Boost CPU Performance with Intel® Vtune[™] Profiler Jennifer DiMatteo (Intel®)



Legal Notices and Disclaimers

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at [intel.com].

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

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.

Results have been estimated or simulated using internal Intel analysis or architecture simulation or modeling, and provided to you for informational purposes. Any differences in your system hardware, software or configuration may affect your actual performance.

Intel, Core and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.

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

© Intel Corporation.





Agenda

- Introduction to Intel[®] VTune[™] Profiler
- Tachyon Ray Tracing Sample
- Hotspots Collection
- Threading Analysis
- Conclusion
- References

Intel[®] VTune[™] Profiler

Advanced sampling profiler allows you to quickly identify CPU bottlenecks causing slow frames and tasks.

Hotspots Hotspots by C	PU Utiliza	tion 🔻	3 10	
Analysis Configuration Collect	ion Log	Summary	Bottom-up	Caller/Callee
 Elapsed Time[®]: 2 CPU Time[®]: Instructions Retired: Microarchitecture Usag Total Thread Count: Paused Time[®]: 	2 8.691s 145,5 3e [©] :	42.63 4 87,400,00 39.6 5 0	ls 0 % <mark>ሾ of Pipelin</mark> 1 ⁰s	e Slots
Frame Count: Top Hotspots This section lists the most an hotspot functions typically r	ctive functi results in in	91 ons in your aproving ov	4 application. O /erall applicatio	ptimizing these in performance.
Function		Mod	ule	CPU Time ^③
Enlighten::Impl::GetProbeIr	nterpolants		unityplayer.dl	5.304s
Enlighten::SolveProbesL2			unityplayer.dl	3.856s
func@0x1800ab2a0		mone	o-2.0-bdwgc.dl	0.955s
Enlighten::SolveDirectional	Irradiance	<1>	unityplayer.dl	0.757s
RtICopyMemory			ntoskrnl.exe	0.536s
[Others]				31.226s
 Top Tasks This section lists the most ar Task Type 	ctive tasks Task Time	in your app ⑦ Task C	lication.	
Idle	160.07	5s 1,1	76,134	
Semaphore.WaitForSignal	133.94	2s	42,906	
Audio.Thread	25.27	4s	6,937	
PlayerLoop	22.96	3s	913	
Camera.Render	15.37	7s	3,656	
[Others]	161.08	9s 2,6	28,856	
Hotspo	ot /	An	alv	sis

Identifies functions consuming the most CPU time



Thread Performance

Visualize thread behavior to quickly identify concurrency problems

Grouping: Task Domain / Task Type / Function / Call Sta	ack				• 🔨	ρ	Q ₄₀
	Microarchitecture Usage						
lask Domain / lask lype / Function / Call Stack	CPU Time V	Instructions Retired	Microarchitecture Usage CPI Rate		lask Time		
▼ UE4Domain	65.847s	211,632,115,937	29.8%	0.990	155.2	52s	
FDeferredShadingSceneRenderer_Render	8.956s 📒	20,558,580,980	23.8%	1.350	15.5	75s	
FDeferredShadingSceneRenderer_InitViews	1.822s	4,945,551,288	26.5%	1.178	3.3	40s	
FSceneRenderer_ComputeViewVisibility	1.582s	4,621,327,612	26.9%	1.112	2.9	03s	
UWorld_Tick	1.463s	5,770,821,626	28.6%	0.887	4.1	09s	
FCompression_UncompressMemory	1.187s	4,825,032,389	60.5%	0.712	1.6	92s	
FScene_UpdateAllPrimitiveSceneInfos	0.580s	1,380,673,014	18.9%	1.356	0.8	51s	
FScene_AddPrimitiveSceneInfos	0.558s	1,331,746,702	18.2%	1.348	0.8	15s	
FScene_AddPrimitiveSceneInfoToScene	0.535s	1,312,085,470	18.5%	1.328	0.7	95s	
FDeferredShadingSceneRenderer_RenderLights	0.333s	673,865,649	22.1%	1.554	0.5	64s	
Slate::Tick	0.219s	203,536,533	30.3%	3.053	0.6	35s	
▶ FViewport_Draw	0.166s	43,686,757	20.5%	2.930	0.2	91s	
Slate::DrawWindows	0.164s	181,458,102	34.6%	2.761	0.5	39s	
FAudioDevice_Update	0.159s	37,496,378	59.6%	2.715	0.2	07s	
FDeferredShadingSceneRenderer_InitViewsPossibl	0.147s	206,229,370	27.9%	2.055	0.2	70s	
Slate::DrawWindow_RenderThread	0.112s	204,785,341	35.5%	1.737	0.1	72s	
FSceneRenderer_InitDynamicShadows	0.103s	152,822,069	27.6%	1.938	0.1	88s	
▶ Frame 0	0.085s	199,353,144	22.4%	1.395	1.3	40s	
Slate::DrawWindow	0.078s	24,953,960	16.6%	9.499	0.1	26s	
Frame 324	0.072s	271,605,376	64.5%	0.872	0.1	45s	
▶ Frame 387	0.070s	263,047,990	2.9%	0.882	0.1	57s	
Frame 214	0.069s	260,711,662	0.0%	0.896	0.1	56s	
Frame 211	0.069s	265,593,751	11.8%	0.867	0.1	55s	
▶ Frame 377	0.069s	254,510,381	0.0%	0.892	0.1	55s	
Frame 248	0.069s	268,360,639	2.0%	0.864	0.1	56s	
Frame 379	0.068s	239,726,238	4.7%	0.936	0.1	57s	
Frame 269	0.068s	263,348,608	0.0%	0.851	0.1	46s	
Frame 217	0.068s	262,593,844	0.0%	0.867	0.1	48s	
Frame 272	0.067s	254,704,545	0.0%	0.875	0.1	55s	
Frame 821	0.067s	240,466,446	17.1%	0.871	0.1	44s	
Frame 231	0.067s	249,131,002	0.0%	0.879	0.1	45s	
▶ Frame 357	0.067s	257,373,132	0.0%	0.864	0.1	39s	
Frame 224	0.067s	249,544,739	31.6%	0.881	0.1	62s	
▶ Frame 370	0.067s	261,285,508	5.4%	0.857	0.1	42s	
▶ Frame 381	0.067s	238,690,251	40.2%	0.897	0.1	58s	
▶ Frame 233	0.067s	250,635,455	37.1%	0.876	0.1	54s	
Frame 229	0.067s	254,875,318	11.2%	0.882	0.1	55s	

Instrumentation API

Extensive API enables frame and task markup for better results

Intel[®] VTune[™] Profiler Features

CPU Performance Profiling

- Locate bottlenecks in the CPU due to lock \bullet contention, synchronization issues, inefficient cache/memory utilization, and more
- Supports advanced profiling for Windows* and \bullet Linux* applications for unmanaged or managed code, or a mix
- Mark up frames and tasks for more focused analysi with the VTune ITT API
 - Now integrated into Unity* 2019.3 and Unreal \bullet Engine* 4.19
- Extensive online documentation includes tutorials. tuning guides, support forums, and more
- Free download with community support \bullet
- Take advantage of **Priority Support** \bullet
 - Connects customers to Intel engineers for confidential inquiries (paid versions)

EXAMPLE Thread	ing Efficiency	• ?									INTEL VTUNE PROFILER
Analysis Configuration	Collection Log	Summary	Bottom-up	Caller/C	Callee To	p-down Tree Platfo	rm				
Grouping: Task Domain / Ta	ask Type / Func	tion / Call Sta	ck						•	<u></u> % ρ %	Inactive Sync Wait Time
		CPU	Time 🔻		(K)	Inactive Wait	Time 🔍	Inactive Wa	ait Count 🔍		Viewing < 1 of 3041 • selected state
Task Domain / Task Type / Function / Call Stack	Effective Idle Poor	Time by Utilizat Ok 🚦 Ideal	ion 🏾 🤊 I 🚦 Over	Spin Time	Overhead Time	Inactive Sync Wait [®] Time	Preemption [®] Wait Time	Inactive Sync Wait Count	Preemption ² Wait Count	Task Time	68.6% (129.859s of 189.405s) KernelBase.dll!WaitForSingleOb
▼ com.unity.vtune	49.205s			19.499s	Os	222.729s	1.693s	266,002	11,319	328.312s	UnityPlayer.dll!UnityClassic::Bas
Camera.Render	7.674s 📕			1.314s	Os	2.332s	0.040s	5,013	213	16.493s	UnityPlayer.dll!JobQueue::Proce
PlayerLoop	4.932s 📒			2.493s	Os	7.806s	0.133s	12,780	1,386	19.806s	UnityPlayer.dll!JobQueue::Work
Drawing	4.626s 📕			0.502s	Os	1.580s	0.035s	1,528	60	9.642s	UnityPlayer.dll: Inread::Run Inre
PostLateUpdate.Finisl	3.063s 📙			1.072s	Os	6.966s	0.031s	4,185	186	14.480s	Remei Sz.dii: Base Thread Mit Thur
▶ Idle	0.024s			3.424s	Os	130.143s	0.917s	186,053	4,792	136.671s	fituit.dif:Rtioser fifeadStart+0x2
Render.OpaqueGeome	2.745s 📒			0.352s	Os	0.726s	0.029s	1,105	53	5.230s	
Semaphore.WaitForSig	0.130s			2.446s	Os	30.208s	0.018s	10,950	437	31.117s	
RenderDeferred.Light	2.045s 📙			0.173s	Os	0.115s	0.004s	187	2	2.086s	
RenderDeferred.Light	1.933s 📕			0.159s	Os	0.105s	0.004s	171	1	1.933s	
WaitForJobGroupID	0.461s			1.462s	Os	0.368s	0.024s	6,274	486	1.823s	
Render.TransparentGe	1.222s			0.123s	Os	0.718s	0.000s	274	3	2.804s	
CommandBuffer.Before	1.270s 💧			0.060s	Os	0.306s	0.000s	127	6	2.036s	
Culling	0.809s			0.500s	Os	0.084s	0.001s	1,962	82	2.004s	
RenderForward.Rende	1.174s 📘			0.102s	Os	0.779s	0.000s	251	5	2.913s	
Post-processing	1.200s 🚦			0.057s	Os	0.297s	0.000s	120	6	1.901s	
RenderForwardAlpha.	0.962s 🛔			0.085s	Os	0.573s	0.000s	161	2	2.315s	
BuiltinStack	0.922s			0.042s	Os	0.246s	0.000s	101	5	1.478s	
FixedUpdate.PhysicsF	0.401s			0.390s	Os	0.273s	0.040s	3,184	795	0.822s	
SconeCulling	0.205-			0.402-	0-	0.075-	0.001	1 051	00	07476	1
Q: 🕇 — 1	🖝 🖉 Os	25 45	6s	8s	10s	12s 14s	16s 18s	20s 22s	24s 2	26s 28s	Scale Markers:
Frame Rate				and the lates	n juli sing tine sa ing p	Ala televisione presidente de la seconda	وأحجب وبإرادة لمتجمع ومعرفه والمعرف	Nikolashan an arasa i	محتوا بالمأوية ومقتله وارمت متتبا	day a deberation in the	Frame
문 Thread (TID: 6708)											Frame Rate
2 Thread (TID: 12724)											Frame Rate
Thread (TID: 8208)						inter Contraction Advantages			ومراملين والمراملين المرامل وروا	and a second product of the second	Thread 🔻
Thread (TID: 15444)											Context Switches
Thread (TID: 17109)											Preemption
Thread (TID: 1/100)											Synchronization
Inread (IID: 15516)											CPU Time
Thread (TID: 16412)											Spin and Overhead
Thread (TID: 5816)											Clocktick Sample
Thread (TID: 17216)											COLUTI
Thread (TID: 15340)											CPU Time
Thread (TID: 12574)											CPU Time
CPLITime											Spin and Overhead

Learn more: software.intel.com/vtune





Simplified UI

More welcoming "welcome page"

 Quick access to documentation and training

Built-in sample code, precollected results

• Easier to explore tutorials

Help tour overlay

• Quickly learn essential user interface controls

Better command line help

• Better structured and more informative



Tachyon Sample

A simple ray tracer

Once packaged as a sample with Intel® Vtune™ Profiler, this simple ray tracer is written in C++ and uses Intel Threading Building Blocks (TBB) for its threading implementation.

Let's take a look at this sample and see where its performance can be improved...





Hotspots Collection

Start with hotspots to see the most active functions running during the collection.

- **User-mode sampling** uses OS interrupts and collects call stacks automatically.
- Hardware event-based sampling uses a driver to collect CPU data, such as clock cycles per instruction (CPI rate).
 Stack collection uses an additional driver and is optional.
- Both methods of hotspot sampling enable collection of additional performance insights by default, which uses the sampling driver to generate a performance overview.



Configure Analysis 🛗



Summary View

Top Hotspot – grid_intersect

- Optimizing this function will provide the best performance gain.
- But this doesn't tell the whole story...

A	nalysis Configuration	Collection Log	Summary	Bottom-up	Caller/Callee
	 CPU Time[®]: Instructions Re Microarchitect Total Thread Co 	tired: 22,6 ure Usage [®] : punt:	8.487s 594,668,221 36.1% 23	of Pipeline	e Slots
(Top Hotspot This section lists th	S e most active func	tions in your a	application. C	Optimizing these
-	Function		Module	CPU	J Time [®]
	grid_intersect		analyze_loc	ks.exe	3.617s
	sphere_intersect		analyze_lock	ks.exe	2.718s

Function	Module	CPU Time 💿
grid_intersect	analyze_locks.exe	3.617s
sphere_intersect	analyze_locks.exe	2.718s
GdipDrawImagePointRectI	GdiPlus.dll	0.941s
[TBB parallel_for on class draw_task]	analyze_locks.exe	0.172s
trace	analyze_locks.exe	0.126s
[Others]		0.913s

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

⊘ Top Tasks

This section lists the most active tasks in your application.

Task Type	Task Time	Task Count 🕐
tbb_parallel_for	88.599s	9
******		,

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

⊘ Effective CPU Utilization Histogram

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



Streetive CPU Utilization 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 utilization value.



More on the Summary View

CPU Utilization Histogram

Even though the summary reports 23 threads, the majority of the application only runs on a single core at a time. This indicates a big concurrency problem.





Bottom-up View

Detailed view of function and thread performance

- VTune highlights metrics below or above a certain threshold where performance can be improved.
- The thread timeline view visualizes thread and CPU activity.

Frouping: Function / Call Stack						· × 0 %	CPU Time *
	3	1	Microarchitecture Usar	(a)	i energia		Viewing + 1 of 93 + selected stack(s)
Function / Call Stack	CPU Time V	Instructions Retired	Microarchitecture Usage	CPI Rate	Module	1	22.5% (0.813s of 3.617s)
grid_intersect	3.617s	9,337,955,156	31.0%	0.860	analyze_locks.exe	grid_intersect	analyze_locks.exe!grid_intersect - grid.cpp
sphere_intersect	2.718s	8,722,608,219	45.6%	0.794	analyze_locks.exe	sphere_intersect	analyze_locks.exe!trace+0x412 - trace_rest.cpp:114
GdipDrawImagePointRectI	0.941s 💼	1,827,741,791	44.6%	0.801	GdiPlus.dll	GdipDrawlmagePointF	analyze_locks.exe!render_one_pixel+0x23F-analyze_locks.cpp:101
[TBB parallel_for on class draw_task]	0.172s	19,397,788	9,3%	7.513	analyze_locks.exe	tbb::interface9::interna	analyze_locks.exel[18B parallel_for on class draw_task]+0x44 · parallel_for.h:143
trace	0.126s	299,141,429	51.4%	0.796	analyze_locks.exe	trace	tob.dill[IBB Scheduler Internals]+0x100 - custom_scheduler.h:469
func@0x1401bce90	0.121s	186,798,022	0.0%	0.922	ntoskrnl.exe	func@0x1401bce90	tbb.dil: TBB Dispatch Loop)+0x14a - custom_scheduler.nco42
• tri_intersect	0.058s	195,558,824	47.5%	0.602	analyze_locks.exe	tri_intersect	tbb.dil:tbb:linternal:market:process+0x30 - market.cpp:000
_stdlo_common_vfscanf	0.047s	140,540,333	0.0%	0.709	ucrtbase.dll	_stdlo_common_vfscan	tbb.dit(166 worker)+0x61 - private_server.cpp:607
light_intersect	0.038s	72,589,972	38.1%	0.730	analyze_locks.exe	light_intersect	tobulini meau start vonsigned int (stotali //void // Foxsis - thread.cpp:115
KiUserCallbackDispatcher	0.030s	25,708,370	62.6%	2.155	ntdil.dll	KiUserCallbackDispate	etelli dillane 20/db2n65/n40x21 - Janknown source file]
func@0x6b101cb0	0.030s	514,698,956	3.9%	1.216	wow64cpu.dll	func@0x6b101cb0	ntdl.dllfunc@0x4b2c65c2+0x2h - Junknown source file)
func@0x18000fa70	0.029s	451,515,791	1.5%	1.524	wow64win.dll	func@0x18000fa70	Literon transfer and the feature for the feature and the met
PeekMessageA	0.021s	3,987,943	64.8%	1.561	user32.dll	PeekMessageA	
libm sse2 pow	0.021s	72,883,986	21.1%	0.709	libmmd.dll	libm sse2 pow	
func@0x69e23532	0.019s	13,912,542	16.5%	2.267	user32.dll	funci@0x69e23532	
render one pixel	0.0195	49,442,508	52.0%	0.531	analyze locks.exe	render one pixel	
TBB Scheduler Internals]	0.019s	6.094,199	0.0%	6.538	tbb.dll	tbbs:internal::custom s	
MsgWaltForMultipleObjects	0.013s	3.862.331	100.0%	0.524	user32.dll	MsgWaltForMultipleO	
VNorm	0.012s	42,437,784	42.7%	0.793	analyze locks exe	VNorm	
the internal: itt task begin v7	0.009s	1,935,388	38.4%	3,780	thb.dll	thb::internal::itt task h	
funci@0x6b82243c	0.008s	6.216.412	85.1%	2.863	kernel32.dll	func@0x6b82243c	
GdioCreateFromHDC	0.0076	5.812.533	89.4%	0.731	GdiPlus dll	GdipCreateFromHDC	
func@0x1401c43bd	0.007s	13,083,117	20.7%	0.956	ntoskral.exe	func@0x1401c43bd	
DispatchMessageA	0.007s	0	0.0%		user32.dll	DispatchMessageA	
6.0000001400004410	0.0046	6 603.041	15.2W	4.400	introduced even	func@Out400ndd40	
9							
0: 🕇 = 🐖 🖉 🔄	ts	2.095s 35	**************************************		05 75	85	es 10s 11s 12s 🕑 Thread 🔻
素 Thread (TID: 22956)					desidentia (all pice)		🖉 🔜 Running
Thread (TID: 18172)			-				Z ■ CPUTime
TBB Worker Thread (TID: 25	1.4		a he h	and the state of t			Spin and Overhead
TBB Worker Thread (TID: 14		Thread (TID: 181)	72)				✓ Contact stript
TBB Worker Thread (TID: 54		CPU Time		1.		1 1 10	CPU Time
TBB Worker Thread (TID: 87		6.2%				-	∠ CPU Time
TBB Worker Thread (TID: 21		Spin and Overhea	d Time	1		abled they be 1	Spin and Overnead
TBB Worker Thread (TID: 21				11	1 1		
TBB Worker Thread (TID: 20							15
Thread (TID: 21676)	a desired and the		mater to are a				
Thread (TID: 18656)							
Thread (TID: 18604)							
Thread (TID: 26896)							
Thread (TID: 14768)							
CONTRACT							

Any Utilization

Module Any Module

Thread, Any Thread

FILTER T 100.0% C Process Any Process

Call Stack Mode User functions

 Loop Mode Functions onl

 Inline Mode Show inline fu

@IntelSoftware @IntelGraphics



Q: ╋ = ⊮ ⊮	0s	1s	2.0	9 <mark>55</mark> 3s	4s 5s		6s	7s	8s		10s	11s		25
Thread (TID: 22956)					A A A Mar March					1				ľ
Thread (TID: 18172)	1 h.			and the second							A.U.	and much second	a da construire	
TBB Worker Thread (TID: 25			-							4				
TBB Worker Thread (TID: 14			3	Thread (TID: 18172)						1				
TBB Worker Thread (TID: 54		1		CPU Time										
TBB Worker Thread (TID: 87				6.2%							A			
TBB Worker Thread (TID: 21		1		0.0%	le la	1	L. L		abiai ka ka	1		<u> </u>		
TBB Worker Thread (TID: 21			1.1					<u> </u>				1	. 14	
TBB Worker Thread (TID: 20			ш. —	14									1	
Thread (TID: 21676)														1
Thread (TID: 18656)	Alex 1													
Thread (TID: 18604)														
Thread (TID: 26896)														
Thread(TID: 14768)														
CPU Time														

Thread/CPU Timeline

Visualize CPU activity and thread concurrency

The timeline shows that although there are many threads, they are not executing at the same time. On a multi-core system, this generally means there is a lock preventing simultaneous execution.





Evention (Call Shade	CDUTing T	Instructions Dational	Microarchitecture Usa	Microarchitecture Usage 🛛 🖉 Mod		
Function / Call Stack	CPU Time V	Instructions Retired	Microarchitecture Usage	CarRate	Module	
grid_intersect	3.617s	9,337,955,156	31.0%	0.860	at alyze_locks.exe	grid_i
sphere_intersect	2.718s	8,722,608,219	45.6%	0.794	analyze_locks.exe	spher
▶ GdipDrawImagePointRectI	0.941s 🛑	1,827,741,791	44.6%	0.801	GdiPlus.dll	Gdip[
[TBB parallel_for on class draw_task]	0.172s	19,397,788	9.3%	7.513	analyze_locks.exe	tbb::i
▶ trace	0.126s	299,141,429	51.4%	0.796	analyze_locks.exe	trace
▶ func@0x1401bce90	0.121s	186,798,022	0.0%	0.922	ntoskrnl.exe	func@
tri_intersect	0.058s	195,558,824	47.5%	0.602	analyze_locks.exe	tri_in
_stdio_common_vfscanf	0.047s	140,540,333	0.0%	0.709	ucrtbase.dll	_stdic
light_intersect	0.038s	72,589,972	38.1%	0.730	analyze_locks.exe	light_
KiUserCallbackDispatcher	0.030s	25,708,370	62.6%	2.155	ntdll.dll	KiUse
▶ func@0x6b101cb0	0.030s	514,698,956	3.9%	1.216	wow64cpu.dll	func@
▶ func@0x18000fa70	0.029s	451,515,791	1.5%	1.524	wow64win.dll	func@

Where's the bottleneck?

CPI rate for grid_intersect is low

The CPI rate is generally a good indicator of function performance. VTune considers 1 or higher to be poor performance, with 0.250 (four instructions per clock cycle) being the best.

Fixing the threading issue is the priority at this point.



Total Thread Count: 22

```
Inactive Wait Time with poor CPU Utilization 215.443s (100.0% from Inactive Wait Time) (2)
Inactive Sync Wait Time 215.095s
Preemotion Wait Time 2 0.348s
```

🕙 Top functions by Inactive Wait Time with Poor CPU Utilization. 🖗

This section lists the functions sorted by the time spent waiting on synchronization or thread preemption with poor CPU Utilization.

Function	Module	Inactive Wait Time 🗇	Inactive Sync Wait Time $^{\odot}$	Inactive Sync Wait Count ®	Preemption Wait Time $^{\odot}$	Preemption Wait Count $^{\oslash}$
[TBB parallel_for on class draw_task]	analyze_locks.exe	78.265s	78.210s	1,069	0.054s	16
NtWaltForWorkViaWorkerFactory	ntdil.dll	75.861s	75.8564	53	0.005s	3
NtWaltForSingleObject	ntdil.dll	24.817s	24.6795	₽ 8	0.138s	1
NtUserGetMessage	win32u.dll	12.264s	12.263s	53		4
func@0x180017cb0	wow64win.dll	11.991s	11.9915	7		1

"NIA is applied to non-summable metrics.

Spin and Overhead Time : 0.382s (4.2% of CPU Time)

Threading Analysis

The threading analysis types provide insights into synchronization objects and context switches

Class draw_task has a mutex lock which prevents parallel execution. In this example, the mutex lock is not needed.





The "After"

Removing the unneeded lock improved performance by over 2x!

- Elapsed time of original sample was 12.775s, reduced to 5.323s
- There is still room for improvement...

Control control based of the second of th	三 習 時 ▶ 去 Ø № ⑦ Welcome ※ r000hs ※ r001tr ※ r002hs ※	
	🗧 Hotspots Hotspots by CPU Utilization 🔹 🕐 📖	INTEL VTUNE PROFILER
 C Elsevent me ²: 5.223 S Howard in the first of the second second	Analysis Configuration Collection Log Summary Bottom-up Caller/Callee Top-down Tree Platform	
	 ✓ Elapsed Time ⁽³⁾: 5.323s S CPU Time ⁽²⁾: 10.449s Instruction Retired: 20.135.629.831 	Hotspots Insights If you see significant hotspots in the Top Hotspots list, switch to the Bottom- up view for in-depth analysis per function. Otherwise, use the Caller/Callee view to track critical paths for these hotspots.
	Microarchiteture Usage 416% of Pipeline Slots CPI Rate 1 CPI Rate 2 Total Thread Count: 22 Dear Tare 2 Dear Tare 2	Explore Additional Insights Parallelism © : 25.1% (2012 out of 8 logical CPUs) N Use The Threading to explore more opportunities to increase parallelism in your application.
 Control processing Control processi		Microarchitecture Usage : 41.65 Use Microarchitecture Exploration to explore how efficiently your application runs on the used hardware.
	Op Hotspots This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.	Vector Register Utilization : 0: 25.0% ►
 Contract of the section of the sectio	Function Module CPU Time ®	Use Intel Advisor to learn more on vectorization efficiency of your application.
Solution of the set	grid intersect analyze locks.exe 5.078s	
Second Participation and Second Part of the second Part of	sphere_intersect analyze_locks.exe 3.454s	
 market of albedow model with the most and the model with the model w	GdipDrawlmagePointRectl GdIPlus.dll 0.780s	
Other 0.000 *Vel applied to consummable metric. *The applied to consummable metric. *The section lists the number of the table is a king our application. *The section list the number of the table is a king our application. *The section list the number of the table is a king our application. *The section list table is table in the table is a constrained in the add to the idle CPU utilization value. (Constrained is by a percentage of the wall time the specific number of CPUs were running simultaneously. Soin and Overhead time adds to the idle CPU utilization value.	trace analyze_locks.exe 0.151s	
 (interminence of the constraint of	func@0x1401bce90 ntoskmi.exe 0.100s	
 Na A applied tensors. Por Tasks € This section flats the most active tasks in your application. Task Time © Task Court © to parallel_for 20.3746 20.3746 20.3746 20.3746 Potective CPU Utilization Histogram This hatogrand displays a percentage of the well time the specific number of CPUs were running simultaneously. Soin and Overhead time adds to the Idle CPU utilization value. O Effective Out of the well time the specific number of CPUs were running simultaneously. Soin and Overhead time adds to the Idle CPU utilization value. O Effective Out of the well time the specific number of CPUs were running simultaneously. Soin and Overhead time adds to the Idle CPU utilization value. O Effective Definition of the well time the specific number of CPUs were running simultaneously. Soin and Overhead time adds to the Idle CPU utilization value. O Effective Definition of the well time the specific number of CPUs were running simultaneously. Soin and Overhead time adds to the Idle CPU utilization value. O Effective Definition of the well time the specific number of CPUs were running simultaneously. Soin and Overhead time adds to the Idle CPU utilization value. 	[Others] 0.886s	
 C Top Tasks ① This section lists the most active tasks in your application. <u>This hype for the forward of the solution</u> ¹ We acceled to nor-normalize metrics C Effective CPU Utilization Histogram This listogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the idle CPU utilization value. O Effective Optimized of the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the idle CPU utilization value. 	NAI is applied to non-summable metrics.	
	✓ Top Tasks St_2 This section lists the most active tasks in your application. Task Type Task Time [⊕] Task Count [⊕] Task Time [⊕] Task Count [⊕]	
Collection and Platform Info	100_Datalle_ror 20.0745 0 70.04 is apolled to non-summable metrics.	
Collection and Platform Info	Effective CBUUItilization Histogram	
Collection and Platform Info	This histogram displays a percentage of the wali time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU utilization	on value.
Collection and Platform Info	1000-00-01	
Collection and Platform Info		
Collection and Platform Info		
Source		
Collection and Platform Info		
Collection and Platform Info	000mm - 20 20 20 10 10 10 10 10 10 10 10 10 10 10 10 10	
Collection and Platform Info		
Scollection and Platform Info	400mi -	
Collection and Platform Info	200#2	
Collection and Platform Info		
Collection and Platform Info	0 1 2 3 4 8 8 7 8	
© Collection and Platform Info	late Poor Ox deal	
O Collection and Platform Info	5 5 5	
🖓 Collection and Platform Info		
💬 Collection and Platform Info		
	🖂 Collection and Platform Info	





About that CPI Rate...

Microarchitecture analysis

- Overall performance is better due to \bullet improved concurrency, but a microarchitecture analysis shows CPI rate is worse?
- The work-stealing nature of TBB may be • causing a higher CPI rate, or improved threading exposed other bottlenecks
- This sample uses linked lists, which can \bullet cause higher memory latency due to cache misses, and make it harder for the compiler to predict branching. Vectors may be a better option.

Elapsed Time $^{\odot}$: 3.517s 🐴
Clockticks:
Instructions Retired:
CPI Rate [®] :
MUX Reliability [®] :
Retiring [®] :
⊘ Front-End Latency [®] :
ICache Misses [®] :
ITLB Overhead [®] :
O Branch Resteers [®] :
DSB Switches [®] :
Length Changing Prefixes [®] :
MS Switches [®] :
Front-End Bandwidth MITE [®] :
Front-End Bandwidth DSB ⁽²⁾ :
(Info) DSB Coverage ⁽²⁾ :
➢ Bad Speculation [™] :
Branch Mispredict [®] :
Machine Clears [®] :
⊘ Back-End Bound [®] :
Memory Bound [®] :
⊘ Core Bound [®] :
Divider ®

 (\bigtriangledown)

25,636,700,000			
21,519,400,000			
1.191 🛤			
0.967		30.5% - Front-End Bound	
44.4%	of Pipeline Slots		
30.5% 🎙	of Pipeline Slots		
19.4% 🏲	of Pipeline Slots		
1.3%	of Clockticks		
3.5%	of Clockticks	44.4% - Retiring	
3.7%	of Clockticks		
9.9% 🎙	of Clockticks		
0.0%	of Clockticks		
0.0%	of Clockticks		
11.2% 🏲	of Pipeline Slots	21.6% - Bad Speculation	
16.0% 🏲	of Clockticks		
6.7%	of Clockticks	-	
35.0% 🏲		This diagram represents ineffic	
21.6% 🏲	of Pipeline Slots	equal to the "pipe efficiency	
21.6% 🏲	of Pipeline Slots	Instruction Retired). If there are	
0.0%	of Pipeline Slots		
3.4%	of Pipeline Slots		
1.9%	of Pipeline Slots		
1.5%	of Pipeline Slots		
2.3%	of Clockticks		



Learn More

• Download Intel[®] VTune[™] Profiler for free:

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

• Documentation:

https://software.intel.com/en-us/vtune/documentation/view-all

Cookbook:

https://software.intel.com/en-us/vtune-cookbook

• Tutorials:

https://software.intel.com/en-us/articles/vtune-tutorials

• Windows* Tips:

https://software.intel.com/en-us/articles/vtune-amplifer-tips-for-profiling-in-windows



