intel® Software

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 (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. Tune the #1 cause of slow threaded performance: – waiting with idle cores.	**Microarchitecture Exploration** Where is the biggest opportunity? Cache misses? Branch mispredictions?
	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 Type | Summary | Bottom-up

Elapsed Time: 5.554s

CPU Time:	10.504s
Instructions Retired:	21,698,000,000
CPI Rate:	1.257
CPU Frequency Ratio:	1.041
Total Thread Count:	9
Paused Time:	0s

Top Hotspots

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

Function	Module	CPU Time
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

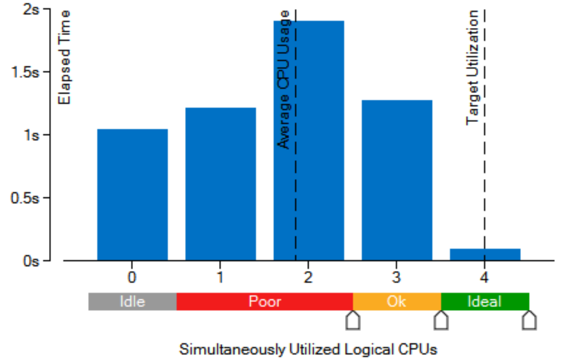
**N/A is applied to non-summable metrics.*

Average Bandwidth

Package	Total, GB/sec	Read, GB/sec	Write, GB/sec
package_0	5.715	3.504	2.212

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.

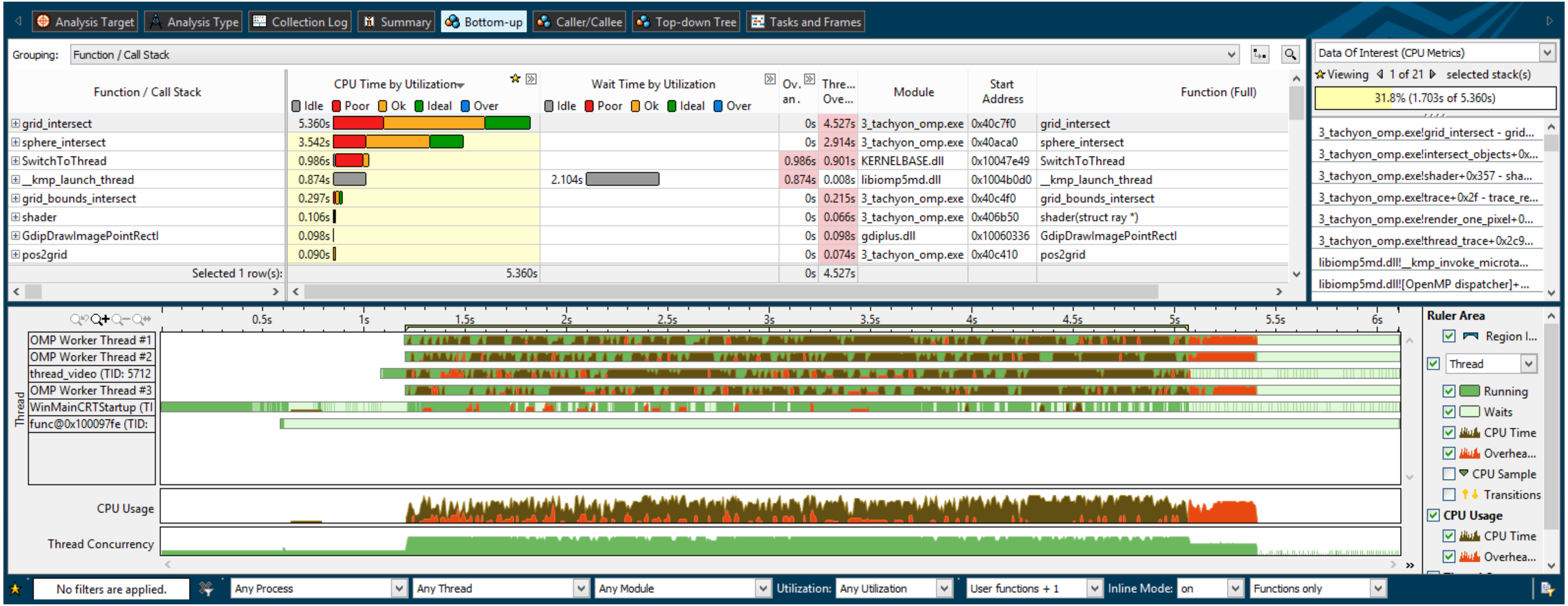


Collection and Platform Info

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

Example: Threading Analysis

Bottom-up View



Find Answers Fast

Intel® VTune™ Amplifier

Adjust Data Grouping

Function - Call Stack
Module - Function - Call Stack
Source File - Function - Call Stack
Thread - Function - Call Stack

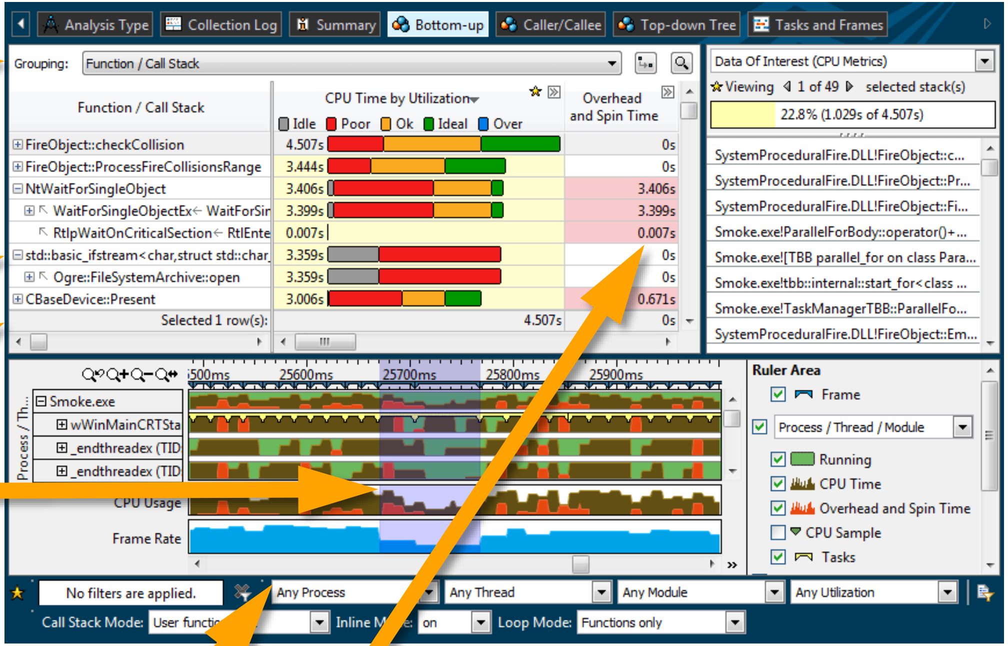
… (Partial list shown)

Double Click Function to View Source

Click [+] for Call Stack

Filter by Timeline Selection (or by Grid Selection)

Zoom In And Filter On Selection
Filter In by Selection
Remove All Filters



Filter by Process & Other Controls

Tuning Opportunities Shown in Pink. Hover for Tips

See Profile Data On Source / Asm

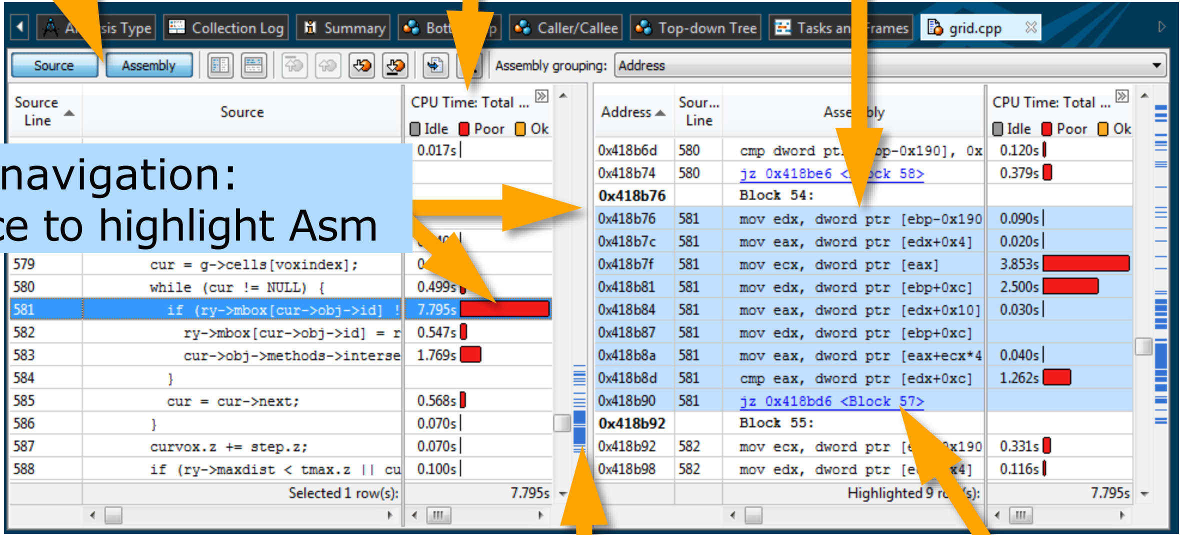
Double Click from Grid or Timeline

View Source / Asm or both

CPU Time

Right click for instruction reference manual

Quick Asm navigation:
Select source to highlight Asm



Scroll Bar “Heat Map” is an overview of hot spots

Click jump to scroll Asm

Command Line Interface

Automate analysis

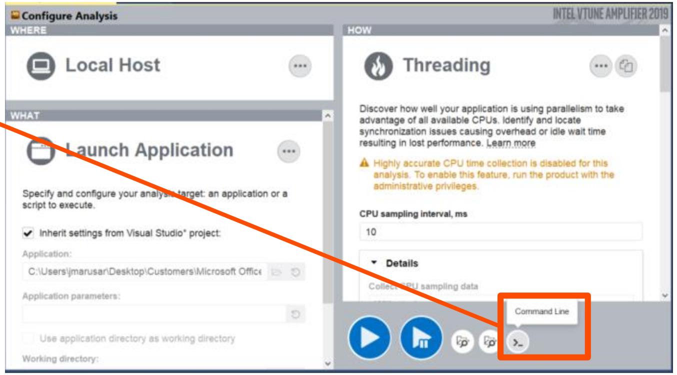
amplxe-cl is the command line:

- **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

Compare Results Quickly - Sort By Difference

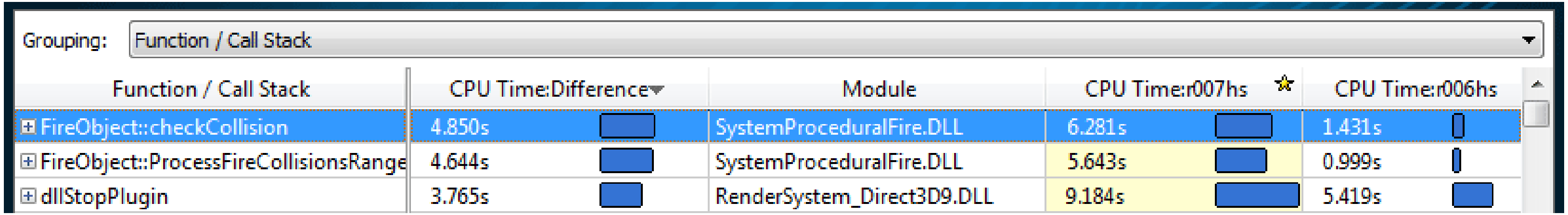
Intel® VTune™ Amplifier

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
FireObject::ProcessFireCollisionsRange	4.644s	SystemProceduralFire.DLL	5.643s	0.999s
dllStopPlugin	3.765s	RenderSystem_Direct3D9.DLL	9.184s	5.419s

Introduction to Performance Tuning

System

Application

Processor

H/W tuning:
- BIOS (TB, HT)
- Memory
- Network I/O
- Disk I/O

OS tuning:
- Page size
- Swap file
- RAM Disk
- Power settings
- Network protocols

Better application design:
- Parallelization
- Fast algorithms / data bases
- Programming language and RT libs
- Performance libraries
- Driver tuning

Tuning for Microarchitecture:
- Compiler settings/Vectorization
- Memory/Cache usage
- CPU pitfalls

Design – Think performance wise (app/sys level)

Prototyping – Choose performance. effective solutions

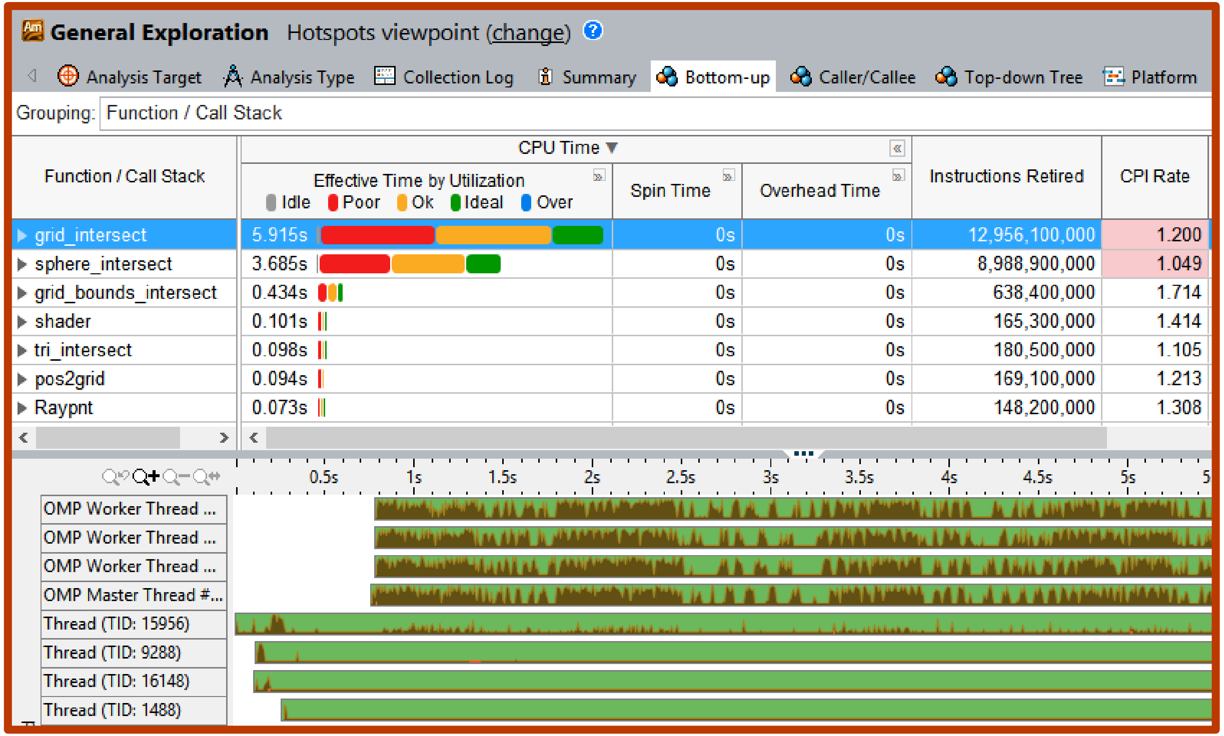
Implementation – Apply performance optimization and check results

Testing – Add performance regressions to test stage

Release – Collect and analyze performance related issues from users

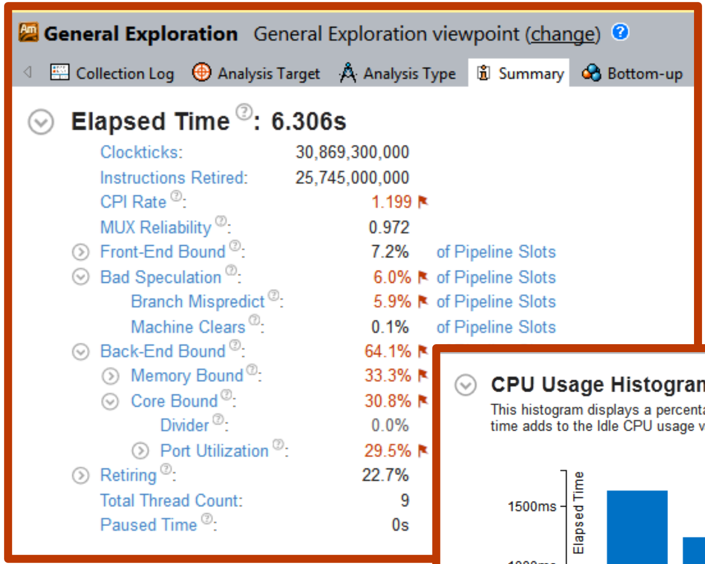
Introduction to Intel VTune Amplifier

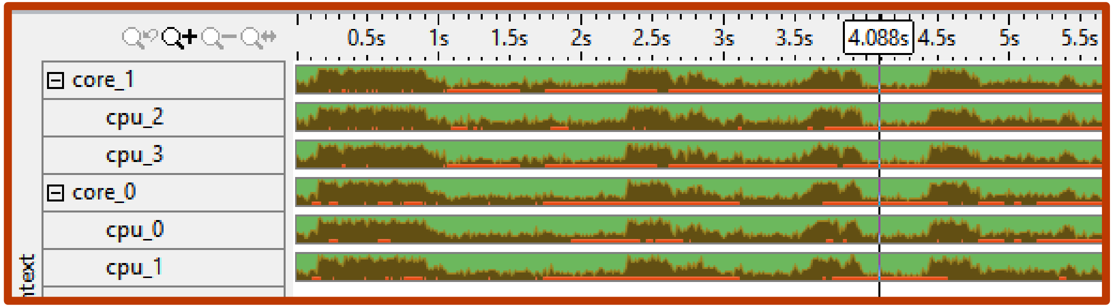
- Accurate Data - Low Overhead
  - CPU, GPU, FPU, threading, bandwidth, and more…
  - Profile applications or systems
- Meaningful Analysis
  - Threading and hardware utilization efficiency
  - Memory and storage device analysis
- Easy
  - Data displayed by source code
  - Expert advice built-in
  -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

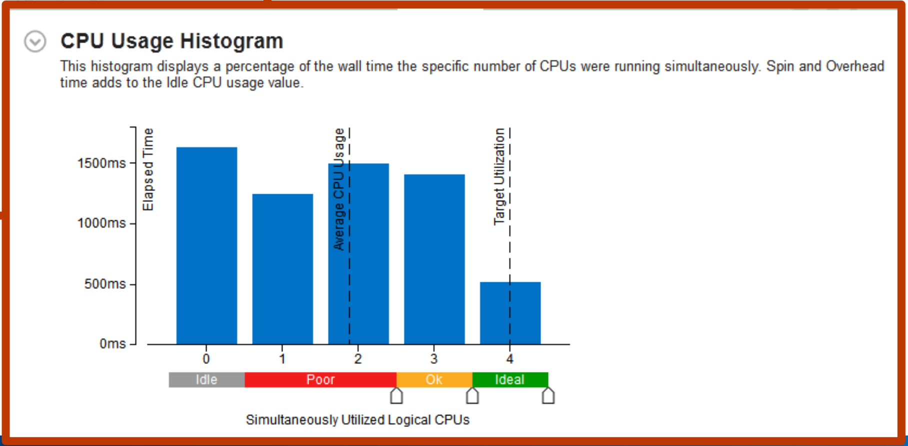



```
> amplxe-cl -help collect
```

System-Level Profiling – High-level Overviews





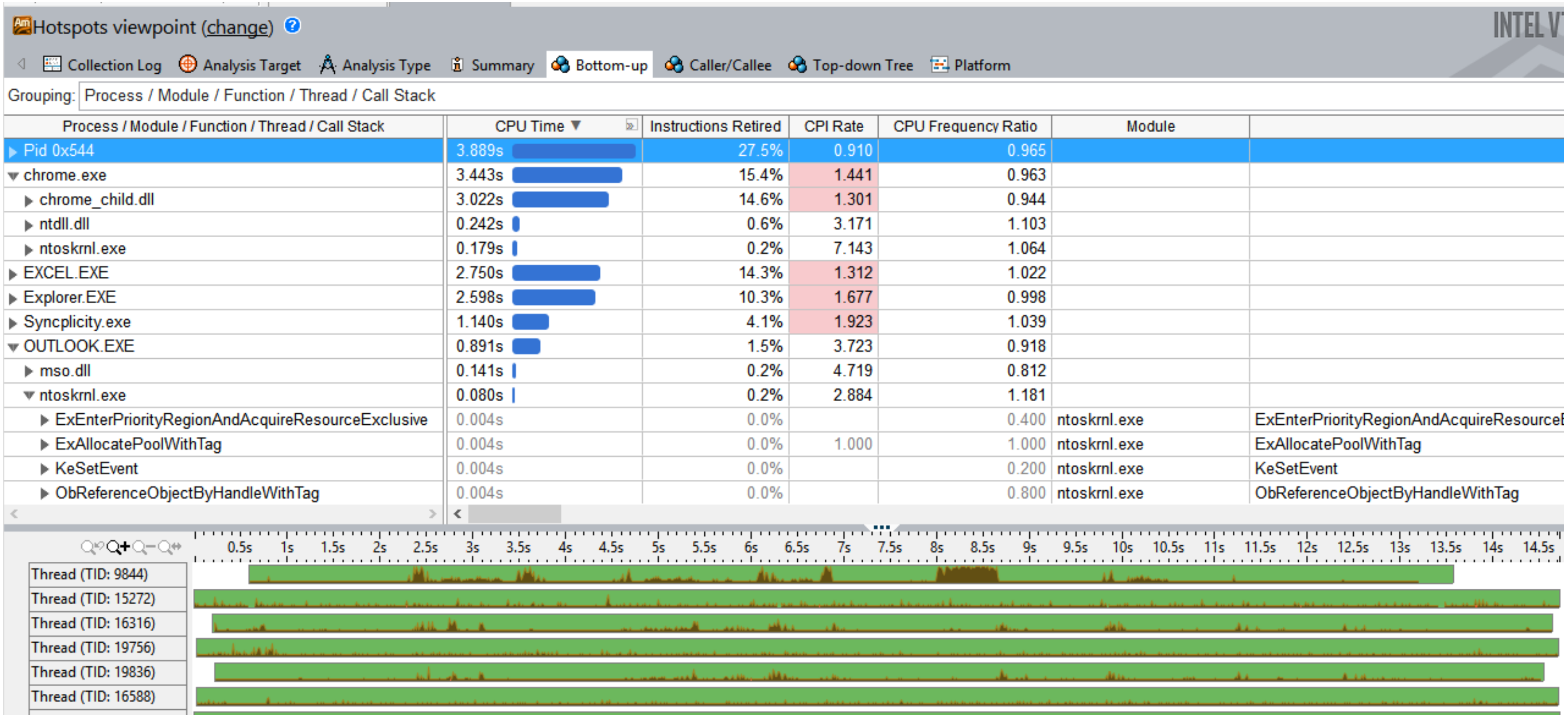


System-Level Profiling – Process/Module Breakdowns

Processes

Modules

Functions



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		
chrome.exe	3.443s	15.4%	1.441	0.963		
chrome_child.dll	3.022s	14.6%	1.301	0.944		
ntdll.dll	0.242s	0.6%	3.171	1.103		
ntoskrnl.exe	0.179s	0.2%	7.143	1.064		
EXCEL.EXE	2.750s	14.3%	1.312	1.022		
Explorer.EXE	2.598s	10.3%	1.677	0.998		
Syncplicity.exe	1.140s	4.1%	1.923	1.039		
OUTLOOK.EXE	0.891s	1.5%	3.723	0.918		
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	ExEnterPriorityRegionAndAcquireResourceE
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

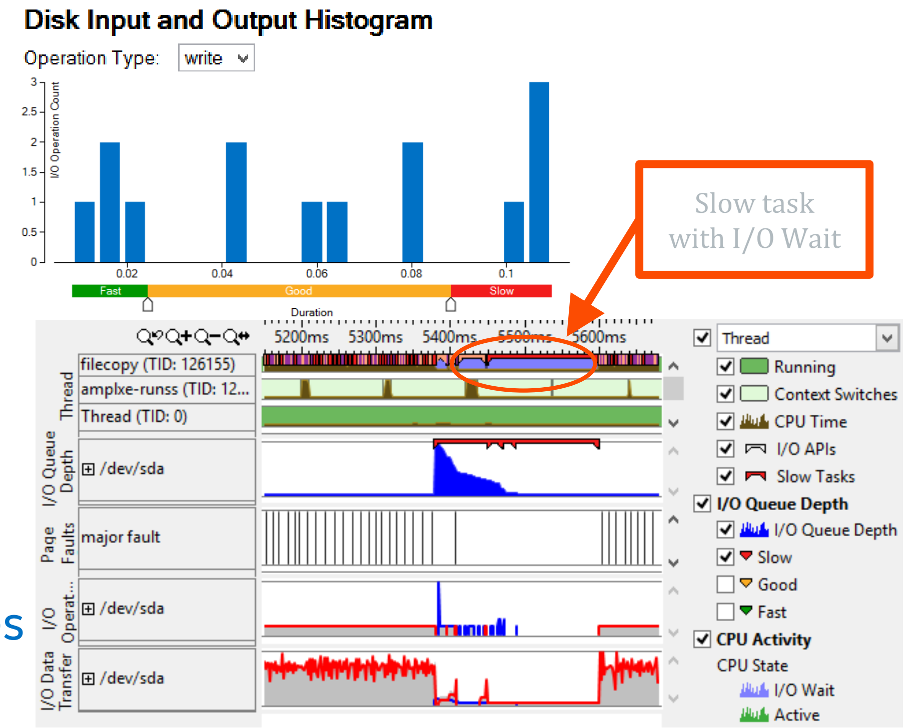
System-Level Profiling – I/O Analysis

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



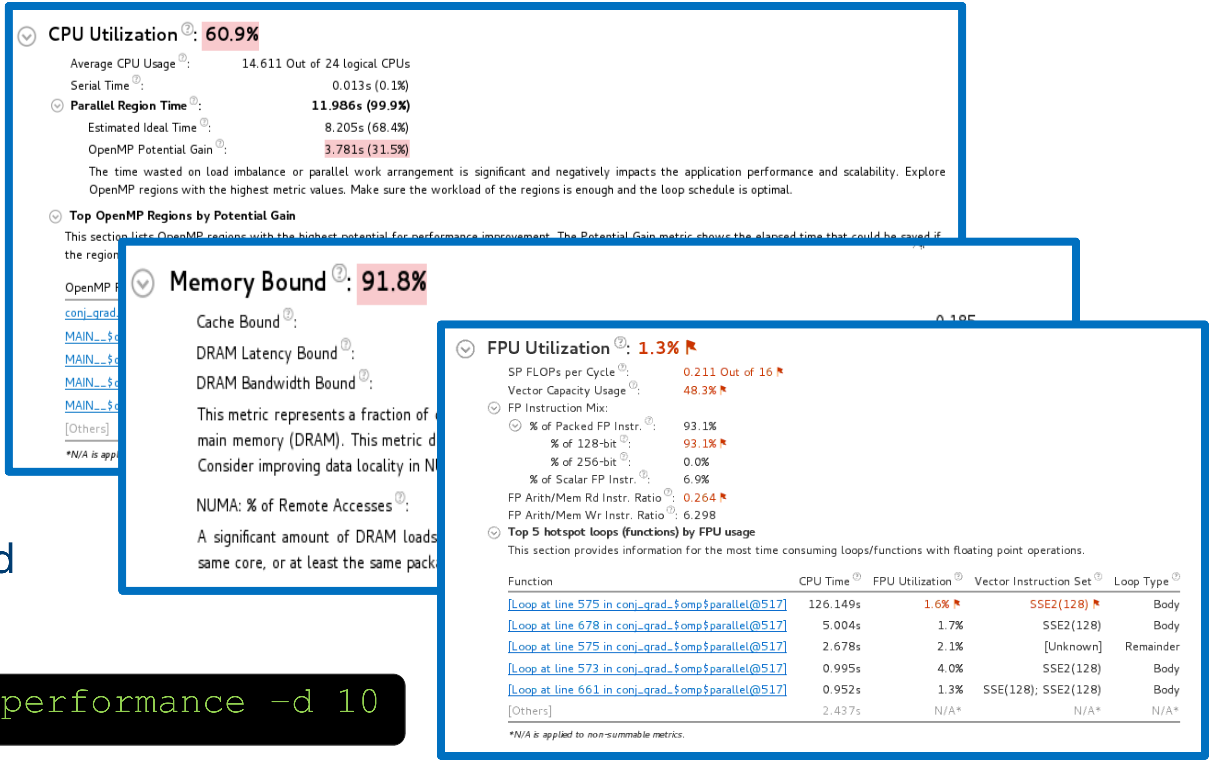

```
> amplxe-cl -collect io -d 10
```

System-Level Profiling – HPC Characterizaton

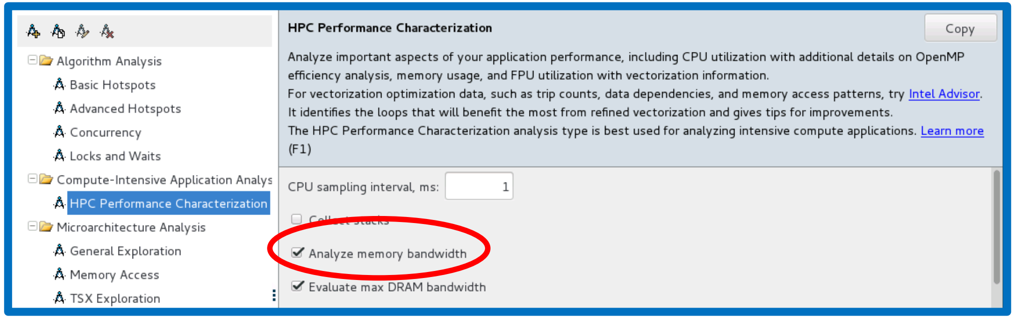
Three Metric Classes

- 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

```
> amplxe-cl -collect hpc-performance -d 10
```

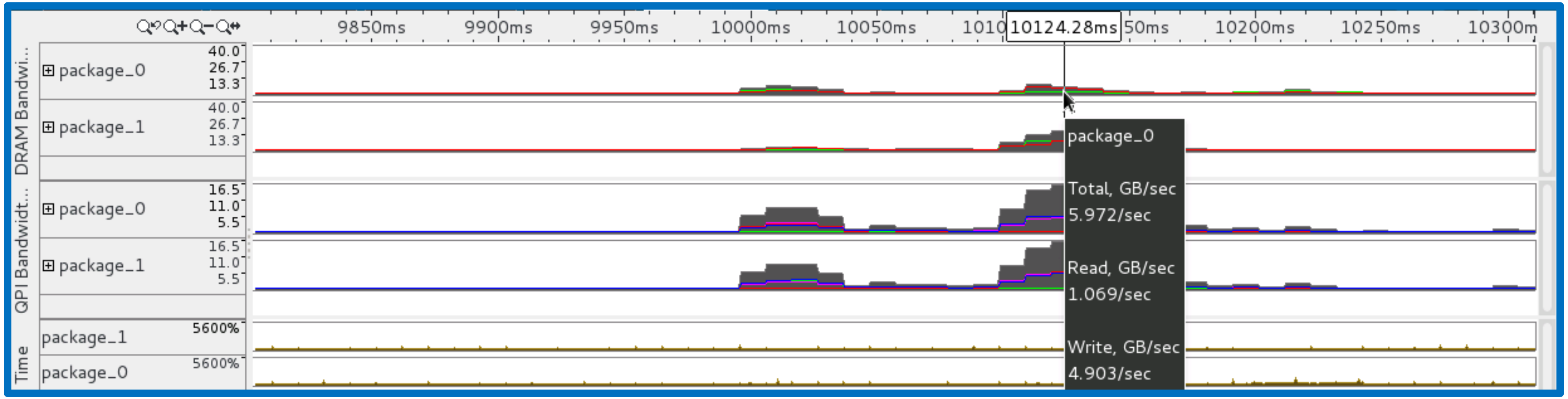


System-Level Profiling – Memory Bandwidth

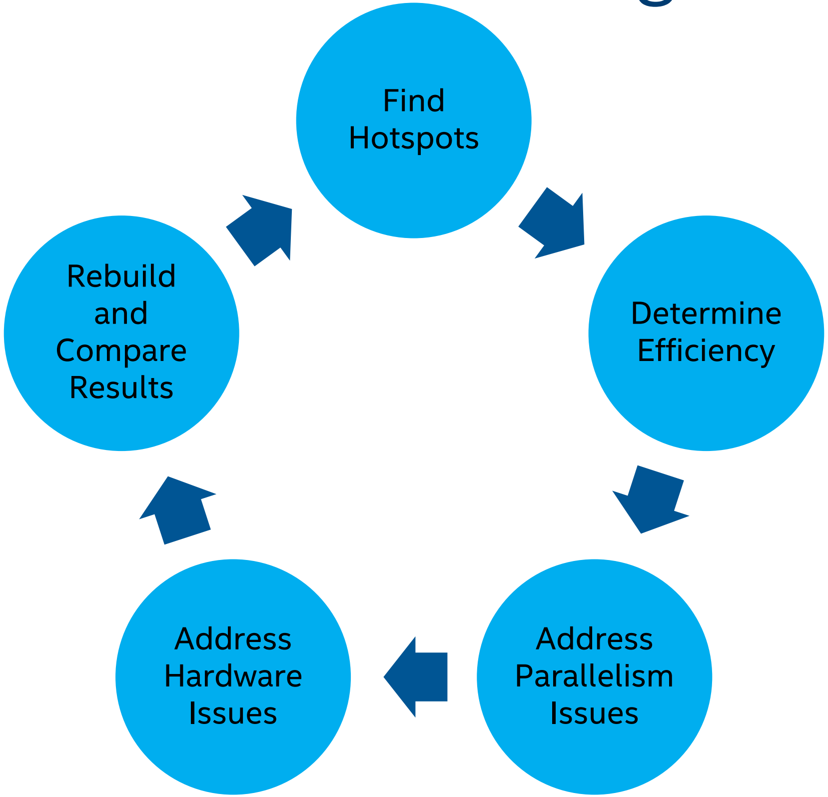


Find areas of high and low bandwidth usage. Compare to max system bandwidth based on Stream benchmarks.

```
-knob collect-memory-bandwidth=true
```

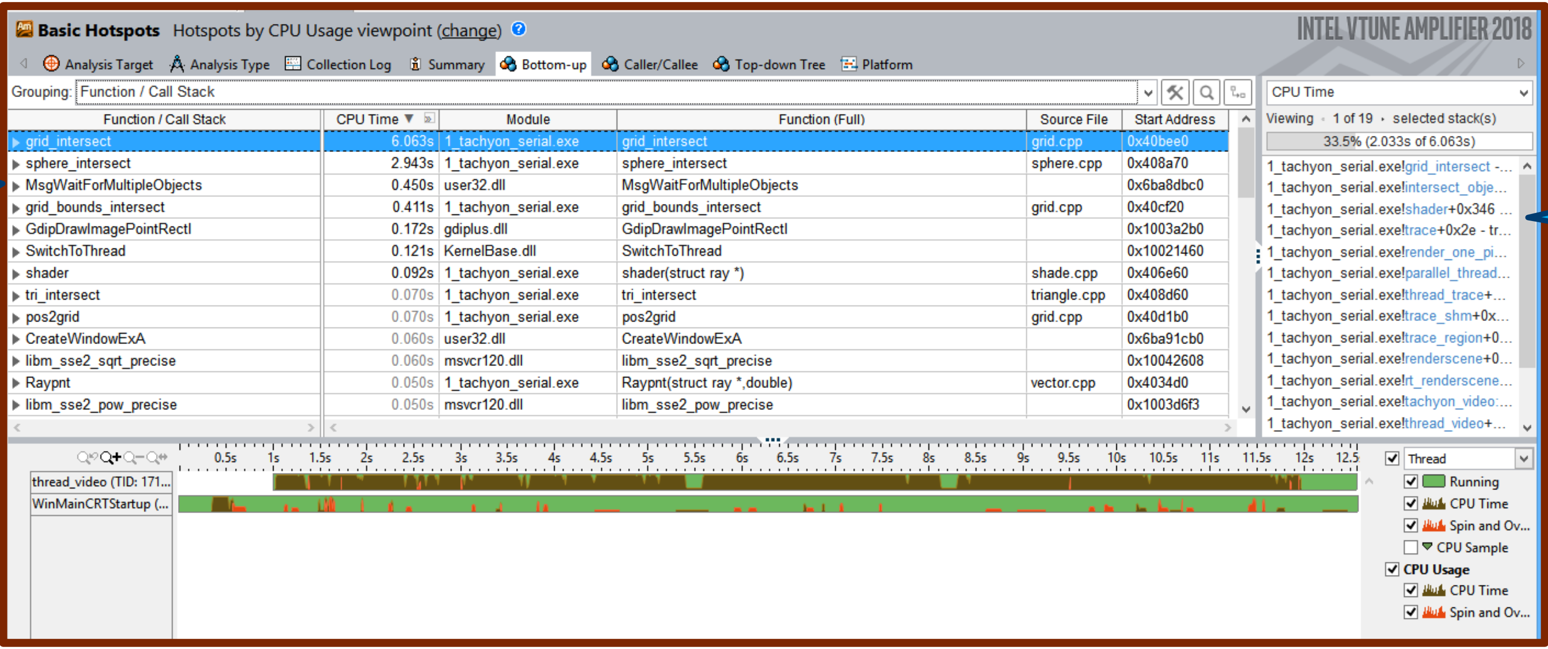


Application Performance Tuning Process



Find Hotspots

Functions



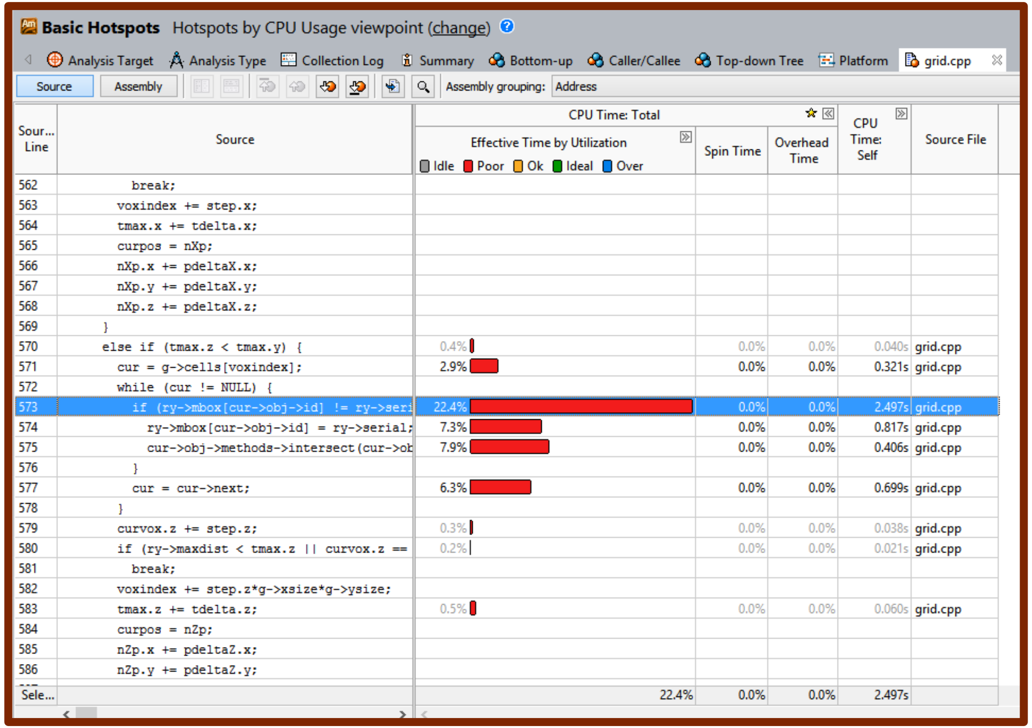
Call Stacks

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

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



Determine Efficiency

General Exploration Hotspots viewpoint (change)

Analysis Target | Analysis Type | Collection Log | Summary | Bottom-up | Caller/Callee | Top

Grouping: Function / Call Stack

Function / Call Stack	CPU Time: Effective Time by Utilization (Idle, Poor, Ok, Ideal, Over)	CPU Time: Spin Time	CPU Time: Overhead Time
grid_intersect	5.915s	0s	0s
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 Exploration General Exploration viewpoint (change)

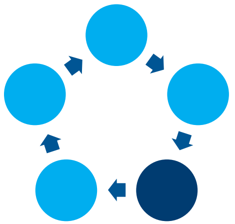
Analysis Target | Analysis Type | Collection Log | Summary | Bottom-up

Grouping: Function / Call 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%	

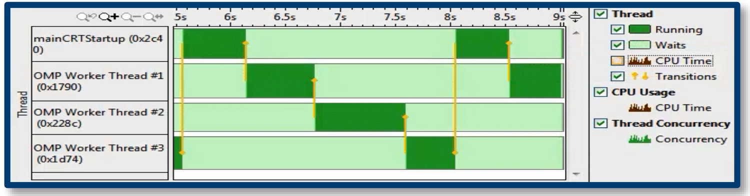
Look for Parallelism, Cycles-per-Instruction (CPI), and Retiring %

Address Parallelism Issues

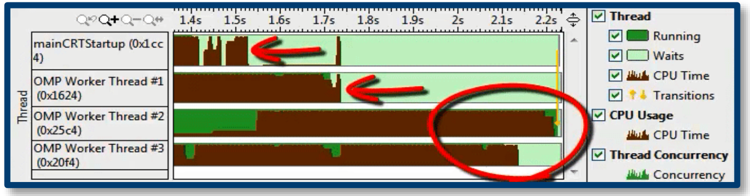


- Use Concurrency Analysis to ensure you're using all your threads as often as possible.
- Common concurrency problems can often be diagnosed in the timeline.
- Switch to the Locks And Waits viewpoint or run a Locks and Waits analysis to investigate contention.

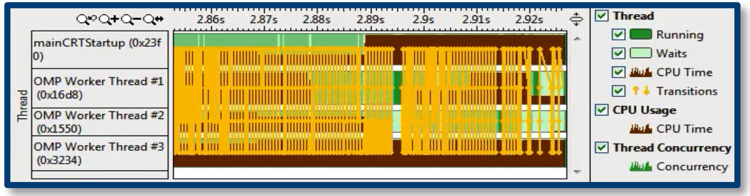
Coarse-Grain Locks



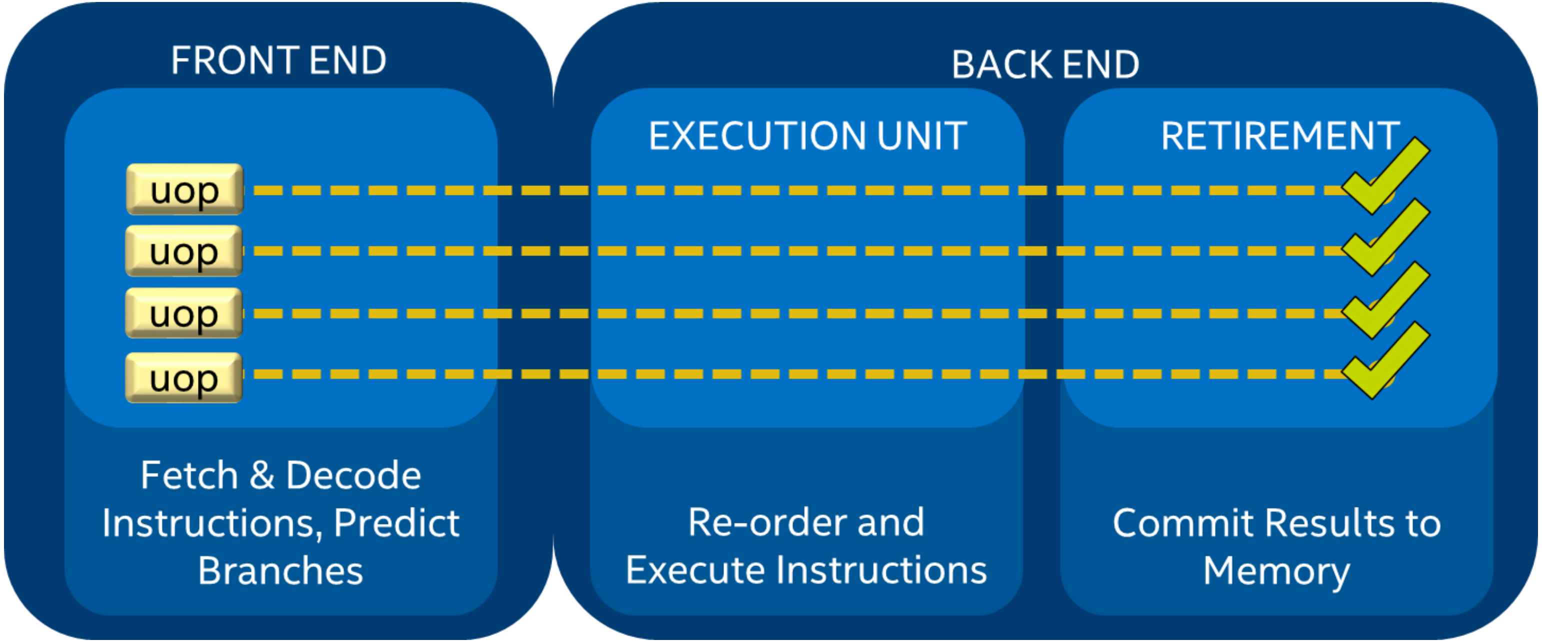
Thread Imbalance



High Lock Contention



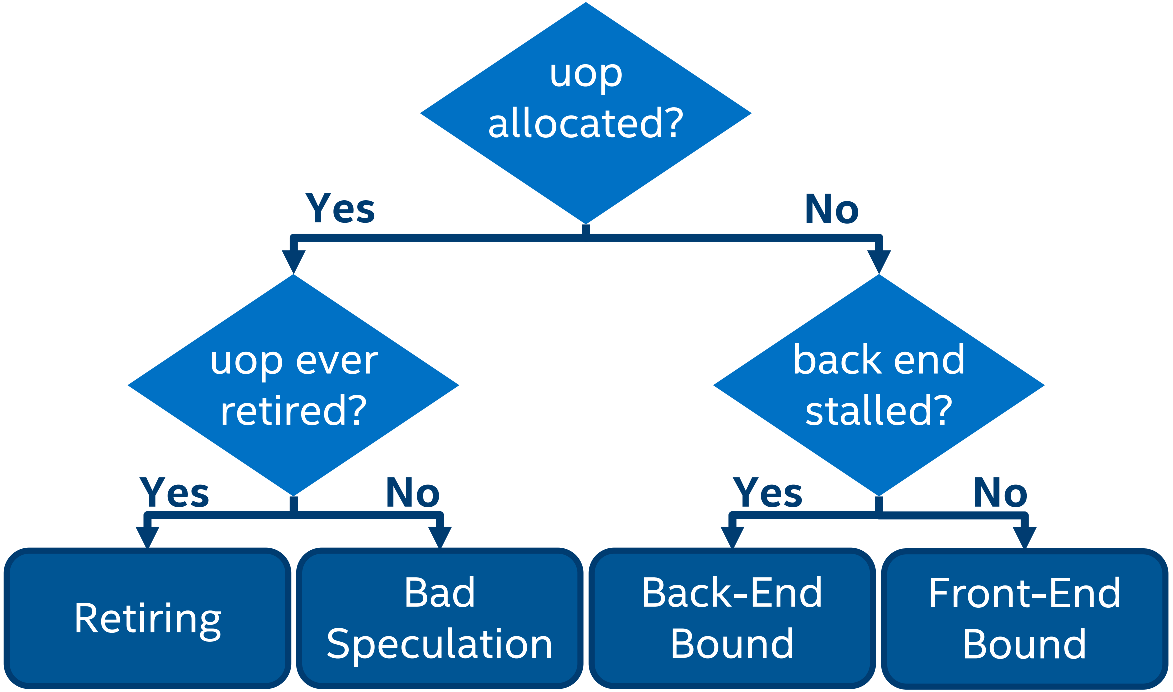
Address Hardware Issues



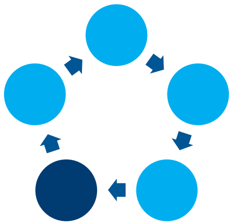
The X86 Processor Pipeline (simplified)

Address Hardware Issues

For each pipeline slot on each cycle:



Address Hardware Issues



Analysis Target | Analysis Type | Collection Log | Summary | Bottom-up | Event Count | Platform

Grouping: Function / Call Stack

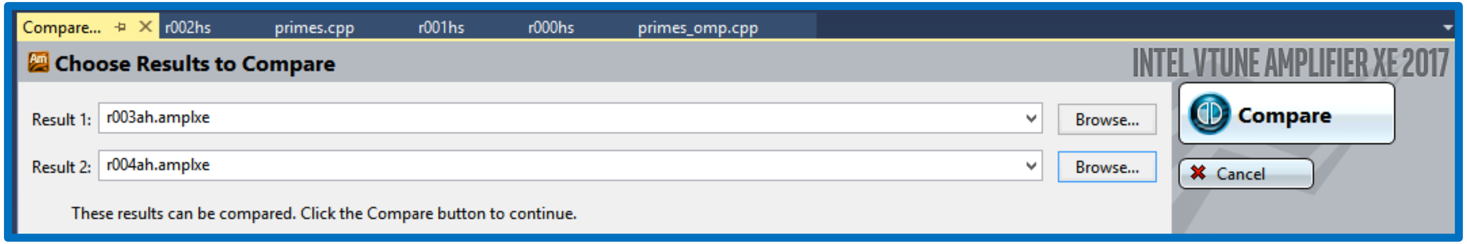
Function / Call Stack	Retiring	Front-End Bound	Bad Speculation	Back-End Bound		Module
				Memory Bound	Core Bound	
grid_intersect	22.5%	6.5%	4.5%	34.6%	31.8%	3_tachyon_omp.exe
sphere_intersect	23.9%	6.2%	11.5%	29.0%	29.4%	3_tachyon_omp.exe
grid_bounds_intersect	16.5%	11.3%	8.7%	31.8%	31.8%	3_tachyon_omp.exe
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_omp.exe
Raypnt	39.2%	4.9%	0.0%	0.0%	90.2%	3_tachyon_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

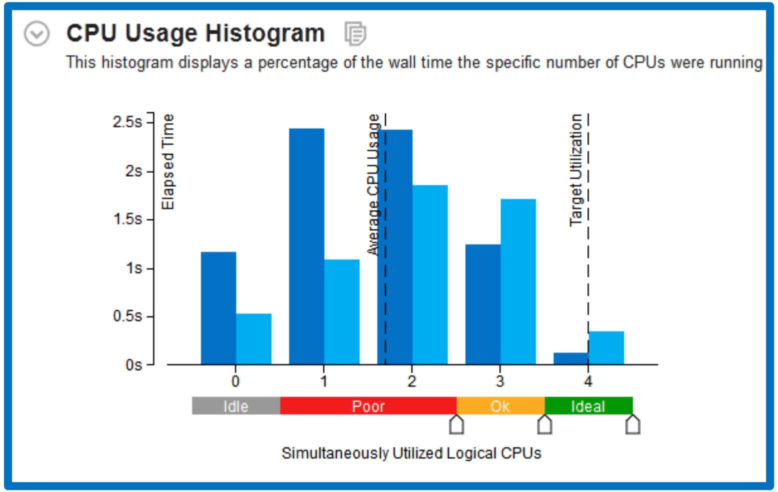
This data is collected statistically with event multiplexing. Gray data has low confidence levels.

Microarchitecture Exploration Analysis Shows the Hardware Bottleneck

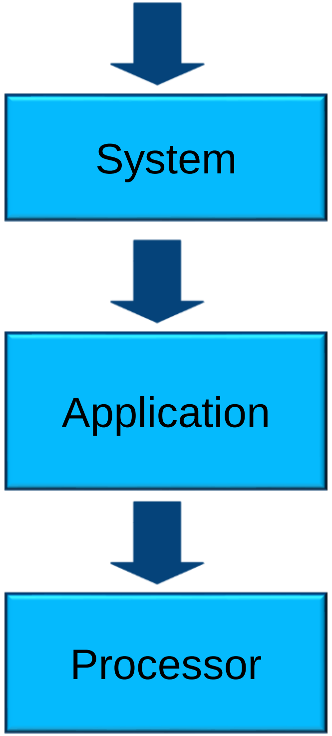
```
> amplxe-cl -collect uarch-exploration -- ./myapp.out
```

Rebuild and Compare Results

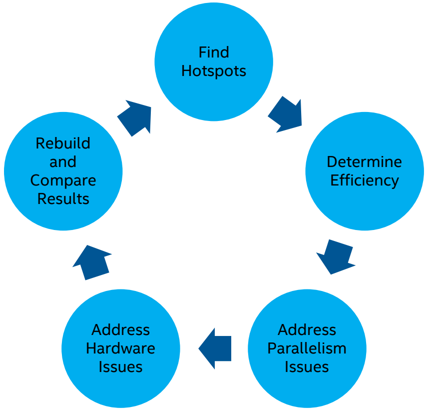




Summary



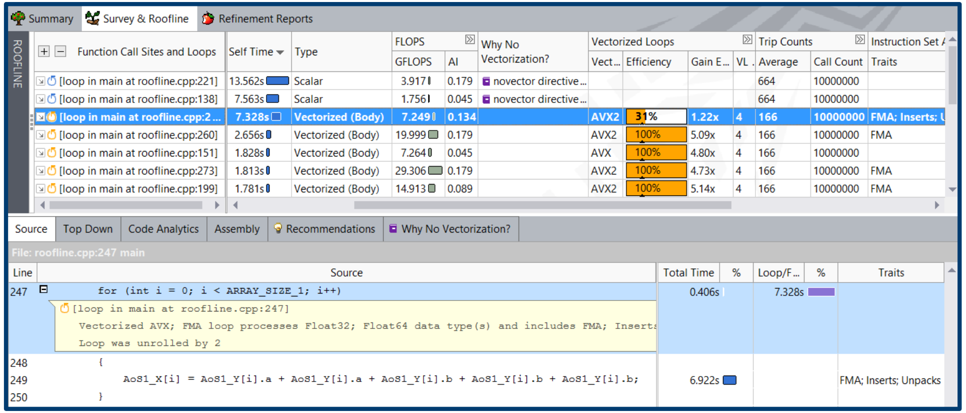
- Start with the lowest hanging fruit for performance tuning
- Use Intel® VTune™ Amplifier for system and application profiling
- Hotspots, HPC Characterization, and General Exploration are good starting points
- Performance tuning is an iterative process

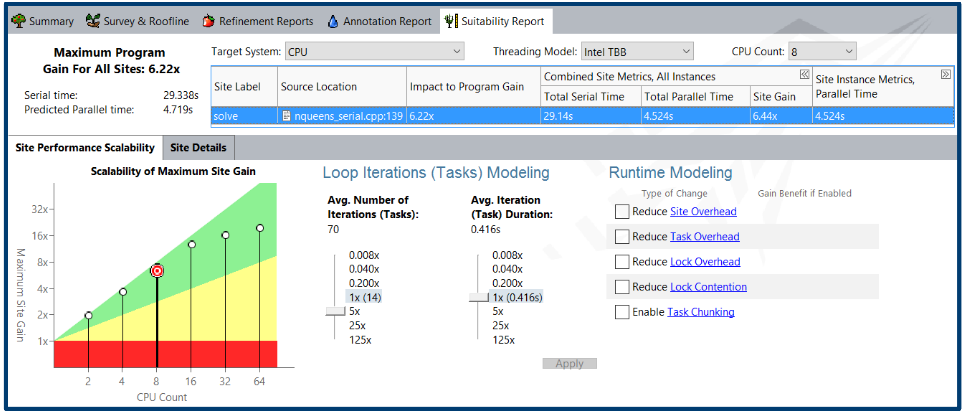


INTEL® ADVISOR

Agenda

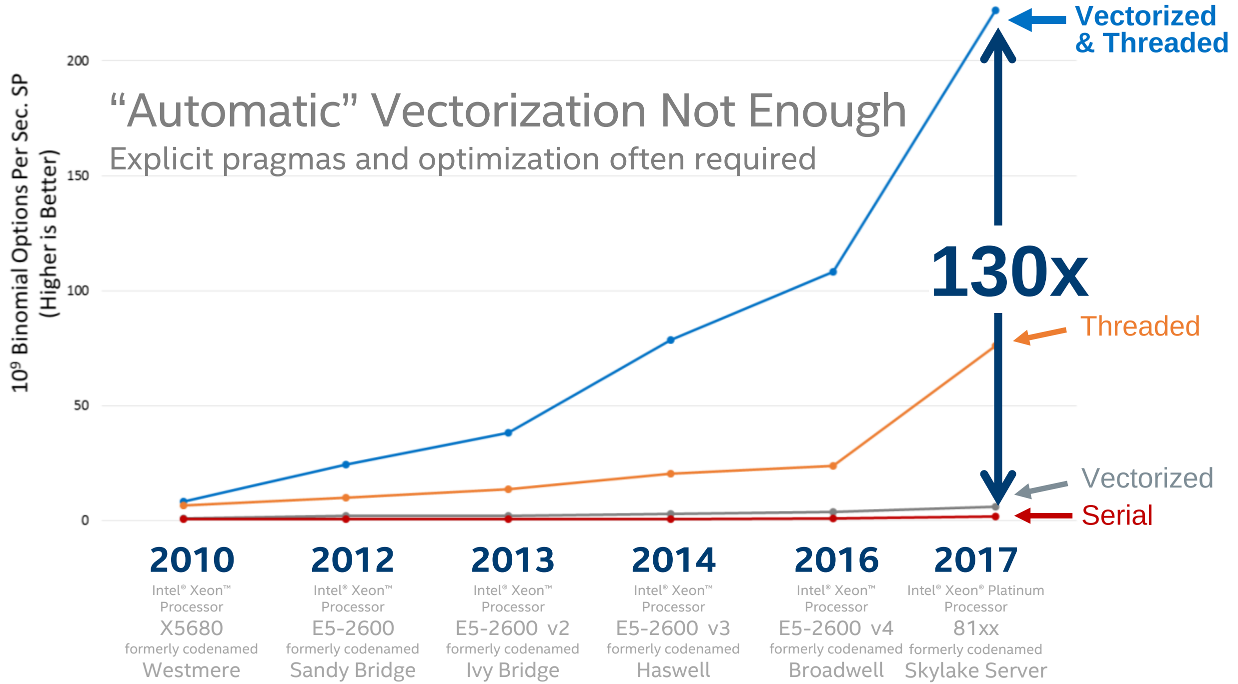
- Motivation
- Threading Advisor
  - Threading Advisor Workflow
  - Advisor Interface
  - Survey Report
  - Annotations
  - Suitability Analysis
  - Dependencies Analysis
- Vectorization Advisor & Roofline
  - Vectorization Advisor recap
  - Roofline
  - Memory Access Patterns Analysis
  - Dependencies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





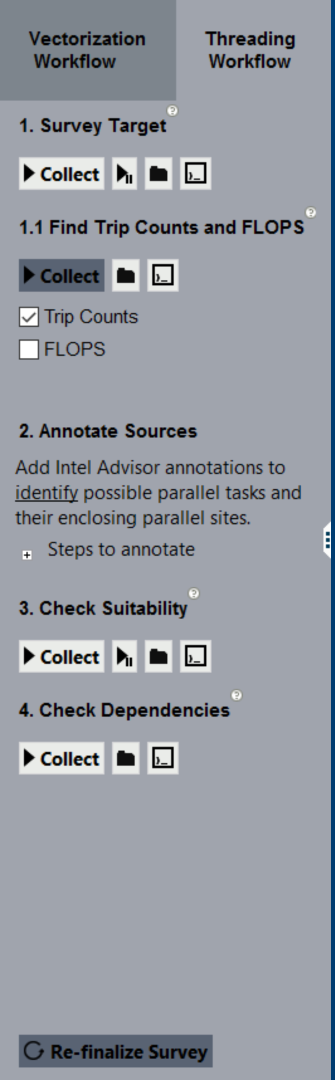
Vectorize & Thread or Performance Dies

Threaded + Vectorized can be much faster than either one alone



The Difference Is Growing With Each New Generation of Hardware

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.



THREADING ADVISOR

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

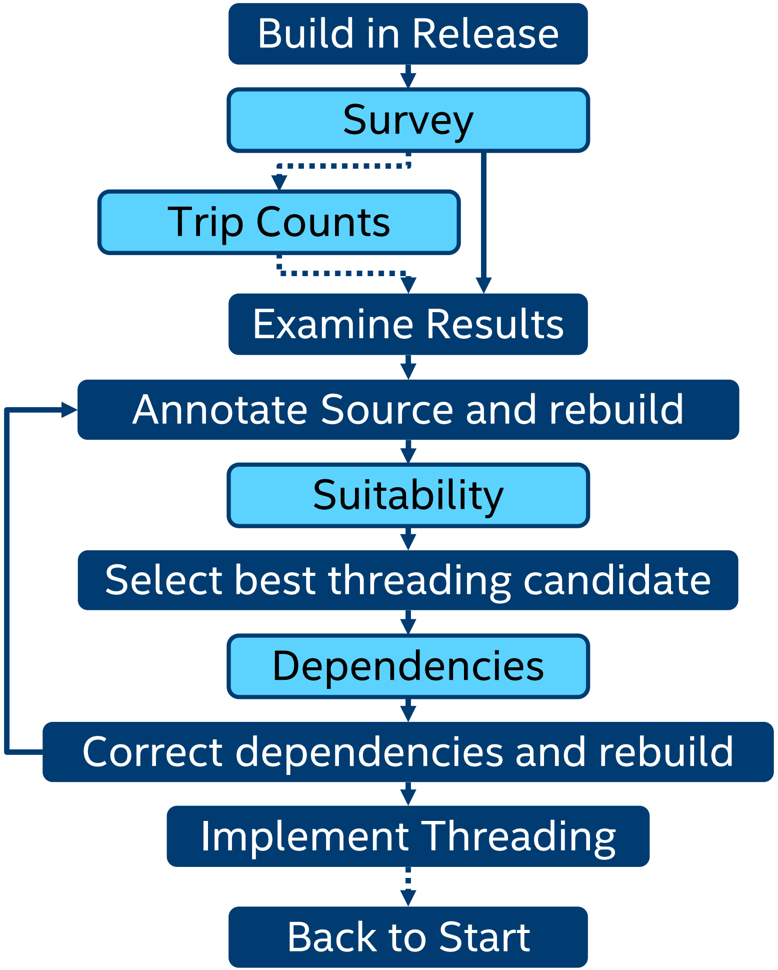
3) All of your test suites should still pass.
Validate the correctness of your transformations.

4) You can use Advisor on partially or completely parallelized code.

Design, measure and test <u>before</u> 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.



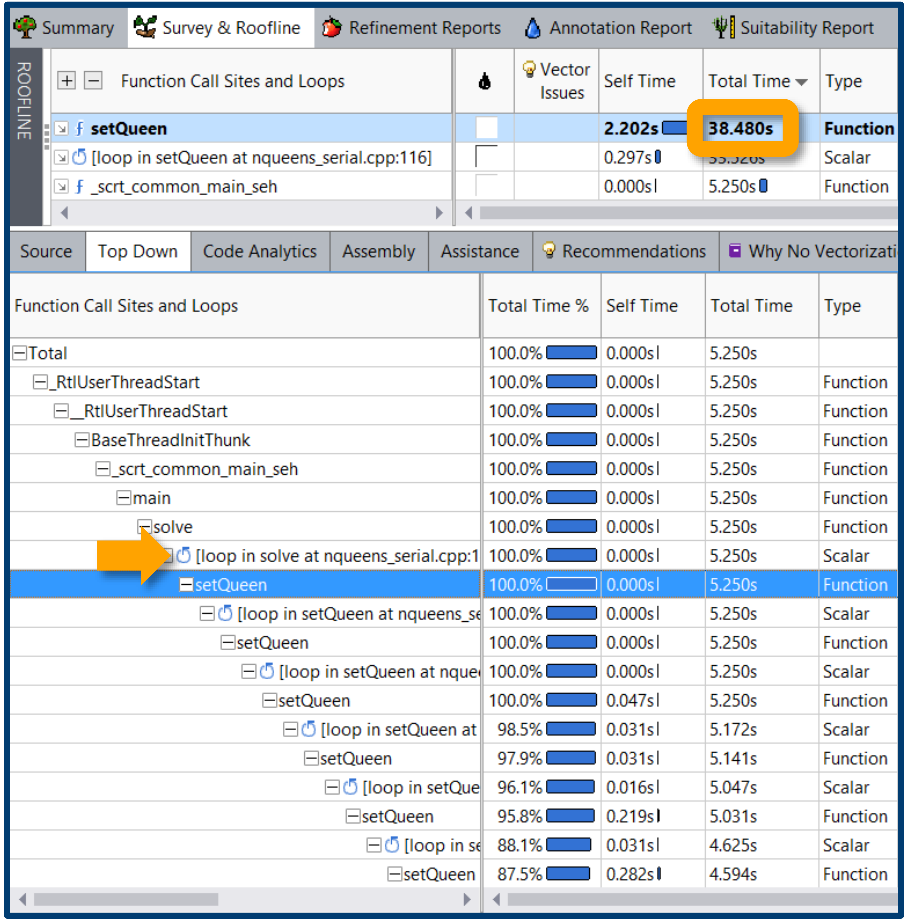
Survey Report

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.



Annotating Your Code

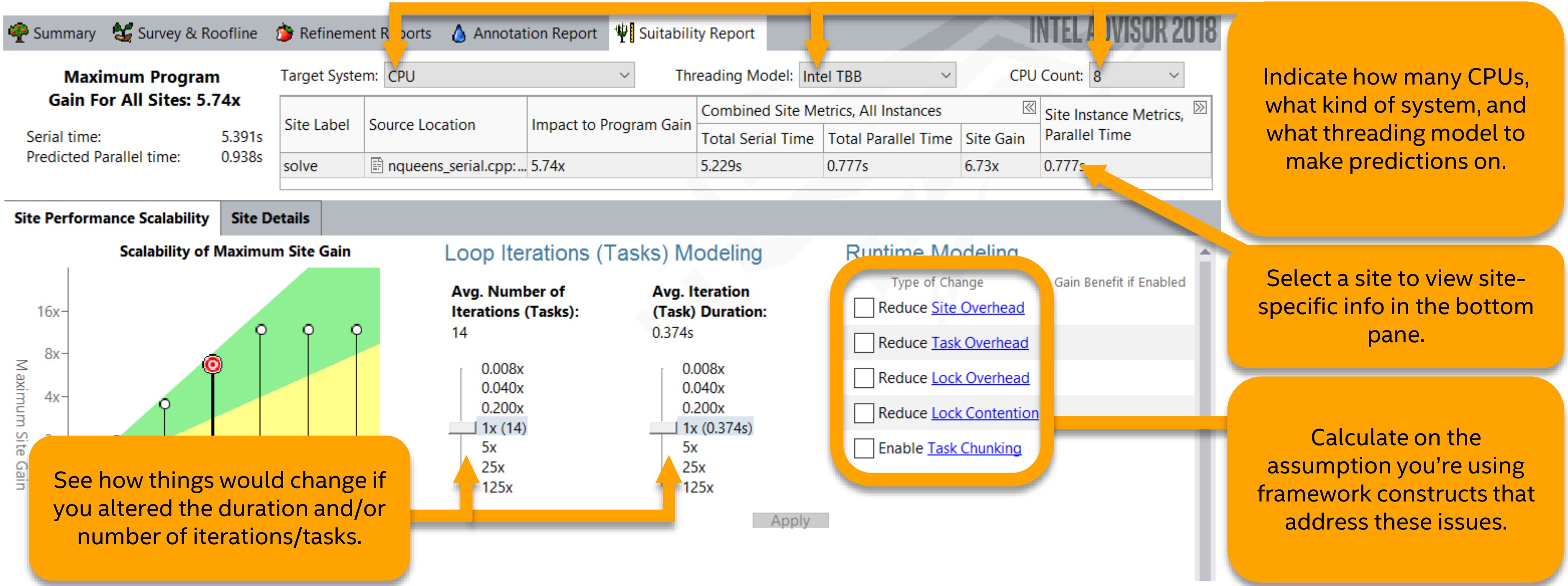
- Annotations are notes to Advisor. They are *not* parallelization commands. They do not affect the way the program itself runs.
- They mark places Advisor should treat as locks or parallel sites.
- To use annotations, you must include the appropriate header/module.

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

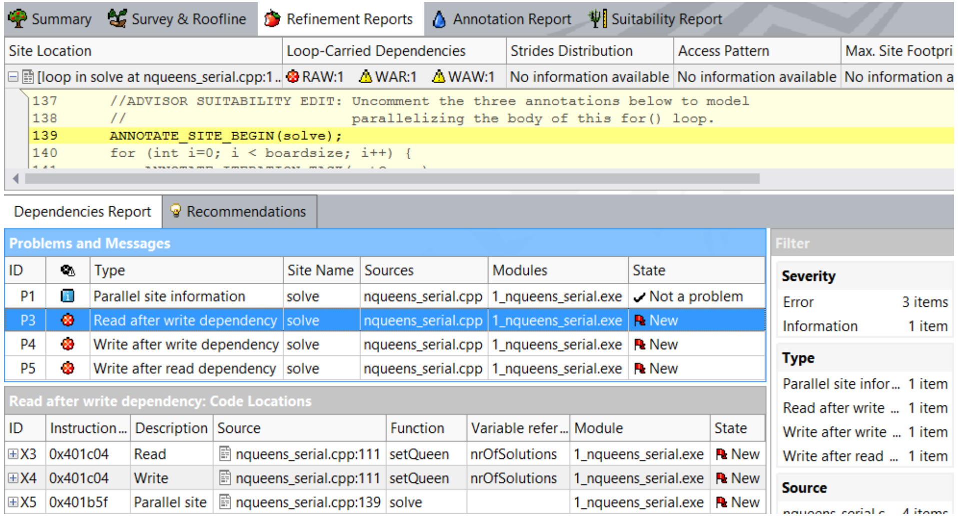
- Using your annotations, Advisor models how the program would behave in parallel, and predicts performance in specified hypothetical circumstances.



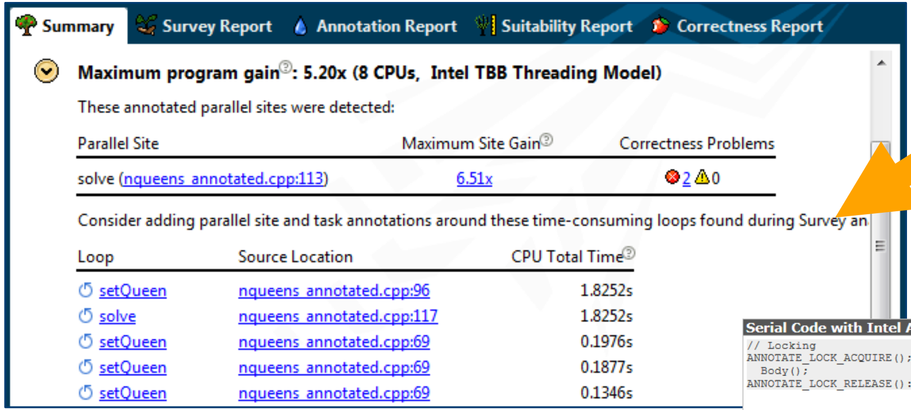
Dependencies Analysis

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 re-run the analysis to confirm the problem has been resolved.
- 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.

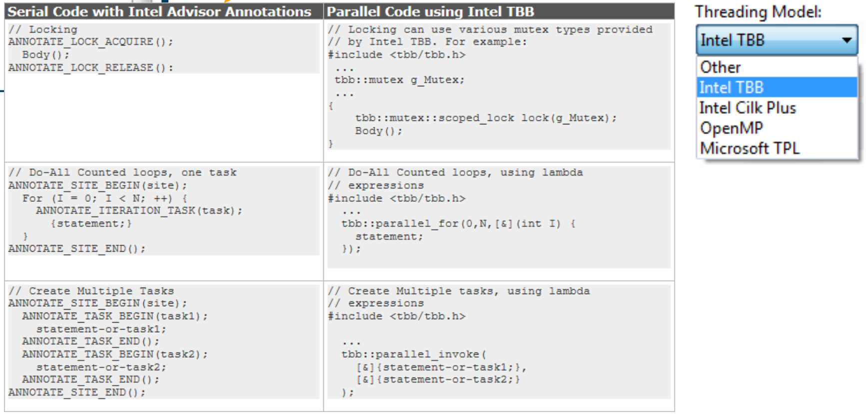


Add Parallel Framework



Here is the list of source locations

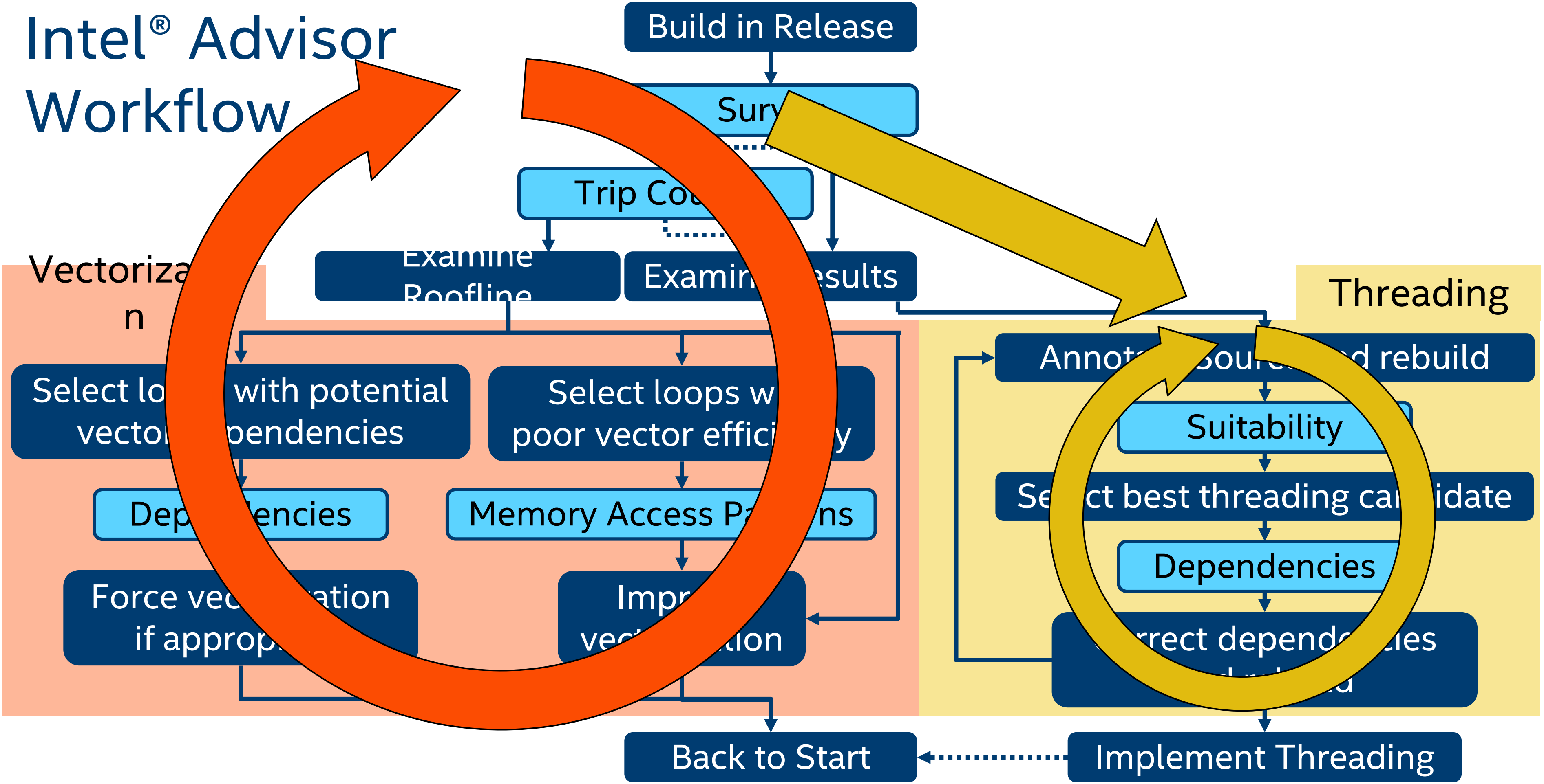
Here are templates for popular parallel frameworks

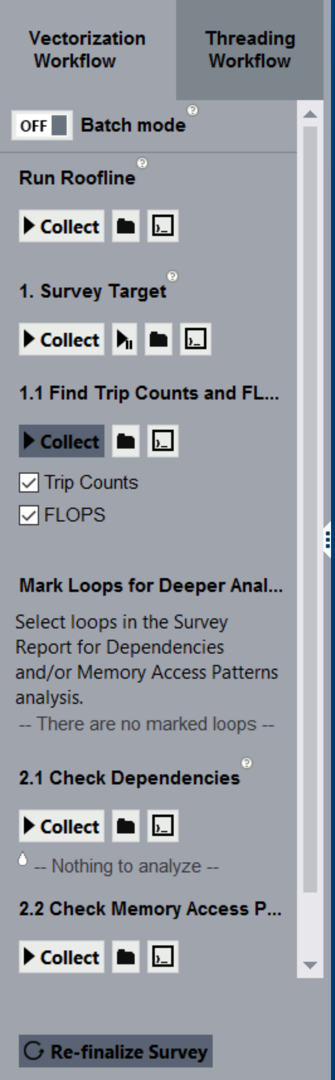


Intel® Advisor

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

Intel® Advisor Workflow

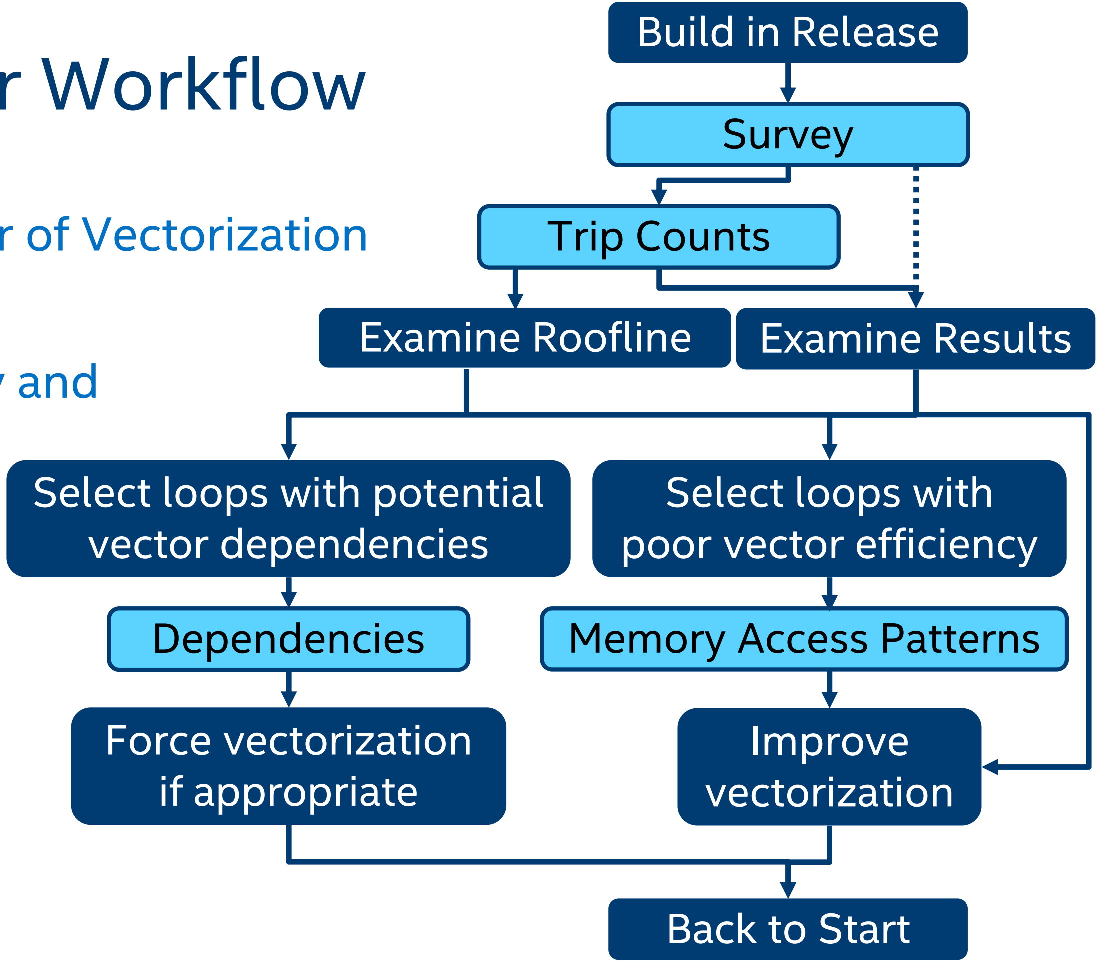




VECTORIZATION ADVISOR & ROOFLINE

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 important!

Efficiency=100% $\frac{\text{Speedup}}{\text{Vec. Length}}$

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

- 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		Vector Issues	Self Time	Total Time	Type	Why No Vectorization?	Vectorized Loops: Vect...	Efficiency	Gain...	VL ...
[loop in main at example.cpp:38]		1 Assumed depend...	0.391s	0.391s	Scalar	vector depen...				
[loop in main at example.cpp:64]		1 Possible ineffficien...	0.297s	0.297s	Vector...		AVX2	2%	0.37x	16
[loop in main at example.cpp:51]		1 Possible ineffficien...	0.094s	0.094s	Vector...	1 vectorizatio...	AVX2	8%	1.23x	16
[loop in main at example.cpp:26]			0.030s	0.030s	Vector...		AVX2	100%	7.98x	8
[loop in main at example.cpp:14]		3 Assumed depend...	0.000s	0.000s	Scalar	vector depen...				
[loop in main at example.cpp:23]			0.000s	0.030s	Scalar	inner lo... w...				

Expand a vectorized loop to see it split into body, peel, and remainder (if applicable).

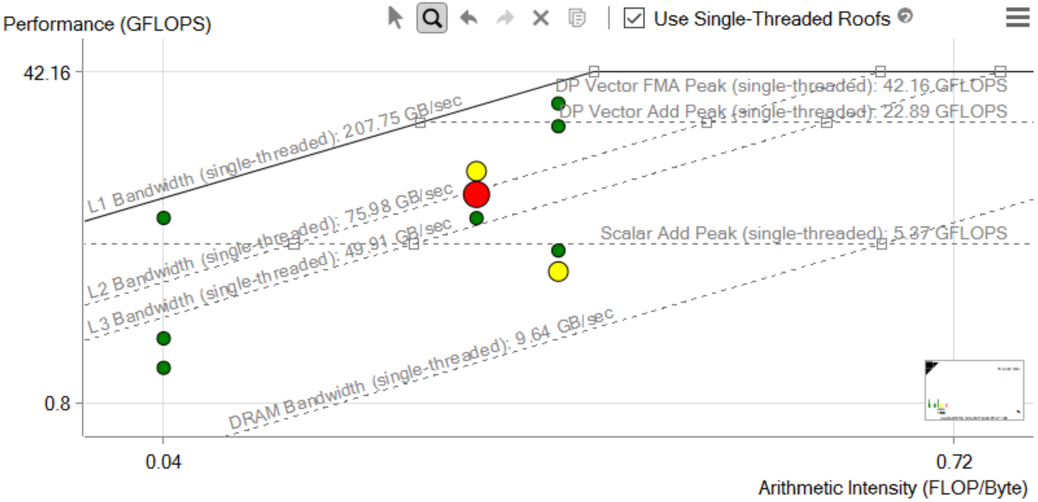
Advisor *advises* you on potential vector issues. This is often your cue to run MAP or Dependencies. Click the icon to see an explanation in the bottom pane.

The Intel Compiler embeds extra information that Advisor can report in addition to its sampled data, such as why loops failed to vectorize.

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:
Roofline: An Insightful Visual Performance Model for Multicore Architectures, 2009
Cache-aware variant proposed by University of Lisbon:
Cache-Aware Roofline Model: Upgrading the Loft, 2013

Roofline Metrics

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

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

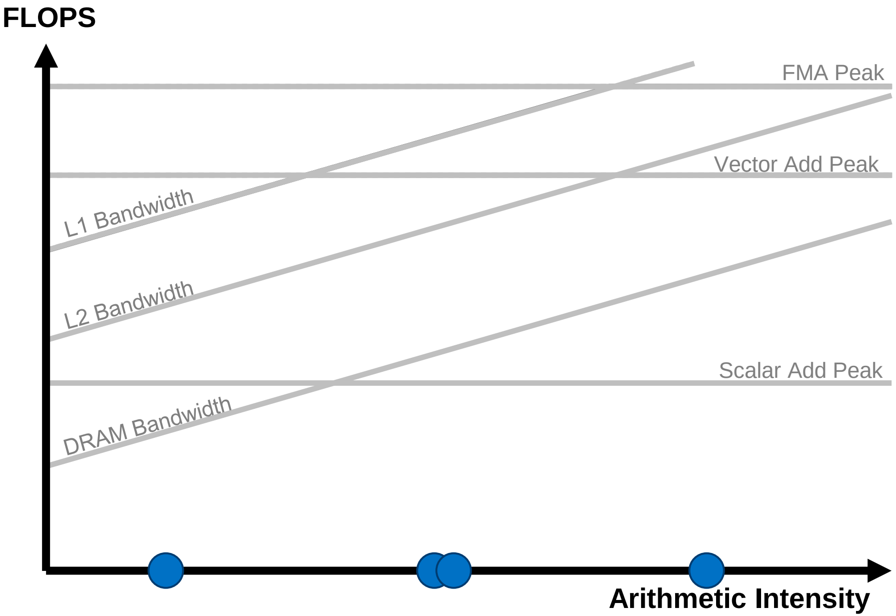
- **FLOPS**: **Fl**oating-Point **Op**erations / **S**econd
  - Is a measure of an implementation (it achieves a certain FLOPS)
  - ***And*** there is a maximum that a platform can provide

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*



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

Cache-Aware Roofline

Next Steps

If Under the Vector Add Peak

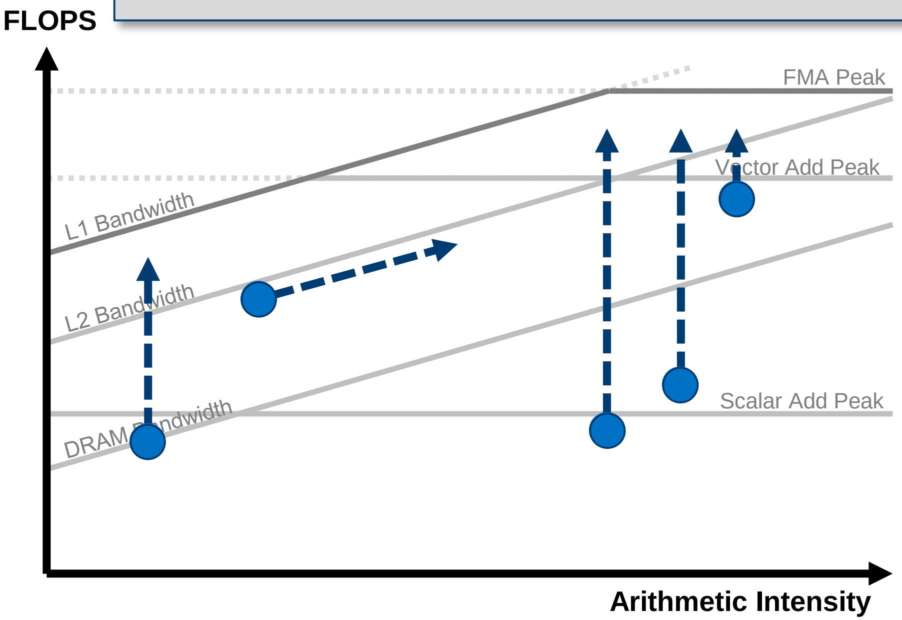
Check “Traits” in the Survey to see if FMAs are used. If not, try altering your code or compiler flags to **induce FMA usage.**

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 or near a memory roof…

- Try a MAP analysis.
  Make any appropriate **cache optimizations**.
- If cache optimization is impossible, try **reworking the algorithm to have a higher AI.**



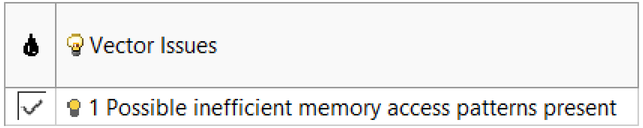
If under the Scalar Add Peak…

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.

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.



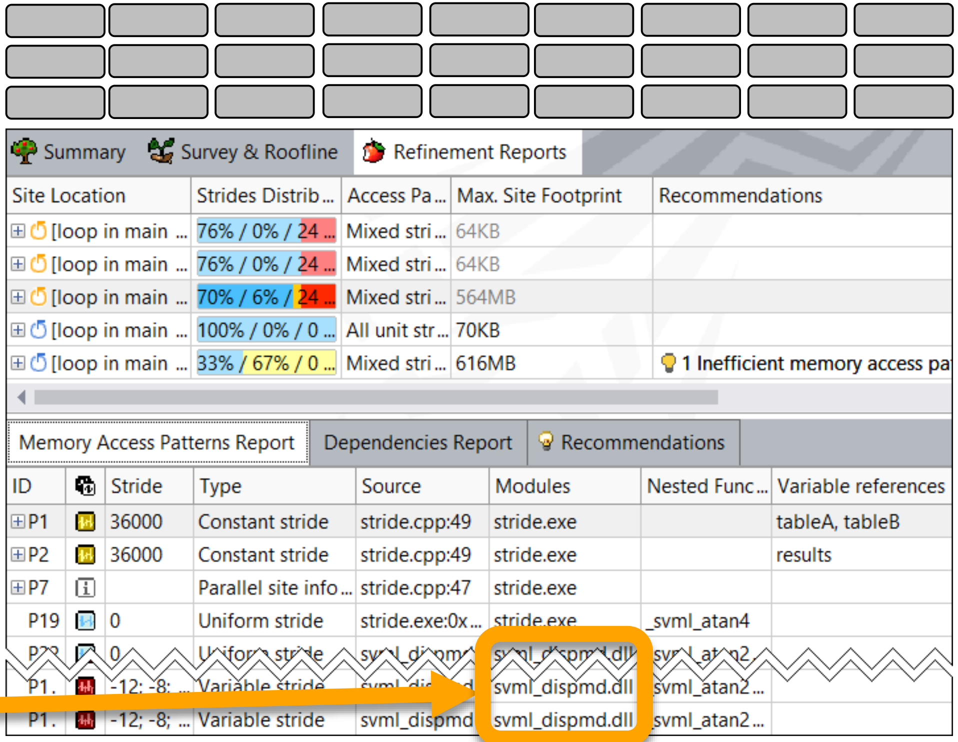
- Memory access patterns affect vectorization efficiency because they affect how data is loaded into and stored from the vector registers.
- Select the loops you want to run the MAP on using the checkboxes. It may be helpful to reduce the problem size, as MAP only needs to detect patterns, and has high 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!



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

> Recommendation: Confirm dependency is real — Confidence: Need More Data
>
> There is no confirmation that a real (proven) dependency is present in the loop. To confirm: Run a Dependencies analysis.

> Recommendation: Check dependencies for outer loop — Confidence: Low
>
> 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.

- Use the survey checkboxes to select which loops to analyze.
- If no dependencies are found, it's safe to force vectorization.
- Otherwise, use the reported variable read/write information to see if you can rework the code to eliminate the dependency.

Summary | Survey & Roofline | Refinement Reports

Site Location	Loop-Carried Dependencies
[loop in main at example.c...	No dependencies found
[loop in main at example.c...	RAW:1

Summary

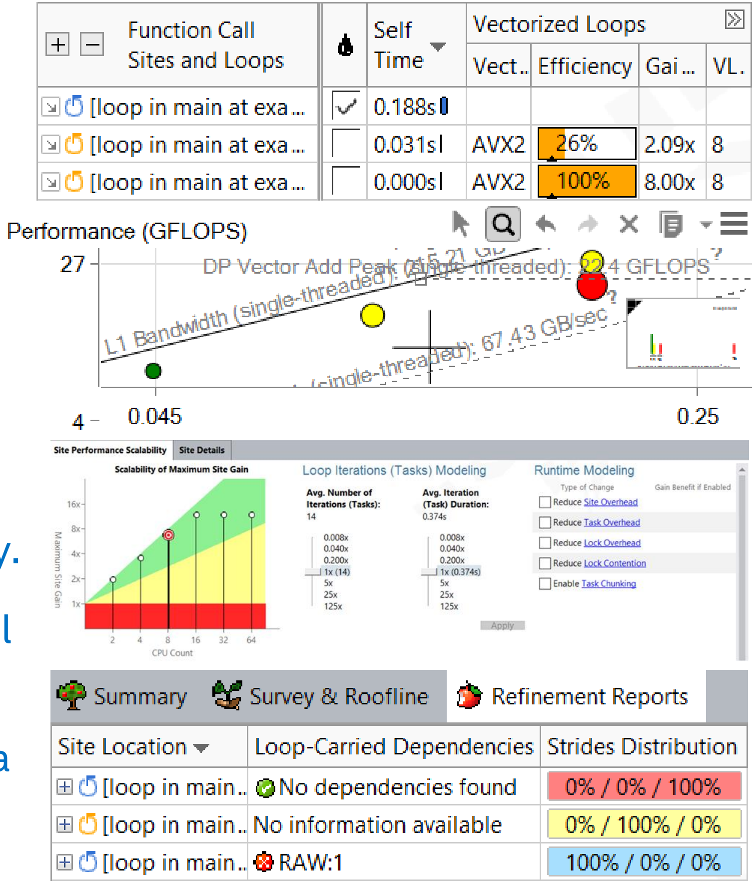
V T **Survey** – Find the most promising sites for threading, see the meat of the vectorization information, and get recommendations from Advisor.

V T **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.

T **Suitability** – Predict how well your proposed threading model will scale under certain conditions quickly and easily.

V T **Dependencies** – Prove or disprove the existence of parallel dependencies and learn how to fix them.

V **Memory Access Patterns** – See how you traverse your data and how it affects your vector efficiency and cache bandwidth usage.



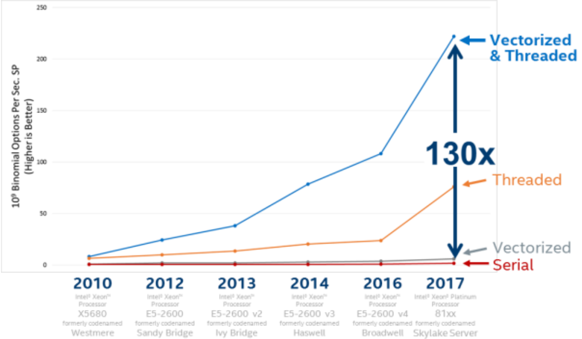
intel®

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

	Platform	Unscaled Core Frequency	Cores/ Socket	Num Sockets	L1 Data Cache	L2 Cache	L3 Cache	Memory	Memory Frequency	Memory Access	H/W Prefetchers Enabled	HT Enabled	Turbo Enabled	C States	O/S 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	Y	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	Y	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	Y	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	Y	Y	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	Y	Disabled	RHEL 7.0	3.10.0-123. el7.x86_64	icc version 17.0.2
BDW	Intel® Xeon™ E5 2600v4 Processor	2.2 GHz	22	2	32K	256K	56 MB	128 GB	2133 MHz	NUMA	Y	Y	Y	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

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 regarding 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 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.
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 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.

Notice revision #20110804

Software