# Boftware filing your application with Intel® Vtune<sup>TM</sup> Amplifier and Intel® Advisor

Paulius Velesko

### Tuning at Multiple Hardware Levels

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

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



# Intel® Compiler Reports

**FREE\*** performance metrics

# Compile with -qopt-report=5

- Which loops were vectorized
  - Vector Length
  - Estimated Gain
  - Alignment
  - Scatter/Gather

- Prefetching
- Issues preventing vectorization
- Inline reports
- Interprocedural optimizations
- Register Spills/Fills

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

#### 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<sup>™</sup> Amplifier's Application Performance Snapshot

High-level overview of application performance

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



## Usage on Theta

Launch all profiling jobs from /projects rather than /home

No module available, so setup the environment manually:

- \$ source /opt/intel/vtune\_amplifier/apsvars.sh
- \$ export PMI\_NO\_FORK=1

Launch your job in interactive or batch mode:

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

Produce text and html reports:



#### **APS HTML Report**

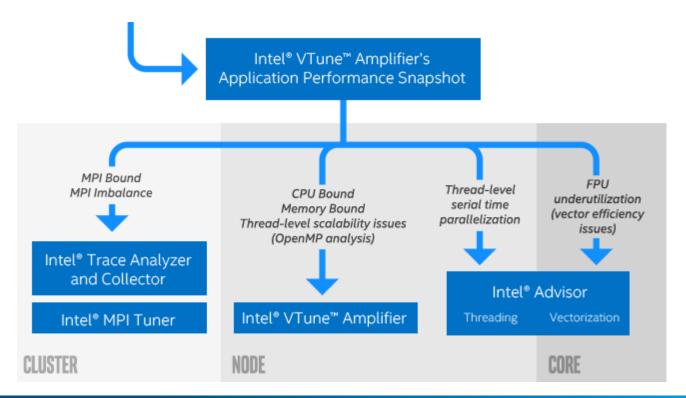
**Application Performance Snapshot** Application: heart\_demo Report creation date: 2017-08-01 12:08:48 Number of ranks: 144 Your application is MPI bound. Ranks per node: 18 This may be caused by high busy wait time inside the library (imbalance), non-OpenMP threads per rank: 2 optimal communication schema or MPI library settings. Use MPI profiling tools HW Platform: Intel(R) Xeon(R) Processor code named Broadwell-EP Logical Core Count per node: 72 like Intel® Trace Analyzer and Collector to explore performance bottlenecks. 121.39s Current.run Target Delta 53.74%▶ <10% MPI Time Elapsed Time OpenMP Imbalance 0.43% <10% Memory Stalls 14.70% <20% FPU Utilization 0.30% ► >50% 50.98 0.68 I/O Bound 0.00% <10% SP FLOPS CPI (MAX 0.81, MIN 0.65) **MPI** Time **OpenMP** Imbalance Memory Stalls FPU Utilization 53.74% ► of Elapsed Time 0.43% of Elapsed Time 14.70% of pipeline slots 0.30% (0.52s) (65.23s) Cache Stalls SP FLOPs per Cycle MPI Imbalance 12.84% of cycles 0.08 Out of 32.00 11.03% of Elapsed Time Memory Footprint DRAM Stalls Vector Capacity Usage (13.39s) Resident: 0.18% of cycles 25.84% TOP 5 MPI Functions % Per node NUMA FP Instruction Mix Waitall 37.35 Peak: 786.96 MB 31.79% of remote accesses % of Packed FP Instr.: 3.54% Average: 687.49 MB Isend 6.48 % of 128-bit: 3.54% Per rank: 5.52 Barrier % of 256-bit: 0.00% Peak: 127.62 MB % of Scalar FP Instr.: 96.46% Average: 38.19 MB Irecv 3.70 Virtual: 0.00 Scatterv FP Arith/Mem Rd Instr. Ratio Per node 0.07 Peak: 9173.34 MB Average: 9064.92 MB FP Arith/Mem Wr Instr. Ratio I/O Bound (intel) Per rank: 0.30 0.00% Peak: 566.52 MB (AVG 0.00, PEAK 0.00) Average: 503.61 MB

#### 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 Threading

#### Intel® Advisor

Modern HPC processors explore different level of parallelism:

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

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

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

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

# **Typical Vectorization Optimization Workflow**

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

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



# Cache-Aware Roofline

**FLOPS** 

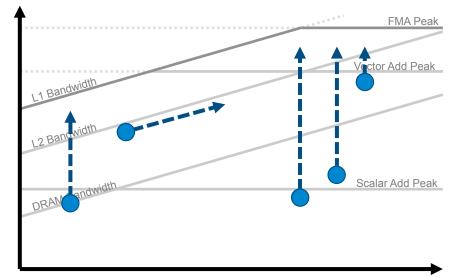
Next Steps

# 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 Al.

#### 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.

#### **Arithmetic Intensity**

#### 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.

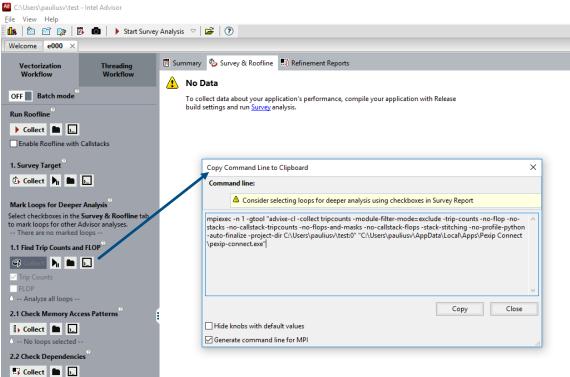
advixe-cl --snapshot --cache-sources --cache-binaries ./advixe\_res\_dir

Some things to note:

- Use /projects rather than /home for profiling jobs
- Set your environment:
  - \$ source /opt/intel/advisor/advixe-vars.sh
  - \$ export LD\_LIBRARY\_PATH=/opt/intel/advisor/lib64:\$LD\_LIBRARY\_PATH
  - \$ export PMI\_NO\_FORK=1



#### Using Intel® Advisor on Theta

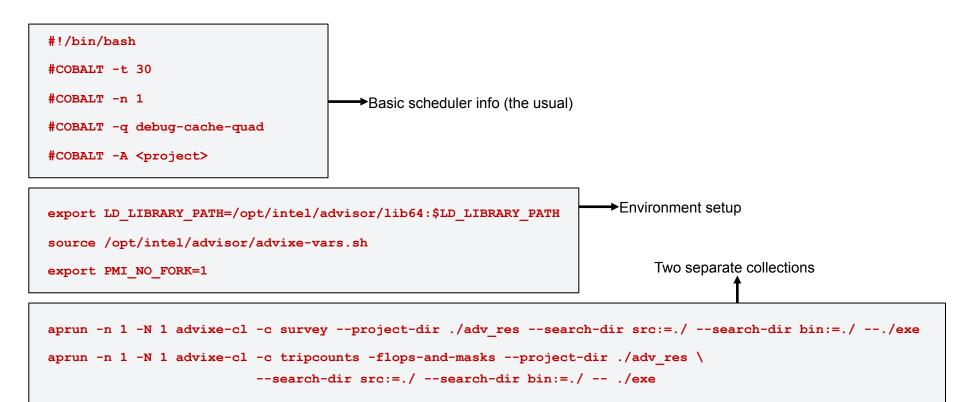


Optimization Notice Copyright © 2018, Intel Corporation. All rights reserved. \*Other names and brands may be claimed as the property of others.

O -- No loops selected --



#### Sample 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

# Nbody gravity simulation

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

Let's consider a distribution of point masses m\_1,...,m\_n located at r\_1,...,r\_n.

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

```
struct Particle
                                              for (i = 0; i < n; i++){
                                                                              // update acceleration
                                                 for (j = 0; j < n; j++)
 public:
                                                   real type distance, dx, dy, dz;
                                                   real type distanceSqr = 0.0;
   Particle() { init();}
   void init()
                                                   real type distanceInv = 0.0;
    pos[0] = 0.; pos[1] = 0.; pos[2] = 0.;
                                                   dx = particles[j].pos[0] - particles[i].pos[0];
     vel[0] = 0.; vel[1] = 0.; vel[2] = 0.;
                                                   •••
     acc[0] = 0.; acc[1] = 0.; acc[2] = 0.;
     mass = 0.;
                                                   distanceSqr = dx*dx + dy*dy + dz*dz + softeningSquared;
                                                   distanceInv = 1.0 / sqrt(distanceSqr);
   real type pos[3];
   real type vel[3];
                                                   particles[i].acc[0] += dx * G * particles[j].mass *
   real type acc[3];
                                                                      distanceInv * distanceInv * distanceInv;
   real type mass;
                                                   particles[i].acc[1] += ...
                                                   particles[i].acc[2] += ...
};
```

#### **Collect Roofline Data**

Starting with version 2 of the code we collect both survey and tripcounts data:
 export LD\_LIBRARY\_PATH=/opt/intel/advisor/lib64:\$LD\_LIBRARY\_PATH
 source /opt/intel/advisor/advixe-vars.sh
 export PMI NO FORK=1

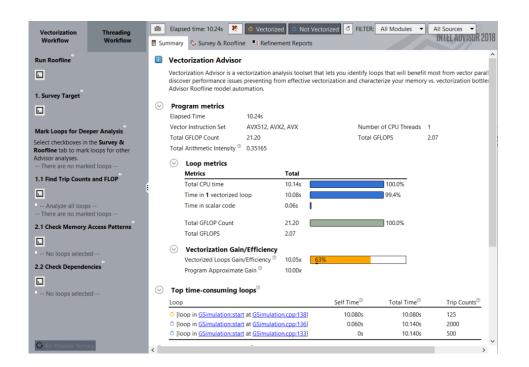
aprun -n 1 -N 1 advixe-cl --collect roofline --project-dir ./adv\_res --search-dir src:=./ \
--search-dir bin:=./ -- ./nbody.x

And generate a portable snapshot to analyze anywhere: advixe-cl --snapshot --project-dir ./adv\_res --pack --cache-sources \ --cache-binaries --search-dir src:=./ --search-dir bin:=./ -- nbody\_naive

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



# **Summary Report**



GUI left panel provides access to further tests

Summary provides overall performance characteristics

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

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

# Survey Report (Source)

			1	-	-							1523	FLOPS
+	- Function Call Sites and Loop	ips	Performance Issues	Self Time 🔻	Total Time	Туре	Why No Vectorization	?	orized Lo	· ·	in E 1		FLOPS Self GFLO
	[loop in GSimulation::start at (	GSimulation con:1381	© 2 Inefficient gather/sc	10.080	10.080	Vectorized (Body)			5 6		05x 1		2.093
	[loop in GSimulation::start at GS		© 1 Opportunity for outer I.		10.140s		inner loop was alrea			10.			1.700
	start			0.000sl	10.140s								
⊳ f	•			0.000sl	10.140s	Function							
∋f(	GSimulation::start			0.000s1	10.140s								
50	[loop in GSimulation::start at GS	Simulation.cpp:1331	@ 1 Data type conversions .	0.000sl	10.140s	Scalar	a inner loop was alrea	dv v					
cach	Top Down Code Analytics e_ee70d097069d33cb5b0f6e4		ion.cpp:138 GSimulation::star Source					Total Time	%	Loop/Functio	on Time	e %	Traits
	const double to = time.	start();	Source					Total Time	10	coop/runcin		. //	mana
Ħ	for (int s=1; s<=get_ns	steps(); ++s)											
	{												
-	<pre>ts0 += time.start();</pre>												
Ŧ	for (i = 0; i < n; i+	<pre>+)// update accel</pre>	eration										
	{ for (j = 0; j < n;												
•	<pre>for (j = 0; j &lt; n; 0 [loop in GSimulation::</pre>							0.100	5		10.080	Os 💻	
			loop processes Float32	. Tot 22. II	Tot22 data	tume (a) and inclu	udes 2-Source Der						
	No loop transformati		roop processes risecon	, 11002, 0.	LIIUUL GUUG	offic (b) and ther	adob E bodico roin						
L	(												
	real_type dx, dy, d	iz;											
	real_type distanceS	3qr = 0.0f;											
	real type distanceI	(ny = 0.0f)											
	real_type distance:												
	real_type distance:						Selected (Total Time):						

Inline information regarding loop characteristics

#### ISA used

. . .

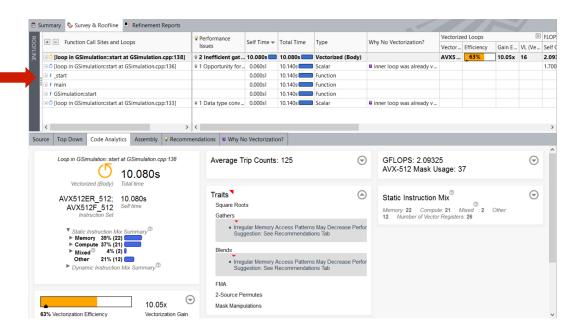
- Types processed
  - Compiler transformations applied
- Vector length used



# Survey Report (Code Analytics)

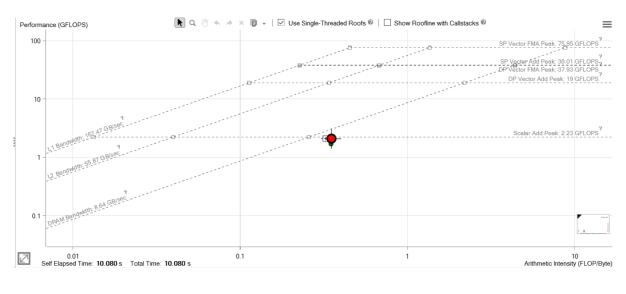
#### **Detailed loop information**

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





### **CARM** Analysis



Using single threaded roof

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

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

#### Next step is clear: perform a Memory Access Pattern analysis



#### Memory Access Pattern Analysis (Refinement)

#### 

Su	ımmai	ry 🗞 S	urvey & Roofli	ne 📲 Refine	ement Repo	rts 🚯 MA	AP Source: GS	imulation.c	ор						INTEL ADVISOR 20
ite l	ocatio	on		Loop-C	Carried Dep	endencies	Strides Dis	tribution	Access Pattern	Max. Site Footprint	Site Name	Recomme	ndations		
E [	loop ii	n start at	GSimulation.c	pp:1 No info	ormation ava	ailable	33% / 3	3% / 33%	Mixed strides	5KB	loop_site_1	@ 2 Ineffic	ient gather/	scatter instruct	ions present
Men	nory A	ccess Pa	tterns Report	Dependence	ies Report	♀ Recomn	nendations								
D		Stride	Туре		Source	1	Nested Funct	ion Vari	able references		Max. Site	Footprint	Modules	Site Name	Access Type
P1		10; 40	Constant st	tride	GSimulatio	n.cpp:144		bloc	k 0x60a0b0 alloca	ated at GSimulation.cpp	:109 4KB		nbody.x	loop_site_1	Read
1	42	2	eal type d	istanceInv	= 0.0f;										
	43														
	44		lx = partic					//1fl							
	45 46		ly = partic lz = partic					//1fl //1fl							
P2	M		Gather strid		GSimulatio		/			ated at GSimulation.cpp	:109 5KB		nbody.x	loop site 1	Read
-	42		eal type d					0.00		ited at opinionationicpp			noodyn	icop_one_r	
	43	-	.cur_cypc u	100411001117	0.02,										
	44		ix = partic					//1fl							
	45 46		ly = partic lz = partic					//1fl //1fl							
	10			e information			].pos[2];	//111	op				a la a al con	In an aite d	
P3						n.cpp:144							nbody.x	loop_site_1	
	42 43	1	eal_type d	istanceInv	= 0.0f;										
	44	c	ix = partic	les[j].pos	[0] - par	ticles[i	].pos[0];	//1fl	qo						
	45	c	ly = partic	les[j].pos	[1] - par	ticles[i	].pos[1];	//1fl	op						
1	46	ć	lz = partic	les[j].pos	[2] - par	ticles[i	].pos[2];	//1fl	op						
P5		0	Uniform str	ride	GSimulatio	n.cpp:149					4B		nbody.x	loop_site_1	Read
	47														
	48		listanceSqr						//6flops						
	49 50	c	listanceInv	= 1.0f /	sqrtf(dis	tanceSqr	);	//1d	iv+1sqrt						
	51	F	articles[i	].acc[0] +	= dx * G	* partic	les[j].ma:	ss * dist	anceInv * dis	stanceInv * distan	ceInv; /	/6flops			
_						-									

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

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

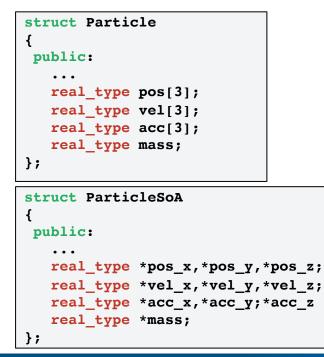
- 33% unit
- 33% uniform, non-unit
- 33% non-uniform

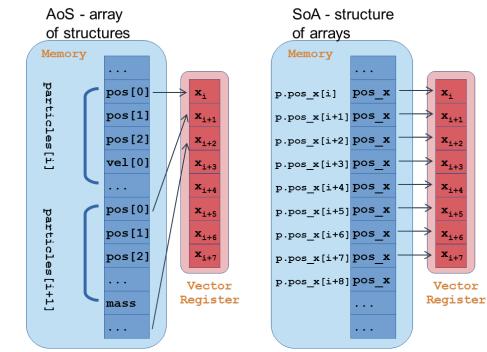
Re-structuring the code into a Structure Of Arrays (SOA) may lead to unit stride access and more effective vectorization



#### Vectorization: gather/scatter operation

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





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

+ -	Function Call Sites and Loops	Performance Issues	Self Time 🔻	Total Time	Туре	Why No Vectorization?		Vectorized Loops			FLOPS	_
									Gain E	VL (Ve	Self GFLOPS	
	oop in GSimulation::start at GSimulation.cpp:151]	· · ·					tor dependence pre	•			1.122	
	alar loop. Not vectorized: vector dep	No loop trans			or dependend	e prevent:	s vectorization					
u 🖱 [lo	pop in GSimulation::start at GSimulation.cpp:171]	9 1 Assumed depe	0.040s	0.040sl	Scalar	vec	tor dependence preve.				0.475	(
∍∮_st	tart		0.000sl	46.400s	Function							
🤉 🕈 ma	ain		0.000s1	46.400s	Function							
S F GS	Simulation::start >		0.000s1	46.400s	Function							>
	Scalar Total time						AVX-512 Mask	g				
	Scalar Total time 46.360s Self time	Memory: 8	ctruction N Compute: 1 r of Vector Re	11 Mixed :	11 Other:	$\odot$	Code Optimizations Compiler: Intel(R) C++ Intel(R) 64 Compiler for applications running on Intel(R) 64, Version: 18.0.0.128 Build 20170811					
	<ul> <li>✓ Static Instruction Mix Summary<sup>®</sup></li> <li>▶ Memory 24% (8)</li> <li>▶ Compute 32% (11)</li> <li>▶ Mixed<sup>®</sup> 32% (11)</li> <li>Cther 12% (4)</li> <li>▶ Dynamic Instruction Mix Summary<sup>®</sup></li> </ul>											

#### Next step is clear: perform a Dependencies analysis



26

#### **Dependencies Analysis (Refinement)**

#### 

												-						
Site Location Loop-Carried Dependencies						_						Recommendations						
🖻 [ d	oop in st	tart at GSimulatio	n.cpp:157]	RAW:4		No infe	ormation avail	lable N	o informatio	n available	No information available	e loop_site_1	© 1 Prov	I Proven (real) dependency present				
Mem	ory Acco	ess Patterns Repo	rt Depen	dencies Report	♀ Recomm	endations												
Probl	lems and	d Messages												Filter				
ID	● 1	Туре		Site Name	Sources		Modules	State						Severity				
P1	🔲 P	Parallel site information loop_site_1 GSimulat			GSimulation.c	op	nbody.x	✓ No	t a problem					Error	4 item			
P3	• R	Read after write dependency loop_site_1 GSimula		GSimulation.c	on.cpp nbody.x 🎙			N			Information	1 iten						
P4	+	Read after write dependency loop_site_1 GSimulation.cpp;					op nbody.x	R Ne						Туре				
P5		Read after write d			GSimulation.c		nbody.x R New							Parallel site information	1 iter			
P6	R	Read after write d	ependency	loop_site_1	GSimulation.c	op	nbody.x	Ne	N					Read after write depend.				
Read	after w	rite dependency	Code Loca	tions										Source				
ID	Instruct	tion Address	Description	Source	F	unction	Variable refer	rences	Module	State			^	GSimulation.cpp	5 iten			
X3	0x401c	:85	arallel site	🗄 GSimulati	ion.cpp:157 st	art			nbody.x	New				main.cpp	1 iter			
	55	real_type	distance	Inv = 0.0f;										Module				
	56 57	dx = parti	cles->po:	s x[j] - par	ticles->po	s x[i];	//1flop							nbody.x	5 item			
15	5.8		cles->po	s_y[j] - par	ticles->po	s_y[i];	//1flop //1flop							State				
15		and an increase of			ticles->po		register XMM		nbody.x	P New			_	New	4 item			
15	59	dz = parti		I GSimulati	ion con:164 ct									Mark a secolulars	1 iter			
15 15	59 0x401c	:b8, 0x401d17	lead		ion.cpp:164 st									Not a problem				
15 15 15	59 0x401c 62 63	b8, 0x401d17   distanceI	Read	f / sqrtf(di	stanceSqr)	;	//1di	v+lsq:	t					Not a problem	1110			
15 15 15	59 0x401c 62 63 64	b8, 0x401d17 distanceI	Read nv = 1.0: >acc_x[i]	f / sqrtf(di ] += dx * G	stanceSqr)	; s->mass[	//1di j] * dista	v+lsq: nceInv	t * distar		distanceInv; //6fl			Not a problem				
15 15 15 16 16 16	59 0x401c 62 63	b8,0x401d17 distanceI particles- particles-	Read nv = 1.0 >acc_x[i >acc_y[i	f / sqrtf(di ] <b>+= dx * G</b> ] <b>+=</b> dy * G	<pre>stanceSqr) * particle * particle</pre>	; <mark>s-&gt;mass[</mark> s->mass[	//1di j] * dista j] * dista	v+1sq; nceInv	t * distar * distar	ceInv *	distanceInv; //6fl distanceInv; //6fl distanceInv; //6fl	pa		Not a problem				
15 15 15 16 16 16 16	59 0x401c 62 63 64 65	b8,0x401d17 distanceI particles- particles- particles-	Read nv = 1.0 >acc_x[i >acc_y[i	f / sqrtf(di ] += dx * G ] += dy * G ] += dz * G	<pre>stanceSqr) * particle * particle</pre>	; <mark>s-&gt;mass[</mark> s->mass[ s->mass[	//1di j] * dista j] * dista	v+1sq; nceInv	t * distar * distar	ceInv *	distanceInv; //6fl	pa		Not a problem				
15 15 15 16 16 16 16 16 16 16 16	59 0x401c 62 63 64 65 66	b8, 0x401d17 distanceI particles- particles- particles- d1e	Read nv = 1.0: >acc_x[i >acc_y[i >acc_z[i Vrite	f / sqrtf(di ] += dx * G ] += dy * G ] += dz * G	<pre>* particle * particle * particle ion.cpp:164 st</pre>	; s->mass[ s->mass[ s->mass[ art	//1di j] * dista j] * dista	v+1sqr nceInv nceInv nceInv	* distar * distar * distar nbody.x	ceInv * ceInv *	distanceInv; //6fl	pa		Not a problem	1			

Dependencies analysis has high overhead:

- Run on reduced workload
- Advisor Findings:
- RAW dependency
- Multiple reduction-type dependencies

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:

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** 

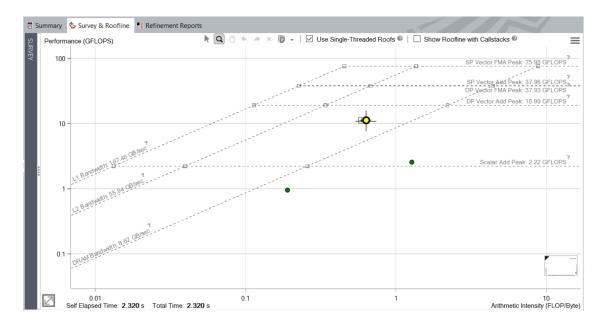
• If there is a reduction pattern dependency in the loop, enable vectorization using the directive #pragma omp simd reduction(operator:list). For example:

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

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.



# Intel® VTUNE™ Amplifier

Core-level hardware metrics

## Intel® VTune<sup>™</sup> 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



31

#### Predefined Collections

Many available analysis types:

- advanced-hotspots Advanced Hotspots
- Concurrency concurrency
  - **Disk Input and Output** disk-io

**GPU Hotspots** 

**Basic Hotspots** 

Locks and Waits

Memory Access

System Overview

Memory Consumption

General microarchitecture exploration

- general-exploration
- gpu-hotspots
- gpu-profiling **GPU In-kernel Profiling**
- hotspots

. . .

- hpc-performance
- locksandwaits
- memory-access
- memory-consumption
- system-overview

**HPC** Performance Characterization **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

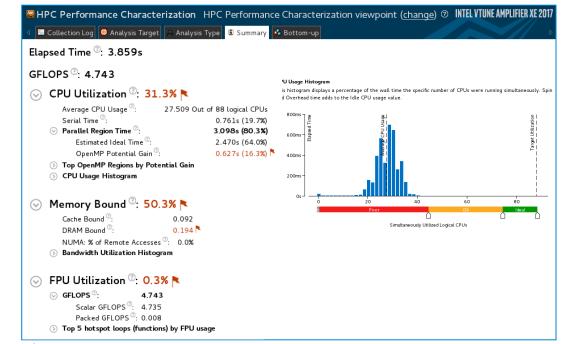
- Serial vs. Parallel time
- Top OpenMP regions by potential gain
- Tip: Use hotspot OpenMP region analysis for more detail

#### Memory Access Efficiency

- Stalls by memory hierarchy
- Bandwidth utilization
- Tip: Use Memory Access analysis

#### Vectorization: FPU Utilization

- FLOPS <sup>†</sup> estimates from sampling
- Tip: Use Intel Advisor for precise metrics and vectorization optimization



<sup>†</sup> For 3rd, 5th, 6th Generation Intel® Core<sup>™</sup> processors and second generation Intel® Xeon Phi<sup>™</sup> processor code named Knights Landing.

#### **Memory Access Analysis**

#### Tune data structures for performance

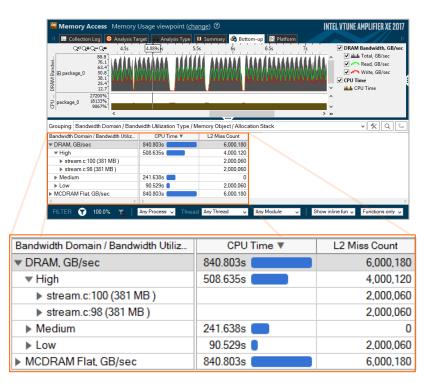
- Attribute cache misses to data structures (not just the code causing the miss)
- Support for custom memory allocators

#### **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<sup>®</sup> Xeon Phi<sup>™</sup> processor MCDRAM (high bandwidth memory) analysis



34

### Using Intel® VTune<sup>™</sup> Amplifier on Theta

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

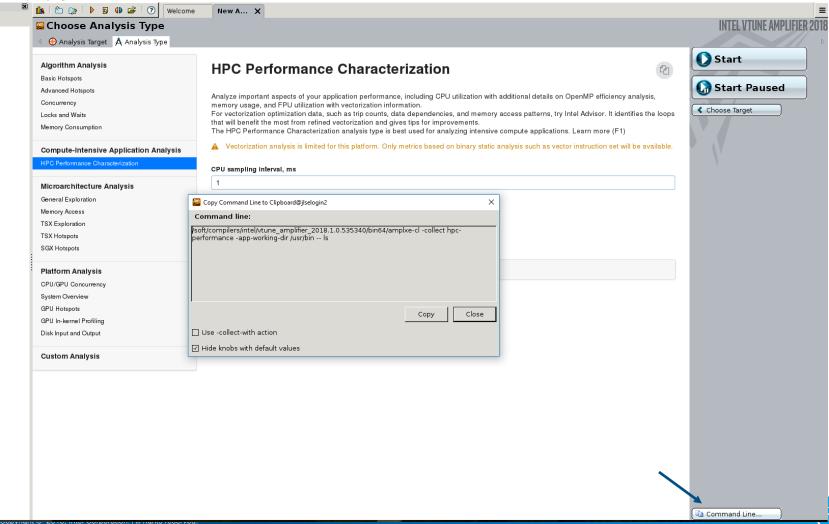
I will focus on the command line since it is better suited for batch execution, but the GUI provides the same capabilities in a user-friendly interface.

Some things of note:

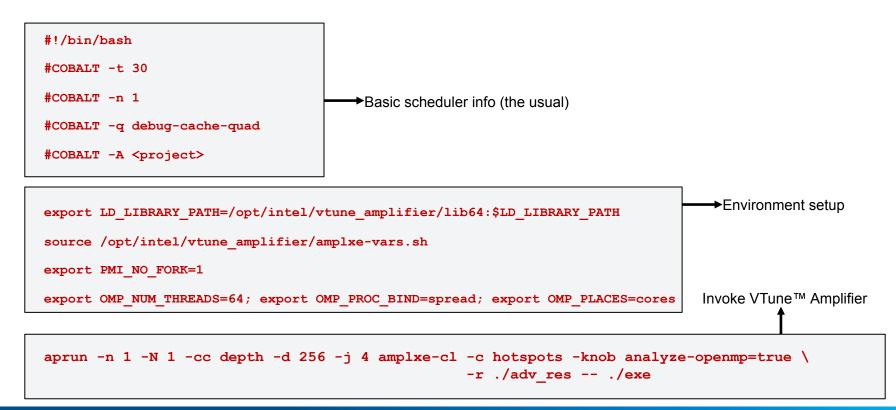
- Use /projects rather than /home for profiling jobs
- Set your environment:
- \$ source /opt/intel/vtune\_amplifier/amplxe-vars.sh
- \$ export LD\_LIBRARY\_PATH=/opt/intel/vtune\_amplifier/lib64:\$LD\_LIBRARY\_PATH
- \$ export PMI\_NO\_FORK=1



35

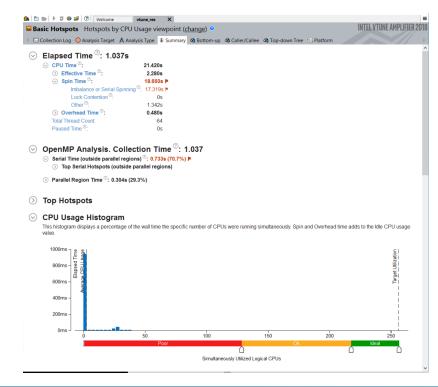


## Sample Script



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

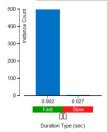
# Hotspots analysis for nbody demo (ver7: threaded)



#### 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: start\$omp\$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 a 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**

🗉 🗔 Collection Log  🕀 Analysis Target 🧍	🖎 Analysis Type 🗈 Summary 🛛 🐼 Bottom-up	Caller/Caller	e 🙈 Top-down Tree	🔁 Platform	Simulation	
Grouping: Module / Function / Call Stack					~ <b>%</b> Q %	+0 CPU Time
	CPU Time 🔻	1	*			Viewing < 1 of 1 → selected stack
Module / Function / Call Stack	Effective Time by Utilization	Spin Time »	Overhead Time	Module		100.0% (2.260s of 2.260s) nbody.x1GSimulation::start\$omp
libiomp5.so	0s	18.660s	0.320s			libiomp5.sol[OpenMP dispatche
r nbody.x	2.260s	0s	0.160s			libiomp5.so![OpenMP fork]+0x1
▶ GSimulation::start\$omp\$parallel_for@	2.260s	0s	0s	nbody.x	GSimulation::start\$omp\$	nbody.xIGSimulation::start+0x6
▶ GSimulation::start	0s	0s	0.160s	nbody.x	GSimulation::start(void)	nbody.x!main+0x86 - main.cpp:/
Unknown]	0.020s	0s	0s			nbody.x!_start+0x28 - start.S:11
	<					>
O         +         -         Image: Image	0.1s 0.2s 0.3s 0.4	s 0.5s	0.6s 0	.7s 0.8s	0.9s 1s	Ruler Area:

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.



### **Bottom-up HPC Characterization View**

		<u> </u>					INTEL VEHICLE AND LEFT
HPC Performance Characterization			ization view	point ( <u>chang</u>	<u>e)</u>		INTEL VTUNE AMPLIFIEF
🗉 ⊡ Collection Log   Onalysis Target 🔺 Analysis Type 👔	Summary 🛭 😪 Bot	ttom-up					
Grouping: Function / Call Stack							🔻 🔍 🖓 🖏 Elapsed Time: 48.882s
Function / Call Stack	Serial CPU Time	CPU Time	Back-En	d Bound	SIMD Instructions per Cycle	CPI Rate	CPU Utilization: 0.6% 🎙 📀
Punction / Gain Stack	Senal CFU Time	CFO TIMe	L2 Hit Bound	L2 Miss Bound	anno manuciona per cycle	OFINALE	Average CPU Usage: 1.490
[Loop at line 2928 in timestep_\$omp\$parallel@2513]	0.020s	8.840s	31.2%	7.4%	0.000	1.860	Average CFO Osage: 1.490
[Loop at line 5335 in gwce_new_\$omp\$parallel@5265]	0.000s	8.590s	95.4%	100.0%	0.365	1.691	Serial Time (outside paralle
[Loop at line 2473 in timestep_\$omp\$parallel_for@2473]	0s	8.530s	27.1%	0.0%	0.000	1.785	
svml_dpow_cout_rare	5.372s	5.372s	0.0%	0.0%	0.084	0.783	Parallel Region Time: 30.3
[Loop at line 7387 in mom_eqs_new_nc_\$omp\$parallel@6789	0s	4.992s 📃 🔤	59.3%	93.6%	0.355	2.004	CPU Usage Histogram
[Loop at line 55 in pjac_\$omp\$parallel_for@49]	0.030s	3.408s 🛑	43.2%	100.0%	0.227	2.858	
[Loop at line 7929 in mom_eqs_new_nc_\$omp\$parallel@6789	0s	2.616s 🛑	28.4%	54.9%	0.377	1.157	Back-End Bound: 60.9% of
[Loop at line 2557 in itpackv_mp_unscal_\$omp\$parallel_for@2	0.010s	2.526s 🛑	79.9%	54.0%	0.188	2.758	
svml_pow8_b3	1.624s	1.624s 📒	0.0%	0.0%	0.478	1.438	L2 Hit Bound:
[Loop at line 2459 in itpackv_mp_scal_\$omp\$parallel_for@245	0s	1.443s 📒	70.0%	100.0%	0.176	3.925	L2 Miss Bound:
[Loop at line 2025 in itpackv_mp_pmult_\$omp\$parallel_for@20	0.010s	1.353s 📒	82.4%	100.0%	0.156	2.561	Demand Misses:
[Loop at line 5659 in gwce_new_\$omp\$parallel@5265]	0s	1.253s 📒	30.3%	100.0%	0.334	1.160	HW Prefetcher:
[Loop at line 4105 in timestep_\$omp\$parallel_for@4105]	0.010s	1.042s 📒	30.5%	0.0%	0.535	0.856	MCDRAM Bandwidth Bound:
[Loop at line 5304 in gwce_new_\$omp\$parallel@5265]	0s	0.932s 📒	18.7%	0.0%	0.176	5.600	DRAM Bandwidth Bound:
[Loop at line 304 in timestep_\$omp\$parallel_for@304]	0s	0.862s 📒	12.6%	0.0%	0.069	1.106	Bandwidth Utilization Histo
[Loop at line 6824 in mom_eqs_new_nc_\$omp\$parallel@6789	0s	0.782s 📒	74.7%	0.0%	0.200	3.348	SIMD Instructions per Cycle:
[Loop@0x6463c0 in _f90_reduction_final_strided]	0.020s	0.702s 🔋	51.0%	0.0%	0.383	1.188	EP Instruction Mix:
[Loop at line 5945 in gwce_new]	0.521s	0.521s 💧	19.0%	0.0%	0.224	3.438	% of Packed SIMD Instr.
[Loop at line 5707 in gwce_new]	0.501s	0.501s 🔋	82.6%	0.0%	0.076	4.385	% of Scalar SIMD Instr.:
[Loop@0x6a70bd insvml_pow8_b3]	0.441s	0.441s	0.0%	0.0%	0.406	1.375	76 OF OCALAL ON IND HISLI.
[Loop at line 4279 in timestep]	0.441s	0.441s	0.0%	0.0%	0.357	0.962	
I oop at line 1455 in applv2dbottomfriction	0.421s	0.421s	24.3%	0.0%	0.186	1.433	
D: + -           05         55           Lump           55           CPU Time	105	15.		20s	255	305	355 405 455 Huter Area: □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
b package_0 0.675 C Loading	• •				· · ·		C CPU Time C me CPU
▶ package_0 10.645 ↓ Loading							✓ Total, GB/sec

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

## **Bottom-up HPC Characterization View - Thread**

💯 <no current project> - Intel VTune Amplifier@jlselogin2

💿 🕨 🗊 🕼 🚔 🕜 🛛 Welcome 🚯 🛾 🖄

vtune\_hpc\_... vtune\_... 🗙

#### HPC Performance Characterization HPC Performance Characterization viewpoint (change) @

🕘 🖾 Collection Log \varTheta Analysis Target 🙏 Analysis Type 🖞 Summary 🔗 Bottom-up

Grouping: OpenMP Region / Thread / Function / Call Stack								▼					
OpenMP Region / Thread / Function / Call Stack	Elapsed Time	Serial CPU Time		Op	enMP Potenti	al Gain		66	CPU Time	» Back-End Bound		SIMD Instructions per	
Openine Region / Thread / Function / Call Stack	Elapsed Time	Senai CPO Time	Imbalance	Lock Contention	Creation	Scheduling	Reduction	Atomics	CPOTIme	L2 Hit Bound	L2 Miss Bound	onvo marticiona per	
[Serial - outside parallel regions]	18.568s	18.202s							19.024s	14.5%	30.9%		
▼ gwce_new_\$omp\$parallel:2@unknown:5265:5673	6.316s	0s	0.259s	0.005s	0s	0.025s	0s	0s	11.887s	73.0%	95.4%		
OMP Master Thread #0 (TID: 183872)		0s							6.204s 🛑	70.3%	64.1%		
OMP Worker Thread #1 (TID: 184145)		0s							5.683s 🛑	76.0%	100.0%		
mom_eqs_new_nc_\$omp\$parallel:2@unknown:6789:7951	5.170s	0s	0.184s	0.020s	0s	0.045s	0.005s	0s	9.903s 📃 🔤	46.3%	61.3%		
timestep_\$omp\$parallel:2@unknown:2513:2966	4.814s	0.000s	0.166s	0s	0s	0s	0s	Os	9.201s	29.6%	7.0%		
timestep_\$omp\$parallel:2@unknown:2473:2502	4.863s	0s	0.161s	0s	0.005s	0.030s	0s	0s	9.151s 📃	25.4%	0.0%		
pjac_\$omp\$parallel:2@unknown:49:59	2.490s	0s	0.117s	0.005s	0.010s	0.035s	0s	0s	4.671s 🛑	37.2%	100.0%		
itpackv_mp_unscal_\$omp\$parallel:2@unknown:2552:2561	1.537s	0s	0.112s	0s	0.010s	0.010s	0s	0s	2.847s 📒	75.0%	50.7%		
timestep_\$omp\$parallel:2@unknown:4105:4134	0.960s	0s	0.061s	0s	0.010s	0s	0s	0s	1.664s 🏮	22.3%	0.0%		
itpackv_mp_scal_\$omp\$parallel:2@unknown:2454:2463	0.884s	0s	0.047s	0.005s	0s	0.005s	0s	0s	1.644s 🏮	59.1%	100.0%		
itpackv_mp_pmult_\$omp\$parallel:2@unknown:2019:2029	0.843s	0s	0.056s	0s	0s	0.015s	0s	0s	1.503s 🔋	68.7%	100.0%		
timestep_\$omp\$parallel:2@unknown:304:345	0.774s	0s	0.075s	0s	0s	0.005s	0s	0s	1.153s 🚦	13.6%	0.0%		
itpackv_mp_sdot_\$omp\$parallel:2@unknown:2950:2954	0.870s	0.000s	0.274s	0.005s	0.025s	0.040s	0.005s		1.093s 🚦	22.3%	50.2%		
mom_eqs_new_nc_\$omp\$parallel:2@unknown:7971:8045	0.392s	0s	0.037s	0s	0.005s	0.020s	0s	0s	0.692s	86.0%	100.0%		
itpackv_mp_unscal_\$omp\$parallel:2@unknown:2506:2540	0.203s	0.000s	0.019s	0s	0.005s	0.010s	0s	0s	0.371s	29.1%	0.0%		
itpackv_mp_itjcg_\$omp\$parallel:2@unknown:570:575	0.174s	0s	0.030s	0s	0s	0s	0s	0s	0.291s	100.0%	100.0%		
gwce_new_\$omp\$parallel:2@unknown:4639:4762	0.023s	0s	0.001s	0s	0s	Os	0.005s	0s	0.050s	0.0%	0.0%		

41

### **Bottom-up General Exploration**

<no current="" project=""> - Intel VTune Amplifier@jlselogin2</no>															- 0	' X
🏡 🖆 💿 🕨 🗿 🕼 🚔 🕜 🛛 Welcome 🔹 vtune_hpc vtur	ne_hpc vtun/	e X														=
General Exploration General Exploration viewpoint (change) @										/IER 2018						
🔄 🖽 Collection Log   Əhalysis Target Å Analysis Type 🗴 Summary	😪 Bottom-up	😪 Event Cour	nt 🙁 Platfo	ərm												Þ
Grouping: Source Function / Function / Call Stack															▼ %	, Q
		Front-End	Bound	«	»					Br	ack-En	nd Boun	ıd			A
Source Function / Function / Call Stack	ITLB Overhead	BACLEARS	MS Entry	ICache Line Fetch	Bad Speculation				Memory Latency	,						/
	THE OTHER		,			L1 Hit Rate	L2 Hit Rate	L2 Hit Bound	L2 Miss Bound	UTLB Overhead	SI	SI	C F		Split Loads §	Split Stor
▶ [Outside any loop]	4.5%	5.1%		31.8%	2.3%	93.0%	93.7%	14.5%	13.3%	3.2%			0	7.6%	2.3%	0.7
▶ [Loop at line 865 in _INTERNAL_25src_kmp_barrier_cpp_ce635	0.0%		6.7%		0.0%					0.0%			0		0.0%	0.0
▶ [Loop at line 5247 in gwce_new_\$omp\$parallel_for@5247]	0.0%	0.0%	0.0%			73.8%	96.0%	100.0%	86.1%			4		10.9%	0.0%	0.0
▶ [Loop at line 56 in pjac_\$omp\$parallel_for@49]	0.0%	0.0%	0.0%			83.3%		56.7%	0.0%			9		0.0%	25.9%	0.0
▶ [Loop at line 7339 in mom_eqs_new_nc_\$omp\$parallel_for@7339]	0.0%	0.0%	0.0%	4.3%	0.9%	50.0%	96.3%	100.0%	97.9%	40.9%	0	1	0	9.4%	0.0%	0.0
[Loop@0x6727bd in _f90_reduction_final_strided]	0.0%	0.0%	0.0%	2.0%	0.0%	94.4%	100.0%	34.2%	0.0%	6.0%	1	23	0	1.0%	0.0%	0.0
[Loop at line 2482 in timestep_\$omp\$parallel_for@2480]	0.0%	0.0%	0.0%	1.0%	0.0%	100.0%	100.0%	17.8%	0.0%	37.7%	0	0	0	1.0%	0.0%	0.0
▶ [Loop at line 2915 in timestep]	0.0%	0.0%	0.0%	0.0%	0.0%	100.0%	0.0%	0.0%	0.0%	14.0%	0	0	0	0.0%	0.0%	0.0
▶ [Loop at line 4082 in timestep]	0.0%	6.0%	0.0%	1.2%	7.2%	100.0%	100.0%	100.0%	0.0%	0.0%	1	7	0	4.8%	0.0%	0.0
▶ [Loop at line 55 in pjac_\$omp\$parallel_for@49]	0.0%	0.0%	0.0%	3.2%	0.0%	82.4%	100.0%	81.6%	0.0%	0.0%	1	6	0	3.2%	29.4%	0.0
▶ [Loop@0x6727a0 in _f90_reduction_final_strided]	0.0%	0.0%	0.0%	0.0%	0.0%	88.9%	100.0%	100.0%	0.0%	0.0%	1	4	0	2.0%	0.0%	0.0
▶ [Loop at line 1840 inkmp_fork_call]	0.0%	0.0%	0.0%	43.2%	0.0%	100.0%	100.0%	38.6%	0.0%	0.0%	0	0	0	6.8%	0.0%	0.0
▶ [Loop at line 1949 inkmp_join_barrier]	2.8%	0.0%	0.0%	45.1%	8.5%	80.0%	0.0%	0.0%	0.0%	0.0%	0	0	0	14.1%	0.0%	0.0
▶ [Loop at line 772 inkmpc_for_static_init_4]	8.3%	20.8%	12.5%	58.3%	4.2%	100.0%	0.0%	0.0%	0.0%	0.0%	1		0	20.8%	0.0%	0.0
▶ [Loop@0x6c0f46 in cvtas_x_to_a]	0.0%	44.4%	0.0%	44.4%	0.0%	100.0%	0.0%	0.0%	0.0%	0.0%	0	0	0	4.4%	0.0%	0.0
▶ [Loop@0x6808e1 in _f90_reduction_init_array]	0.0%	0.0%	0.0%	0.0%	0.0%	100.0%	0.0%	0.0%	0.0%	0.0%			0	0.0%	0.0%	0.0
▶ func@0x13a00	0.0%	27.8%	0.0%	5.6%	0.0%	100.0%	0.0%	0.0%	0.0%	0.0%	1		0	5.6%	0.0%	0.0
▶ [Loop at line 957 in chgcon]	0.0%	0.0%	0.0%	40.0%	0.0%	100.0%	0.0%	0.0%	0.0%	0.0%			0	5.7%	0.0%	0.0
▶ [Loop at line 1130 in _INTERNAL_22src_kmp_lock_cpp_7c9c5b	11.4%	0.0%	0.0%	11.4%	0.0%	100.0%	0.0%	0.0%	0.0%	0.0%	0	0	0	5.7%	0.0%	0.0
▶ [Loop at line 2122 in prbndx]	0.0%	0.0%	0.0%	0.0%	0.0%	100.0%	0.0%	0.0%	0.0%	0.0%	1		0	0.0%	28.6%	0.0
▶ [loon@0x380e0 in getenv]	0.0%	31.3%	0.0%	0.0%	12.5%	0,0%	100.0%	100.0%	0.0%	0.0%	0	0	0	0.0%	0.0%	0.0 ▼
	<u> </u>															



Profiling Python is straightforward in VTune<sup>™</sup> 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:



43

### Simple Python Example on Theta

### 

**INTEL VTUNE AMPLIFIER 2018** 

Basic Hotspots Hotspots by CPU Usage viewpoint (change)

🗉 🖸 Collection Log 😌 Analysis Target 🗍 Analysis Type 🔹 Summary 🐼 Bottom-up 🔞 Caller/Callee 🐼 Top-down Tree 🗈 Platform 🔓 cov.py

#### Elapsed Time<sup>®</sup>: 209.598s

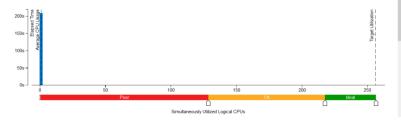
#### Solution → Solutio

This section lists the most active functions in your application. Optimizing these holspot functions typically results in improving overall application performance

Function	Module	CPU Time
naive	cov.py	113.533s
<genexpr></genexpr>	cov.py	91.587s
[Outside any known module]		1.460s
[Unknown stack frame(s)]		1.260s
<module></module>	cov.py	0.588s
[Others]		0.532s

#### ⊘ CPU Usage 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 usage value



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**

Basic Hotspots Hotspots b	y CPU Usage viewpoint ( <u>change</u> ) <b>e</b>					INTEL VTUNE AMPLIFIER 2018
Collection Log O Analysis Target	Å Analysis Type 🔹 Summary 🗞 Bottom-up	o 🗞 Caller/Ca	illee 🔌 Top-down	Tree 🖪 Platform	Cov.py	D
Grouping: Module / Function / Call Stack					~ 🛠 Q 🖁	CPU Time ~
	CPU Time 🔻		K			<ul> <li>Viewing &lt; 1 of 1 &gt; selected stack(s)</li> </ul>
Module / Function / Call Stack	Effective Time by Utilization	Spin Time	Overhead Time	Module		100.0% (112.473s of 112.473s) cov.py/naive - cov.py
▼ cov.py	203.728s	2.280s	0s			cov.pylmain+0x42 - cov.py:200
▼ naive	111.873s	1.660s	0s	cov.py	naive(fullArray)	cov.py! <module>+0x221 - cov.py:</module>
▼ main	110.833s	1.660s	0s	cov.py	main()	python2.7!_start+0x28 - [unknow
► <module> ← _start</module>	110.813s	1.660s		cov.py	<module></module>	
▶ 🔼 main ← <module> ← _star</module>	0.020s	0s	0s	cov.py	main()	1
► naive ← main ← <module> ←</module>	1.040s	0s	0s	cov.py	naive(fullArray)	
▶ <genexpr></genexpr>	90.967s	0.620s	0s	cov.py	naive@ <genexpr>i</genexpr>	
▶ <module></module>	0.588s	0s	0s	cov.py	<module></module>	
▶ main	0.300s	0s	0s	cov.py	main()	
[Unknown]	2.720s	0s	0s			
libc-dynamic.so	0.132s	0s	0s			
python2.7	0.060s	0s	0s			
libpin3dwarf.so	0.020s	0s	0s			
⊾ trankriane en <	0 020e	0e	Λe		>	~
,D:+- ⊭ ⊮ <sup>0s</sup>	50s	100s		150s	200s	Thread ~ ^
9 python (TID: 218893)						Running     B CPU Time     Spin and Overhead Ti…     CPU Sample

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

Source	Assembly 🗉 🔄 💿 🧐 🇐 🚱 🖉 Assembly grouping: Function Range / Basic	Block / Address
Sou.		CPU Time:
Line	Source	Effective Time by Utili
		🔲 Idle 📕 Poor 📙 Ok 📕 Idea
59		
60	# calculate norm arrays and populate norm arrays dict	
61	for i in range(numCols):	
62	<pre>normArrays.append(np.zeros((numRows, 1), dtype=float))</pre>	
63	for j in range(numRows):	0.0%
64	<pre>normArrays[i][j]=fullArray[:, i][j]-np.mean(fullArray[:, i]</pre>	6.3%
65		
66		
67	# calculate covariance and populate results array	
68	for i in range(numCols):	
69	for j in range(numCols):	0.0%
70	result[i,j] = sum(p*q for p,q in zip(	
71	<pre>normArrays[i],normArrays[j]))/(numRows)</pre>	47.2%
72		
73	<pre>end = time.time()</pre>	
74	<pre>print('overall runtime = ' + str(end - start))</pre>	

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.

## **Useful Options on Theta**

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

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

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

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

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



### Resources

### **Product Pages**

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

**Detailed Articles** 

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



# 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

## **EMON** Collection

General Exploration analysis may be performed using EMON

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

### export AMPLXE\_EXPERIMENTAL=emon

aprun [...] amplxe-cl -c general-exploration -knob summary-mode=true[...]

### VTune Cheat Sheet

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

- --result-dir=./vtune\_output\_dir
- --search-dir src:=../src --search-dir bin:=./
- -knob enable-stack-collection=true -knob collect-memorybandwidth=false
- -knob analyze-openmp=true
- -finalization-mode=deferred
- -data-limit=125 🗲 in mb
- -trace-mpi

#### **Optimization Notice**

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



### **Advisor Cheat Sheet**

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

- --project-dir=./advixe\_output\_dir
- --search-dir src:=../src --search-dir bin:=./
- -no-auto-finalize
- --interval 1
- -data-limit=125 ← in mb

#### Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved. \*Other names and brands may be claimed as the property of others. https://software.intel.com/en-us/advisor-help-lincommand-line-interface-reference



Profiling a single rank (for a 4 node, 256 rank job) profile1.sh: mpirun -n 1 \ #!/bin/bash amplxe-cl -c hotspots \ export PE RANK=\$ALPS APP PE -- ./exe \ export PMI NO FORK=1 : -n 255 ./exe if [ "\$PE RANK" == 0];then \$1 -- \$2 else \$2 fi aprun -n 256 -N 64 profile1.sh "amplxe-cl -c hotspots" "exe" Intel + Cray(Theta)

Intel (JLSE/BEBOP)

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