

INTEL® PERFORMANCE TOOLS

Jackson Marusarz - Intel



INTEL[®] VTUNE[™] AMPLIFIER

Agenda

- Introduction to Performance Tuning
- Introduction to Intel VTune Amplifier
- System-Level Profiling
 - HPC Characterization
 - Disk I/O Analysis
- Application Performance Tuning Process
 - Find Hotspots
 - Determine Efficiency
 - Address Parallelism Issues
 - Address Hardware Issues
 - Rebuild and Compare
- Summary



Two Great Ways to Collect Data

Intel[®] VTune[™] Amplifier

Software Collector	Hardware Collector				
Uses OS interrupts	Uses the on chip Performance Monitoring Unit (PMU)				
Collects from a single process tree	Collect system wide or from a single process tree.				
~10ms default resolution	~1ms default resolution (finer granularity - finds small functions)				
Either an Intel [®] or a compatible processor	Requires a genuine Intel [®] processor for collection				
Call stacks show calling sequence	Optionally collect call stacks				
Works in virtual environments	Works in a VM only when supported by the VM				
works in virtual environments	(e.g., vSphere*, KVM)				
No driver required	Requires a driver - Easy to install on Windows - Linux requires root (or use default perf driver)				

No special recompiles - C, C++, C#, Fortran, Java, Assembly



A Rich Set of Performance Data

Intel[®] VTune[™] Amplifier

Software Collector	Hardware Collector		
Hotspots Which functions use the most time?	Hotspots Which functions use the most time? Where to inline? – Statistical call counts		
Threading Tune parallelism. Colors show number of cores used.	Microarchitecture Exploration Where is the biggest opportunity? Cache misses? Branch mispredictions?		
Tune the #1 cause of slow threaded performance: – waiting with idle cores.	Advanced Analysis Memory-access, HPC Characterization, etc		
Any IA86 processor, any VM, no driver	Higher res., lower overhead, system wide		

No special recompiles - C, C++, C#, Fortran, Java, Assembly



Example: Hotspots Analysis Summary View

📟 Collection Log 🤇	Analysis Target	Analysis T	ype 🗓 Su	immary
Elapsed Tir	me ^② : 5.554	s		
O CPU Time		10.504s		
Instructions R	letired: 21,69	8,000,000		
CPI Rate [®] :		1.257 🏲		
CPU Frequer	ncy Ratio [®] :	1.041		
Total Thread	Count:	9		
Paused Time	2 ⁽⁰⁾ :	0s		

O Top Hotspots

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

🗣 Bottom-up

Function	Module	CPU Time 🛛
grid intersect	3_tachyon_omp.exe	5.539s
sphere intersect	3_tachyon_omp.exe	3.247s
func@0x1002e59d	libiomp5md.dll	0.148s
shader	3_tachyon_omp.exe	0.117s
KeDelayExecutionThread	ntoskrnl.exe	0.091s
[Others]	N/A*	1.361s

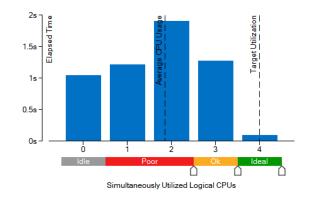
*N/A is applied to non-summable metrics.

Average Bandwidth

Package	Total, GB/sec	Read, GB/sec	Write, GB/sec
<u>package 0</u>	5.715	3.504	2.212

OPU 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.



Collection and Platform Info

This section provides information about this collection, including result set size and collection platform data.



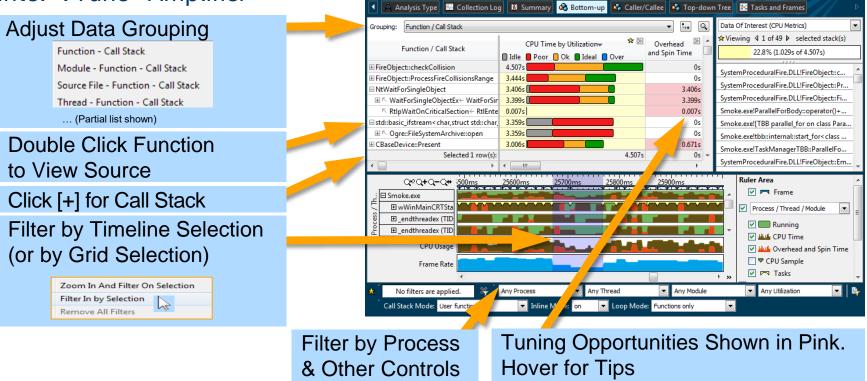
Example: Threading Analysis Bottom-up View

Grouping: Function / Call Stack			🗳 Caller/Callee 🗳 Top-down Tree 🗄					✓ L.	Q Data Of Interest	(CPU Metrics)
Function / Call Stack	CPU Time by		Wait Time by Utilization	≫ Ov.≫ an.	Thre Ove	Module	Start Address	Function (Full)		of 21
grid_intersect	5.360s			0s	4.527s	3_tachyon_omp.exe	0x40c7f0	grid_intersect	24.4	
	3.542s			0s	2.914s	3_tachyon_omp.exe	0x40aca0	sphere_intersect		.exelgrid_intersect - grid '
SwitchToThread	0.986s			0.986s	0.901s	KERNELBASE.dll	0x10047e49	SwitchToThread	3_tachyon_omp	.exe!intersect_objects+0x
kmp_launch_thread	0.874s		2.104s	0.874s	0.008s	libiomp5md.dll	0x1004b0d0	kmp_launch_thread	3_tachyon_omp	.exelshader+0x357 - sha
grid_bounds_intersect	0.297s 🚺			0s	0.215s	3_tachyon_omp.exe	0x40c4f0	grid_bounds_intersect	3_tachyon_omp	.exeltrace+0x2f - trace_re
shader	0.106s			0s	0.066s	3_tachyon_omp.exe	0x406b50	shader(struct ray *)	3_tachyon_omp	.exe!render_one_pixel+0
GdipDrawImagePointRectI	0.098s			0s	0.098s	gdiplus.dll	0x10060336	GdipDrawImagePointRectI	3 tachyon omn	.exe!thread_trace+0x2c9
pos2grid	0.090s			0s	0.074s	3_tachyon_omp.exe	0x40c410	pos2grid		_kmp_invoke_microta
Selected 1 row(s):	5.360	s	0s	4.527s				× .	
	> <							>	libiompomd.dll	[OpenMP dispatcher]+
Q©Q+Q−Q⇔ 0.5s	1s	1.5s	2s 2.5s	3s		3.5s	4s	4.5s 5s 5.5s	s <mark>6</mark> s	Ruler Area
OMP Worker Thread #1				1.11	4.1	11.010				🔨 🔽 Region I
OMP Worker Thread #2		1 PROVADENCIAL					r ru w			Thread V
thread_video (TID: 5712 DMP Worker Thread #3								1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1		
WinMainCRTStartup (TI										Running
func@0x100097fe (TID:					-					Vaits
	-									🗹 🚧 CPU Time
										🗹 🚧 Overhea
		. All days become load	ladella, Maria in manification de la desta de la d	1. A						Transitions
		A BLACK ALL AND A DOWN						-1-10-10-10-01/0		CPU Usage
CPU Usage		المراقعية والالفاقية والمراد								
		11-11-0-11-0-1-0-1-0-1-0-1-0-1-0-1-0-1-	M. 1977. 1.124 STI 11							CPU Time
CPU Usage									ala 16 kita 181 karan 1640 at 101 16 101 101 164 kit	



Find Answers Fast

Intel[®] VTune[™] Amplifier



Optimization Notice

See Profile Data On Source / Asm

Double Click from Grid or Timeline

View Source / Asm or both CPU Time Right click for instruction reference manual

	Source	Assembly					Tree 🛛 🔁 Tasks an 🚽 rames	🚡 grid.c	pp 🕺
			Assembly 🥵	groupir	ng: Address				
	Source Line	Source	CPU Time: Total 🕅 🔲 Idle 📕 Poor 📒 Ok	*	Address 🔺	Sour Line	Asse bly		CPU Time: Total 🕅 🖍
· · · · ·			0.017s		0x418b6d	580	cmp dword pt. op-0x	190], Ox	0.120s
Quick Asm r	าลงเ	dation:			0x418b74	580	jz 0x418be6 <	>	0.379s
-					0x418b76		Block 54:		
Select source	e to	highlight Asm		and the second	0x418b76	581	mov edx, dword ptr [e	bp-0x190	0.090s
Sciece Source		Ingringine Asiri	<u>v</u>		0x418b7c	581	mov eax, dword ptr [e	dx+0x4]	0.020s
	579	<pre>cur = g->cells[voxindex];</pre>	0		0x418b7f	581	mov ecx, dword ptr [e	ax]	3.853s
	580	while (cur != NULL) {	0.499s 🖥		0x418b81	581	mov edx, dword ptr [e	bp+0xc]	2.500s
	581	if (ry->mbox[cur->obj->id] !	7.795s		0x418b84	581	mov eax, dword ptr [e	dx+0x10]	0.030s
	582	ry->mbox[cur->obj->id] = r	0.547s		0x418b87	581	mov edx, dword ptr [e	bp+0xc]	
	583	cur->obj->methods->interse	1.769s	_	0x418b8a	581	mov eax, dword ptr [e	ax+ecx*4	0.040s
	584	}		≣	0x418b8d	581	cmp eax, dword ptr [e	dx+0xc]	1.262s
	585	<pre>cur = cur->next;</pre>	0.568s	≣	0x418b90	581	jz 0x418bd6 <block 57<="" td=""><td>></td><td></td></block>	>	
	586	}	0.070s		0x418b92		Block 55:		
	587	curvox.z += step.z;	0.070s		0x418b92	582	mov ecx, dword ptr [⁰ x190	0.331s
	588	if (ry->maxdist < tmax.z cu	0.100s		0x418b98	582	mov edx, dword ptr [e	x 4]	0.116s
		Selected 1 row(s):	7.795s	-			Highlighte	d 9 ru (s):	7.795s 👻
		•	<				•		۲ III ۲

Scroll Bar "Heat Map" is an overview of hot spots

Click jump to scroll Asm

Command Line Interface

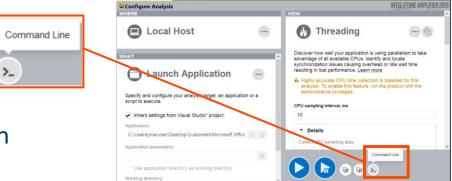
Automate analysis

amplxe-cl is the command line:

- -Windows: C:\Program Files (x86)\IntelSWTools\VTune Amplifier\bin[32|64]\amplxe-cl.exe
- -Linux: /opt/intel/vtune_amplifier/bin[32|64]/amplxe-cl

Help: amplxe-cl -help

Use UI to setup
1) Configure analysis in UI
2) Press "Command Line..." button
3) Copy & paste command



Great for regression analysis – send results file to developer Command line results can also be opened in the UI

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



10

Compare Results Quickly - Sort By Difference

Intel[®] VTune[™] Amplifier

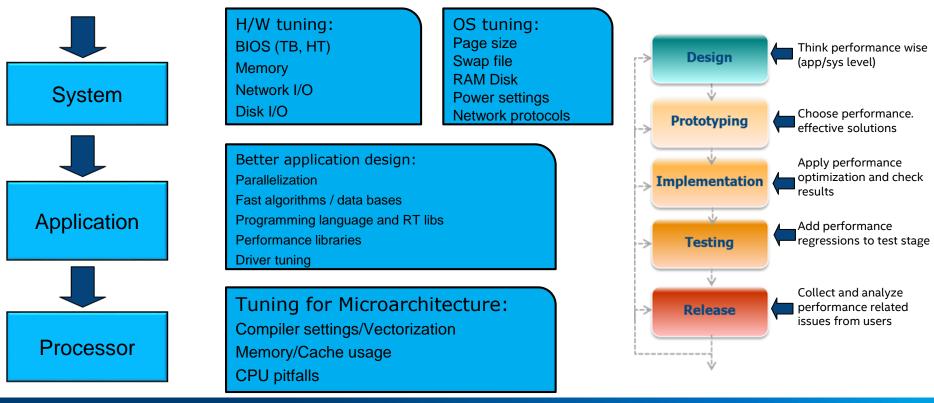
Quickly identify cause of regressions.

- Run a command line analysis daily
- Identify the function responsible so you know who to alert
- Compare 2 optimizations What improved?

Compare 2 systems – What didn't speed up as much?

Grouping: Function / Call Stack					•
Function / Call Stack	CPU Time:Difference	Module	CPU Time:r007hs 🔺	CPU Time:r006hs	-
■ FireObject::checkCollision	4.850s	SystemProceduralFire.DLL	6.281s	1.431s	
Image: Barrier Herrichten Bereichnen Ber	4.644s	SystemProceduralFire.DLL	5.643s	0.999s	
	3.765s	RenderSystem_Direct3D9.DLL	9.184s	5.419s	

Introduction to Performance Tuning





Introduction to Intel VTune Amplifier

- Accurate Data Low Overhead
 - CPU, GPU, FPU, threading, bandwidth, and more...
 - Profile applications or systems
- Meaningful Analysis
 - Threading and hardware utilization efficiency
 - Memory and storage device analysis
- Easy
 - Data displayed by source code
 - Expert advice built-in
 - Easy set-up, no special compiles

🔛 General Explorati	ion Hotspots viewpoint (chang	<u>e)</u>				
🖣 💮 Analysis Target 🧳	🕻 Analysis Type 🔛 Collection Log 🗂	Summary	😪 Bottom-u	p 🗞 Caller/Callee	😪 Top-down Tree	🔁 Platform
Grouping: Function / Call	Stack					
	CPU	J Time 🔻		>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>		
Function / Call Stack	Effective Time by Utilization	ver Sp	pin Time 📓 🤇	Overhead Time	Instructions Retired	CPI Rate
grid_intersect	5.915s (0s	0s	12,956,100,000	1.200
sphere_intersect	3.685s		0s	0s	8,988,900,000	1.049
grid_bounds_intersect	0.434s 🏮		0s	0s	638,400,000	1.714
▶ shader	0.101s		0s	0s	165,300,000	1.414
tri_intersect	0.098s		0s	0s	180,500,000	1.105
▶ pos2grid	0.094s		0s	0s	169,100,000	1.213
▶ Raypnt	0.073s		0s	0s	148,200,000	1.308
< >						
્∾ ્+ ્−્+	0.5s 1s 1.5s	2s	2.5s	3s 3.5s	4s 4.5s	5s 5
OMP Worker Thread	Construction of the second	The state of the s	MALMAN.			Part Control I. A.
OMP Worker Thread	An all a subsection of the second	harden fal	1111. A. H. M. A.			and have and the
OMP Worker Thread	Lotan and and some of the A. A. A. A.	Little and a second			A RAY ON A RAN PRAY	and the second by
OMP Master Thread #	Constant Street Stre	Law of the Also		A.A. NEW WIND		MICHANNEL IA
Thread (TID: 15956)	La Aller		and a state	· · · Antonio estatuana	يستناب والمتعادية	a sugar to a
Thread (TID: 9288)	A					
Thread (TID: 16148)	14					
Thread (TID: 1488)						

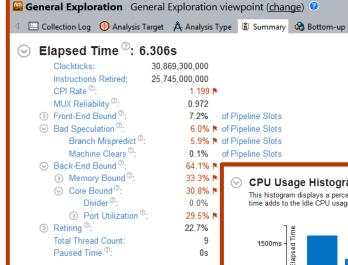
> amplxe-cl -help collect

Optimization Notice

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

(intel)

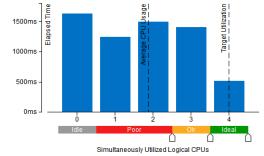
System-Level Profiling – High-level Overviews



Г	್ಲಿ ೦+ ೦−೦⇔	0.5s 1s 1.5s 2s 2.5s 3s 3.5s 4.088s 4.5s 5s 5.5s
	⊡ core_1	and the second
	cpu_2	
	cpu_3	and the provide state of the second state in the second state of the
	⊡ core_0	an a beil all a beil a state and an an an a bein an an a beil an
	cpu_0	and with the first the second s
itext	cpu_1	

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.







System-Level Profiling – Process/Module Breakdowns

(And the second s	Hotspots viewpoint (<u>change</u>) 😢						INTEL V
4	🛛 🔜 Collection Log \varTheta Analysis Target 🕺 Analysis Type	🖞 Summary 🔗 Bottom-u	p 🚱 Caller/Callee 🛛	名 Top-down	n Tree 🛛 😤 Platform		
Gr	ouping: Process / Module / Function / Thread / Call Stack						
	Process / Module / Function / Thread / Call Stack	CPU Time 🔻 💿	Instructions Retired	CPI Rate	CPU Frequency Ratio	Module	
	Pid 0x544	3.889s	27.5%	0.910	0.965		
Processes	chrome.exe	3.443s	15.4%	1.441	0.963		
I	▶ chrome_child.dll	3.022s	14.6%	1.301	0.944		
1	▶ ntdll.dll	0.242s 💧	0.6%	3.171	1.103		
1	▶ ntoskrnl.exe	0.179s 🛔	0.2%	7.143	1.064		
▶ [EXCEL.EXE	2.750s	14.3%	1.312	1.022		
Modules Functions	Explorer.EXE	2.598s	10.3%	1.677	0.998		
>	Syncplicity.exe	1.140s 💼	4.1%	1.923	1.039		
Modulos	OUTLOOK.EXE	0.891s 🛑	1.5%	3.723	0.918		
Modules	▶ mso.dll	0.141s	0.2%	4.719	0.812		
	▼ ntoskrnl.exe	0.080s	0.2%	2.884	1.181		
	ExEnterPriorityRegionAndAcquireResourceExclusive	0.004s	0.0%		0.400	ntoskrnl.exe	ExEnterPriorityRegionAndAcquireResource
Even etti even	ExAllocatePoolWithTag	0.004s	0.0%	1.000	1.000	ntoskrnl.exe	ExAllocatePoolWithTag
	▶ KeSetEvent	0.004s	0.0%		0.200	ntoskrnl.exe	KeSetEvent
	ObReferenceObjectByHandleWithTag	0.004s	0.0%		0.800	ntoskrnl.exe	ObReferenceObjectByHandleWithTag
	Q**Q+Q=Q** 0.5s 1s 1.5s 2s 2.5s Thread (TID: 9844) Intread (TID: 15272) Intread (TID: 16316) Intread (TID: 19756) Thread (TID: 19836) Intread (TID: 16588) Intread (TID: 16588) Intread (TID: 16588)	3s 3.5s 4s 4.5s 4 4 4 4 4	5s 5.5s 6s 6	.5s 7s	7.5s 8s 8.5s 9s	9.5s 10s 10.5s 11s	11.5s 12s 12.5s 13s 13.5s 14s 14.5s

Optimization Notice

System-Level Profiling – I/O Analysis

Are You I/O Bound or CPU Bound?

- Explore imbalance between I/O opera (async & sync) and compute
- Storage accesses mapped to the source code
- See when CPU is waiting for I/O
- Measure bus bandwidth to storage
- Latency analysis
- Tune storage accesses with latency histogram
- Distribution of I/O over multiple devices

> amplxe-cl -collect io -d 10

Disk Input and Output Histogram





System-Level Profiling – HPC Characterizaton

Three Metric Classes

- CPU Utilization
 - Logical core % usage •
 - Includes parallelism and **OpenMP** information
- Memory Bound •
 - Break down each level of • the memory hierarchy
- FPU Utilization
 - Floating point GFLOPS and • density

CPU Utilization Average CPU Usage						I.	
Serial Time ⁽²⁾ :	0.013s (0.1%)					1	
Parallel Region Tir						1	
Estimated Ideal 1						í.	
OpenMP Potent						í.	
The time waste	ed on load imbalance or parallel work arrangeme	ent is significant and negatively impacts th	ie application perform	ance and scal:	ability. Explore	1	
OpenMP regions	s with the highest metric values. Make sure the w	orkload of the regions is enough and the lo	op schedule is optimal			í.	
	ions by Potential Gain					i	
	enMP regions with the highest potential for perfor	rmance improvement. The Potential Gain m	atric shows the elapser	time that co	ild be sayed if		
the region							
OpenMP F 🕑	Memory Bound ^② : 91.8%						
<u>conj_grad.</u>	Cache Bound [©] :				0.10	<u></u>	
MAIN\$c MAIN\$c	DRAM Latency Bound [©] :		% 🏲				
MAIN\$0	DRAM Bandwidth Bound [®] :	SP FLOPs per Cycle ⁽²⁾ :	0.211 Out of 16 🎙				
MAIN\$0		Vector Capacity Usage ⁽²⁾ :	48.3% 🏲				
	This metric represents a fraction of (FP Instruction Mix: % of Packed FP Instr.⁽³⁾: 	93.1%				
[Others]	main memory (DRAM). This metric d	% of 128-bit ⁽²⁾ :	93.1% 🏲				
*N/A is appl	Consider improving data locality in N	% of 256-bit [@] :	0.0%				
	, , ,	% of Scalar FP Instr. ⁽²⁾ : FP Arith/Mem Rd Instr. Ratio ⁽²⁾	6.9%				
	NUMA: % of Remote Accesses $^{\odot}$:	FP Arith/Mem Rd Instr. Ratio FP Arith/Mem Wr Instr. Ratio					
	A significant amount of DRAM loads	⊘ Top 5 hotspot loops (function	s) by FPU usage				
	same core, or at least the same pack	This section provides informati	on for the most time co	insuming loops	/functions with flo	ating point operations.	
	Same core, of at tease the same pack	Function		CPU Time 🕐	FPU Utilization [®]	Vector Instruction Set $^{\odot}$	Loop Type 🖉
		[Loop at line 575 in conj_grad.	_\$omp\$parallel@517]	126.149s	1.6% 🎙	SSE2(128) 🎙	Body
		[Loop at line 678 in conj_grad.	.\$omp\$parallel@517]	5.004s	1.7%	SSE2(128)	Body
		[Loop at line 575 in conj_grad.	.\$omp\$parallel@517]	2.678s	2.1%	[Unknown]	Remainder
		[Loop at line 573 in conj_grad.	.\$omp\$parallel@517]	0.995s	4.0%	SSE2(128)	Body
C	ance -d 10	[Loop at line 661 in conj_grad.	\$omp\$parallel@517	0.952s	1.3%	SSE(128); SSE2(128)	Body
$r \pm \circ r m \Rightarrow$	n a - a + i + i						

*N/A is applied to non-summable metric

amplxe-cl -collect hpc-per: >



System-Level Profiling – Memory Bandwidth

As As Ar Ax	HPC Performance Characterization Copy	
🗆 🦢 Algorithm Analysis	Analyze important aspects of your application performance, including CPU utilization with additional details on OpenMP	Find areas of high ar
A Basic Hotspots	efficiency analysis, memory usage, and FPU utilization with vectorization information. For vectorization optimization data, such as trip counts, data dependencies, and memory access patterns, try Intel Advisor.	
🗚 Advanced Hotspots	It identifies the loops that will benefit the most from refined vectorization and gives tips for improvements.	usage. Compare
Å Concurrency	The HPC Performance Characterization analysis type is best used for analyzing intensive compute applications. Learn more	bandwidth base
A Locks and Waits	(F1)	banchm
🗆 🦢 Compute-Intensive Application Analys	CPU sampling interval, ms:	benchm
A HPC Performance Characterization		
🗆 🦢 Microarchitecture Analysis		
🗚 General Exploration	🖉 Analyze memory bandwidth	-knob collect-memor
Å Memory Access	✓ Evaluate max DRAM bandwidth	
A TSX Exploration		

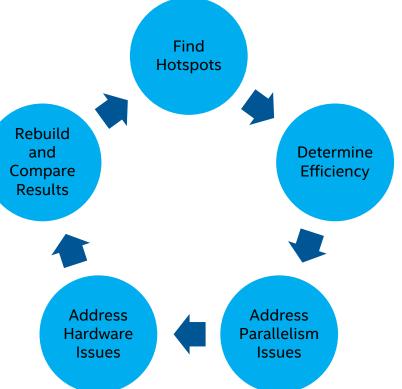
nd low bandwidth to max system ed on Stream arks.

-bandwidth=true

Г		Q+Q-Q#	9850ms 9900ms 9950ms 10000ms 10050ms 1010 <mark>10124.28ms</mark> 50ms 10200ms 10250ms 10300m
ndwi	⊞ package_0	40.0 26.7 13.3	
RAM Bai	⊞ package_0 ⊞ package_1	40.0 26.7 13.3	package_0
		16.5 11.0 5.5	Total, GB/sec
QPI Bandwidt.	⊞ package_1	16.5 11.0 5.5	Read, GB/sec
	package_1 package_0	5600% ⁻ 5600% ⁻	Write, GB/sec



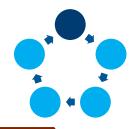
Application Performance Tuning Process





Find Hotspots

Functions



Basic Hotspots Hotspots by CPU U	5 1						INTEL VIONE APIFEITIEN ZU	10			
	Analysis Target Å Analysis Type 🖸 Collection Log 🗓 Summary 🗞 Bottom-up 🗞 Caller/Callee 🗞 Top-down Tree 🔁 Platform Grouping: Function / Call Stack										
Function / Call Stack	CPU Time 🔻 🔊	Module	Function (Full)	Source File	Start Address		Viewing • 1 of 19 • selected stack(s)				
▶ grid_intersect		1_tachyon_serial.exe	grid_intersect	grid.cpp	0x40bee0 0x408a70		33.5% (2.033s of 6.063s)	_			
sphere_intersect MsoWaitForMultipleObjects		1_tachyon_serial.exe user32.dll	sphere_intersect MsgWaitForMultipleObjects	sphere.cpp	0x406a70 0x6ba8dbc0		1_tachyon_serial.exe!grid_intersect				
grid bounds intersect		1 tachyon serial.exe	grid bounds intersect	grid.cpp	0x60a60bc0		1_tachyon_serial.exelintersect_obje				
GdipDrawImagePointRectI		gdiplus.dll	GdipDrawlmagePointRectl	gna.cpp	0x40Cl20		1_tachyon_serial.exe!shader+0x346 1 tachyon_serial.exe!trace+0x2e - tr				
SwitchToThread		KernelBase.dll	SwitchToThread		0x10034200		1_tachyon_serial.exe!race+ox2e - tr				
▶ shader							1 tachyon serial.exe!parallel thread				
▶ tri intersect		1 tachyon serial.exe	tri intersect	triangle.cpp	0x406e60 0x408d60		1 tachyon serial.exe!thread trace+				
▶ pos2arid	-	1 tachvon serial.exe	pos2arid	grid.cpp	0x40ddb0		1 tachyon serial.exe!trace shm+0x				
CreateWindowExA		user32.dll	CreateWindowExA	gnu.opp	0x6ba91cb0		1 tachyon serial.exe!trace region+0				
libm sse2 sqrt precise		msvcr120.dll	libm sse2 sqrt precise		0x10042608		1 tachyon serial.exe!renderscene+0				
▶ Ravpnt		1 tachyon serial.exe	Raypnt(struct ray *,double)	vector.cpp	0x4034d0		1 tachyon serial.exe!rt renderscene				
libm sse2 pow precise		msvcr120.dll	libm sse2 pow precise		0x1003d6f3		1_tachyon_serial.exe!tachyon_video:				
<	<			1			1_tachyon_serial.exe!thread_video+	v			
Q ≈Q+ Q−Q# 0.5s 1s	5c 2c 25c	3c 35c 4c 45	s 5s 5.5s 6s 6.5s 7s 7.5s 8s 8.5s	9s 95s 1(ls 10.5s 11s	11	.5s 12s 12.5: ✔ Thread				
thread video (TID: 171				1	10155 115						
WinMainCRTStartup (110 1 1 1						CPU Time				
							Spin and O				
							CPU Sample				
							✓ CPU Usage				
							✓ ₩₩ CPU Time				
							Spin and O				

> amplxe-cl -collect hotspots -- ./myapp.out

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



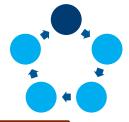
Call Stacks

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

Find Hotspots

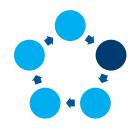
- Drill to source or assembly
- Hottest areas easy to ID
- Is this the expected behavior
- Pay special attention to loops and memory accesses
- Learn how your code behaves
- What did the compiler generate
- What are the expensive statements

🔛 Ba	sic Hotspots Hotspots by CPU Usage viewpoin	nt (<u>change</u>) 😢					I
⊲ ⊕	Analysis Target 🙏 Analysis Type 🔛 Collection Log 👔	Summary 🖓 Bottom-up 🔗 Caller/Callee	🔗 Top-dov	n Tree 🖂	Platform	arid.con	×
Source		Assembly grouping: Address	o rop arm		/ lat.onine _	groupp	
				-A+ [27]			-
Sour		CPU Time: Total	1	*	CPU 🔊		
Line	Source	Effective Time by Utilization	Spin Time	Overhead	Time:	Source File	
	I	🔲 Idle 📕 Poor 🚺 Ok 📕 Ideal 📕 Over	opinc	Time	Self		
562	break;						1
563	<pre>voxindex += step.x;</pre>	1					
564	<pre>tmax.x += tdelta.x;</pre>	()					
565	curpos = nXp;		ļ]			Ļ	
566	nXp.x += pdeltaX.x;		ļ]				
567	nXp.y += pdeltaX.y;	()	l				
568	nXp.z += pdeltaX.z;	()	ļ	ļ		ļ	
569	}		0.00/	0.00%	0.040	<u> </u>	
570	else if (tmax.z < tmax.y) {	0.4%	0.0%			grid.cpp	
571 572	<pre>cur = g->cells[voxindex];</pre>	2.9%	0.0%	0.0%	0.3215	grid.cpp	
572	<pre>while (cur != NULL) { if (ry->mbox[cur->obj->id] != ry->seri</pre>	22.4%	0.0%	0.0%	2 407-	grid.cpp	
574	<pre>if (ry->mbox[cur->obj->id] != ry->seri ry->mbox[cur->obj->id] = ry->serial;</pre>	1	0.0%			grid.cpp	٩
575	cur->obj->methods->intersect(cur->ob		0.0%			grid.cpp	
576	}	1.976	01035	0.075	0.0003	gnucpp	
577	cur = cur->next;	6.3%	0.0%	0.0%	0.699s	grid.cpp	
578	}					9	
579	curvox.z += step.z;	0.3%	0.0%	0.0%	0.038s	grid.cpp	
580	if (ry->maxdist < tmax.z curvox.z ==	0.2%	0.0%	0.0%		grid.cpp	
581	break;						
582	<pre>voxindex += step.z*g->xsize*g->ysize;</pre>						
583	<pre>tmax.z += tdelta.z;</pre>	0.5%	0.0%	0.0%	0.060s	grid.cpp	
584	curpos = nZp;						
585	nZp.x += pdeltaZ.x;						
586	nZp.y += pdeltaZ.y;	1					
Sele		22.4%	0.0%	0.0%	2.497s		





Determine Efficiency



General Exploration Hotspots viewpoint (change)										
🔄 ⊕ Analysis Target 🙏 Analysis Type 🔛 Collection Log 🖺 Summary 🐼 Bottom-up 🐼 Caller/Callee 🐼 Top										
Grouping: Function / Call Stack										
	CPU Time	•	»							
Function / Call Stack	Effective Time by Utilization	Spin Time	Overhead Time							
▶ grid_intersect	5.915s (1997)	0s	Os							
sphere_intersect	3.685s 📕	0s	0s							
grid_bounds_intersect	0.434s 📕	0s	0s							
▶ shader	0.101s	0s	0s							
▶ tri_intersect	0.098s	0s	0s							
▶ pos2grid	0.094s	0s	0s							
▶ Raypnt	0.073s	0s	0s							

💹 General Explor	ation General	Exploration view	vpoint (<u>chan</u>	<u>ge</u>) 😮
Analysis Target	A Analysis Type	🔛 Collection Log	🖞 Summary	😪 Bottom-up
Grouping: Function / Ca	all Stack			

Function / Call Stack	CPI Rate	Retiring »	Fro
▶ grid_intersect	1.200	22.5%	
sphere_intersect	1.049	23.9%	
grid_bounds_intersect	1.714	16.5%	
▶ shader	1.414	16.3%	
▶ pos2grid	1.213	50.9%	
▶ tri_intersect	1.105	23.8%	
▶ Raypnt	1.308	39.2%	
▶ func@0x140150ef0	9.714	80.9%	
libm_sse2_sqrt_precise	2.241	0.0%	

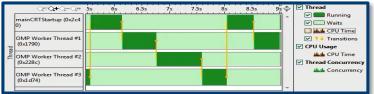
Look for Parallelism, Cycles-per-Instruction (CPI), and Retiring %



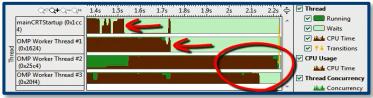
Address Parallelism Issues

- Use Concurrency Analysis to ensure you're using all your threads as often as possible.
- Common concurrency problems can often be diagnosed in the timeline.
- Switch to the Locks And Waits viewpoint or run a Locks and Waits analysis to investigate contention.

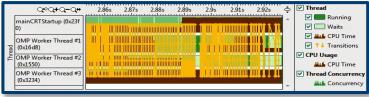
Coarse-Grain Locks



Thread Imbalance

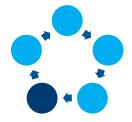


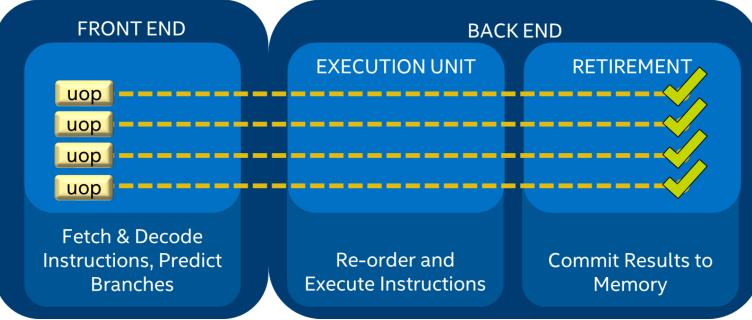
High Lock Contention





Address Hardware Issues



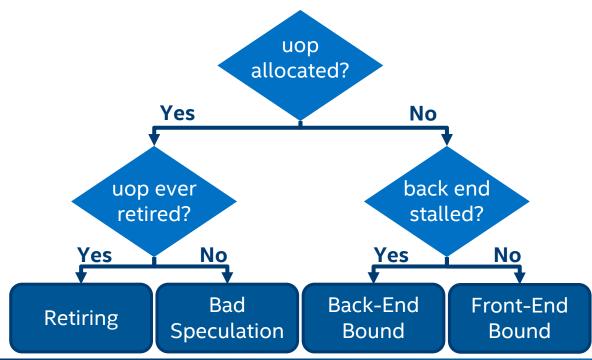


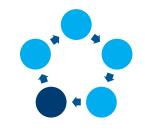
The X86 Processor Pipeline (simplified)



Address Hardware Issues

For each pipeline slot on each cycle:





Optimization Notice Copyright © 2018, Intel Corporation. All rights reserved.

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



Address Hardware Issues

🕘 💮 Analysis Target 🛛 Å Analysis Type	🔛 Collection Log 🖞	Summary 😪 Bottom	-up 🛛 😪 Event Count	🖼 Platform						
Grouping: Function / Call Stack										
Function / Call Stack	Retiring	Front-End Bound	Bad Speculation	Back-End B	Bound «	Module				
Function / Call Stack	Realing	From-End Bound	Bad Speculation	Memory Bound 💿	Core Bound 💿	Module				
▶ grid_intersect	22.5%	6.5%	4.5%	34.6%	31.8%	3_tachyon_omp.exe				
sphere_intersect	23.9%	6.2%	11.5%	29.0%	29.4%	3_tachyon_omp.exe				
grid_bounds_intersect	16.5%	11.3%	8.7%	31.8%	31.8%	3_tachyon_omp.exe				
▶ shader	16.3%	20.3%	4.1%	100.0%	0.0%	3_tachyon_omp.exe				
▶ pos2grid	50.9%	4.6%	0.0%	72.2%	0.0%	3_tachyon_omp.exe				
▶ tri_intersect	23.8%	14.3%	0.0%			3_tachyon_unp.exe				
▶ Raypnt	39.2%	4.9%	0.0%	0.0%	90.2%	achyon_omp.exe				
▶ func@0x140150ef0	80.9%	0.0%	0.0%	15.6%	10.9%	ntoskrnl.exe				
libm_sse2_sqrt_precise	0.0%	30.8%	38.5%	0.0%	30.8%	msvcr120.dll				
▶ aullrem	46.9%	0.0%	0.0%	26.6%	26.6%	libiomp5md.dll				
▶ func@0x10013010	41.0%	16.4%	0.0%	0.0%	50.8%	gdiplus.dll				
_kmp_linear_barrier_release	33.3%	0.0%	41.7%	7.1%	17.9%	libiomp5md.dll				
libm_sse2_pow_precise	0.0%	9.1%	18.2%			msvcr120.dll				
ColorScale	30.6%	0.0%	0.0%			3_tachyon_omp.exe				
intersect_objects	20.8%	10.4%	0.0%	0.0%	100.0%	3_tachyon_omp.exe				
▶ func@0x10009c00	35.7%	23.8%	0.0%	0.0%	64.3%	gdiplus.dll				



This data is collected statistically with event multiplexing. Gray data has low confidence levels.

Microarchitecture Exploration Analysis Shows the Hardware Bottleneck

> amplxe-cl -collect uarch-exploration -- ./myapp.out



Rebuild and Compare Results

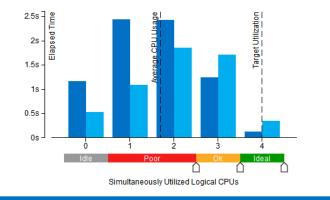


Compare	-12	× r002h	;	primes.cpp	r001hs	r000hs	primes_omp.cpp		-
🔛 Choo	ose	Results	to Co	mpare				INT	EL VTUNE AMPLIFIER XE 2017
Result 1:	r00	3ah.amplxe	2					✓ Browse	Compare
Result 2:	r00	4ah.amplxe	:					✓ Browse	X Cancel
The	se re	esults can b	e compa	red. Click the C	ompare button to	continue.			



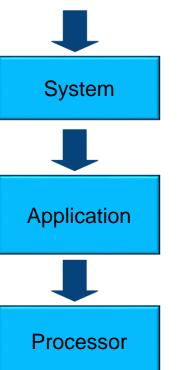
📀 CPU Usage Histogram 📑

This histogram displays a percentage of the wall time the specific number of CPUs were running





Summary



- Start with the lowest hanging fruit for performance tuning
- Use Intel[®] VTune[™] Amplifier for system and application profiling
- Hotspots, HPC Characterization, and General Exploration are good starting points
- Performance tuning is an iterative process





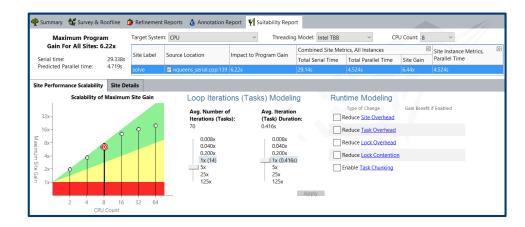


INTEL[®] ADVISOR

Agenda

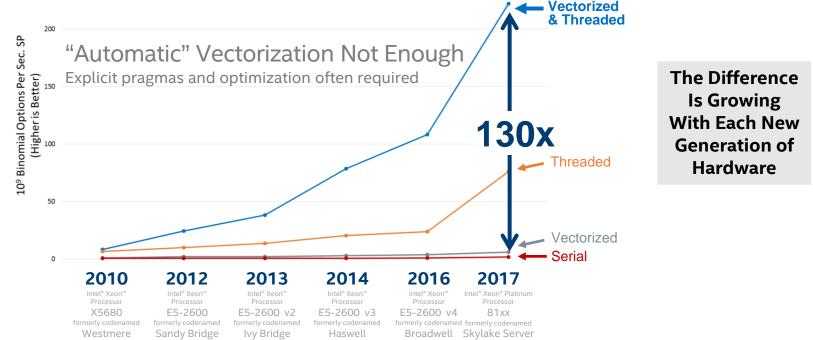
- Motivation
- Threading Advisor
 - Threading Advisor Workflow
 - Advisor Interface
 - Survey Report
 - Annotations
 - Suitability Analysis
 - Dependencies Analysis
- Vectorization Advisor & Roofline
 - Vectorization Advisor recap
 - Roofline
 - Memory Access Patterns Analysis
 - Dependencies Analysis
- Summary

E	. Constant	C-11 Cites	CHIT		FLOPS	\geq	Why No	Vector	rized Loo	ps	\gg	Trip Coun	ts 📎	Instruction Se	et A
+	Function	Call Sites and Loops	Self Time *	Туре	GFLOPS	AI	Vectorization?	Vect	Efficien	cy Gain E	VL	. Average	Call Count	Traits	
<u>ا</u> ن	[loop in main	at roofline.cpp:221] 13.562s 🗖	Scalar	3.9171	0.179	a novector directive					664	10000000		
¥.	[loop in main	at roofline.cpp:138] 7.563s 🗖	Scalar	1.7561	0.045	a novector directive					664	1000000		
	[loop in main	n at roofline.cpp:2.	7.328s 🗆	Vectorized (Body)	7.2490	0.134		AVX2	31%	1.22x	4	166	1000000	FMA; Inserts	;; l
" 🛛 🕐	[loop in main	at roofline.cpp:260] 2.656s	Vectorized (Body)	19.999 🔲	0.179		AVX2	100%	5.09x	4	166	10000000	FMA	
<u>ان</u> ا	[loop in main	at roofline.cpp:151] 1.828s 🛙	Vectorized (Body)	7.2640	0.045		AVX	100%	4.80x	4	166	10000000		
<u>ان</u> ا	[loop in main	at roofline.cpp:273] 1.813s0	Vectorized (Body)	29.306	0.179		AVX2	100%	4.73x	4	166	10000000	FMA	
<u>ان</u> ا	[loop in main	at roofline.cpp:199] 1.781s0	Vectorized (Body)	14.913	0.089		AVX2	100%	5.14x	4	166	10000000	FMA	
			▶ .												Þ
ource	Top Down	Code Analytics	Assembly	Recommendations	Why No	Vectori	zation?								
le: roof															
ne				Source		_		_		Total Time	%	Loop/F	%	Traits	1
7 🖻	for	(int i = 0; i	< ARRAY_SI	SE_1; i++)						0.406s		7.328s			
5	🖞 [loop in	main at roofl	ine.cpp:247	1											
	Vectori	zed AVX; FMA 1	oop proces	ses Float32; Floa	t64 data	type	(s) and includes H	MA; I	inserts						
	Loop wa	s unrolled by	2												
в	(
	9 AoS1 X[i] = AoS1 Y[i].a + AoS1 Y[i].b + AoS1 Y[i].b + AoS1 Y[i].b; 6922														
9		AoS1_X[i] = Ao	81_Y[i].a ·	• AoS1_Y[i].a + A	.oS1_Y[i]	.b + 1	loS1_Y[i].b + AoS1	_Y[i]	.b;	6.922s 🗧			FMA;	Inserts; Unpacl	ks



Vectorize & Thread or Performance Dies

Threaded + Vectorized can be much faster than either one alone



Benchmark results were obtained prior to implementation of recent software patches and firmware updates intended to address exploits referred to as "Spectre" and "Meltdown". Implementation of these updates may make these results inapplicable to your device or system. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks See Vectorize & Thread or Performance Dies Configurations for 2010-2016 Benchmarks in Backup. Benchmarks source: Intel Corporation.

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others. 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, & SSSE3 instruction sets & 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 reparding the specific instruction sets covered by this notice. Notice Revision #20110804



Vectorization Workflow

on Threading Workflow

rkflow

1. Survey Target

🕨 Collect 🏼 🖡 🖿 📃

1.1 Find Trip Counts and FLOPS

Collect 🖿 🛌

✓ Trip Counts
FLOPS

2. Annotate Sources

Add Intel Advisor annotations to identify possible parallel tasks and their enclosing parallel sites.

+ Steps to annotate

3. Check Suitability

Collect 片 🖿 🗔

4. Check Dependencies

Collect 🖿 📘

THREADING ADVISOR

○ Re-finalize Survey

Serial Modeling Has Multiple Benefits

Intel[®] Advisor

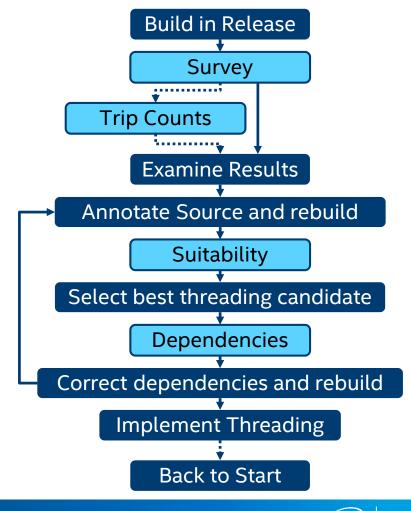
- 1) Your application can't fail due to bugs caused by incorrect parallel execution. (It's running serially.)
- 2) You can easily experiment with several different proposals before committing to the expense of implementation.
 - a) Measure performance focus on where it will pay off.
 - b) Predict scalability, load balancing and overheads.
 - c) Predict (and avoid) data races
- All of your test suites should still pass. Validate the correctness of your transformations.
- 4) You can use Advisor on partially or completely parallelized code.

Design, measure and test before implementation



Threading Advisor Workflow

- Use the **Survey** to find good potential threading sites.
 - Optionally, follow up with **Trip Counts** to find information about iteration and call counts.
- Annotate your code.
- Use **Suitability** to predict how much performance improvement the proposed threading model will create under specific, editable conditions.
- Use **Dependencies** to determine whether the proposed model is safe, and what needs to be done to correct it.



Survey Report Threading Advisor

Tip:

Survey sorts by Self Time by default. This is good for Vector Advisor, but for Threading Advisor, you may want to sort by Total Time.

- The Survey Report has lots of information, but most of it is more relevant to Vector Advisor.
- Look for outer loops or functions with high Total Time.
- In this example, setQueen has a high Total Time. It's recursive, but is originally called from a loop in Solve. That makes the loop in Solve a good potential candidate.

🌪 Summary	🖌 Surv	ey & Roofline	🍃 Refinemer	nt Re	epor	ts	💧 Annot	ation Report	₩ Suitability	Report
	unction (Call Sites and Loo	ps		đ	•	Sector Issues	Self Time	Total Time 🔻	Туре
🗧 : 🖸 🛉 setQu	leen							2.202s 🥅	38.480s	Function
🛛 🗂 [loop	in setQu	een at nqueens_s	erial.cpp:116]	Г	-		0.297s	33.3205	Scalar
∍ f _scrt_o	commor	n_main_seh						0.000sl	5.250s	Function
•				►						
Source Top	Down	Code Analytics	Assembly	As	ssista	ance	😪 Reco	mmendation	s 📮 Why No	Vectorizat
Function Call Sit	tes and I	loops				Tota	al Time %	Self Time	Total Time	Туре
Total						100.	0% 📖	0.000s1	5.250s	
_RtIUserThr	eadStar	t				100.	0% 📖	0.000s1	5.250s	Function
RtIUse	rThread	Start				100.	0% 📖	0.000s	5.250s	Function
BaseT	hreadIn	itThunk				100.	0% 📖	0.000sl	5.250s	Function
⊟_sc	rt_comn	non_main_seh				100.	0% 📖	0.000s	5.250s	Function
	main					100.	0% 📖	0.000sl	5.250s	Function
	solve					100.	0% 📖	0.000s	5.250s	Function
	DC	[loop in solve at i	nqueens_seria	al.cp	p:1	100.	0% 📖	0.000s	5.250s	Scalar
	Ξ	setQueen				100.	0% 💷	0.000s1	5.250s	Function
		🗆 🗂 [loop in set	Queen at nqu	ieen	s_s€	100.	0% 📖	0.000s	5.250s	Scalar
		_setQueen				100.	0% 💷	0.000s	5.250s	Function
		🗏 🗂 [loop i	in setQueen a	at ng	que	100.	0% 📖	0.000s	5.250s	Scalar
		-setQue	een			100.	0% 💷	0.047s	5.250s	Function
		ا 🖱 🖂	loop in setQu	leen	at	98.	5% 💷	0.031sl	5.172s	Scalar
		—s	etQueen		97.	9% 💷	0.031sl	5.141s	Function	
		(🗆 🗂 [loop in	Que	96.	1% 💶	0.016s	5.047s	Scalar	
			_setQuee	n		95.	8% 💶	0.219s1	5.031s	Function
			🗆 🗂 [lo	op i	n se	88.	1% 💶	0.031s	4.625s	Scalar
			⊟set	Que	en	87.	5% 💷	0.282s	4.594s	Function
•					•					



Annotating Your Code

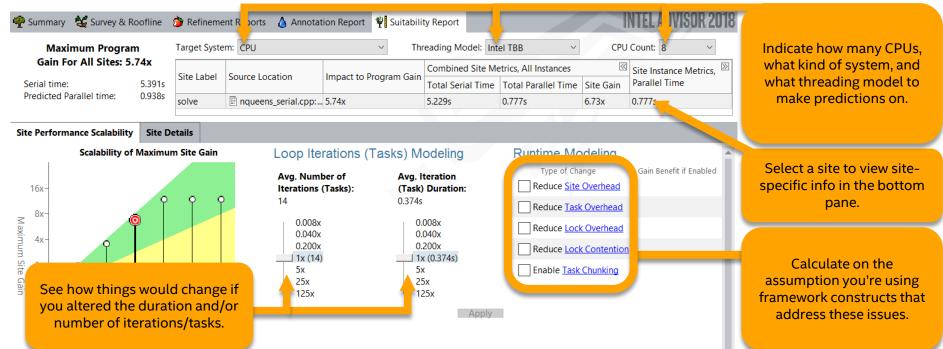
- Annotations are notes to Advisor. They are *not* parallelization commands. They do not affect the way the program itself runs.
- They mark places Advisor should treat as locks or parallel sites.
- To use annotations, you must include the appropriate header/module.

C/C++	FORTRAN	C#
 In source files where annotations are used, add: #include <advisor-< li=""> </advisor-<>	 In source files where annotations are used, add: use advisor_annotate 	 In source files where annotations are used, add: using AdvisorAnnotate;
annotate.h>	• Add	• Add the C# annotations
• Add <install_dir>/include to your include directories.</install_dir>	<install_dir>/include to your include directories.</install_dir>	definition file to your project.

• The Advisor User's Guide contains a section on Annotations with full documentation, examples, and instructions on the above if you forget.

Suitability Analysis

• Using your annotations, Advisor models how the program would behave in parallel, and predicts performance in specified hypothetical circumstances.



Optimization Notice

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

Dependencies Analysis

Threading Advisor

- This is the same analysis as in Vectorization Advisor. It works with annotations as well as selections in the survey report.
- Add lock annotations or reorganize code to resolve reported dependencies, then rerun the analysis to confirm the problem has been resolved.
 Annotation Report Strides Distribution
 Annotation Report Strides Distribution
 Access Pattern
 Max. Site Fried Composition and Strides Distribution
 Composition and Strides Distri
- Run suitability again to check that you still get good improvement.
- Once you're happy with Advisor's predictions, replace the annotations with actual parallelism and locks.

Site L	ocatio	n			Loop-Carrie	ed Depen	dencies	Strides D	istribution	n A	ccess Pattern	1	Max. Site	Footp
= 🛱 (loop i	n solve	at nqueens_s	erial.cpp:1.	BRAW:1	AWAR:1	∆ WAW	V:1 No inform	nation av	ailable N	o informatio	n available	No inform	nation
1	37 38	11			P			hree annot he body of						
1	39 40		NOTATE_SI r (int i=		solve); ardsize;	i++) {	•				_			
Dep	enden	cies Rep	oort 💡 Ree	commendat	ions									
		nd Me										Filter		
ID	۹	Type	Type Site Name Sources Modules State								Severity			
P1		Parall	el site inforn	nation	solve	nqueens_serial.cpp nqueens_serial.cpp		cpp 1_nqueens_serial.e		erial.exe 🖌 Not a pro		Error		3 item
	•	Read	after write d	ependency	solve			0 1_nqueens_s	erial.exe 🖪 New			Informat	ion	1 iten
P4	- 😌	Write	after write o	lependency	solve nqueens_serial.cpp			1_nqueens_s	l_nqueens_serial.exe 🔒 New			True		
P5	- 😌	Write after read dependency solve nqueens_serial.cpp 1_nqueens_serial.exe R New									Туре			
		write d	ependency:	Code Locat									ite infor	
ID			Description			Euro	tion V	ariable refer	Modulo		State		er write	
	-								able refer Module				er write	
± X3 ± X4	0x40		Read					rOfSolutions	1_nqueens_serial			Write aft	er read	1 item
	0x401c04 Write		write	nqueer	nqueens_serial.cpp:111		lueen ni	rOfSolutions			xe 🎙 New	Source		
± X4	0x40		Parallel site	-	s serial.cpp:						xe 🖪 New	Source		

38

Add Parallel Framework

mmary 🛛 💥 Surve	ey Report 💧 Annotation Report	📲 Suitability Repo	rt 🛛 🍅 Correctness Re
Maximum prog	gram gain [®] : 5.20x (8 CPUs, Intel	TBB Threading M	odel)
These annotated	parallel sites were detected:		
Parallel Site	Maximum	Site Gain [®]	Correctness Problems
solve (naveens a	nnotated.cpp:113) 6.5	1x	◎ 2▲0
	parallel site and task annotations arour Source Location		ming loops found durin
Consider adding	parallel site and task annotations aroun	d these time-consur	ning loops found durin ຼ
Consider adding	parallel site and task annotations arour Source Location	d these time-consur CPU Total Time	ming loops found durin গ্র 2s 2s
Consider adding Loop © <u>setQueen</u>	parallel site and task annotations aroun Source Location <u>nqueens annotated.cpp:96</u>	d these time-consur CPU Total Time 1.825	ming loops found durin
Consider adding Loop රී <u>setQueen</u> රී <u>solve</u>	parallel site and task annotations aroun Source Location <u>nqueens annotated.cpp:96</u> <u>nqueens annotated.cpp:117</u>	d these time-consur CPU Total Time 1.825 1.825	ming loops found durin

Intel[®] Advisor

- Contains overhead metrics for popular parallel frameworks
- Quickly prototype and evaluate alternatives
- Detailed help pages for popular parallel frameworks

Here is the list of source locations

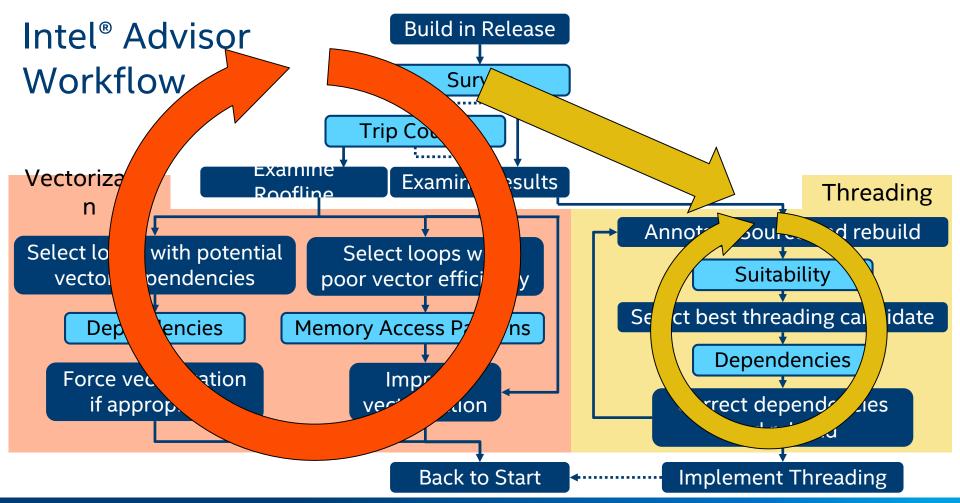
Here are templates for popular parallel frameworks

Threading Model: Intel TBB Other Intel TBB Intel Cilk Plus OpenMP

Microsoft TPL

Serial Code with Intel Advisor Annotations	Parallel Code using Intel TBB
<pre>// Locking AnnotAte_Lock_Acquire(); Body(); AnnotAte_Lock_Release();</pre>	<pre>// Locking can use various mutex types provided // by Intel TBB. For example: #include <tbb tbb.h=""> tbb::mutex g_Mutex; { tbb::mutex::scoped_lock lock(g_Mutex); Body(); }</tbb></pre>
<pre>// Do-All Counted loops, one task AlNOTATE_SITE_BEGIN(site); For (I = 0; I < N; ++) { ANNOTATE_ITERATION_TASK(task); (statement;) } ANNOTATE_SITE_END();</pre>	<pre>// Do-All Counted loops, using lambda // expressions #include <tbb tbb.h=""> tbb::parallel_for(0,N,[6](int I) { statement; });</tbb></pre>
<pre>// Create Multiple Tasks ANNOTATE ITS BEGIN(site); ANNOTATE ITASK BEGIN(task1); statement-or-task1; ANNOTATE TASK EGIN(task2); statement-or-task2; ANNOTATE TASK EGIN(task2); ANNOTATE TASK EDIN();</pre>	<pre>// Create Multiple tasks, using lambda // expressions #include <tbb tbb.h=""> tbb::parallel invoke([6](statement-or-task1;), [6](statement-or-task2;));</tbb></pre>





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

40

'inte



VECTORIZATION ADVISOR & ROOFLINE

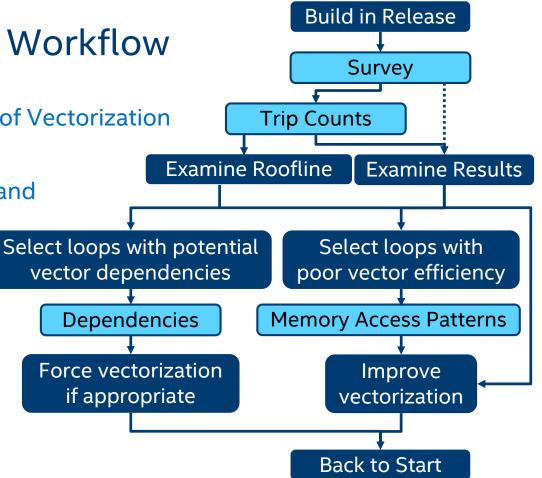
⊖ Re-finalize Survey

Collect 🖿 📐

2.2 Check Memory Access P...

Vectorization Advisor Workflow

- **Survey** is the bread and butter of Vectorization Advisor! All else builds on it!
- Trip Counts adds onto Survey and enables the Roofline.
- **Dependencies** determines whether it's safe to force a scalar loop to vectorize.
- Memory Access Patterns diagnoses vectorization inefficiency caused by poor memory striding.



Survey Vectorization Advisor

Tip:

For vectorization, you generally only care about loops. Set the type dropdown to "Loops".

Efficiency is importentlup

fficiency=100%

Vec. Length

The black arrow is 1x. Gray means you got less than that. Gold means you got more. You want to get this value as high as possible!

Function/Loop Icons

- **F** Scalar Function
- Vector Function
 Scalar Loop
- Vector Loop

Vectorizing a loop is usually best done on innermost loops. Since it effectively divides duration by vector length, you want to target loops with high self time.

Function Call Sites and Loops		Q Vector Issues	Self	Total	Tuno	Why No	Vector	rized Loops	Þ	
Function can sites and coops	œ	a vector issues	Time	Time	Туре	Vectorization?	Vect	Efficiency	Gain	VL .
🗵 🖱 [loop in main at example.cpp:38]		9 1 Assumed depend	0.391s 💳	0.391s 🗖	Scalar	vector depen				
🗵 🗂 [loop in main at example.cpp:64]		9 1 Possible inefficien	0.297s 🔲	0.297s 🗖	Vector		AVX2	2%	0.37x	16
🕀 🗂 [loop in main at example.cpp:51]		9 1 Possible inefficien	0.094s 🛙	0.094s I	Vector	1 vectorizatio	AVX2	8%	1.23x	16
🕂 🗂 [loop in main at example.cpp:26]			0.030s I	0.030s1	Vector		AVX2	100%	7.98x	8
[loop in main at example.cpp:14]		S Assumed depend	0.000s1	0.000s1	Scalar	vector depen				
[O [loop in main at example.cpp:23]			0.000sl	0.030s1	Scalar	🖬 inner lou 🧹 w				

Expand a vectorized loop to see it split into body, peel, and remainder (if applicable). Advisor *advises* you on potential vector issues. This is often your cue to run MAP or Dependencies. Click the icon to see an explanation in the bottom pane. The Intel Compiler embeds extra information that Advisor can report in addition to its sampled data, such as why loops failed to vectorize.

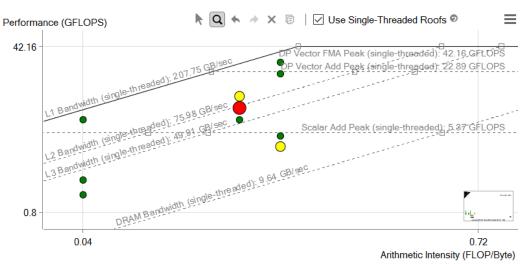
Optimization Notice

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

What is a Roofline Chart?

A Roofline Chart plots application performance against hardware limitations.

- Where are the bottlenecks?
- How much performance is being left on the table?
- Which bottlenecks can be addressed, and which should be addressed?
- What's the most likely cause?
- What are the next steps?



Roofline first proposed by University of California at Berkeley: <u>Roofline: An Insightful Visual Performance Model for Multicore Architectures</u>, 2009 Cache-aware variant proposed by University of Lisbon: <u>Cache-Aware Roofline Model: Upgrading the Loft</u>, 2013

Roofline Metrics

Roofline is based on Arithmetic Intensity (AI) and FLOPS.

- Arithmetic Intensity: FLOP / Byte Accessed
 - This is a characteristic of your algorithm



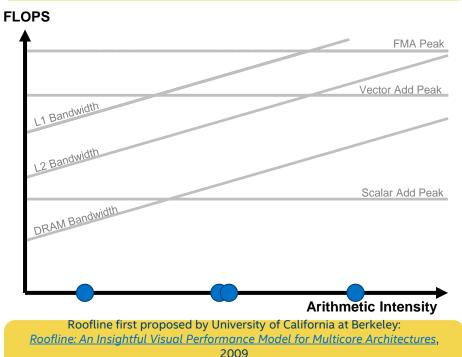
- FLOPS: <u>Fl</u>oating-Point <u>Op</u>erations / <u>Second</u>
 - Is a measure of an implementation (it achieves a certain FLOPS)
 - And there is a maximum that a platform can provide



Cache-Aware Roofline Concept

- Prior to collecting data, Advisor runs quick benchmarks to measure hardware limitations.
 - Computational limitations
 - Memory Bandwidth limitations
- These form the performance "roofs".
- Loops and functions have algorithms and therefore a specific AI.
- Their performance in FLOPS is also measured.
- Optimization changes performance. The goal is to go as far up as possible.

Video Available: Roofline Analysis in Intel® Advisor 2017



Cache aware variant proposed by Technical University of Lisbon: <u>Cache-Aware Roofline Model: Upgrading the Loft</u>, 2013

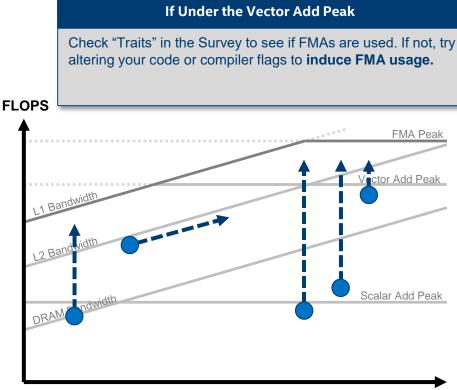
46

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 Al.



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.



Memory Access Patterns Analysis Collecting a MAP

- If you have low vector efficiency, or see that a loop did not vectorize because it was deemed "possible but inefficient", you may want to run a MAP analysis.
- Advisor will also recommend a MAP analysis if it detects a possible inefficient access pattern.

Vector Issues

Possible inefficient memory access patterns present

- Memory access patterns affect vectorization efficiency because they affect how data is loaded into and stored from the vector registers.
- Select the loops you want to run the MAP on using the checkboxes. It may be helpful to reduce the problem size, as MAP only needs to detect patterns, and has high overhead.
 - Note that if changing the problem size requires recompiling, you will need to re-collect the survey before running MAP.



Memory Access Patterns Analysis Reading a MAP

Videos Available: Stride and Memory Access Patterns and Memory Access 101

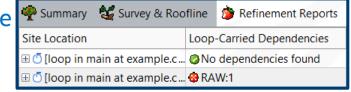
- MAP is color coded by stride type. From best to worst:
 - Blue is unit/uniform (stepping by 1 or 0)
 - Yellow is constant (stepping a set distance)
 - **Red** is variable (a changing step distance)
- Click a loop in the top pane to see a detailed report below.
 - The strides that contribute to the loop are broken down in this table.
 - Important information includes the size of the stride, the variable being accessed, and the source.
 - Not all strides will come from your code!

🌪 Su	Imma	ary 💥 S	Gurvey & Roofline	🍅 Refinemen	t Reports				
Site Lo	ocati	on	Strides Distrib	Access Pa Ma	x. Site Footprint	Recommend	ations		
ا) 🖰 🖽	оор	in main	76% / 0% / <mark>24</mark> 1	Mixed stri 64K	B				
🕀 🗂 [оор	in main	76% / 0% / <mark>24</mark>	Mixed stri 64K	B				
🕀 🝊 (I	оор	in main	70% / 6% / <mark>24</mark> 1	Mixed stri 564	MB				
🕀 🗂 [оор	in main	100% / 0% / 0 /	All unit str 70K	В				
H 🖸 [ا	оор	in main	<mark>33% / 67% / 0</mark> 1	Mixed stri 616	MB	1 Inefficient memory access particular			
•									
Mem	ory /	Access Pat	terns Report De	pendencies Rep	ort 💡 Recomm	endations			
ID	8	Stride	Туре	Source	Modules	Nested Func	Variable references		
± P1	-	36000	Constant stride	stride.cpp:49	stride.exe		tableA, tableB		
± P2	14	36000	Constant stride	stride.cpp:49	stride.exe		results		
± P7	ī		Parallel site info	. stride.cpp:47	stride.exe				
P19		0	Uniform stride	stride.exe:0x	stride.exe	_svml_atan4			
P28			Uniforn strike	sur l di om	singly the provided	VAVAVA			
¥1.		-12: -8:	Variable stride	fum die d	svml_dispmd.dll	švml_atan2			

Dependencies Analysis

Vectorization Advisor

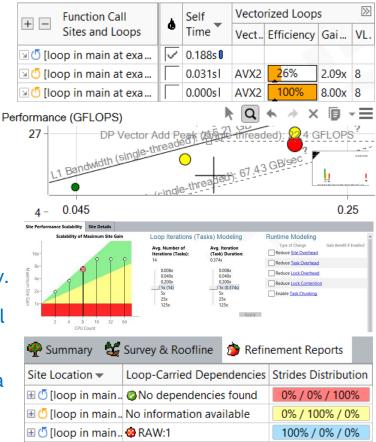
- Generally, you don't need to run Dependencies analysis unless Advisor tells you to. It produces recommendations to do so if it detects:
 - Loops that remained unvectorized • because the compiler was playing it safe with autovectorization.
 - Outer loop vectorization opportunities ٠
- Use the survey checkboxes to select which loops to analyze.
- Recommendation: Confirm Confidence *Need More dependency is real Data There is no confirmation that a real (proven) dependency is present in the loop. To confirm: Run a Dependencies analysis.
- Recommendation: Check Confidence: @Low dependencies for outer loop It is not safe to force vectorization without knowing that there are no dependencies. Disable inner vectorization before check Dependency. To check: Run a Dependencies analysis.
- If no dependencies are found, it's safe to force vectorization.
- Otherwise, use the reported variable read/write Summary Survey & Roofline information to see if you can rework the code to eliminate the dependency.



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

Summary

- Survey Find the most promising sites for threading, see
 the meat of the vectorization information, and get
 recommendations from Advisor.
- Trip Counts & FLOPS Add to your Survey report to help
 fine-tune vector efficiency and capability, as well as unlock the powerful Roofline to visualize your bottlenecks and help direct your efforts.
- **Suitability** Predict how well your proposed threading model will scale under certain conditions quickly and easily.
 - **Dependencies** Prove or disprove the existence of parallel dependencies and learn how to fix them.
- Memory Access Patterns See how you traverse your data and how it affects your vector efficiency and cache bandwidth usage.



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



Software

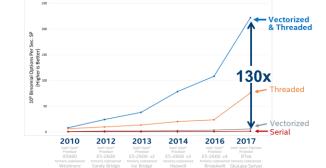
Configurations for 2010-2017 Benchmarks

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

Performance measured in Intel Labs by Intel employees

Platform Hardware and Software Configuration



		Unscaled Core	Cores/	Num	L1 Data	L2	L3		Memory	Memory	H/W Prefetchers	нт	Turbo		0/S		
	Platform	Frequency	Socket	Sockets	Cache	Cache	Cache	Memory	Frequency	Access	Enabled	Enabled	Enabled	C States	Name	Operating System	Compiler Version
WSM	Intel® Xeon™ X5680 Processor	3.33 GHZ	6	2	32K	256K	12 MB	48 MB	1333 MHz	NUMA	Y	Y	Υ	Disabled	Fedora 20	3.11.10-301.fc20	icc version 17.0.2
SNB	Intel® Xeon™ E5 2690 Processor	2.9 GHZ	8	2	32K	256K	20 MB	64 GB	1600 MHz	NUMA	Y	Y	Υ	Disabled	Fedora 20	3.11.10-301.fc20	icc version 17.0.2
IVB	Intel® Xeon™ E5 2697v2 Processor	2.7 GHZ	12	2	32K	256K	30 MB	64 GB	1867 MHz	NUMA	Y	Υ	Y	Disabled	RHEL 7.1	3.10.0-229.el7.x86_64	icc version 17.0.2
HSW	Intel® Xeon™ E5 2600v3 Processor	2.2 GHz	18	2	32K	256K	46 MB	128 GB	2133 MHz	NUMA	Y	Υ	Υ	Disabled	Fedora 20	3.15.10- 200.fc20.x86_64	icc version 17.0.2
BDW	Intel® Xeon™ E5 2600v4 Processor	2.3 GHz	18	2	32K	256K	46 MB	256 GB	2400 MHz	NUMA	Y	Y	Υ	Disabled		3.10.0-123. el7.x86_64	
BDW	Intel® Xeon™ E5 2600v4 Processor	2.2 GHz	22	2	32K	256K	56 MB	128 GB	2133 MHz	NUMA	Y	Y	Υ	Disabled	CentOS 7.2	3.10.0-327. el7.x86_64	icc version 17.0.2
SKX	Intel® Xeon® Platinum 81xx Processor	2.5 GHz	28	2	32K	1024K	40 MB	192 GB	2666 MHz	NUMA	Y	Y	Y	Disabled	CentOS 7.3	3.10.0- 514.10.2.el7.x86_64	icc version 17.0.2

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 information go to http://www.intel.com/performance

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others. 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, & SSSE3 instruction sets & 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 reparding the specific instruction sets covered by this notice. Notice Revision #20110804



Legal Disclaimer & Optimization Notice

Benchmark results were obtained prior to implementation of recent software patches and firmware updates intended to address exploits referred to as "Spectre" and "Meltdown". Implementation of these updates may make these results inapplicable to your device or system.

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>.

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.

Copyright © 2018, Intel Corporation. All rights reserved. Intel, Xeon, Xeon Phi, Core, VTune, 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

Legal Disclaimer & Optimization Notice

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>.

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.

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