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 <u>www.intel.com/benchmarks</u>.

Copyright © 2020, 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





CPU & GPU Performance analysis Vtune & Advisor

Paulius Velesko

paulius.velesko@intel.com

Application Engineer

Intel[®] Software Development Tools for Tuning

- Compiler Optimization Reports Key to identify issues preventing automated optimization
- Intel[®] VTune[™] Application Performance Snapshot Overall performance
- Intel[®] Advisor Core and socket performance (vectorization and threading)
- Intel[®] *VTune*[™] Profiler Node level performance (memory and more)
- Intel[®] Trace Analyzer and Collector Cluster level performance (network)

Get the tools

Intel profiling tools are now FREE

Current Parallel Studio Tools:

https://software.intel.com/en-us/vtune/choose-download

https://software.intel.com/en-us/advisor/choose-download





Next-Gen OneAPI Tools:

https://software.intel.com/content/www/us/en/develop/tools/oneapi.html



Agenda

- Vtune
 - CPU Architecture Performance Analysis
 - GPU Profiling
- Advisor
 - CPU Vectorization
 - GPU Roofline
 - Offload Advisor



Intel[®] Advisor

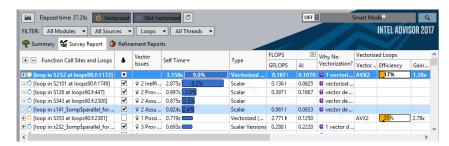
Intel[®] Advisor – Vectorization Optimization

Faster Vectorization Optimization:

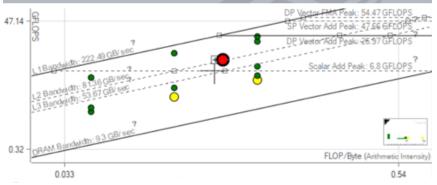
- Vectorize where it will pay off most
- Quickly ID what is blocking vectorization
- Tips for effective vectorization
- Safely force compiler vectorization
- Optimize memory stride

Roofline model analysis:

- Automatically generate roofline model
- Evaluate current performance
- Identify boundedness



INTEL ADVISOR



http://intel.ly/advisor-xe

Add Parallelism with Less Effort, Less Risk and More Impact

Optimization Notice



Typical Vectorization Optimization Workflow

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

- 1. Collect survey (overhead ~5%) advixe-cl -c survey
 - Basic info (static analysis) ISA, time spent, etc.
- 2. Collect Tripcounts and Flops (overhead 1-10x) advixe-cl -c tripcounts -flop
 - Investigate application place within roofline model
 - Determine vectorization efficiency and opportunities for improvement
- 3. Collect dependencies (overhead 5-1000x) advixe-cl -c dependencies
 - Differentiate between real and assumed issues blocking vectorization
- 4. Collect Memory Access Patterns advixe-cl -c map



Survey

Starting point for all Advisor analyses

- Where is time being spent? ٠
- What is vectorized?
- **Issues preventing Vectorization** ۲

RUER All Sources Loops And Functions All Threads Initial Sources Loops And Functions All Threads Initial Sources	٩	0	٩,
Function Call Sites and Loops Function Call Sites and Performance Total Time Set Time Verter Total Time Verter	ISOR B	OR I	2 BI
			
■0 Opon nc (J, and, J, Sompfa) 1 1 Potential under: 10211a Vectorized flooty P 5522 3552 181a 2 ■0 Opon nc (J, and, J, Sompfa) 2 Unpertised Tab. 2 Stol 5504 5504 5504 5604	35	3	T
■0 Opon nc (J, and, J, Sompfa) 1 1 Potential under: 10211a Vectorized flooty P 5522 3552 181a 2 ■0 Opon nc (J, and, J, Sompfa) 2 Unpertised Tab. 2 Stol 5504 5504 5504 5604	/e		
EPI Dipopis nod Langle ang, shamp & 2 Unoptimized II., 65404 65404 Vectorized Body P. 5522 532 143x 2 Tail Boopin nod Langle, ang, shamp 4 30910 Vectorized Body P. 1 vectorized Cody P. 1 vectorized Cody P. 1 vectorized Cody P. Tail Boopin nod Langle, ang, shamp 4 30910 Vectorized Cody P. 1 vectorized Cody P. 2 vectorized Cody P. 1 vectorized Cody P. 2 vectorized Vectorized Cody P. 2 vectorized Cody P. 2 vectorized Vectorized Vectorized Cody P. 2 vectorized Vec			T
It is loopin coll, avails avails avail av			
Image: State Stat			1
I Praction I Praction Identify			1
I Bicub-Kokkon:Device +Kokkon:Device +Kokkon:Device +Kokkon:Device +Kokkon:Device +Kokkon:Device +Kokkon:Device +Kokkon:Device +Kokkon:Device +Kokkon:Device + Kokkon:Device + Kokkon:Dev	1		1
c > c currel Top Down Code Analytics Assembly Recommendations If Why No Vectorization? Consult: Expensions Dop vectorization possible but seems inefficient Case: The compiler vectorized determined the loop will not benefit from vectorization. Common reasons include: • Non-unit stride memory access • Indirect memory access			
Come Code Analytics Assembly Recommendations Why No Vectorization? Come/or Disensities Opposition Why No Vectorization? Come/or Secondary come/or Disensities The complete vectorized determined the loop will not benefit from vectorization. Common reasons include: Non-unit stride memory access • Non-unit stride memory access • Indirect memory access •			
Comment: Desemble Second Secon	>	>	>
oop vectorization possible but seems inefficient Cause: The compiler vectorizer determined the loop will not benefit from vectorization. Common reasons include: • Non-unit stride memory access • Indirect memory access			
oop vectorization possible but seems inefficient Cause: The compiler vectorizer determined the loop will not benefit from vectorization. Common reasons include: • Non-unit stride memory access • Indirect memory access			
Cause: The compiler vectorizer determined the loop will not benefit from vectorization. Common reasons include: • Non-unit stride memory access • Indirect memory access			
Cause: The compiler vectorizer determined the loop will not benefit from vectorization. Common reasons include: Non-unit stride memory access Indirect memory access			L
Non-unit stride memory access Indirect memory access			I.
Indirect memory access			
Low iteration count			
ue detection is in progress[0%]			

Program metrics

Elapsed Time

Number of CPU Threads 1

78.39s Vector Instruction Set AVX2, AVX, SSE2, SSE

⊘ Performance characteristics

Metrics	Total	K MKL details 🤒
Total CPU time	76.91s	100%
Time in 31 vectorized loops	51.65s	67.2%
Time in scalar code	25.26s	32.8%
Sectorization Gain/Effic Vectorized Loops Gain/Efficiency Program Approximate Gain	iency [®] 1.80x 1.53x	~89%

Optimization Notice

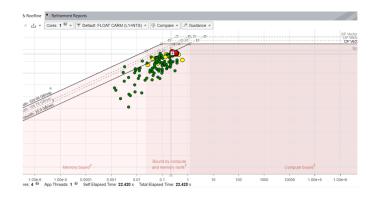


Tripcounts & Flops

Can be run after Survey is complete

- Loop Tripcounts
- Flops
- Memory Traffic
- Arithmetic Intensity
- <u>Roofline</u>

Function Call Sites and	Compute Perform	nance		
+ - Loops	Self GFLOPS	Total GFLOPS	Self Al	Total AI
∃ [loop in col_f_e_and_d_m_\$o	r 4.728 📖	4.728	0.348	0.348
∃ [©] [loop in col_f_e_and_d_s_\$om	¢ 6.067	6.067	0.250	0.250
∃ [©] [loop in col_f_angle_avg_m_\$	4.559	4.559	0.328	0.328
Ioop in col_f_angle_avg_m_\$	0.0001	0	0	0
80 [loop in ellip_agm_v at ellipti	3.917 🔲	3.917	0.150	0.150
J PetscCheckPointer	0.0001	0	0	0
Iloop in col_f_e_and_d_s_\$om	c 0.499 0	5.460	0.027	0.231
Bicub < Kokkos::Device < Kokko	s 2.508 🔲	2.314	0.188	0.188
Iloop in ellip_agm at elliptics.	F 1.314 🛙	1.314	0.615	0.615
Ioop in col_f_angle_avg_m_\$	3.768 📖	3.768	0.437	0.438
Iloop in col_f_angle_avg_s_\$o	r 2.349 🔲	1.762	0.132	0.182
Ioop in col_f_angle_avg_m_\$	0.0001	0	0	0
< >	<			



Optimization Notice

Dependencies

Runs through your code tracking operations between memory objects

- Identifies loop carried dependencies
- Allows to safely force vectorization
 - #pragma omp simd
 - #pragma ivdep
- High overhead
 - Run on smallest possible input

Welco	ome	snapshot000000 (r	ead-only)	testnopack0	00 (read-or	nly) ×		
FILTER	· · ·		Vectorized	d 0 Not Vect	orized 🦉			INTEL ADVISOR B
🗒 Sur	nmary	survey & Ro	ofline 📲	Refinement Rep	orts			
Site Lo	ocation	ı				Loop-Carri	ed Dependencies	Strides Distri
🖹 [lo	op in	MPIDI_OFI_mpi_ini	t_hook at of	i_init.c:1034]		No Depe	endencies Found	No Informati
🖻 [lo	oop in	MPIDI_OFI_mpi_ini	t_hook at of	i_init.c:1034]		@ RAW:1	A WAW:1	No Informati
🖹 [lo	oop in	_GLOBALsub_l_a	cl_platform.c	pp at libalterad	l.so:0x1e0c	No Informa	ation Available	0% / 100% /
۵ II د ۲	on in	M disnose at has	ic string h/31	1281		ANo Dene	andencies Found	No Strides Fo
Memo	ory Ac	cess Patterns Repo	rt Depend	dencies Report	♀ Recomm	nendations		
Proble	ems ai	nd Messages					Filter	
ID	۹	Туре		Site Name	Sources	^	Severity	
P1	Δ	Inconsistent lock	use	loop_site_13	[Unknown];	ofi_init.e	Error	2 items
P5	1	Parallel site inform	mation	loop_site_13	ofi_init.c		Information	1 item
P16	٥	Read after write o	lependency	loop_site_13	[Unknown];	ofi_init.c	Warning	1 item
P17	٥	Write after write of	dependency	loop_site_13	[Unknown];	ofi_init.c ~	Туре	
×		the design of the later					Inconsistent lock	use 1 item
		t lock use: Code L			Course	0	Parallel site inform	mati 1 item
ID		ction Address	Description		Source		Read after write of	depe 1 item
	0x7f1	-	Parallel site		🖹 libfabr		Write after write	depe 1 item
X2 X3)7, 0x1737	Read		Cofi_init		Source	
	UX1/3	35, 0x1746, 0x1716	write		🖹 ofi_init	.c:1034		
X4	0xf94		Circle a second	lock defined	🖹 ofi init	-1024 V		lame 🔰

Optimization Notice

Memory Access Patterns

Observes memory object access patterns in loops

- Classify loops based on pattern
 - Unit Stride
 - Constant Stride
 - Random Access
- Stride size for constant stride

	<u> </u>	sed time: 5.14			Not Vecto	orized 🍯			INTEL ADVISOR	DET
FILTER:	All	Modules	 All S 	Sources 🔻					INTELADVISON	DEN
🗵 Sum	mary	v 🏷 Survey	& Roofli	ne 🏾 🖫 Refinem	nent Repo	orts				
Site Loo	atior	1					Strides Dist	tribution	Loop-Carried Dep	ie (
🖹 [loc	op in	tcf_0 at liba	alteracl.sc	o:0x5f260]			0% / 1	00% / 0%	No Information Av	ve.
🖹 [loc	op in '	fi_ini at libfal	bric.so.1:()x7f10]			52% /	6 <mark>%</mark> / 42%	No Information Av	Vê
🖺 [loo	op in	getCurrentD	SODir[ab	i:cxx11] at libsyc	cl.so.1:0x1	aec30]	33% <mark>/</mark>	1 <mark>3% / 53%</mark>	No Dependenci	е
🖹 [loc	op in '	initialize at li	ibsycl.so.1	I:0x143df0]			No Strides	Found	No Information Av	νê
🖹 [loc	op in	ofi_getifadd	rs at libtc	p-fi.so:0xd2cf]			No Strides	Found	No Information Av	1ĉ
مما اھ <	on in a	queue at que	eue.hpp:6	91			No Informa	ation Available	❷No Dependencie >	P.
Memo	ry Ac	cess Patterns	Report	Dependencies	Report	💡 Recomm	nendations			
ID	•	Stride	Туре		Source		Nested Fu	unction 👻	Variable reference	e
P16.	44 1	41; 49; 80	Variable	stride	libstdc+	+.so.6:0xfb5.	sentry		block 0x24e40a0,	,ī
P16.	44	0; 14	Variable	stride	libstdc+	+.so.6:0xfb5.	sentry			
P86		0	Uniform s	stride	libsycl.sc	o.1:0x5fb70	ignore		_GLOBAL_OFFSET	ĉ
P88		0	Uniform s	stride	libsycl.sc	o.1:0x602f0	ignore		_GLOBAL_OFFSET	í_
P16.	deb;	41; 49; 88	Variable	stride	libstdc+	+.so.6:0xaac	ignore		block 0x24e40a0,	, Ì
P16.	dely.	41; 49; 80	Variable	stride	libstdc+	+.so.6:0xaa	. ignore		block 0x24e40a0,	, Ì
P16.	dely.	41; 114	Variable	stride	libstdc+-	+.so.6:0x10	M_extrac	t_int <unsigne< td=""><td>block 0x24e40a0,</td><td>,</td></unsigne<>	block 0x24e40a0,	,
P87		0	Uniform s	stride	libsycl.sc	o.1:0x602e0	_M_extrac	t <unsigned lo.<="" td=""><td>GLOBAL_OFFSET</td><td>í_</td></unsigned>	GLOBAL_OFFSET	í_
P5		4	Constant	stride	libc.so.6:	0x18a540	[Unknown	1]	block 0x3178cb0	
P6		4	Constant	stride	libc.so.6:	0x18a544	[Unknown	1]	block 0x3178cb0	
<									>	Ĵ

Optimization Notice

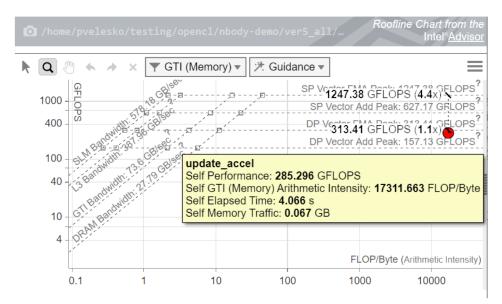
GPU Roofline

Intel Advisor can collect GPU GFLOPs

- Report kernel arithmetic intensity
- GFLOPs
- Generate roofline

Not yet integrated into Advisor GUI

Reports saved as HTML files



Intel[®] VTUNE[™] Amplifier

Intel[®] VTune[™] Amplifier

VTune Amplifier is a full system profiler

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

Memory Access

Concurrency

GPU In-kernel Profiling

GPU Hotspots

Basic Hotspots

Locks and Waits

- hpc-performance HPC Performance Characterization
- memory-access
- disk-io
 Disk Input and Output
- concurrency
- gpu-hotspots
- gpu-profiling
- hotspots
- Iocksandwaits
- memory-consumption Memory Consumption
- system-overview System Overview

Python Support (SW Sampling)



Optimization Notice

...

Copyright © 2020, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Analyze Performance Over Time

Observe individual performance-affecting aspects over time

Slice and dice your data

- HW Event Thread
- Source Line Thread Time Slice

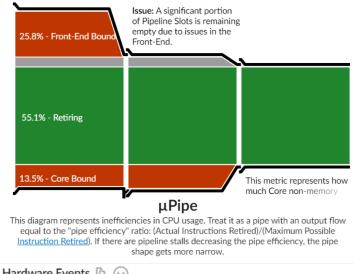
_																			
	D: = ⊮ ⊮	0s	0.2s	.	0.4s	0.6s		0.8s	1s	1.2s		.4s	1.6s	1.8s		2s	2.2s	2.4s	
ead	OMP Worker Thread #6 (TID																		
Type	UOPS_EXECUTED.THREAD		يبلي	يش	11	1	1	ينف ا		يد ا	يىلى	ىيار	يش	, hu	ي ال	ш.	يل ا	4	
ent T	INST_RETIRED.ANY							A is die 44-44-44-48-			utur, dan dan di i				tion through a to a				
re Event	INST_RETIRED.PREC_DIST			, hu	يىلى	يىلى	يىلى	ينغر	all i	تنفر	تنار	1	11	44	л и	ш	11		di.
	FP_ARITH_INST_RETIRED.2		ا بار	. Lu	. Lu	يبلي	ي ال	فعادر	1	4	4	4	ш.	. L.	ينا.	li i		1	4
Hardwa	UOPS_ISSUED.ANY	M	ينار	ي ال	ي ال	للله	illi,	فعادر	4	, 1 1	ي ا	يبلي	يىلى	ين ا	ي ال	1	1	14	
	UOPS_RETIRED.RETIRE_SL	M	يد أن	, du	يبل ا	.	ندل ا	14	<i>.</i>	, L	1	, L u	يل ا	<u>بار</u>	ulu.	1	11		
	IDQ.DSB_UOPS	يبلي	يش	يىلى	ي ال	يل ا		ىلى 1	يل ا	يبلى	سلى	ي أن	للله	1	<u>نان</u>	ш,	يىلى	ينار	
	CPU_CLK_UNHALTED.REF				t to take to be to be		******			i di da i di d							i alalahi ahi		
		14444																	

Optimization Notice

Explore HW Counter Data

Vtune uses Intel's SEP driver to read CPU HW counters

- uArch Exploration
 - Most coverage, time multiplexing
 - Classify functions/loops
 - Front-End Bound
 - Back-End Bound
 - Retiring



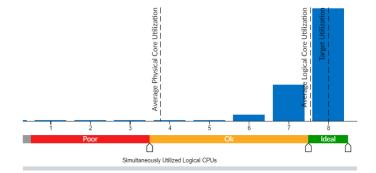
Hardware Events 🗈 🌝	
Hardware Event Type	Hardware Event Count
CPU_CLK_UNHALTED.REF_TSC	65,383,500,000
CPU_CLK_UNHALTED.REF_XCLK	429,212,876
CPU_CLK_UNHALTED.THREAD	65,359,000,000
CPU_CLK_UNHALTED.THREAD_P	62,466,093,699
CYCLE_ACTIVITY.STALLS_MEM_ANY	11,194,016,791
EXE_ACTIVITY.1_PORTS_UTIL	23,606,035,409
EXE_ACTIVITY.2_PORTS_UTIL	18,212,027,318

Optimization Notice

Analyze OpenMP Performance

Analyze performance of individual OpenMP Loops

- Evaluate workload-to-thread ratio
- Overhead Analysis
 - Load imbalance
 - Thread creation
 - Reductions
 - atomics
- GFLOPs per loop/function



D: 🕇 🗕 🗲 🖉	248ms 250ms
Worker Thread #1 (TID	
• Worker Thread #6 (TID	
• Master Thread #0 (TID:	
Worker Thread #2 (TID	
Worker Thread #4 (TID	
Worker Thread #5 (TID	
Worker Thread #7 (TID	
Worker Thread #3 (TID	

Optimization Notice

GPU Offload Profiling

Coarse-grain gpu-offload profile

- Analyze the utilization of GPU
 - Time spent on CPU vs GPU
 - Explore opportunities for asynchronous execution
 - Best for
 - Finding new offload opportunities
 - Balanced view of CPU and GPU work

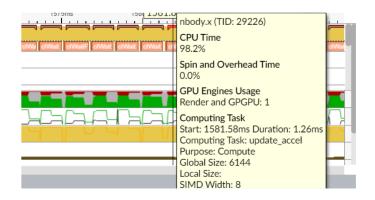
Recommendations

GPU Utilization:28.8%

GPU utilization is low. Switch to the **Bottom-up view** for in-depth analysis of host activity. Poor GPU utilization can prevent the application from offloading effectively.

EU Array Stalled/Idle:47.4%

GPU metrics detect some kernel issues. Use **GPU Compute/Media** Hotspots (preview) to understand how well your application runs on the specified hardware.



Optimization Notice

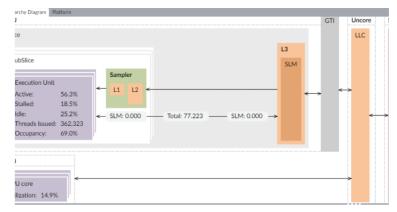
GPU Execution Profiling

Memory Hierarchy Diagram

 Feeds and speeds between all memory layers

HW Event Counters

- EU Occupancy
- EU Stall rate
- IPC
- Mem BW
- SIMD Utilization





Optimization Notice

Source-Assembly View

Identify performance bottlenecks down to a single line of code

Quickly extract assembly for JIT

- OpenMP Target
- SYCL/DPC++
- OpenCL

So	urce Assembly 💵 = 🍻 🏘 🗛 🏘 Assembly grouping: Addre	ess				
	Source	🖕 GPU Instructions Executed		Sourc	Assembly	GPU Instructions Execute
		Control Flow Send Int3	0x50		Block 2:	
108	ifdef MAXTHREADS		0x50	109	(W) mov (1 M0) r127.1<1>:ud r125.0<0;1,0>:ud	384,0
109	for (int i = item.get id()[0] + start; i < end; i+=tota	1,652,000	0x60	109	(W) mov (1 M0) r125.0<1>:ud 0x1:ud	384,0
10	else		0x70		(W) and (1 MO) r125.0<1>:ud sr0.2<0;1,0>:ud r125.	125,0
111	auto i = item.get_id() + start;		0x80	114	(W) cbit (1 M0) r125.0<1>:ud r125.0<0;1,0>:ud	125,0
12	endif		0x90	115	(W) mov (1 M0) r124.0<1>:ud r127.0<0;1,0>:ud	125,0
13	1		0xa0	114	(W) send (1 MO) null r124 0xC 0x4009709 [Data Cac	125,0
14	real type ax i = particles acc x[i];	250.000	0xb0	115	(W) mov (1 MO) r124.0<1>:ud r127.2<0;1,0>:ud	125,0
15	real type ay i = particles acc y[i];	250.000	0xc0	116	(W) mov (1 MO) r125.0<1>:ud r127.1<0;1,0>:ud	125,0
16	real type az i = particles acc z[i];	125,000	0xd0	117	(W) mov (1 M0) r127.2<1>:ud r124.0<0;1,0>:ud	125,0
117	for (int $i = 0; i < n; i++)$ (750,250,000	0xe0		(W) mov (1 MO) r127.1<1>:ud r125.0<0;1,0>:ud	125,0
18	real_type dx, dy, dz;		0xf0		(W) mov (1 M0) r125.0<1>:ud 0xFF:ud	125,0
19	real type distanceSgr = 0.0f;		0x100	117	(W) and (1 MO) r125.0<1>:ud sr0.2<0;1,0>:ud r125.	125,0
120	real type distanceInv = 0.0f;		0x110		(W) cbit (1 M0) r125.0<1>:ud r125.0<0;1,0>:ud	125,0
121			0x120		(W) mov (1 M0) r124.0<1>:ud r127.0<0;1,0>:ud	125,0
122	<pre>dx = particles pos x[j] - particles pos x[i]; /</pre>	750.000.000	0x130		(W) add (1 MO) r124.0<1>:ud r124.0<0;1,0>:ud 0x4:	125,0
23	dy = particles pos y[j] - particles pos y[i]; /	750.000.000	0x140	122	(W) send (1 M0) null r124 0xC 0x4009709 [Data Cac	250,000,0
24	dz = particles pos z[i] - particles pos z[i]; /	750.000.000	0x150	117	(W) mov (1 M0) r124.0<1>:ud r127.2<0;1,0>:ud	250,000,0
25	ar - barororofbog_r()) barororofbog_r(r)) /	730,000,000	0x160	123	(W) mov (1 M0) r125.0<1>:ud r127.1<0;1,0>:ud	250,000,0
126	distanceSgr = dx*dx + dv*dv + dz*dz + softeningSguare	750.000.000	0x170	122	(W) mov (1 M0) r127.2<1>:ud r124.0<0;1,0>:ud	250,000,0
27	distanceInv = 1.0f / sgrt(distanceSgr); //	7 30,000,000	0x180	124	(W) mov (1 M0) r127.1<1>:ud r125.0<0;1,0>:ud	250,000,0
28	disenceine = 1.01 / sqc(disencesql)/ //		0x190	117	(W) mov (1 M0) r125.0<1>:ud 0x1:ud	250,000,0
129	ax i += dx * G * particles mass[j] * distanceInv *	1.000.000.000	0x1a0	123	(W) and (1 MO) r125.0<1>:ud sr0.2<0;1,0>:ud r125.0	250,000,0
130	ay i += dy * G * particles mass[j] * distanceInv *	750.000.000	0x1b0	122	(W) cbit (1 M0) r125.0<1>:ud r125.0<0;1,0>:ud	250,000,0
.30	az i += dz * G * particles mass[j] * distanceInv *	500.000.000	0x1c0	124	(W) mov (1 M0) r124.0<1>:ud r127.0<0;1,0>:ud	250,000,0
.32	I and a comparatores mass()] . discurrently .	500,000,000	0x1d0	129	(W) add (1 MO) r124.0<1>:ud r124.0<0;1,0>:ud 0x8:	250,000,0
.33	particles_acc_x[i] = ax_i;	125.000	0x1e0	123	(W) send (1 M0) null r124 0xC 0x4009709 [Data Cach	250,000,0
.34	particles acc y[i] = ay i;	125,000	0x1f0	124	(W) mov (1 M0) r124.0<1>:ud r127.2<0;1,0>:ud	250,000,0
135	particles acc z[i] = az i;	125,000	0x200	130	(W) mov (1 M0) r125.0<1>:ud r127.1<0;1,0>:ud	250,000,0
135) // end of max threads loop or just scope	125,000	0x210	129	(W) mov (1 M0) r127.2<1>:ud r124.0<0;1,0>:ud	250.000.0

Optimization Notice

Compiling Applications for Profiling

Compile DPC++ Applications with :

-g -gline-tables-only -fdebug-info-for-profiling

All other:

-g



CLI and GUI Profiling

- vtune -c uarch-exploration \
- -result-dir uarch_exp_01 \
- -knob \

collect-memory-bandwith=true \

- -trace-mpi \
- -start-paused \
- -resume-after=1.5 $\$
- -- binary.x arg1 arg2



Analyze CPU microarchitecture bottlenecks affecting the performance of your application. This analysis type is based on the hardware event-based sampling collection. <u>Learn more</u>

- Cannot enable Hardware Event-based Sampling due to a problem with the driver (sep*/sepdrv*). Check that the driver is running and the driver group is in the current user group list. See the "Sampling Drivers" help topic for further details.
- 😢 To collect hardware events, run the product as administrator.

Retry

CPU sampling interval, ms

1

Extend granularity for the top-level metrics:

- Front-End Bound
- Bad Speculation
- Memory Bound

Optimization Notice

OpenCL vs Level Zero

Level Zero support is still under development - for now use OpenCL

OpenMP Target

export LIBOMPTARGET_PLUGIN=OPENCL
export LIBOMPTARGET_PLUGIN=LEVEL0

• DPC++

export SYCL_BE=PI_OPENCL
export SYCL_BE=PI_LEVEL0

Support Aspect	DPC++ application with OpenCL as back end	DPC++ application with Level Zero as back end
Operating System	Linux OS Windows OS	Linux OS only
Data collection	VTune Profiler collects and shows GPU computing tasks and the GPU computing queue.	VTune Profiler collects and shows GPU computing tasks and the GPU computing queue.
Data display	VTune Profiler maps the collected GPU HW metrics to specific kernels and displays them on a diagram.	VTune Profiler maps the collected GPU HW metrics to specific kernels and displays them on a diagram.
Display Host side API calls	Yes	Yes
Source Assembler for computing tasks	Can drill down to Level Zero computing tasks using Source Assembler.	Unavailable

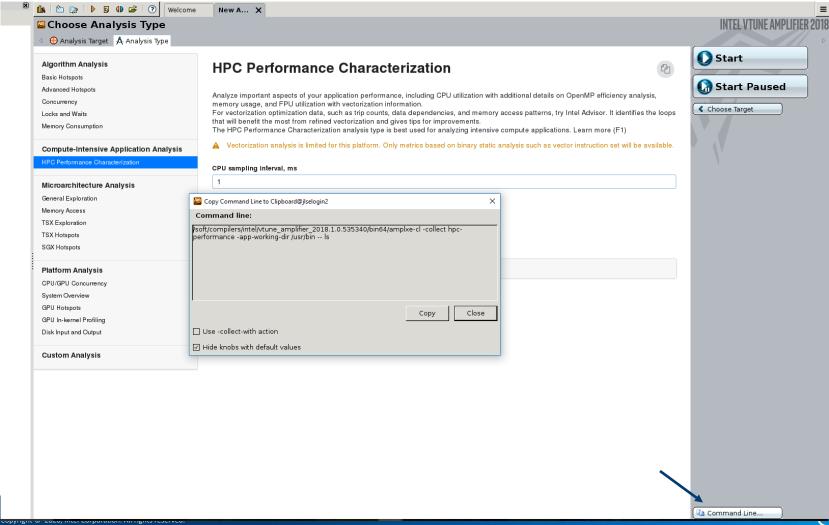
Vtune hands on

Collect uarch-exploration

cd /projects/intel/pvelesko/nody-demo/ver7 vim Makefile # edit to add -dynamic cp /soft/perftools/intel/advisor/amplxe.qsub ./ vim amplxe.qsub # edit collection to "uarch-exploration" qsub ./advixe.qsub ./nbody.x 2000 500

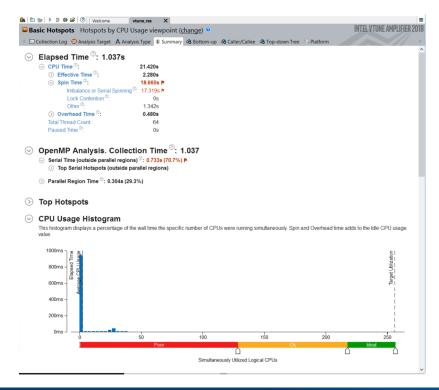
scp result back to your local machine





Hotspots analysis for nbody demo (ver7: threaded)

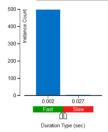
qsub amplxe.qsub ./your_exe ./inputs/inp



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: 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



Bottom-up Hotspots view

	A Analysis Type 🗈 Summary 🗟 Bottom-up						
rouping: Module / Function / Call Stack					~ 🕺	Q 2+0	CPU Time
	CPU Time		×				Viewing < 1 of 1 → selected stac
Module / Function / Call Stack	Effective Time by Utilization	Spin Time 🏾	Overhead Time	Module			100.0% (2.260s of 2.260s) nbody.x!GSimulation::start\$omp
libiomp5.so	0s	18.660s	0.320s				libiomp5.so![OpenMP dispatch
nbody.x	2.260s	0s	0.160s				libiomp5.so![OpenMP fork]+0x
GSimulation::start\$omp\$parallel_for	🧟 2.260s 🚬 📕	0s	0s	nbody.x	GSimulation::sta	art\$omp\$pa	nbody.x!GSimulation::start+0x6
 GSimulation::start 	Os	0s	0.160s	nbody.x	GSimulation::sta	art(void)	nbody.x!main+0x86 - main.cpp
[Unknown]	0.020s	0s	0s				nbody.x!_start+0x28 - start.S:1
	<					>	
OMP Master Thread #0 (TID OMP Worker Thread #60 (TI OMP Worker Thread #66 (TI							
OMP Worker Thread #60 (TI							
OMP Worker Thread #60 (TI OMP Worker Thread #56 (TI OMP Worker Thread #50 (TI OMP Worker Thread #55 (TI OMP Worker Thread #54 (TI							
OMP Worker Thread #60 (TI OMP Worker Thread #56 (TI OMP Worker Thread #50 (TI OMP Worker Thread #55 (TI OMP Worker Thread #54 (TI OMP Worker Thread #49 (TI							OpenMP Barrier- to-Barrier Segment Thread Manning m CPU Tme Spin and Overhea. CPU Sample
OMP Worker Thread #80 (T OMP Worker Thread #56 (T OMP Worker Thread #56 (T OMP Worker Thread #56 (T OMP Worker Thread #49 (T OMP Worker Thread #49 (T OMP Worker Thread #58 (T							OpenMP Barrier- to-Barrier Segment Thread Manning m CPU Tme Spin and Overhea. CPU Sample
OMP Worker Thread #80 (TL OMP Worker Thread #56 (TL OMP Worker Thread #55 (TL OMP Worker Thread #55 (TL OMP Worker Thread #56 (TL OMP Worker Thread #58 (TL							OpenMP Barrier- to-Barrier Segment Thread Manning Mac CPU Tme Application CPU Tme CPU Sample
OMP Worker Thread #80 (TL OMP Worker Thread #56 (TL OMP Worker Thread #55 (TL OMP Worker Thread #55 (TL OMP Worker Thread #54 (TL OMP Worker Thread #54 (TL OMP Worker Thread #59 (TL OMP Worker Thread #59 (TL OMP Worker Thread #59 (TL OMP Worker Thread #51 (TL							OpenMP Barrier- to-Barrier Segment Thread Manning Mac CPU Tme Application CPU Tme CPU Sample
OMP Worker Thread #50 (TL. OMP Worker Thread #56 (TL. OMP Worker Thread #56 (TL. OMP Worker Thread #55 (TL. OMP Worker Thread #54 (TL. OMP Worker Thread #58 (TL. OMP Worker Thread #58 (TL. OMP Worker Thread #58 (TL. OMP Worker Thread #59 (TL. OMP Worker Thread #51 (TL. OMP Worker Thread #52 (TL. OMP Worker Thread #52 (TL. OMP Worker Thread #52 (TL.							OpenMP Barrier- to-Barrier Segment Thread Manning Mac CPU Tme Application CPU Tme CPU Sample
OMP Worker Thread #80 (TL OMP Worker Thread #56 (TL OMP Worker Thread #55 (TL OMP Worker Thread #56 (TL OMP Worker Thread #54 (TL OMP Worker Thread #58 (TL OMP Worker Thread #59 (TL OMP Worker Thread #50 (TL OMP Worker Thread #50 (TL OMP Worker Thread #51 (TL OMP Worker Thread #51 (TL OMP Worker Thread #51 (TL							OpenMP Barrier- to-Barrier Segment Thread Manning m CPU Tme Spin and Overhea. CPU Sample

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.

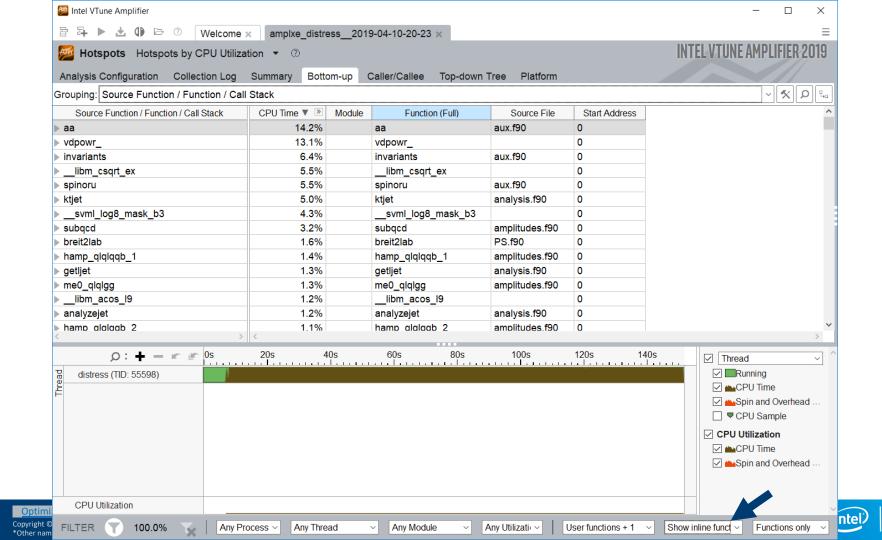
Next steps would include additional analysis to continue the optimization process.

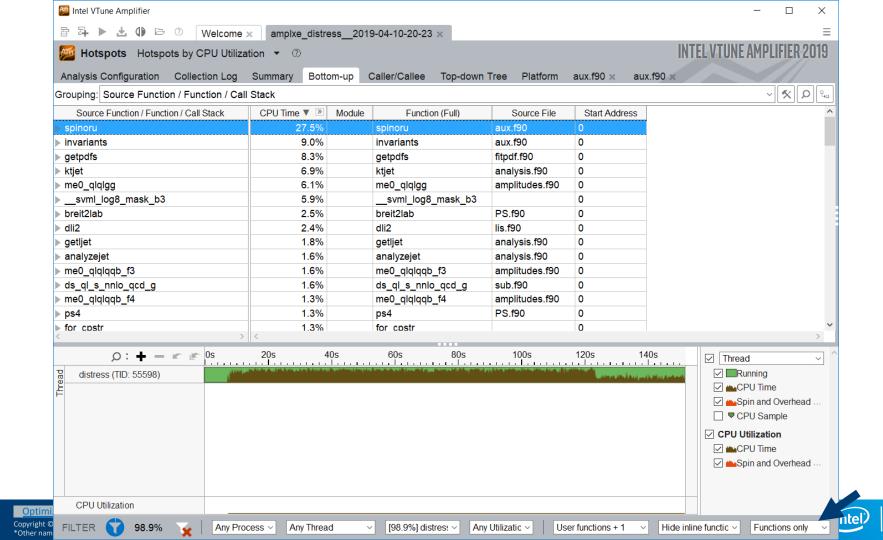
Optimization Notice

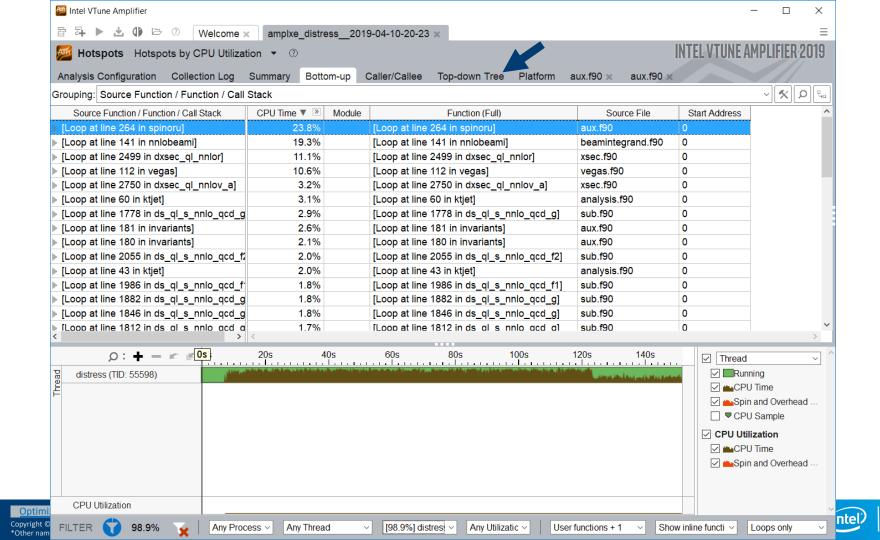
Mintel VTune Amplifier					- 🗆 ×
답 🕂 🕨 📩 🕩 🗁 🕐 Welcome »	amplxe_distr	ess2019-04-10-20-23	3 ×		=
Motspots Hotspots by CPU Utilizat	ion 🔻			INTEL VTUN	E AMPLIFIER 2019
Analysis Configuration Collection Log	Summary Bett	om-up Caller/Callee	Top-down Tree Platform		
	Summary Boll	om-up Caller/Callee	Top-down Tree Platform		
Grouping: Function / Call Stack			1		<u>~ ≪ ₽ </u> ₽
Function / Call Stack	CPU Time 🔻 🔌	Module	Function (Full)	Source File	Start Address
vdpowr_		libmkl_intel_lp64.so	vdpowr_		0x695310
▶ aa		distress	aa	aux.f90	0x41ec1c
▶ aa		distress	aa	aux.f90	0x41ec9a
invariants		distress	invariants	aux.f90	0x41d550
▶libm_csqrt_ex		libimf.so	libm_csqrt_ex		0xc7a50
spinoru		distress	spinoru	aux.f90	0x41e9e0
▶ ktjet		distress	ktjet	analysis.f90	0x420ae0
svml_log8_mask_b3		distress	svml_log8_mask_b3		0x532f50
▶ breit2lab		distress	breit2lab	PS.f90	0x4602d0
▶ getljet		distress	getljet	analysis.f90	0x421830
▶ me0_qlqlgg		distress	me0_qlqlgg	amplitudes.f90	0x4408d0
▶libm_acos_l9		libimf.so	libm_acos_l9		0xedd80
▶ analyzejet		distress	analyzejet	analysis.f90	0x422050
▶ ds_ql_s_nnlo_qcd_g		distress	ds_ql_s_nnlo_qcd_g	sub.f90	0x4694e0
scart	1.384s	libimf.so	csart		0x1d430
	20s	40s 60s	80s 100s 120s 140		and
distress (TID: 55598)					Running CPU Time
F					Spin and Overhead
					CPU Sample
					Utilization
					CPU Time
					Spin and Overhead
CPU Utilization					
FILTER 7 100.0%					
FILTER 📉 100.0% 🐷 Any Pro	cess ~ Any Thre	ead 🗸 🗸 Any Modu	ule V Any Utilizati V User functions + 1 V	Show inline funct	 Functions only

Hotspots Hotspots by CPU Utiliza	tion 🔻 🕐				INTEL	VI UNE AMPLIFIER 2019
Analysis Configuration Collect Log	Summary Botte	om-up Caller/Callee	Top-down Tree Platfo	orm		
Grouping: Function / Call Stack						~ <u>≮</u> ₽ ⊑
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		distress	invariants	aux.f90	0x41d550	
▶libm_csqrt_ex		libimf.so	libm_csqrt_ex		0xc7a50	
▶ spinoru		distress	spinoru	aux.f90	0x41e9e0	
▶ ktjet		distress	ktjet	analysis.f90	0x420ae0	
svml_log8_mask_b3		distress	svml_log8_mask_b3		0x532f50	
▶ breit2lab		distress	breit2lab	PS.f90	0x4602d0	
▶ getljet		distress	getljet	analysis.f90	0x421830	
▶ me0_qlqlgg		distress	me0_qlqlgg	amplitudes.f90	0x4408d0	
▶libm_acos_l9		libimf.so	_libm_acos_l9		0xedd80	
▶ analyzejet		distress	analyzejet	analysis.f90	0x422050	
▶ ds_ql_s_nnlo_qcd_g		distress	ds_ql_s_nnlo_qcd_g	sub.f90	0x4694e0	
▶ csart	< 1.0%	libimf.so	csart		0x1d430	>
	200	100 600	900 1000	120-2	1400	
	205 4		80s 100s	1205	1405	✓ Thread ✓
distress (TID: 55598)						Running
Ē						CPU Time
						Spin and Overhead
						CPU Sample
						CPU Utilization
						🗹 💼 CPU Time
						Spin and Overhead

- 32







금 목 ▶ ★ ① ▷ ② Welcome >	ampl ¹ a .stress_	_2019-04-10-20-23	×				Ξ
Motspots Hotspots by CPU Utilizat	ion 🔻 🕜				INTEL VT	UNE AMPLIFIEF	12019
Analysis Configuration Collection Log	Summary Bottom-u	o Caller/Callee	Top-down Tree	Platform aux.f90 × aux.f9	0 x		
Grouping: Call Stack						~	0.*
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	1
[Loop at line 100 in vegas]	99.6%	0s	distress	[Loop at line 100 in vegas]	vegas.f90	0x4162c8	1
▼ [Loop at line 112 in vegas]	99.6%	1.531s	distress	[Loop at line 112 in vegas]	vegas.f90	0x416641	
	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 spinor	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%		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:		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%	Os	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	
► ILoop at line 2750 in dxsec al	< 3.8%	4.494s	distress	Loop at line 2750 in dxsec dl	xsec.f90	0x49d2b2	>
	20s 40s	60s	80s	100s 120s 140s		Thread	~
						Running	
distress (TID: 55598)						CPU Time	
F			Spin and Overhead				
						CPU Sample	
	✓ CPU Utilization						
						 Spin and Overh	ead
CPU Utilization							
FILTER 100.0%	cess ~ Any Thread	~ Any Mod	ule 🗸 🛛 Any Util	izatic V User functions + 1 V	Show inline fun	cti ~ Loops only	

 Intel VTune Amplifier 日 日 日					- 🗆 ×
	amplxe_distress2019-04-10-20-23	3 ×			
Hotspots Hotspots by CPU Utilization	• (2)			INTELVIUN	E AMPLIFIER 2019
Analysis Configur HPC Performance Characte	erization ottom-up Caller/Callee	Top-down Tree	Platform aux.f90 × aux.f9	0 ×	
Grouping: Call Sta Hotspots by CPU Utilizati	on				□+□ Q ~
Funda	al 🔻 » CPU Time: Self »	Module	Function (Full)	Source File	Start Address
▼ Total Threading Efficiency	00.0% Os				
 [Outside any loop] 	99.9% 0.020s		[Outside any loop]	0	
[Loop at line 100 in vegas]	99.6% Os	distress	[Loop at line 100 in vegas]	vegas.f90 0	(4162c8
▼ [Loop at line 112 in vegas]	99.6% 1.531s	distress	[Loop at line 112 in vegas]	vegas.f90 0	(416641
[Loop at line 112 in vegas]	98.2% 13.427s	distress	[Loop at line 112 in vegas]	vegas.f90 02	«4166f1
▼ [Loop at line 2499 in dxsec_ql_	36.9% 15.606s	distress	[Loop at line 2499 in dxsec_ql	xsec.f90 02	49ba17
[Loop at line 263 in spinoru]	24.2% 1.422s	distress	[Loop at line 263 in spinoru]	aux.f90 02	41ecd6
[Loop at line 264 in spinor	23.2% 32.939s	distress	[Loop at line 264 in spinoru]	aux.f90 02	k41edcf
[Loop at line 258 in spinoru]		distress	[Loop at line 258 in spinoru]		(41ea94
[Loop at line 260 in spinoru]		distress	[Loop at line 260 in spinoru]		(41ec41
[Loop at line 2487 in LHAPD		libLHAPDF.so	[Loop at line 2487 in LHAPDF:		(669c9
[Loop at line 1169 in LHAPD		libLHAPDF.so	[Loop at line 1169 in LHAPDF:		<66960
[Loop at line 139 in nnlobeami]		distress	[Loop at line 139 in nnlobeami]		(4310f9
[Loop at line 43 in ktjet]		distress	[Loop at line 43 in ktjet]	-	(420c70
► [Loop at line 2750 in dxsec al	3.8% 4.494s	distress	[Loop at line 2750 in dxsec al	xsec.f90 0	×49d2b2 ×
	os 40s 60s	80s	100s 120s 140s	Thr	ead ~
					Running
distress (TID: 55598)					CPU Time
				🖂 🗠	Spin and Overhead
					CPU Sample
				CPU	Utilization
					CPU Time
				🗹 📥	Spin and Overhead
CPU Utilization					~
FILTER 🍸 100.0% 🥁 Any Process	Any Thread < Any Mod	ule 🗸 Any Utiliz	zatic V User functions + 1 V	Show inline functi	 Loops only

簷 Intel VTune Amplifier@jlselogin	1								_	- 🗆 X
물 좌 🕨 🛓 🕕 🗁 🕐	Welcom	ne x roo	0hpc × r001ue ×							=
Microarchitecture	Explora	tion Micr	oarchitecture Expl	oration 👻 🗇					INTEL VTUNE AM	IPLIFIER 2019
Analysis Configuration					tform					
, ,		Log Sumi	nary Bottom-up	Event Count Pla	uom					
Grouping: Function / Call St	аск			[N]				Back-End B	aund	★ ▷ %
Function / Call Stack	CPU 🔻	CPI Rate	Front-End Bound	Bad Speculation			Memory Late		ouna	Mer
runction / can stack	Time *	crinate	Hone-End Boand	bad speculation	L1 Hit Rate	L2 Hit Rate	L2 Hit Bound	L2 Miss Bound	UTLB Overhead	Split Loads
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_folld	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%		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%
Ø: + = ⊮	er ا		50s	100s		150s	183.876s 200)s	✓ Thread	•
OMP Master Thread #0 (.									Run	
OMP Master Thread #0 (. OMP Worker Thread #1 (.									CPU 🗠 CPU	Time
OMP Worker Thread #2 (CPU Tin	
OMP Worker Thread #3 (CPU Ti	ne
OMP Worker Thread #17	pa	used								
OMP Worker Thread #55										
OMP Worker Thread #52										
CPU Time										

FILTER 🝸 100.0%

Any Process

50

E

▼ Thread Any Thread ▼ Module Any Module ▼ User functions + 1

Functions only

Show inline functions

Viewing the result

- Text file reports:
 - amplxe-cl -help report

How do I create a text report?

- amplxe-cl -help report hotspots
 What can I change
- amplxe-cl -R hotspots -r ./res_dir -column=? Which columns are available?
- Ex: Report top 5% of loops, Total time and L2 Cache hit rates
 - amplxe-cl -R hotspots -loops-only

-limit=5 -column="L2_CACHE_HIT, Time Self (%)"

- Vtune GUI
 - unset LD_PRELOAD; amplxe-gui



Using Vtune to ch

General Exploration Microarchitecture

Analysis Configuration Collection Log

Grouping: Function / Call Stack

Function / Call Stack

GSimulation::start

apic_timer_interrupt

native_write_msr_safe

Grouping: Function / Call Stack

Function / Call Stack

GSimulation::start

Isnic nevt deadline

Optimization Notice

Copyright © 2020, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others. amplxe: Using result path `/gpfs/jlse-fs0/users/pvelesko/nbody-demo/ver5/amplxe_knl_nodiv_60k' amplxe: Executing actions 75 % Generating a report Elapsed Time: 2 Elapsed Time: 280.549s Clockticks: 405.093.000.000 Instructions Retired: 342,199,000,000 CPI Rate: 1.184 MUX Reliability: 0.992 Front-End Bound: 1.5% of Pipeline Slots ITLB Overhead: 0.0% of Clockticks BACLEARS: 0.1% of Clockticks MS Entry: 0.0% of Clockticks ICache Line Fetch: 1.0% of Clockticks Bad Speculation: 0.2% of Pipeline Slots Branch Mispredict: 0.2% of Clockticks SMC Machine Clear: 0.0% of Clockticks MO Machine Clear Overhead: 0.0% of Clockticks Back-End Bound: 56.2% of Pipeline Slots A significant proportion of pipeline slots are remaining empty. When operations take too long in the back-end, they introduce bubbles in the pipeline that ultimately cause fewer pipeline slots containing useful work to be retired per cycle than the machine is capable of supporting. This opportunity cost results in slower execution. Long-latency operations like divides and memory operations can cause this, as can too many operations being directed to a single execution port (for example, more multiply operations arriving in the back-end per cycle than the execution unit can support). L1 Hit Rate: 60.2%

1 Hit Rate: 60.2% The L1 cache is the first, and shortest-latency, level in the memory hierarchy. This metric provides the ratio of demand load requests that hat the L1 cache to the total number of demand load requests.

L2 Hit Rate: 98.8%

2 Hit Bound: 100.0% of Clockficks | Issue: A significant portion of cycles is being spent on data | fetches that miss the Ll but hit the L2. This metric includes | coherence penalties for shared data.

Tips:

Summa

 If contested accesses or data sharing are indicated as likely issues, address them first. Otherwise, consider the performance tuning applicable to an 1.2-missing workload: reduce the data working set size, improve data access locality, consider blocking or partitioning your working set so that it fits into the L1, or better exploit hardware prefetchers.

 Consider using software prefetchers, but note that they can interfere with normal loads, potentially increasing latency, as well as increase pressure on the memory system.

L2 Miss Bound: 36.2% of Clockticks | Issue: A high number of CPU cycles is being spent waiting for L2 | load misses to be serviced.

Tips:

 Reduce the data working set size, improve data access locality, blocking and consuming data in chunks that fit into the L2, or better exploit hardware prefetchers.

 Consider using software prefetchers but note that they can increase latency by interfering with normal loads, as well as increase pressure on the memory system.

uTLB Overhead: 4.6% of Clockticks SIMD Compute-to-12 Access Ratic: 1.490 SIMD Compute-to-12 Access Ratic: 4.003 | This metric provides the ratio of SIMD compute instructions to | the total number of memory loads that hit the L2 cache. On this | platform, it is important that this ratio is large to ensure | efficient usage of compute resources.

Contested Accesses (Intra-Tile): 0.0% Page Walk: 4.9% of Clockticks Memory Reissues Split Loads: 0.0% Loads Blocked by Store Forwarding: 0.0% Retiring: 42.1% of Pipeline Slots VPU Utilization: 99.% of Clockticks Divider: 0.0% of Clockticks FP Assists: 0.1% of Clockticks FP Assists: 0.0% of Clockticks Total Thread Count: 1

d Speculation 🔋	Back-End Bound 🔅	Retiring 😕				
0.1%	41.3%	58.6%				
0.0%	46.7%	0.0%				
0.0%	60.0%	0.0%				

Me mory Latency												
L2 Hit Bound	L2 Miss Bound	UTLB Overhead	:									
0.9%	0.0%	0.0%										
0.0%	0.0%	0.0%										



Microarchitecture Exploration - Caches

S	2k	2.5k	30k	35k	50k	60k
L1 Hit %	100%	63.9%	62.4%	48.5%	57.5%	60.2%
L2 Hit %	0%	100%	100%	100%	99.2%	98.8%
L2 Hit Bound %	0%	100%	100%	100%	100%	100%
L2 Miss Bound %	0%	0%	0%	0%	28.6%	36.2%





Profiling PYThon & ML applications



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:



Simple Python Example on Theta

INTEL VTUNE AMPLIFIER 2018

Basic Hotspots Hotspots by CPU Usage viewpoint (change)

🗉 🖸 Collection Log 😌 Analysis Target Å Analysis Type 🔹 Summary 💩 Bottom-up 💩 Caller/Callee 💩 Top-down Tree 🖂 Platform 🔓 cov.py

Elapsed Time⁽²⁾: 209.598s

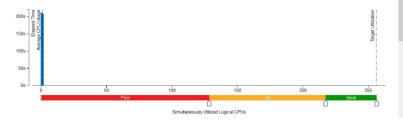
⊘ Top Hotspots

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

Function	Module	CPU Time ⁽²⁾
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

Bottom-up View and Source Code

Basic Hotspots Hotspots b	y CPU Usage viewpoint (<u>change</u>) 🤨					INTEL VTUNE AMPLIFIER 201
🗉 🗔 Collection Log \varTheta Analysis Target	Å Analysis Type 🔹 Summary 🚳 Bottom-u	p 🗞 Caller/Ca	allee 🛛 🗞 Top-dowr	n Tree 🙁 Platform	🔁 cov.py	
Grouping: Module / Function / Call Stack					~ 🛠 Q	CPU Time
	CPU Time 🔻		×.			∧ Viewing < 1 of 1 → selected stack(stack)
Module / Function / Call Stack	Effective Time by Utilization	Spin Time	Overhead Time	Module		100.0% (112.473s of 112.473s) cov.py1naive - cov.py
▼ cov.py	203.728s	2.280s	0s			cov.py!main+0x42 - cov.py:200
▼ naive	111.873s	1.660s	0s	cov.py	naive(fullArray)	cov.py1 <module>+0x221 - cov.py:</module>
▼ main	110.833s	1.660s	0s	cov.py	main()	python2.71 start+0x28 - [unknow
▶ S <module> ← _start</module>	110.813s	1.660s	0s	cov.py	<module></module>	
▶ 🔼 main ← <module> ← _star</module>	0.020s	0s	0s	cov.py	main()	
▶ 🔽 naive ← main ← <module> ←</module>	1.040s	0s	0s	cov.py	naive(fullArray)	
▶ <genexpr></genexpr>	90.967s	0.620s	0s	cov.py	naive@ <genexpr></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	∩e	()e		>	v
,0:+- ⊮ ⊮ ^{0s}	50s	100s		150s	200s	Thread V
python (TID: 218893)						Running
E.						🗹 💼 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 Util
		🔲 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[:, :</pre>	i 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	end = time.time()	
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

Intel[®] VtunE[™] Application Performance Snapshot

Performance overview at you fingertips

VTune[™] Amplifier's Application Performance Snapshot

High-level overview of application performance

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



Usage on Theta

Launch all profiling jobs from /projects rather than /home

No module available, so setup the environment manually:

- \$ module load vtune
- \$ 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:

Optimization Notice



APS HTML Report

Application F	Performance Sn	napshot	
Application: heart_demo Report creation date: 2017-08-01 12:08:48 Number of ranks: 144 Ranks per node: 18 OpenAIP threads per rank: 2 HW Platform: Intel(R) Xeon(R) Processor cod Logical Core Count per node: 72	e named Broadwell-EP	Your application is MPI bot This may be caused by high busy wait ti optimal communication schema or MPI like Intel® Trace Analyzer and Collector	ne inside the library (imbalance), non- library settings. Use <u>MPI profiling tools</u>
121.39s Elapsed.Time		Current.run Target MPI.Time 53.74% ₹ <10% OpenMP.Imbalance 0.43% <10% Memory.Stalls 14.70% <20% Construction 20% <20%	Delta
50.98 0.68		FPU Utilization 0.30% ▶ >50% I/O Bound 0.00% <10%	
SP FLOPS CPI	. <u>MIN</u> 0.65)		
MPL Time 53.74% ► of Elapsed Time (65.23s)	OpenMP Imbalance 0.43% of Elapsed Time (0.52s)	Memory Stalls 14.70% of pipeline slots	FPU Utilization 0.30%►
MPI Imbalance		Cache Stalls 12.84% of cycles	SP FLOPs per Cycle 0.08 Out of 32.00
11.03% of Elapsed Time (13.39s) TOP 5 MPI Functions %	Memory Footprint Resident:	DRAM Stalls 0.18% of cycles	Vector Capacity Usage 25.84%
TOP 5 MPI Functions % Waitall 37.3	Per node: Peak: 786.96 MB	NUMA	FP Instruction Mix
Isend 6.48	Average: 687.49 MB	31.79% of remote accesses	% of Packed FP Instr.: 3.54%
Barrier 5.52	 Per rank: Peak: 127.62 MB 		% of <u>128-bit</u> : 3.54% % of <u>256-bit</u> : 0.00%
Irecv 3.70	Average: 38.19 MB		% of Scalar FP Instr.: 96.46%
Scatterv 0.00	Virtual: Per node:		FP Arith/Mem Rd Instr. Ratio 0.07
1/O Bound	<u>Peak:</u> 9173.34 MB Average: 9064.92 MB Per rank: Peak: 566.52 MB		FP. Arith/Mem. Wr. Instr. Ratio 0.30

Optimization Notice

Copyright © 2020, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

(in

inte

Common issues



No call stack information - check that finalization

Incompatible database scheme - make sure the same version of vtune

Vtune sampling driver.. using perf - use latest vtune/ driver needs a rebuild



Tips and tricks

Speeding up finalization

Advisor

add `--no-auto-finalize` to the aprun

followed by `advixe-cl R survey ...` <u>without</u> <u>aprun</u> will cause to finalize on the momnode rather than KNL.

You can also finalize on thetalogin:

cd your_src_dir;

export SRCDIR=`pwd | xargs realpath`
advixe-cl -R survey --search-dir src:=\${SRCDIR}

Vtune

...

add `--finalization-mode=none` to aprun

followed by `amplxe-cl -R hotspots ...` <u>without</u> <u>aprun</u> will cause to finalize on momnode rather than KNL

You can also finalize on thetalogin: cd your_src_dir; export SRCDIR=`pwd | xargs realpath` amplxe-cl -R hotspots --search-dir src:=\${SRCDIR}

Optimization Notice

. .



Managing overheads

Advisor Dependencies and MAP analyses can have huge overheads

If able, run on reduced problem size. Advisor just needs to figure out the execution flow.

Only analyze loops/functions of interest:

https://software.intel.com/en-us/advisor-user-guide-mark-up-loops



Advisor hands on

Collect survey and tripcounts

cd /projects/intel/pvelesko/nody-demo/ver0 make cp /soft/perftools/intel/advisor/advixe.qsub ./

qsub ./advixe.qsub ./nbody.x 2000 500

scp result back to your local machine

Text report can also be useful:

advixe-cl -R survey





X-forwarding is not recommended.

Tar the result along with sources (if you want to be able to view them)

or

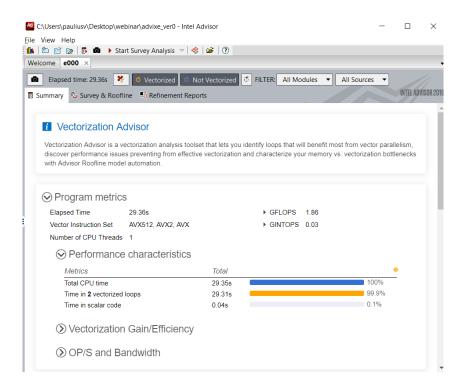
Generate a snapshot:

\$ advixe-cl --snapshot --pack --cache-sources --cache-binaries

then scp to your local machine



Summary Report



Summary provides overall performance characteristics

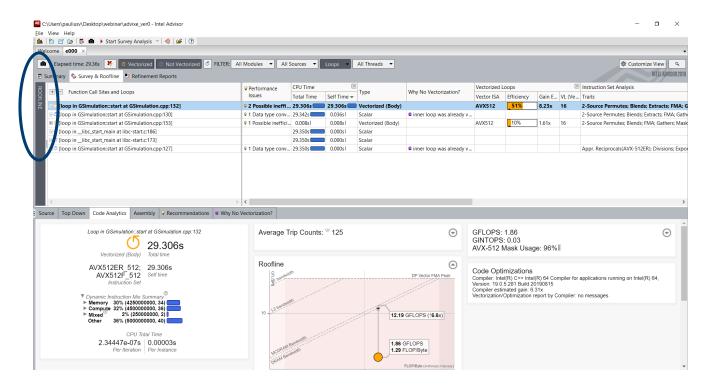
Top time consuming loops are listed individually

Vectorization efficiency is based on used ISA (in this case SSE2/SSE)

Note the warning regarding a higher ISA (in this case -xMIC-AVX512)

Optimization Notice

Survey Report (Code Analytics Tab)



Analytics tab contains a wealth of information

- Instruction set
- Instruction mix
- Traits (sqrt, type conversions, unpacks)
- Vector efficiency
- Floating point statistics

And explanations on how they are measured or calculated expand the box or hover over the question marks.

Optimization Notice

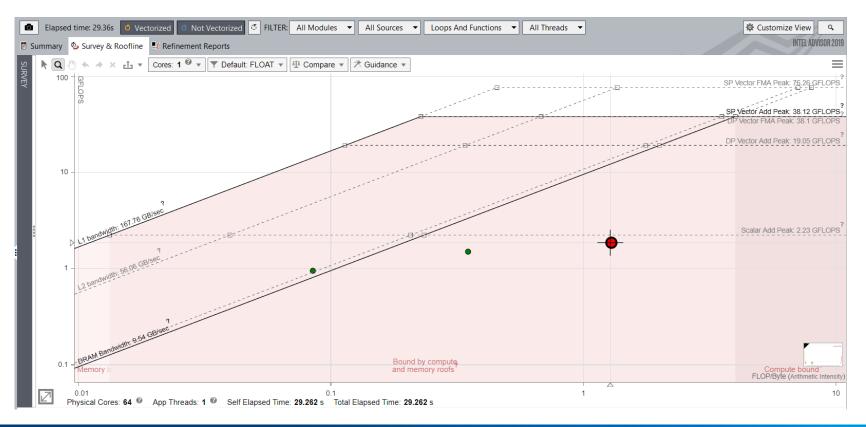
Survey Report (Source Tab)

	Elapsed time: 91.40s 😵 O Vectorized O Not Vector	d Functions All Threads	▼ OF	F	Smart N		. ADVISOR 2		Notice the following:				
🖾 Sun	mmary 🗞 Survey & Roofline 🛄 Refinement Reports												
😸 🧘	Higher instruction set architecture (ISA) avai	lable								▲ 2	of 2 🕨 🧔	×	 Higher ISA available
	Consider recompiling your application using a higher ISA.												ingher is/cavanasie
고		Performance					Vectoriz	ed Loops		➢ FL	OPS	^	 Type conversion
ROOFLINE	Function Call Sites and Loops	Issues	Self Time 🔻	Total Time	Туре	Why No Vectorization?	Vector	Efficiency	Gain E	VL (Ve Se	If GFLOPS	T	.,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
	[0] [loop in GSimulation::start at GSimulation.cpp:138]	🔋 1 Data type con	. 90.600s 🗖	90.600s	Vectorized (Body)		SSE2	91%	1.82x	2 0.	993 📖	t i	Use of square root
	[0] [loop in GSimulation::start at GSimulation.cpp:136]		0.020sl	90.620s	Scalar	inner loop was already v				0.1	150		
	f_start		0.000s1	90.620s	Function							-	
	f main		0.000s1	90.620s	Function								All of these elements may
	f GSimulation::start		0.000sl	90.620s	Function								
	· · · · · · · · · · · · · · · · · · ·	<	0.000 1								>	† ¥	affect performance
Sourc		nendations Why N		on?									
File: ca	che_76525f07bef212a1fcb8f6b3b3ab2632_GSimulatior	n.cpp:138 GSimulatio	n::start										
Line		Sour	ce				Total Tim	ne % L	.oop/Functio	n Time 🛛 🕅	6 Traits	^ L	
133	- IOI (INC 8-1, 80-get_Naceps(), (18)												
134 135	ts0 += time.start();												
136		ration											
130	{	140101											
138	for (j = 0; j < n; j++)						1.02	20s		90.600s			
	O [loop in GSimulation::start at GSimulati	ion.cpp:138]											
	Vectorized SSE; SSE2 loop processes Fl	.oat32; Float64;	Int64 data	type(s) an	d includes Squar	re Roots; Type Convers:							
	No loop transformations applied												
	[] [loop in GSimulation::start at GSimulat:	ion.cpp:138]											
	Scalar remainder loop [not executed]												
	No loop transformations applied												
						Selected (Total Time):	1.02	20s				\sim	
	<					>	<				>		

Optimization Notice

Copyright © 2020, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others. **inte**

Cache-Aware Roofline Model (CARM) Analysis



Follow recommendations and re-test

In this new version (ver2 in github sample) we introduce the following changes:

- Consistently use float types to avoid type conversions in GSimulation.cpp
- Recompile to target Intel[®] Xeon Phi 7230 with -xMIC-AVX512

Note changes in survey report:

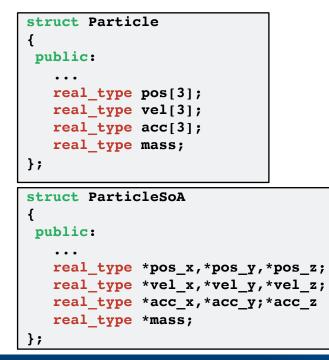
- Reduced vectorization efficiency (harder with 512 bits)
- Type conversions gone
- Gathers/Blends point to memory issues and vector inefficiencies

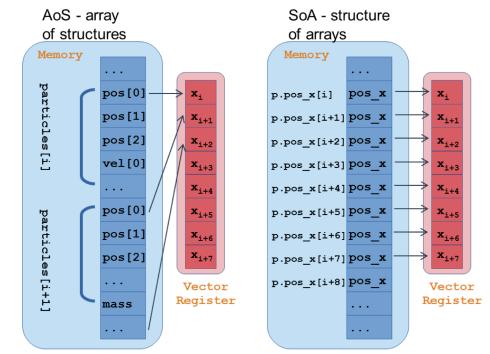
ummary 🗞 Survey & Roofline 📲 Refinement Reports											
+ - Function Call Sites and Loops	Performance	rformance Self Time -		Туре	Wh	v No Vectorization?	Vectoriz	ed Loops		D	FLC
- Tunction can sites and Loops	Issues	Sen Time +	Total Time	Type		y NO VECTORZATION:	Vector	Efficiency	Gain E	VL (Ve.	. Sel
[loop in GSimulation::start at GSimulation.cpp:138]	© 2 Inefficient gat	. 10.080s 🥅	10.080s 🥅	Vectorized (Body)			AVX5	63%	10.05x	16	2.0
Iloop in GSimulation::start at GSimulation.cpp:136	I Opportunity for.	. 0.060sl	10.140s	Scalar	🖣 in	nner loop was already v					1.70
		0.000s	10.140s	Function							
⊇ ∮ main		0.000sl	10.140s	Function							
		0.000sl	10.140s	Function							
Iloop in GSimulation::start at GSimulation.cpp:133	@ 1 Data type conv	0.000sl	10.140s	Scalar	🖬 in	nner loop was already v					
	mendations	lo Vectorizatio	on?								
				-			_			0	
Loop in GSimulation::start at GSimulation.cpp:138	Average T	rip Counts	: 125)	GFLOPS: 2.0932		-		$\overline{\mathbf{A}}$		
🕛 10.080s				AVX-512 Mask Usage: 37							
Vectorized (Body) Total time											
	Traits			(A	Static Instruction Mix [®]						
AVX512ER_512; 10.080s	Square Roo	ts		0							
AVX512F_512 Self time Instruction Set	Gathers					Memory: 22 Compu 12 Number of Vector			Other:		
	•					12 Number of vector	rRegister	8. 20			
▼ Static Instruction Mix Summary [®] ▶ Memory 33% (22) ▶ Compute 37% (21)			Access Patterr Recommendat	or							
► Compute 37% (21)	Blends										
Other 21% (12)	• Irrec	ular Memory A	Access Patterr	or.							
Dynamic Instruction Mix Summary [®]		Suggestion: See Recommendations Tab									
	FMA										
	2-Source Pe	ermutes									
10.05x	Mask Manip										
3% Vectorization Efficiency Vectorization Gair		ulations									

Optimization Notice

Vectorization: gather/scatter operation

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





Optimization Notice

Memory access pattern analysis

How should I access data ?

Unit stride access are faster

for (i=0; i<N; i++)
 A[i] = B[i]*d</pre>

Constant stride are more complex

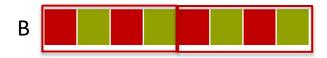
for (i=0; i<N; i+=2)
 A[i] = B[i]*d</pre>

Non predictable access are usually bad

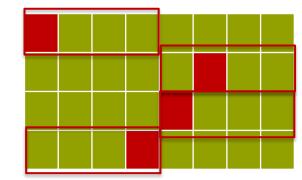
for (i=0; i<N; i++) A[i] = B[C[i]]*d



For B, 1 cache line load computes 4 DP



For B, 2 cache line loads compute 4 DP with reconstructions



В

For B, 4 cache line loads compute 4 DP with reconstructions, prefetching might not work

Optimization Notice

Follow recommendations and re-test

In this new version (ver3 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

🖾 Summary 🗞 Survey & Roofline 📑 Refinement Reports										3		
P + - Function Call Sites and Loops	Performance	Self Time 👻	Total Time	Туре	Marine N	lo Vectorization?	Vectoriz	ed Loops	\gg	FLOPS		^
Function Call Sites and Loops	Issues	Jes Sen Time +		Total time Type w		vo vectorization?	Vector	Gain E	VL (Ve	Self GFLOPS	5	
[Ref [loop in GSimulation::start at GSimulation.cpp:151]	1 Assumed dep	46.360s 🔲	46.360s 📩	Scalar	vec	tor dependence pre				1.122	— (
Scalar loop. Not vectorized: vector dep No loop transformations applied	Scalar loop. No loop trans			or dependence pre	events	vectorization						
Icop in GSimulation::start at GSimulation.cpp:171	1 Assumed depe	0.040s1	0.040s1	Scalar	vect	or dependence preve				0.475 📖	C	
		0.000s1	46.400s	Function								
⊌ ∮ main		0.000s1		Function								
GSimulation::start	(0.000s1	46.400s	Function							0	~
Loop in GSimulation::start at GSimulation.cpp:151	Average Static Ins	No Vectorizat Trip Count truction M Compute: 1 r of Vector Re	s: 2000 ix [®] 1 <i>Mixed</i> : 1	6	9 9	GFLOPS: 1.121 AVX-512 Mask (Code Optimizati Compiler: Intel(R) C++ running on Intel(R) 64, Version: 18.0.0 128 Bu	Jsage: ons	64 Compil	er for appli	cations		^
Compute 32% (11) Mixed [®] 32% (11) Other 12% (4) Dynamic Instruction Mix Summary [®] Traits Source Date EMA	Ð											~

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



Suggested solutions

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



Analyze Result - advixe_ver4

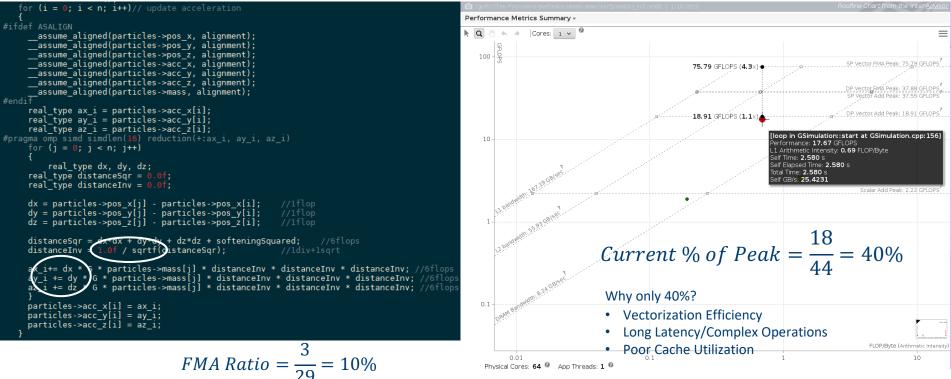
Vectorization time back to normal

Reduced execution time

Optimization Notice



Advisor Roofline – How much further can we go?



Peak = SP Vector ADD * (1+ FMA Ratio) Peak = 40 * (1 + 0.1) = 44 GFLOPS

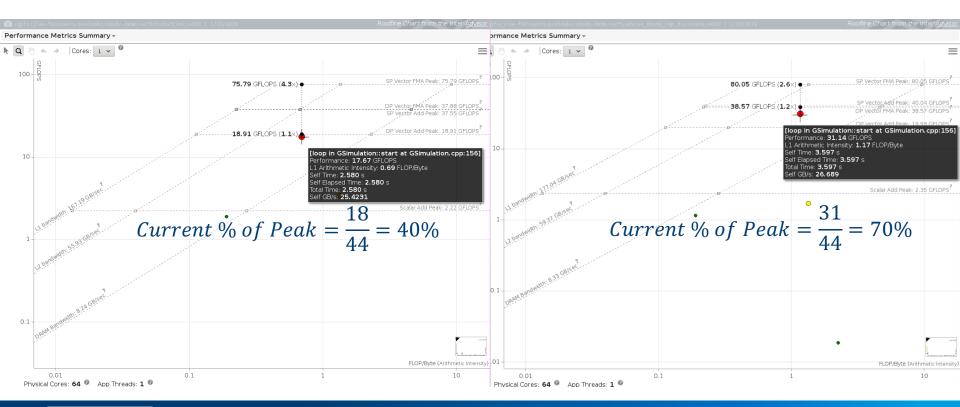
Optimization Notice

Vectorization Efficiency?

	Elapsed	d time: 5.19s	🖰 Vect	orized 👩	Not Ve	ctorized	H 👩 FILTER:	All Mo	dules	•
🗒 Summary 🗞 Survey & Roofline 🛄 Refinement Reports										
7	+ - Function Call Sites and Loops					Vectorized Loops			≫ (
l og	ğ					Vec	Efficiency	Gai	VL (
ΙË	💵 🕘 [loop in GSimulation::start at GSin					AVX	~ <mark>97%</mark>	15	16	
	☑ [loop in GSimulation::start at GSimulati									1



Complex Operations?



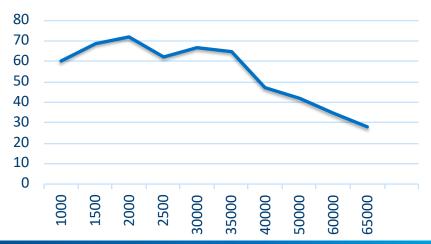
Optimization Notice

Memory Performance



Maximum N before we lose caching? KNL L1-32kB L2-1MB (1 tile/2cores) 32k/(4*4) = 2k (L1) 1MB/(7*4) = 35.7k(L2)





Optimization Notice

backup

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



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-memorybandwidth=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

Optimization Notice



Advisor Cheat Sheet

Compile with -g -dynamic

advixe-cl -c roofline/depencies/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 \leftarrow in mb
- advixe-cl -help

Optimization Notice

How much further can we go?

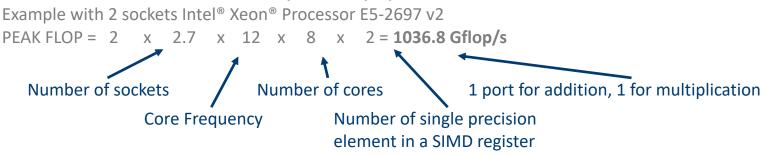
Introducing the Cache-Aware Roofline Model

Platform peak FLOPs

How many floating point operations per second



Theoretical value can be computed by specification



More realistic value can be obtained by running Linpack

=~ 930 Gflop/s on a 2 sockets Intel[®] Xeon[®] Processor E5-2697 v2



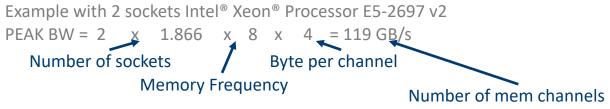
77

Platform PEAK bandwidth

How many bytes can be transferred per second



Theoretical value can be computed by specification

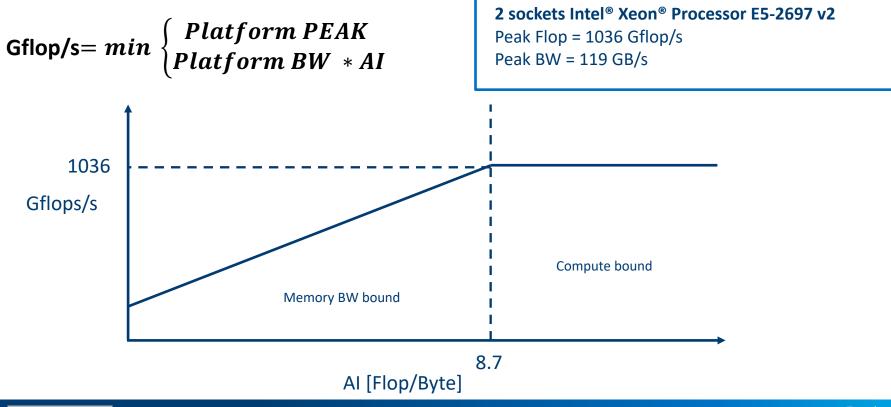


More realistic value can be obtained by running Stream

=~ 100 GB/s on a 2 sockets Intel[®] Xeon[®] Processor E5-2697 v2



Drawing the Roofline



Optimization Notice

Cache-Aware Roofline

Next Steps

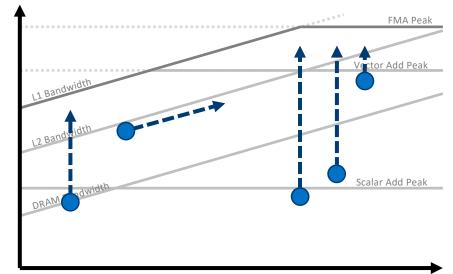
If under or near a memory roof...

 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.**



Arithmetic Intensity

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





Software