



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.



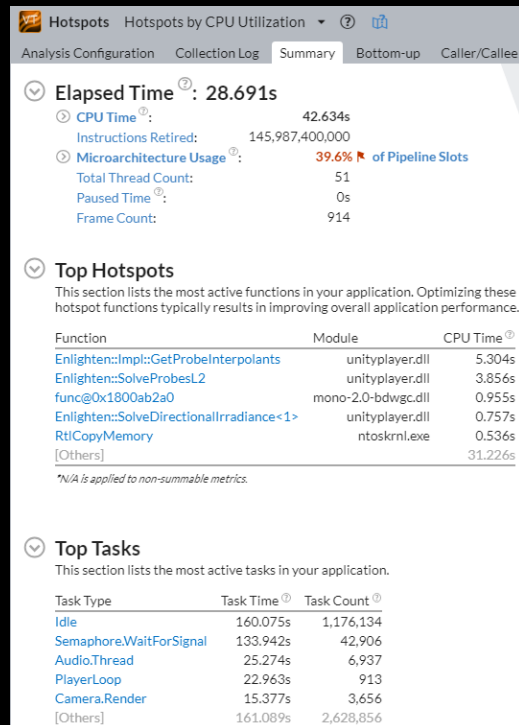
Agenda

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



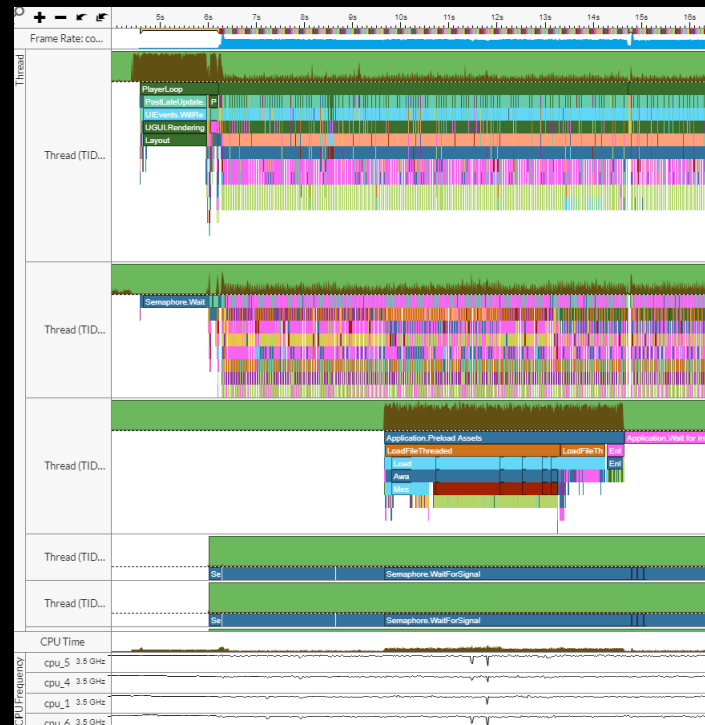
Intel® VTune™ Profiler

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



Hotspot Analysis

Identifies functions consuming the most CPU time



Thread Performance

Visualize thread behavior to quickly identify concurrency problems

Grouping: Task Domain / Task Type / Function / Call Stack					
Task Domain / Task Type / Function / Call Stack	CPU Time	Instructions Retired	Microarchitecture Usage		Task Time
			Microarchitecture Usage	CPI Rate	
UE4Domain	65.847s	211,632,115,937	29.8%	0.990	155.252s
FDeferredShadingSceneRenderer_Render	8.956s	20,558,580,980	23.8%	1.350	15.575s
FDeferredShadingSceneRenderer_InitViews	1.822s	4,945,551,288	26.5%	1.178	3.340s
FSceneRenderer_ComputeViewVisibility	1.582s	4,621,327,612	26.9%	1.112	2.903s
UWorld_Tick	1.463s	5,770,821,626	28.6%	0.887	4.109s
FCompression_UncompressMemory	1.187s	4,825,032,389	60.5%	0.712	1.692s
FScene_UpdateAllPrimitiveSceneInfos	0.580s	1,380,673,014	18.9%	1.356	0.851s
FScene_AddPrimitiveSceneInfos	0.558s	1,331,746,702	18.2%	1.348	0.815s
FScene_AddPrimitiveSceneInfoToScene	0.535s	1,312,085,470	18.5%	1.328	0.795s
FDeferredShadingSceneRenderer_RenderLights	0.333s	673,865,649	22.1%	1.554	0.564s
Slate::Tick	0.219s	203,536,533	30.3%	3.053	0.635s
FViewport_Draw	0.166s	43,686,757	20.5%	2.930	0.291s
Slate::DrawWindows	0.164s	181,458,102	34.6%	2.761	0.539s
FAudioDevice_Update	0.159s	37,496,378	59.6%	2.715	0.207s
FDeferredShadingSceneRenderer_InitViewsPossibl	0.147s	206,229,370	27.9%	2.055	0.270s
Slate::DrawWindow_RenderThread	0.112s	204,785,341	35.5%	1.737	0.172s
FSceneRenderer_InitDynamicShadows	0.103s	152,822,069	27.6%	1.938	0.188s
Frame 0	0.085s	199,353,144	22.4%	1.395	1.340s
Slate::DrawWindow	0.078s	24,953,960	16.6%	9.499	0.126s
Frame 324	0.072s	271,605,376	64.5%	0.872	0.145s
Frame 387	0.070s	263,047,990	2.9%	0.882	0.157s
Frame 214	0.069s	260,711,662	0.0%	0.896	0.156s
Frame 211	0.069s	265,593,751	11.8%	0.867	0.155s
Frame 377	0.069s	254,510,381	0.0%	0.892	0.155s
Frame 248	0.069s	268,360,639	2.0%	0.864	0.156s
Frame 379	0.068s	239,726,238	4.7%	0.936	0.157s
Frame 269	0.068s	263,348,608	0.0%	0.851	0.146s
Frame 217	0.068s	262,593,844	0.0%	0.867	0.148s
Frame 272	0.067s	254,704,545	0.0%	0.875	0.155s
Frame 821	0.067s	240,466,446	17.1%	0.871	0.144s
Frame 231	0.067s	249,131,002	0.0%	0.879	0.145s
Frame 357	0.067s	257,373,132	0.0%	0.864	0.139s
Frame 224	0.067s	249,544,739	31.6%	0.881	0.162s
Frame 370	0.067s	261,285,508	5.4%	0.857	0.142s
Frame 381	0.067s	238,690,251	40.2%	0.897	0.158s
Frame 223	0.067s	250,635,455	37.1%	0.876	0.154s
Frame 229	0.067s	254,875,318	11.2%	0.882	0.155s

Instrumentation API

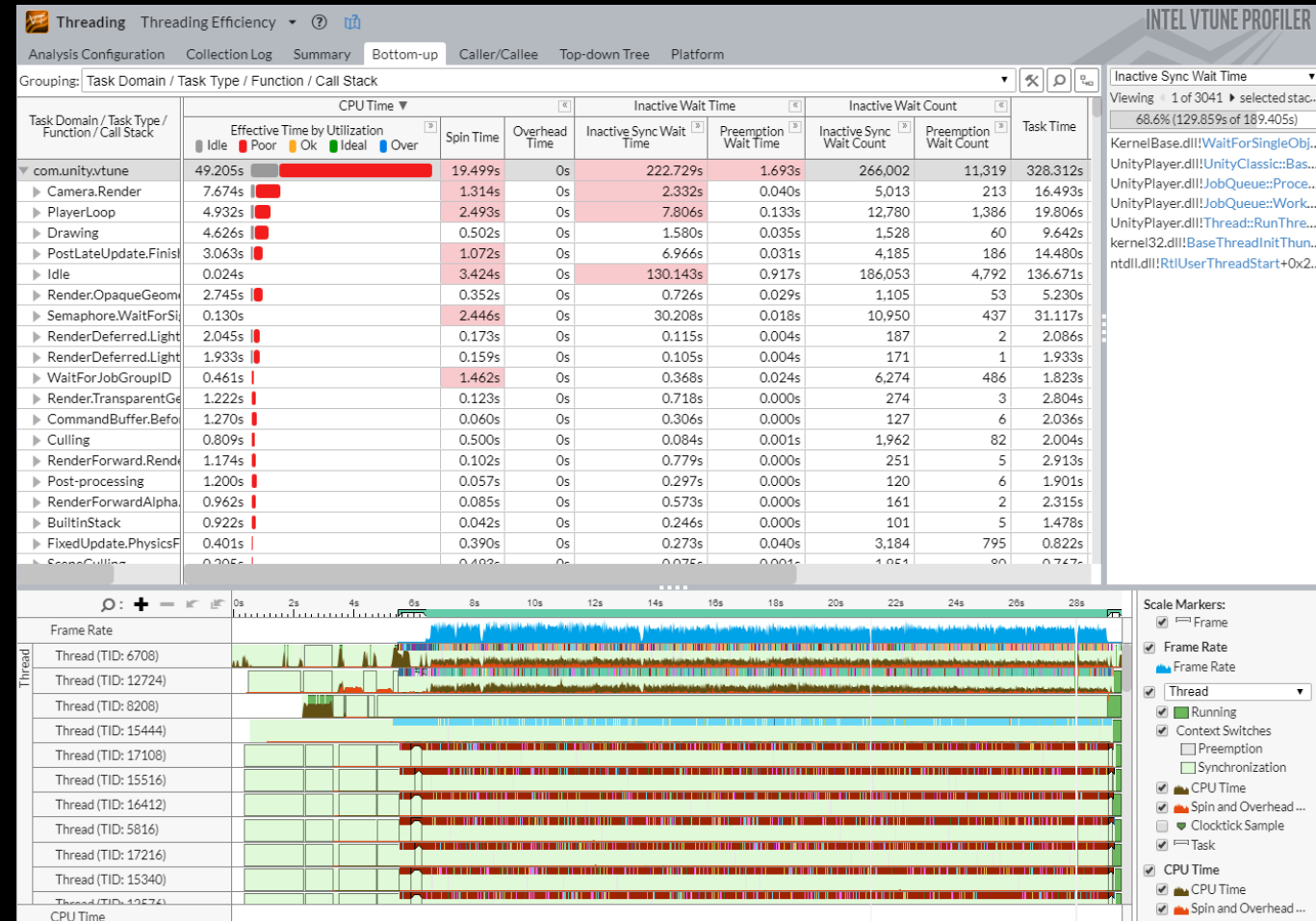
Extensive API enables frame and task markup for better results



Intel® VTune™ Profiler Features

CPU Performance Profiling

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



Learn more: software.intel.com/vtune



Current project: 3_tachyon_omp

 **Configure Analysis...** New Project...

RECENT PROJECTS

> 3_tachyon_omp

 Open Project...

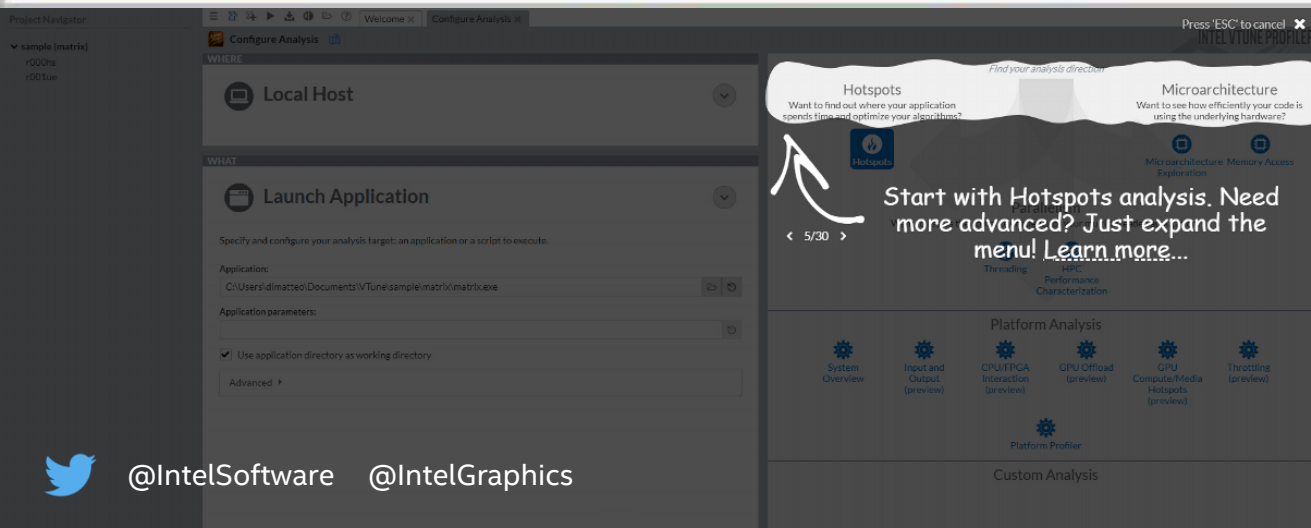
RECENT RESULTS

> r001hs [3_tachyon_omp]

> r011hs [3_tachyon_omp]

 Open Result... Help Tour  Documentation  Cookbook  Get Support  Twitter  Facebook

FEATURED CONTENT...



Simplified UI

More welcoming “welcome page”

- Quick access to documentation and training

Built-in sample code, pre-collected results

- Easier to explore tutorials

Help tour overlay

- Quickly learn essential user interface controls

Better command line help

- Better structured and more informative

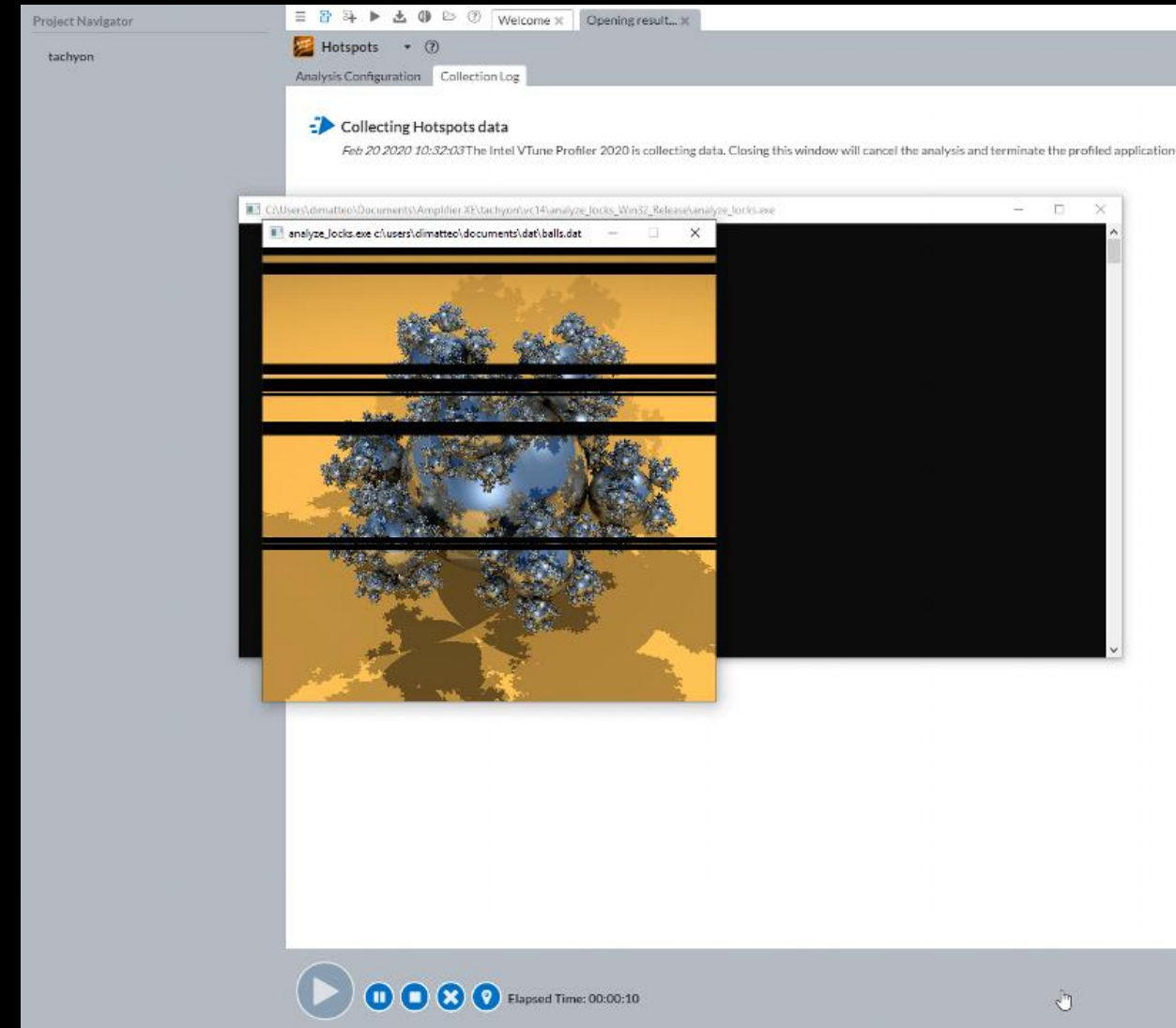


Tachyon Sample

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.

The screenshot shows the 'Configure Analysis' window in Intel VTune Profiler. The window is divided into two main sections: 'WHERE' and 'HOW'.

WHERE Section:

- Local Host:** Selected target.
- WHAT:** 'Launch Application' is selected.
- Specify and configure your analysis target:** An application or a script to execute.
- Application:** C:\Users\dimatteo\Documents\Amplifier XE\tachyon
- Application parameters:** (Empty text box)
- ☐ Use application directory as working directory
- Working directory:** C:\Users\dimatteo\Documents\Amplifier XE\tachyon
- Advanced** (Expandable section)

HOW Section:

- Hotspots:** Selected method.
- Identify the most time consuming functions and drill down to see time spent on each line of source code. Focus optimization efforts on hot code for the greatest performance impact. [Learn more](#)**
- ☐ User-Mode Sampling (?)
- ☒ Hardware Event-Based Sampling (?)
- CPU sampling interval, ms:** 1
- ☒ Collect stacks
- ☒ Show additional performance insights
- Details** (Expandable section)
- Overhead** (Bar chart showing increasing overhead levels)

Bottom Bar: Contains icons for Play, Stop, and other controls.

Summary View

Top Hotspot – grid_intersect

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

Elapsed Time[?]: 12.775s

CPU Time [?] :	8.487s
Instructions Retired:	22,694,668,221
Microarchitecture Usage [?] :	36.1% of Pipeline Slots
Total Thread Count:	23
Paused Time [?] :	0.075s

Top Hotspots

This section lists the most active functions in your application. Optimizing these h



Function	Module	CPU Time [?]
grid_intersect	analyze_locks.exe	3.617s
sphere_intersect	analyze_locks.exe	2.718s
GdiDrawImagePointRectI	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

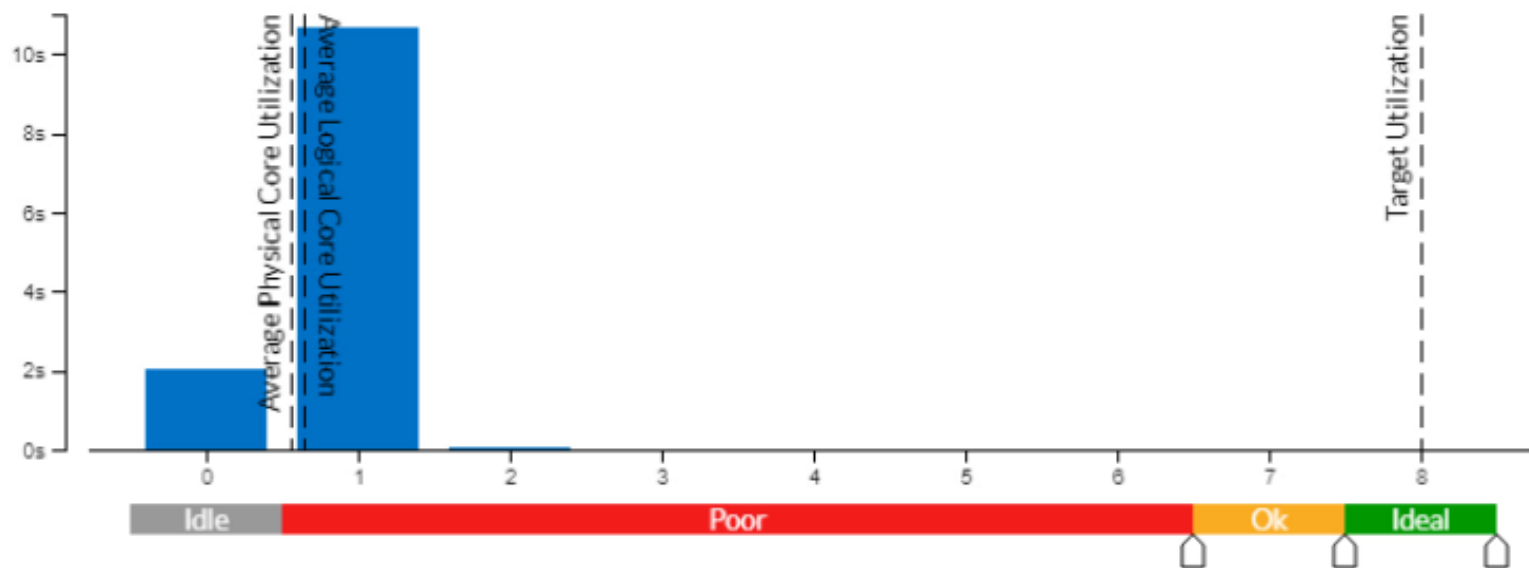
*N/A is applied to non-summable metrics.

Effective CPU Utilization Histogram

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

Effective 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

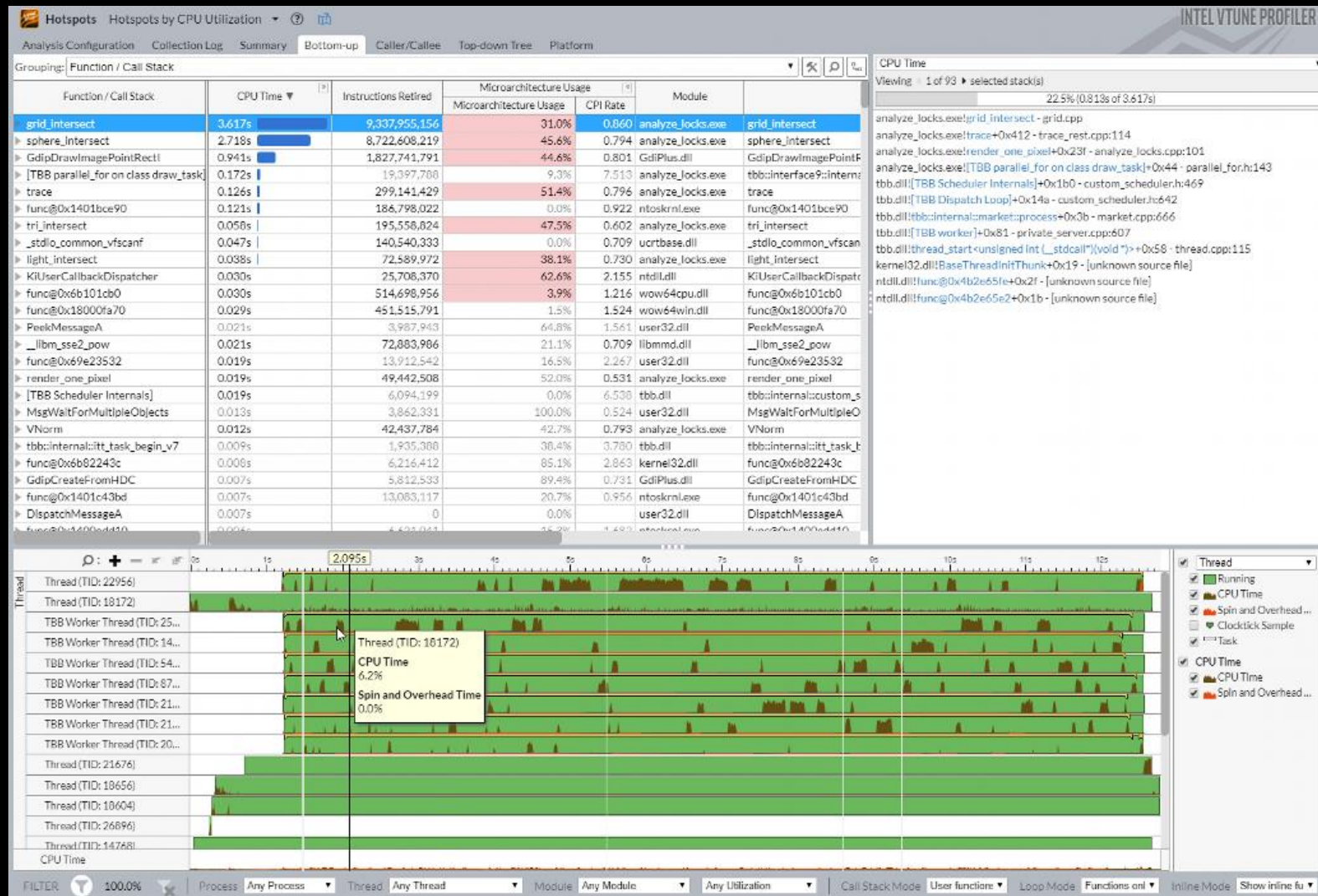
Even though the summary reports 23 threads, the majority of the application only runs on a single core at a time. This indicates a big concurrency problem.

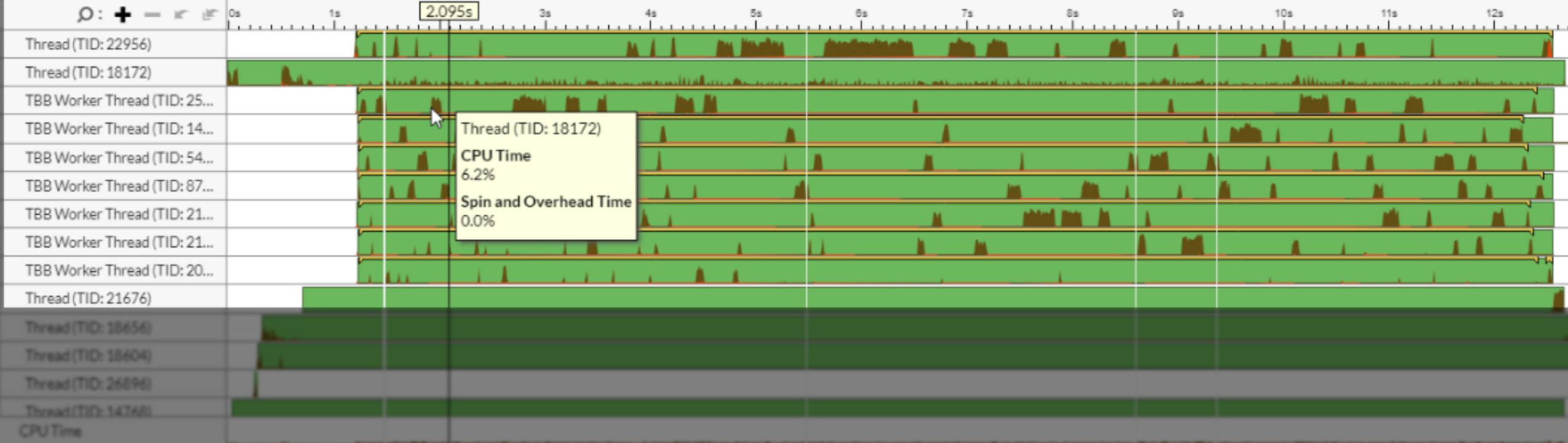


Bottom-up View

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.





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.



Function / Call Stack	CPU Time ▼	Instructions Retired	Microarchitecture Usage		Module	
			Microarchitecture Usage	CPI Rate		
▶ grid_intersect	3.617s	9,337,955,156	31.0%	0.860	analyze_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	GdipF
▶ [TBB parallel_for on class draw_task]	0.172s	19,397,788	9.3%	7.513	analyze_locks.exe	tbb::in
▶ 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_int
▶ _stdio_common_vfscanf	0.047s	140,540,333	0.0%	0.709	ucrtbase.dll	_stdio
▶ light_intersect	0.038s	72,589,972	38.1%	0.730	analyze_locks.exe	light_i
▶ 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)

Inactive Sync Wait Time: 215.095s

Preemption Wait Time: 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	Inactive Sync Wait Count	Preemption Wait Time	Preemption Wait Count
[TBB parallel_for on class draw_task]	analyze_locks.exe	78.265s	78.210s	1,069	0.054s	16
NtWaitForWorkViaWorkerFactory	ntdll.dll	75.861s	75.856s	53	0.005s	3
NtWaitForSingleObject	ntdll.dll	24.817s	24.679s	8	0.138s	1
NtUserGetMessage	win32u.dll	12.264s	12.263s	53	0.000s	4
func@0x180017cb0	wow64win.dll	11.991s	11.991s	7	0.000s	1
[Others]		12.246s	12.095s	6,077	0.151s	523

*N/A 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

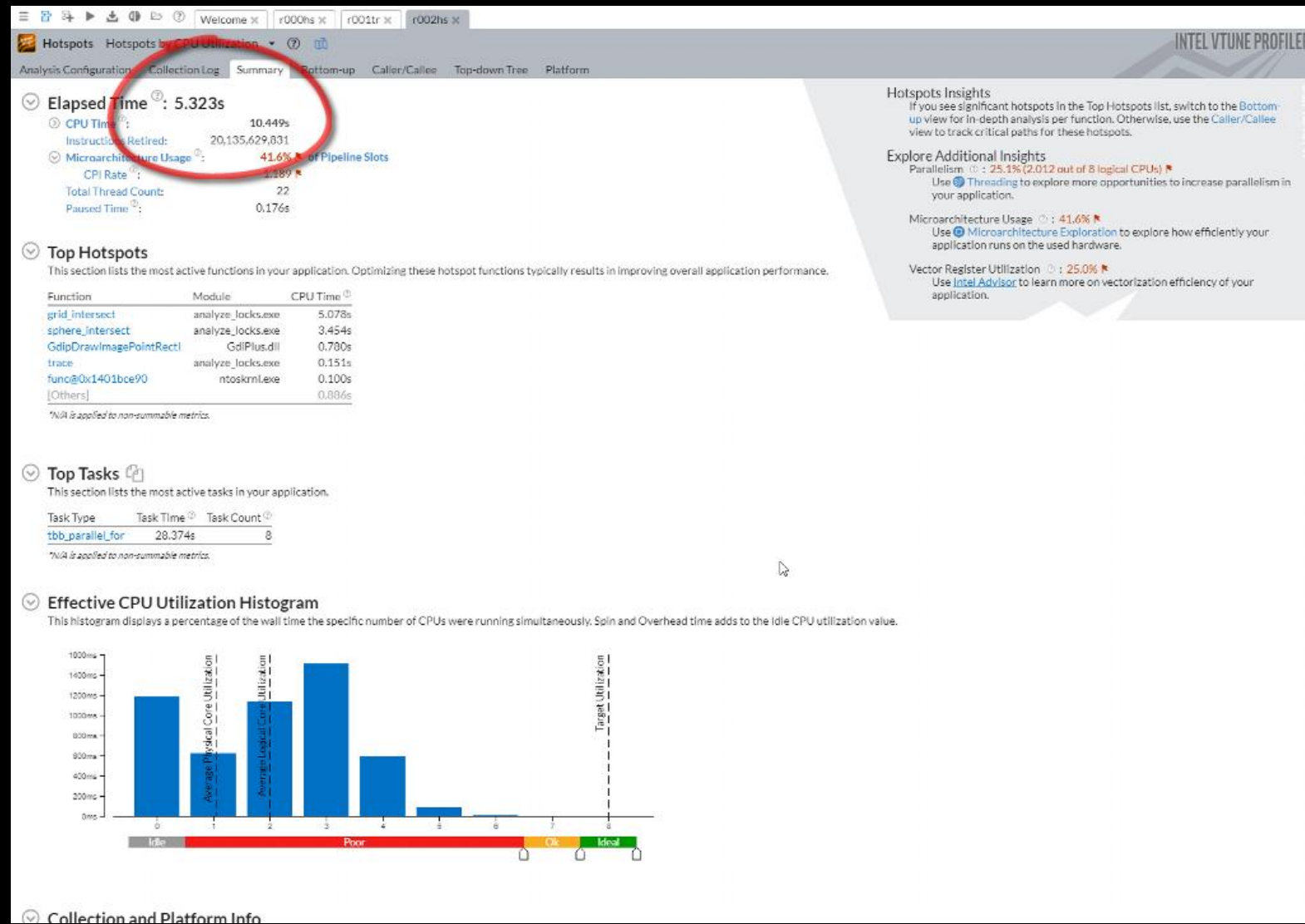
Class draw_task has a mutex lock which prevents parallel execution. In this example, the mutex lock is not needed.



The “After”

Removing the unneeded lock improved performance by over 2x!

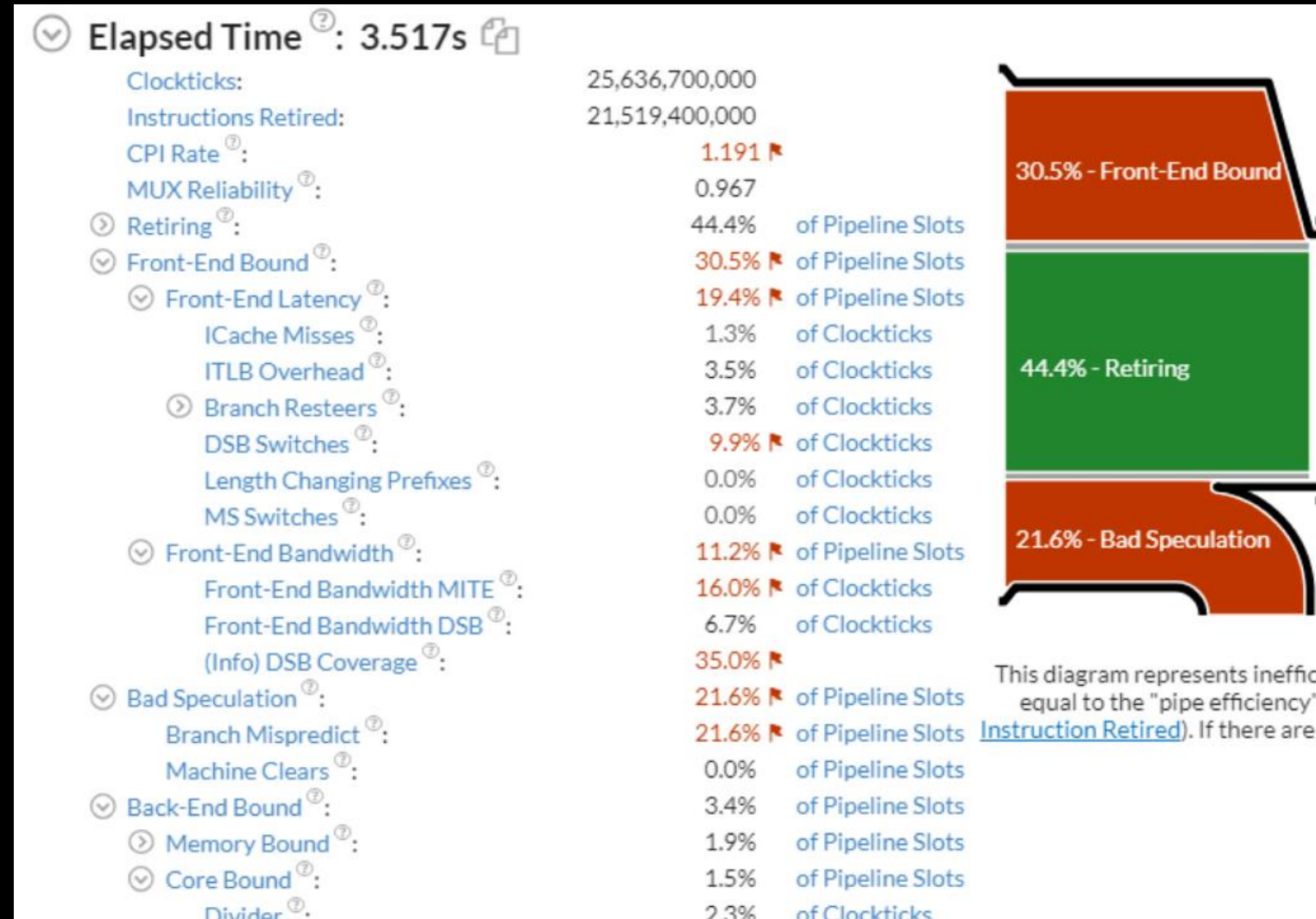
- Elapsed time of original sample was 12.775s, reduced to 5.323s
- There is still room for improvement...



About that CPI Rate...

Microarchitecture analysis

- Overall performance is better due to 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 cause higher memory latency due to cache misses, and make it harder for the compiler to predict branching. Vectors may be a better option.



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-amplifier-tips-for-profiling-in-windows>



