	0
▶ getpdfs	8.3%	getpdfs	getpdfs	fitpdf.f90	0
▶ ktjet	6.9%	ktjet	ktjet	analysis.f90	0
▶ me0_qlqlgg	6.1%	me0_qlqlgg	me0_qlqlgg	amplitudes.f90	0
▶ __svml_log8_mask_b3	5.9%	__svml_log8_mask_b3	__svml_log8_mask_b3		0
▶ breit2lab	2.5%	breit2lab	breit2lab	PS.f90	0
▶ dli2	2.4%	dli2	dli2	lis.f90	0
▶ getljet	1.8%	getljet	getljet	analysis.f90	0
▶ analyzejet	1.6%	analyzejet	analyzejet	analysis.f90	0
▶ me0_qlqlqbb_f3	1.6%	me0_qlqlqbb_f3	me0_qlqlqbb_f3	amplitudes.f90	0
▶ ds_ql_s_nnlo_qcd_g	1.6%	ds_ql_s_nnlo_qcd_g	ds_ql_s_nnlo_qcd_g	sub.f90	0
▶ me0_qlqlqbb_f4	1.3%	me0_qlqlqbb_f4	me0_qlqlqbb_f4	amplitudes.f90	0
▶ ps4	1.3%	ps4	ps4	PS.f90	0
▶ for costr	1.3%	for costr	for costr		0

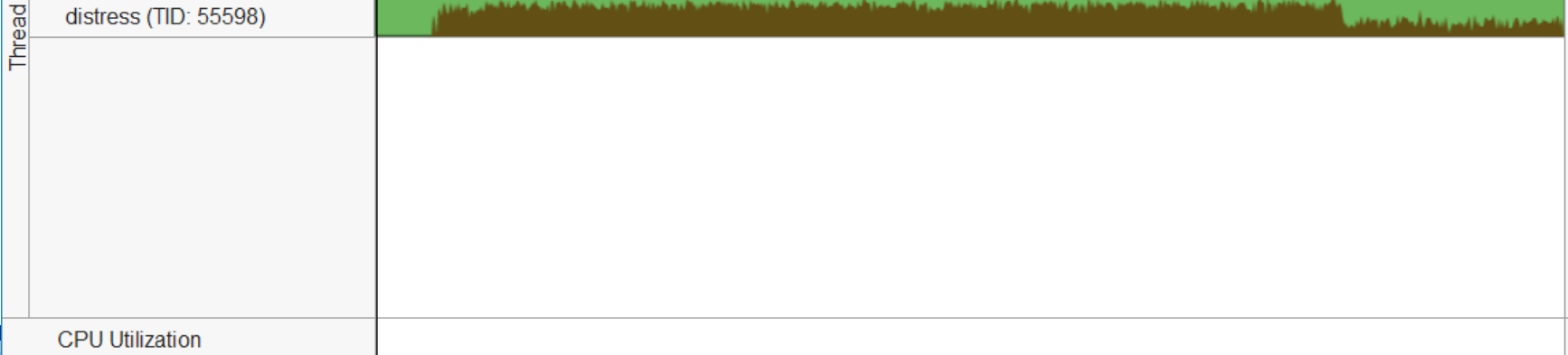
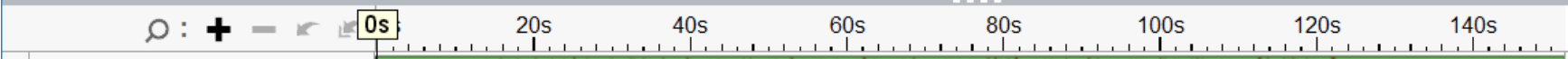


- Thread
 - Running
 - CPU Time
 - Spin and Overhead ...
 - CPU Sample
- CPU Utilization
 - CPU Time
 - Spin and Overhead ...

CPU Utilization

Grouping: Source Function / Function / Call Stack

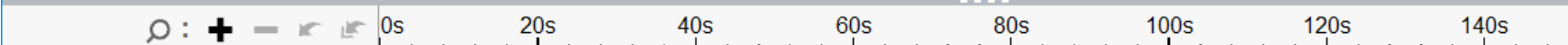
Source Function / Function / Call Stack	CPU Time	Module	Function (Full)	Source File	Start Address
[Loop at line 264 in spinoru]	23.8%		[Loop at line 264 in spinoru]	aux.f90	0
[Loop at line 141 in nnlobeami]	19.3%		[Loop at line 141 in nnlobeami]	beamintegrand.f90	0
[Loop at line 2499 in dxsec_ql_nnlor]	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
 - Running
 - CPU Time
 - Spin and Overhead ...
 - CPU Sample
- CPU Utilization
 - CPU Time
 - Spin and Overhead ...

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 LHAPD:...	stl_algo.h	0x669c9
▶ [Loop at line 1169 in LHAPD	0.1%	0.036s	libLHAPDF.so	[Loop at line 1169 in LHAPD:...	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 dl	3.8%	4.494s	distress	[Loop at line 2750 in dxsec dl...	xsec.f90	0x49d2b2



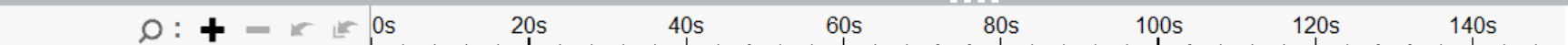
- Thread
 - Running
 - CPU Time
 - Spin and Overhead ...
 - CPU Sample
- CPU Utilization
 - CPU Time
 - Spin and Overhead ...

Hotspots Hotspots by CPU Utilization

Analysis Configur Bottom-up Caller/Callee Top-down Tree Platform aux.f90 x aux.f90 x

Grouping: Call Sta 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 LHAPD...]	0.1%	libLHAPDF.so	[Loop at line 2487 in LHAPD:...	stl_algo.h	0x669c9
▶ [Loop at line 1169 in LHAPD...]	0.1%	libLHAPDF.so	[Loop at line 1169 in LHAPD:...	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 dl...]	3.8%	distress	[Loop at line 2750 in dxsec dl...]	xsec.f90	0x49d2b2



Thread

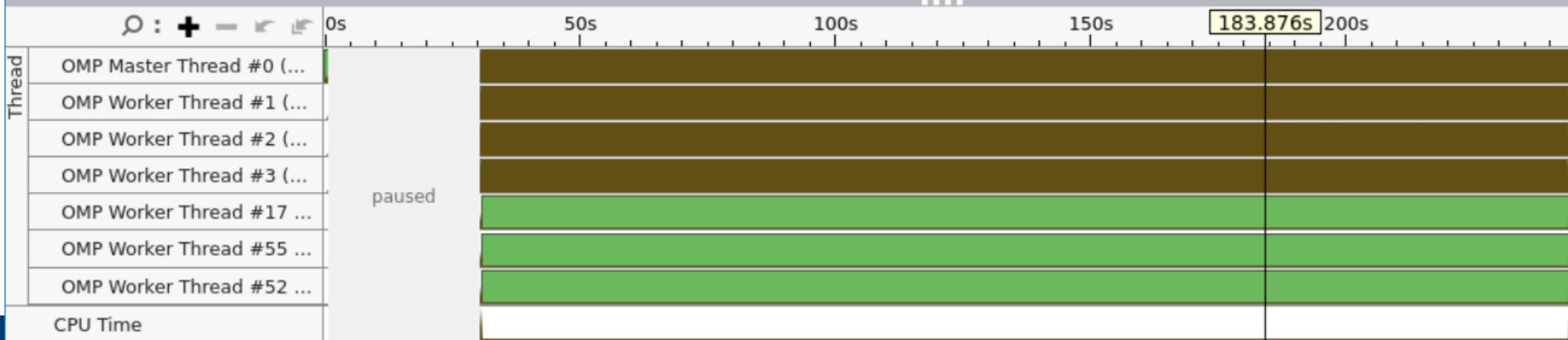
distress (TID: 55598)

CPU Utilization

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

Grouping: Function / Call Stack

Function / Call Stack	CPU Time	CPI Rate	Front-End Bound	Bad Speculation	Back-End Bound					
					Memory Latency					Mer
					L1 Hit Rate	L2 Hit Rate	L2 Hit Bound	L2 Miss Bound	UTLB Overhead	
bicub_interpol1_aio_vec	26.8%	1.092	15.2%	2.3%	97.9%	100.0%	12.2%	0.0%	0.1%	0.0%
bicub_interpol2_aio_vec	11.1%	1.488	36.4%	0.9%	97.8%	100.0%	7.2%	0.0%	0.3%	0.0%
efield_gk_elec2_vec	10.9%	1.850	29.2%	1.0%	85.2%	100.0%	31.0%	0.0%	2.7%	0.0%
derivs_elec_vec	8.7%	2.241	57.9%	0.2%	86.2%	100.0%	28.7%	0.0%	0.3%	0.0%
field_following_pos2_vec	5.7%	0.969	43.6%	1.8%	94.3%	100.0%	33.3%	0.0%	0.2%	0.0%
i_interpol_ider0_aio_vec	5.3%	1.896	12.0%	0.0%	89.5%	100.0%	11.8%	0.0%	0.5%	0.0%
field_vec	4.8%	2.413	57.1%	0.0%	89.9%	100.0%	23.6%	0.0%	0.0%	0.0%
derivs_single_with_e_ele	3.0%	1.734	55.5%	0.0%	88.5%	100.0%	34.4%	0.0%	0.8%	0.0%
fld_vec_modulefield_follo	3.0%	1.189	34.9%	6.7%	74.0%	100.0%	73.0%	0.0%	0.9%	0.0%
bvec_interpol_vec	2.9%	1.131	38.8%	0.0%	91.2%	100.0%	36.2%	0.0%	0.0%	0.0%
pushe_single_vec	2.3%	1.943	43.9%	1.5%	71.3%	100.0%	54.7%	0.0%	1.1%	5.1%
i_interpol_ider0_aio_vec	1.8%	2.803	42.0%	0.1%	90.6%	0.0%	0.0%	0.0%	1.4%	0.0%



Thread

- Running
- CPU Time
- CPU Time

Python

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

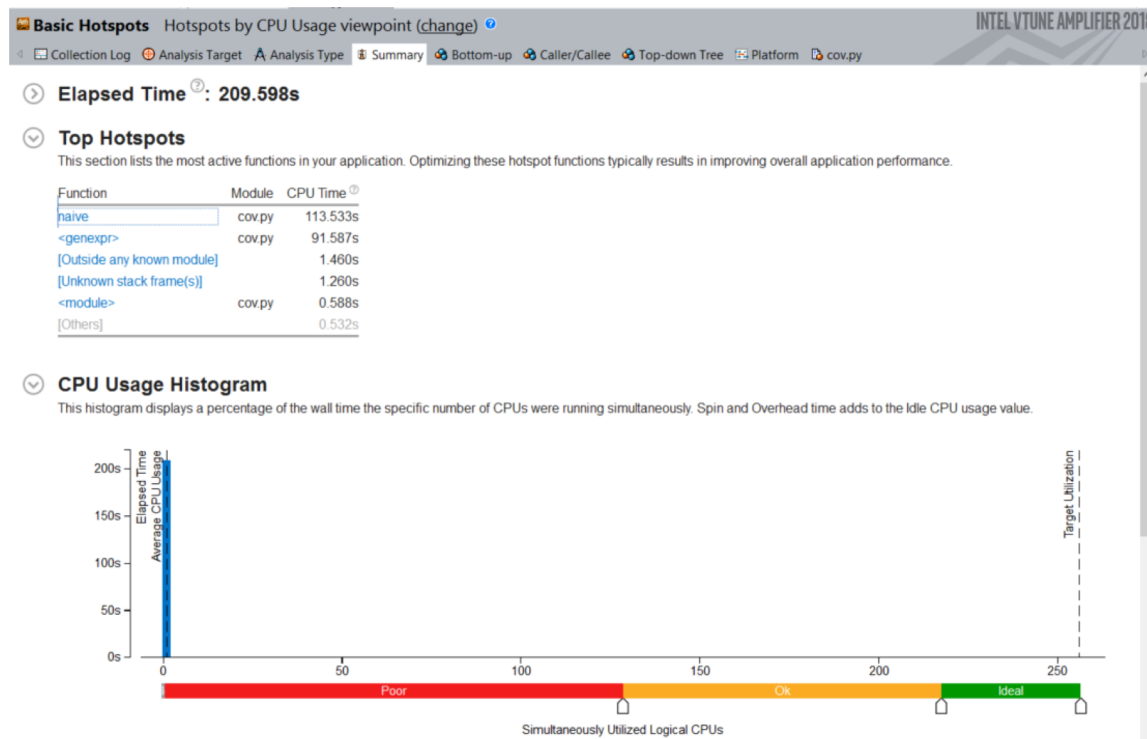
- The “application” should be the full path to the python interpreter used
- The python code should be passed as “arguments” to the “application”

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
```



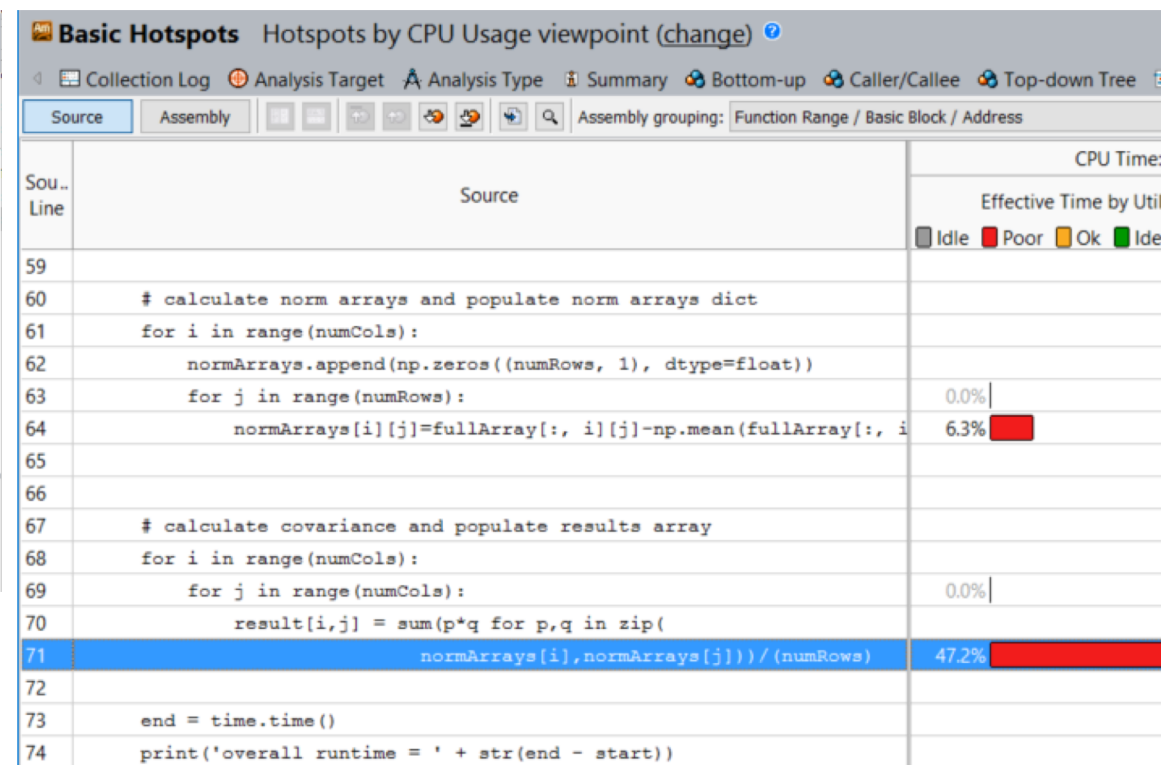
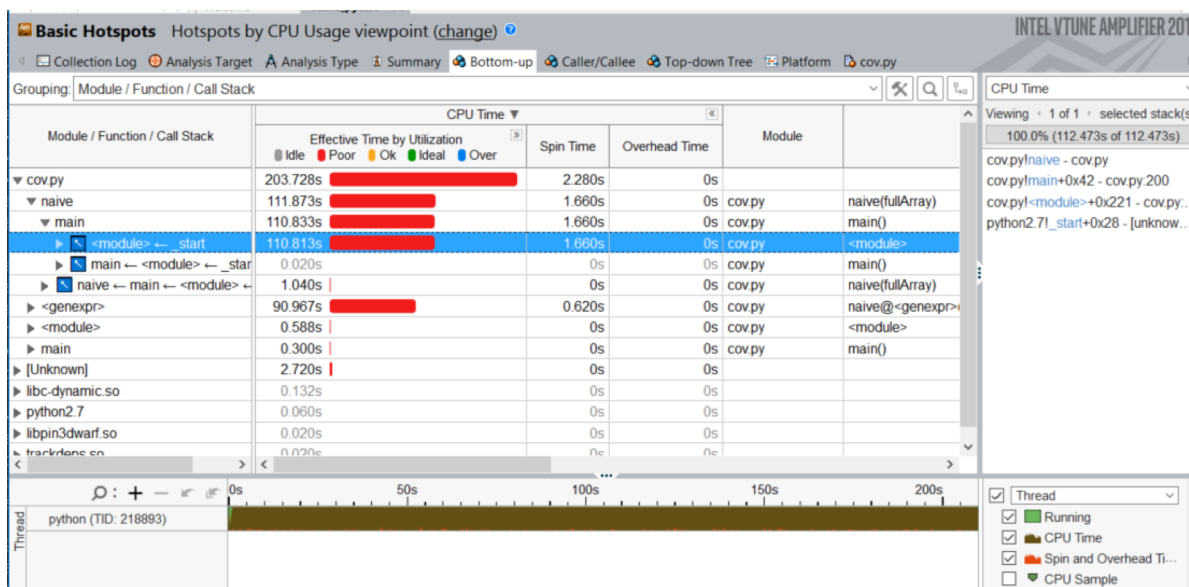
Naïve implementation of the calculation of a covariance matrix

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

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?
- Flow of execution (callstacks)
- What is my vectorization efficiency?
- Can I safely force vectorization?
- Inlining? Data type conversions?
- Roofline

Remember

Compile with `-g` and `-dynamic`

Profile 1 rank and small number of threads - `amplxe.qsub/advixe.qsub`

Advisor for big picture

Vtune for details

Resources

Product Pages

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

Detailed Articles

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

Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

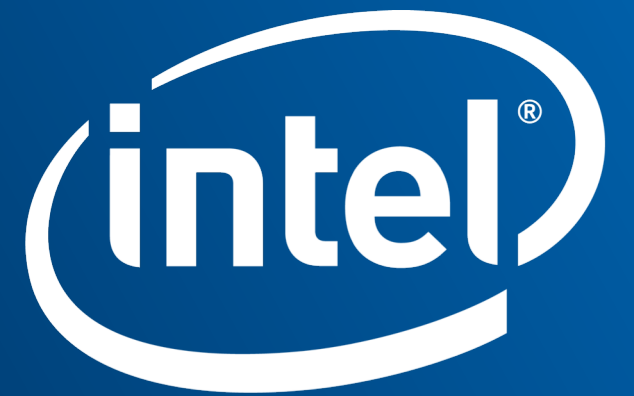
Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks.

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



Software

VTune Cheat Sheet

Compile with `-g -dynamic`

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

- `--result-dir=./vtune_output_dir`
- `--search-dir src:=../src --search-dir bin:=./`
- `-knob enable-stack-collection=true -knob collect-memory-bandwidth=false`
- `-knob analyze-openmp=true`
- `-finalization-mode=deferred` if finalization is taking too long on KNL
- `-data-limit=125` ← in mb
- `-trace-mpi` for MPI metrics on Theta
- `amplxe-cl -help collect survey`

Advisor Cheat Sheet

Compile with `-g -dynamic`

```
advixe-cl -c roofline/dependencies/map -flags -- ./executable
```

- `--project-dir=./advixe_output_dir`
- `--search-dir src:=../src --search-dir bin:=./`
- `-no-auto-finalize` if finalization is taking too long on KNL
- `--interval 1` (sample at 1ms interval, helps for profiling short runs)
- `-data-limit=125` ← in mb
- `advixe-cl -help`