

INTEL® VTUNE[™] AMPLIFIER WORKSHOP



Getting Setup Intel[®] VTune[™] Amplifier Introduction matrix sample Intel[®] Optane[™] DC Persistent Memory Profiling **PMDK Sample Platform Profiler** Server Sample Workloads (not included)





1. License:

https://registrationcenter.intel.com/en/forms/?productid=3218

- 2. Install VTune Amplifier (USB or Download Installer)
- 3. Copy "Results" folder from USB to laptop
- 4. Pass USB to neighbor





INTEL® VTUNE[™] AMPLIFIER INTRO

Faster, Scalable Code, Faster

Intel[®] VTune[™] Amplifier Performance Profiler

Accurate Data - Low Overhead

CPU, GPU, FPU, threading, bandwidth...

Meaningful Analysis

- Threading, OpenMP region efficiency
- Memory access, storage device

Easy

- Data displayed on the source code
- Easy set-up, no special compiles

"Last week, Intel[®] VTune[™] Amplifier helped us find almost 3X performance improvement. This week it helped us improve the performance another 3X."

Claire Cates Principal Developer SAS Institute Inc.

Gro	ouping: Function / Call Stack					~ 🔨	Q	9.a	CPU Time
				CPU Time			«	^	Viewing 1 of 18 selected stack(s)
	Function / Call Stack	Serial CPU Time	Effectiv	e Time by Utilization 🔻 👘	Spin Time	Overhead Time	30		31.9% (0.723s of 2.268s)
			🛚 Idle 🔋 Po	oor 🔋 Ok 🛢 Ideal 🔋 Over	Spin nine	Overnead nine		_	3_tachyon_omp.exelgrid_intersect - grid.cpp
g	rid_intersect	Os	2.268s		Os		0s		3_tachyon_omp.exelintersect_objects+0x18 - intersect
S	phere_intersect	0s	1.854s		Os		0s		3_tachyon_omp.exelshader+0x324 - shade.cpp:132
» G	GdiplusStartup	Os	1.203s		Os		0s		3_tachyon_omp.exeltrace+0x2e - trace_rest.cpp:71
► C	CreateWindowExA	Os	0.5165		0s		Os		3_tachyon_omp.exe!render_one_pixel+0x8a - tachyon
P 11	kegGetValuevv	US	0.4385		US		US		3_tachyon_omp.exeIthread_trace\$omp\$parallel@141+
	unc@0x10030200	05	0.3605		05		0s		libiomp5md.dll![OpenMP dispatcher]+0x79 - kmp_runtin
9	GdipDrawlmagePointRectl	03	0.0565		05		0s	-	libiomp5md.dlll_kmp_fork_call+0xf6a - kmp_runtime.c
<	>	<						>	3 tachyon own exeltbread trace+0x15b - tachyon own
	D: ➡ = # #	0s 1s	2s	s 3s	4s	5s	_	(6s 7s Scale Markers:
pee	WinMainCRTStartup (TID: 1								
THE	OMP Worker Thread #1 (TID				a setter				to Barrier Segment
-	OMP Master Thread #0 (TID:								
F	OMP Worker Thread #2 (TID					R		1	
ŀ	OMP Worker Thread #3 (TID						TT I		Running
-	func@0v10068420 (TID: 115							-	
+	Through (TD) 45504)						_	_	Spin and Overhead
-	Inread (IID: 15524)								CPU Sample
-	func@0x10068430 (TID: 203				-		_	_	L 41 Transitions
	Thread (TID: 22124)								CPU Utilization
	func@0x100570b0 (TID: 226								

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



EDE			INTEL VTUNE AMPLIFIER 2019	HOW		
Local Host		Find your and Hotspots Want to find out where your app spends time and optimize your algorithms?	lysis direction Microarchitecture Want to see how efficiently your code is using the underlying hardware?	Memory Acces Measure a set of metrics to identify mem example, specific for NUMA architecture	bory access related issues (for es). This analysis type is based	
Launch Application		Basic Hotspots	General Exploration	on the hardware event-based sampling c CPU sampling interval, ms	ollection. <u>Learn more</u> (F1)	
pecify and configure your analysis target: an application xecute. Press F1 for more details.	n or a script to	Advanced Hotspots	Memory Access	 Analyze dynamic memory objects Minimal dynamic memory object size to 1024 	track, in bytes	
localdisk/temp/matrix/linux/matrix.gcc	2	Consumption		 Evaluate max DRAM bandwidth 		
plication parameters:		Paral	lelism	Analyze OpenMP regions		
] Use application directory as working directory /orking directory: /localdisk/jmarusar/temp/matrix/linux Advanced	C (Concurrency L	Locks and Waits	 Details Analyze I/O waits Collect I/O API data No Collect stacks 	Ţ	

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

intel

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			
	(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



Example: Hotspots Analysis Summary View

🗸 🔛 Collection Log \varTheta Analysi	s Target 🕺 Analysis Type	🖞 Summary 🥈
Solution State	5.554s	
	10.504s	
Instructions Retired:	21,698,000,000	
CPI Rate [®] :	1.257 🏲	
CPU Frequency Ratio	⁽²⁾ : 1.041	
Total Thread Count:	9	
Paused Time [®] :	0s	

Top Hotspots

 (\checkmark)

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 [©]
grid intersect	3_tachyon_omp.exe	5.539s
sphere intersect	3_tachyon_omp.exe	3.247s
func@0x1002e59d	libiomp5md.dll	0.148s
<u>shader</u>	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

Bottom-up

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.

Optimization Notice

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



Example: Threading Analysis Bottom-up View

A	Threading Hotspots by CPU Utilization - 🕐												
A	nalysis Configuration Colle	ection Log	Summary Bottom-u	p Caller/Callee	Top-down Tree F	Platform							
Gro	ouping: Function / Call Stack							~ 🔨	ρ 🖫	CPU Time			~
	Function / Call Stack		Serial CPU Time	Effective Time	CPU Tin	ne Spin Time	Overhead Time	Wait Time by Utilizatio	on al O	Viewing ∢ 1 of 18	 selected stack(31.9% 	s) 6 (0.723s of 2.26	8s)
⊫ g	rid_intersect		0s	2.268s	Ok Ideal Uver	0s	Os			3_tachyon_omp.exe!grid_intersect - grid.cpp 3_tachyon_omp.exe!intersect_objects+0x18 - intersect.cpp:108			
▶ s	phere_intersect		0s	1.854s		Os	Os			3_tachyon_omp.e	xelshader+0x324	4 - shade.cpp:/	132
▶ C	GdiplusStartup CreateWindowExA		0s	1.203s		0s	0s		-	3_tachyon_omp.e	xe!trace+0x2e -	trace_rest.cpp:	:71
⊫ F	RegGetValueW		03	0.438s		03 05	0s			3_tachyon_omp.e	xe!render_one_p xelthread_traces	Somp\$parallel@	D141+0x175 - tachvon om
⊩ f	unc@0x1003d2b0		0s	0.360s		Os	Os			libiomp5md.dll![OpenMP dispatcher]+0x79 - kmp_runtime.cpp:7137			
▶ g	rid_bounds_intersect		0s	0.126s		0s	0s			libiomp5md.dll!kmp_fork_call+0xf6a - kmp_runtime.cpp:1031			time.cpp:1031
⊫ C ⊫ n			05	0.0565		US OS	US Os			libiomp5md.dll![OpenMP fork]+0x5b - kmp_csupport.cpp:341			port.cpp:341
<		>	<	0.0100					>	3_tachyon_omp.e	xeltroop_ohm LO	FUX15D - tachyo	on_omp.cpp:141
	p: 🕇 🗕 🖝 🖻	- 2s	2.5s	3s	3.5s	4s 4.	5s 5s 5s	5.5s	6s	6.5s	7s	7.5s	Scale Markers: ^
ead	WinMainCRTStartup (TID: 1										للماه م	^	Region Instance
Ê	OMP Worker Thread #1 (TID												to-Barrier Segment
	OMP Master Thread #0 (TID:								<u> </u>	i ' _ ' () ''			Thread V
	OMP Worker Thread #2 (TID							" "	<u>'</u>				Running
	OMP Worker Thread #3 (TID												Waits
	func@0x10068430 (TID: 115												CPU Time
	Thread (TID: 15524)	_											CPU Sample
	func@0x10068430 (TID: 203											~	☐ ↓† Transitions
	CPU Utilization	<											CPU Utilization

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



Identifying and Diagnosing Inefficiency

Microarchitecture Analysis

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

- Microarchitecture Exploration (previously General Exploration) is a hardware events analysis. It is preconfigured to sample the appropriate events on your architecture and calculates the proper metrics from them.
- Potential tuning opportunities are highlighted in pink.
- To check the efficiency of a hotspot, look at the Retiring metric. If it's less than the expected number for your application type, it's probably inefficient.
 - Hotspots with high retiring values may still have room for improvement.

Арр Туре	Expected		alysis Target 🔺 Ana	lysis Type	🛍 Summary 💊 Botto	m-up 🚯 Event Cou	Int 🖻 Platform				
Client/	20-50%	Grouping: Function / Call S	ping: Function / Call Stack								
Desktop		Function / Call Stack	Instructions Retired	CPI Rate	Front-End Bound 🚿	Bad Speculation »	Back-End Bound 🔊	Retiring 🔌 🔨			
Serverl	10_20%	initialize_2D_buffer	85,219,200,000	0.266	0.5%	0.0%	0.0%	100.0%			
Databasa	10-30 %	grid_intersect	10,963,200,000	0.706	4.6%	15.1%	46.4%	33.9%			
Database		sphere_intersect	10,946,400,000	0.601	2.1%	1.6%	47.5%	48.8%			
Distributed		grid_bounds_intersect	480,000,000	1.105	13.0%	2.2%	52.3%	32.5%			
HPC	30-70%	tri_intersect	216,000,000	0.789	0.0%	20.2%	39.3%	40.5%			

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



10

Categorizing and Correcting Inefficiencies Microarchitecture Exploration Analysis

- Intel[®] VTune[™] Amplifier has hierarchical expanding metrics categorized by the four slot types.
- You can expand your way down, following the hotspot, to identify the root cause of the inefficiency.
 - Sub-metrics highlight pink on their own merits, just like top level metrics.
- Hovering over a metric produces a helpful, detailed tooltip (not shown).
 - There are tooltips on Summary tabs too: hover over any ② icon.



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

Matrix Sample

matrix\vc14\VTune Amplifier Results\matrix\matrix.amplxeproj

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





PERSISTENT MEMORY BANDWIDTH USE CASE STUDY

Goal

We now know performance characteristics for Intel® Optane™ DC persistent memory DIMMs



Would like to write memory benchmark achieving max persistent memory bandwidth as close to the limit as possible



Algorithm

Use triad kernel similar to the one used is well-known stream benchmark Original code:

```
for (j = 0; j < REPEATS; j++)
{
    #pragma omp parallel for
    for (i = 0; i < size; i++)
    {
        D_RW(c)[i] = multiplier * D_RO(a)[i] + D_RO(b)[i];
    }
</pre>
```



Initial VTune Amplifier Results (ma_orig)



- Bandwidth peaks at about 12 GB/s. Much lower than expected
- Recall that write bandwidth is much lower for persistent memory. Could be write-limited performance.
- Let's try to avoid writing to persistent memory



Read-only Persistent Memory

Allocated array 'c' in DRAM instead of persistent memory (i.e. use regular malloc instead of PMDK API for it)

```
for (j = 0; j < REPEATS; j++)
{
    #pragma omp parallel for
    for (i = 0; i < size; i++)
    {
        c[i] = multiplier * D_R0(a)[i] + D_R0(b)[i];
    }
}</pre>
```



VTune Amplifier result for read-only persistent memory (ma_read_only)



- A significant improvement bandwidth now peaks at 25 GB/s
- Now let's examine the code more precisely in VTune Amplifier



VTune Amplifier result for read-only persistent memory

Grouping: Function / Call Stack		
Function / Call Stack	CPU Time 🔻	Memory
main\$omp\$parallel_for@63	10.876s	
pmemobj_direct_inline	1.661s 📕	
main\$omp\$parallel_for@63	1.661s 🛑	
pmemobj_direct_inline	1.141s	
main\$omp\$parallel_for@63	1.141s 📒	
_INTERNAL_25src_kmp_barrier_0	0.922s 📒	

We see some 'pmemobj_direct_inline' functions called from our main loop taking some time. What are these?



VTune Amplifier result for read-only persistent memory

		0x40180c		Block 23:
for $(j = 0; j < \text{REPEATS}; j++)$		0x40180c	66	xor r9d, r9d
{		0x40180f		Block 24:
#pragma omp parallel for		0x40180f	66	mov edi, dword ptr [r15+r13*4]
for (i = 0; i < size; i++)	0.67	0x401813	66	imul edi, dword ptr [rsp+0x30]
{	0.00	0x401818	66	add edi, dword ptr [r9+r13*4]
$c[i] = multiplier * D_RO(a)[i] + D_RO(b)[$	10.20	0x40181c	66	mov dword ptr [rbx+r13*4], edi
}		0x401820	64	inc r13
}		0x401823	65	cmp r13, qword ptr [rsp+0x8]
		0v401828	65	ile 0x40172d <block 6=""></block>

- The functions are from the D_RO macro
- This prevents compilers from vectorizing the code as can be seen in the assembly



0

6 0

Move D_RO out of the loop

```
const int* _a = D_R0(a);
const int* _b = D_R0(b);
for (j = 0; j < REPEATS; j++)
{
    #pragma omp parallel for
    for (i = 0; i < size; i++)
    {
        c[i] = multiplier * _a[i] + _b[i];
    }
}
```

- Did a simple modification by moving D_RO out of the main loop
- Let's run VTune and see what this changed



VTune Amplifier result w/o D_RO (ma_vect_default)



- Peak bandwidth grew a little bit to 26 GB/s
- Let's examine the code





VTune Amplifier result w/o D_RO (ma_vect_default)

for (j = 0; j < REPEATS; j++)		0x4018c3	68	psrlq xmm1, 0x20	
{		0x4018c8		Block 17:	
#pragma omp parallel for	0.171	0x4018c8	68	movdqu xmm3, xmmword ptr [r10-	0.036s
for (i = 0; i < size; i++)		0x4018ce	68	movdqa xmm4, xmm2	5.148s
{		0x4018d2	68	pmuludq xmm4, xmm3	0.107s
c[i] = multiplier * _a[i] + _b[i];	13.391	0x4018d6	68	psrlq xmm3, 0x20	0.077s
}		0x4018db	68	pmuludq xmm3, xmm1	0.030s
}		0x4018df	68	pand xmm4, xmm0	0.075s
		0x4018e3	68	nslla ymm3 0y20	0.090s

- As expected compiler was able to vectorize the code
- But it uses SSE
- Let's rebuild with -xCORE-AVX2 and see if using wider vectors will help



VTune Amplifier result for AVX2 vectorization (ma_vect_avx256)



Bandwidth now peaks at more than 27 GB/s

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



VTune Amplifier result for AVX2 vectorization (ma_vect_avx256)

	_	0x401853	65	<u>jb 0x40183f <block 13=""></block></u>	
for (j = 0; j < REPEATS; j++)		0x401855		Block 14:	
{		0x401855	65	vpbroadcastd ymm0, dword ptr [rsp]	
#pragma omp parallel for	0.325s	0x40185b		Block 15:	
for (i = 0; i < size; i++)	_	0x40185b	68	vpmulld ymm1, ymm0, ymmword ptr [r10+r8*4]	0.03
{	10.011	0x401861	68	vpaddd ymm2, ymm1, ymmword ptr [rdi+r8*4]	6.84
<pre>c[i] = multiplier * _a[i] + _b[i];</pre>	12.911s	0x401867	68	vmovdqu ymmword ptr [r9+r8*4], ymm2	6.02
}		0x40186d	65	add r8, 0x8	0.32
}		0x401871	65	cmp r8, r15	
		0.401074	65	Sh 0+4010Eh ZDlock 1EN	

We can confirm that the loop is now uses AVX instructions (256-bit YMM registers)

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





PLATFORM PROFILER DEMO



See Platform Profiler in Action

ent: sample hos-20140326-1	250 digeren ke-Tûrsin 🛢		Select Very	Dererv	3
		-			
	i mentente m	and formular 1	(*************************************	****	
5	5000		5004		
	en l				
	Time Range Select	er (Danatum + Birts)	0.5 +		
-	Tree Barge Saler via	ar (bassus i kris) bis	103 ti	-	
	Tres large failer ain	ar Surator - Brit Ma	03.1 ain	uiu.	
Overview OVerview	Time Range Salect	ar thasann a bro	alan alan	nin.	_
Overview OVerview OURsels	Time Barge Sales	ar durator - b e t	ana	10 100 • 1000 •	021
United States	Timo Bargo Salan Sila R - #10012 - #10041	ar bastor - i est	ougliper Reality	100 1001 • 1001 • 1	007 1 907 (Marine 10 1 10 1
Overview OUtwala In Internet In Internet In Internet In Internet	The Arge Select	n bastor () e (ada ada oogfgot Herics	40. 1447 - 1447 - 14	00/1 900/1000 101/1000 101/1000
Overview SUBarlo	7 mu Aarqu dalar 300a 41 - 10042 - 100441		ada oogfgort Renics	40 100 - 100 - 100	
Dvervlew SUBasis	Tree Aurope Select		ada ada ada ada ada ada ada ada ada ada		OV 1 Noveland Internetion Internetion Internetion
United States	Tree Aurop dated		outon outon outon outon outon outon outon outon outon outon outon outon outon outon	60x	
Diverview SPUMarks	Tres karge feler of a 5 5042 • 5042 • 6 • 6		Contractions	000 1000 • 1000 • 1000	
COVERVIEW COUNTS COUNTS COUNTS CONTENTENT CONT	Time Auropa Select Select 97 - 100421 - 100421 		Alt a adda ong/ippet Wetrics There fragment of "Instrument of Instrument of I		

<page-header>

Matrix Multiplication

This sample was collected using an application derived from the Intel® Math Kernel Library Matrix Multiplication C Sample. The sample is not heavily CPU-bound and demonstrates efficient memory usage and throughput. However, the application is only running on a few cores and could benefit from parallelization. The sample was collected on an Ubuntu* Linux system with 64GB of DRAM and 36 cores.

File Copy

This sample was collected while continuously copying 30+ randomly sized files for five minutes on socket 1 followed by five minutes on socket 0. It shows a marked increase in efficiency when using socket 0 and demonstrates that applications that use a lot of file operations can benefit from using the socket closest to storage. The sample was collected on an Ubuntu* Linux system with two sockets and one NVMe* disk.

<section-header><section-header><section-header><section-header><section-header><text><text><text><text><section-header><section-header><section-header><section-header><section-header>

Memory Sample

This sample monitors a memory traversal program that walks a 32GB allocation in varying steps. The same program is run five times using different core combinations and has a 10 second delay between each run. The sample was collected on an Ubuntu* Linux system with 64GB of DRAM and 200GB swap space.

