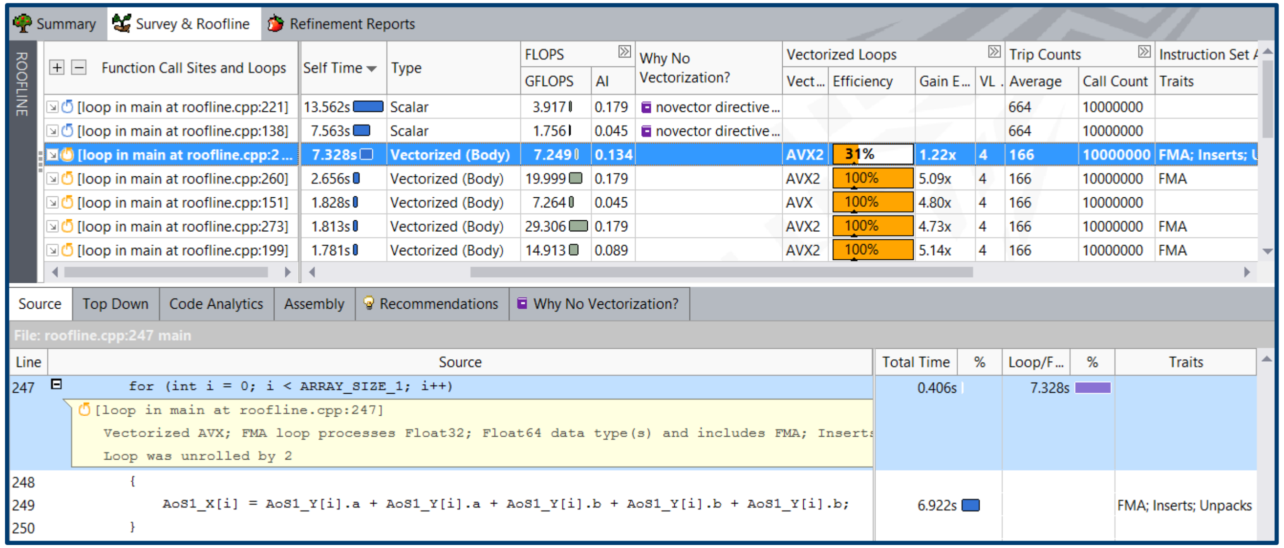
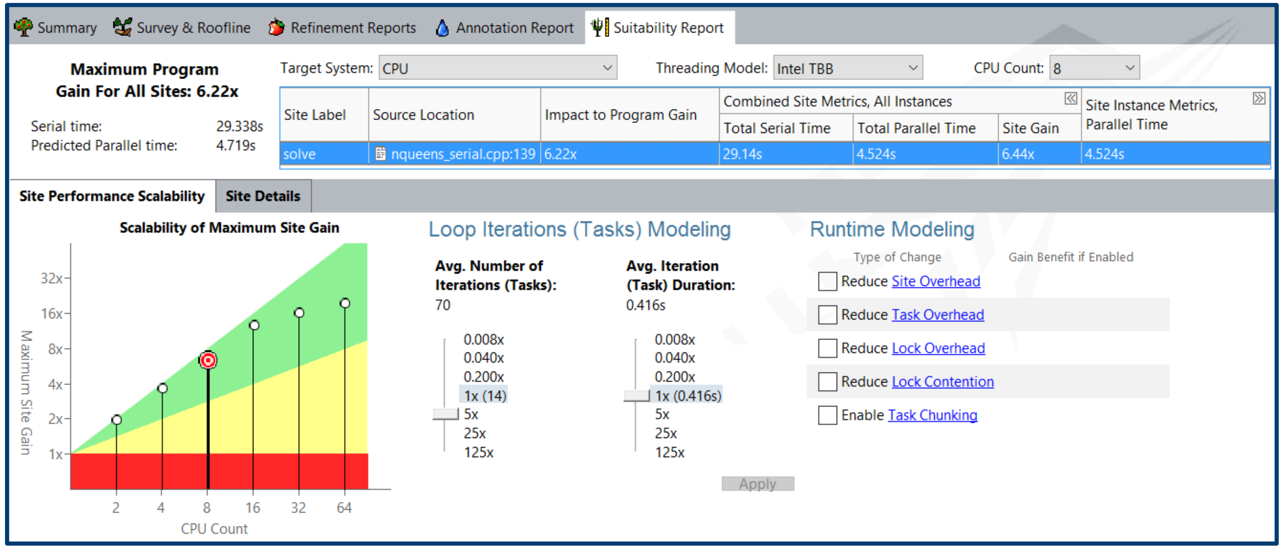
INTEL® ADVISOR

Jackson Marusarz – Intel®

Agenda

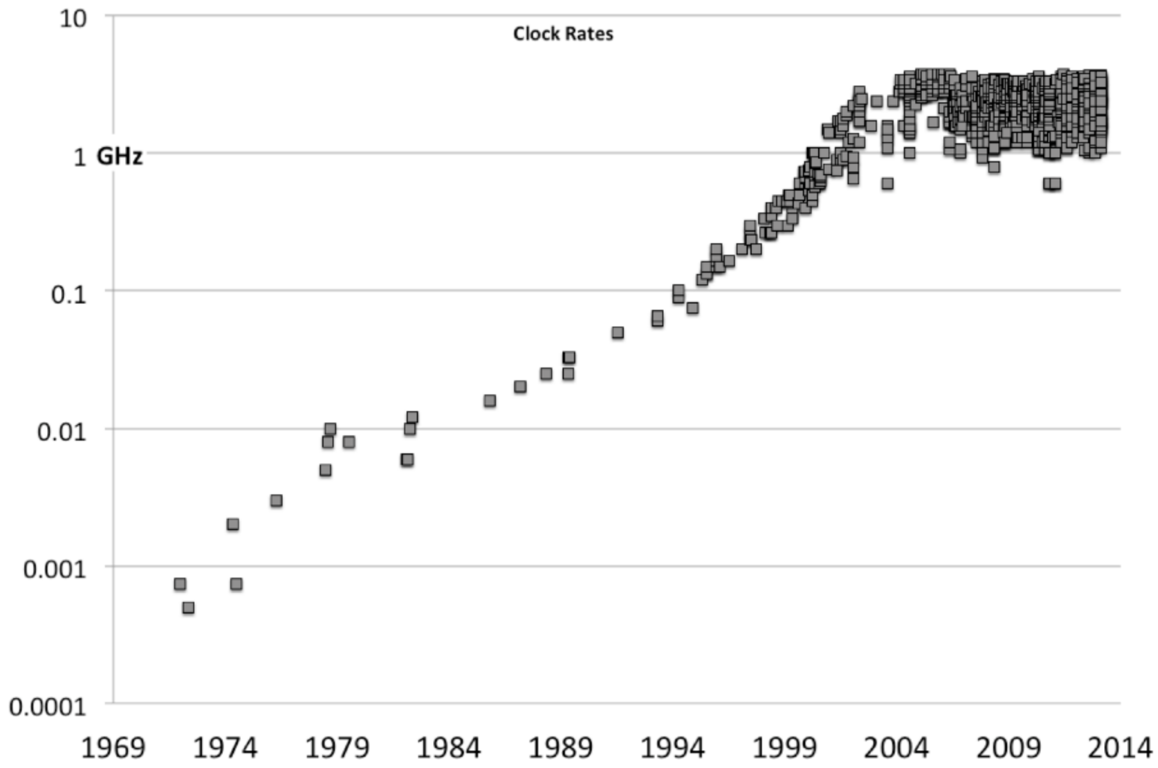
- Motivation
- Threading Advisor
  - Threading Advisor Workflow
  - Advisor Interface
  - Survey Report
  - Annotations
  - Suitability Analysis
  - Dependencies Analysis
- Vectorization Advisor & Roofline
  - Vectorization Advisor recap
  - Roofline
  - Memory Access Patterns Analysis
  - Dependencies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





The "Free Lunch" is over, really

Processor clock rate growth halted around 2005

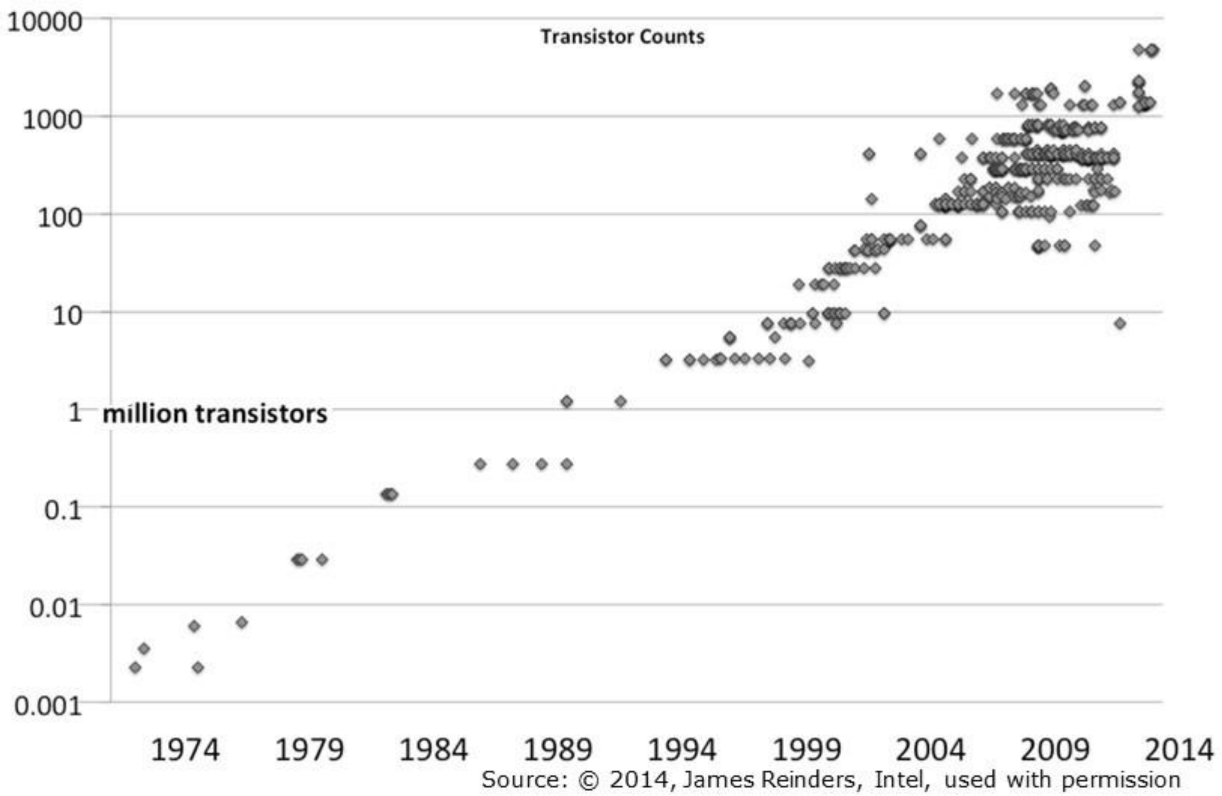


Source: © 2014, James Reinders, Intel, used with permission

Software must be parallelized to realize all the potential performance

Moore’s Law Is Going Strong

Hardware performance continues to grow exponentially

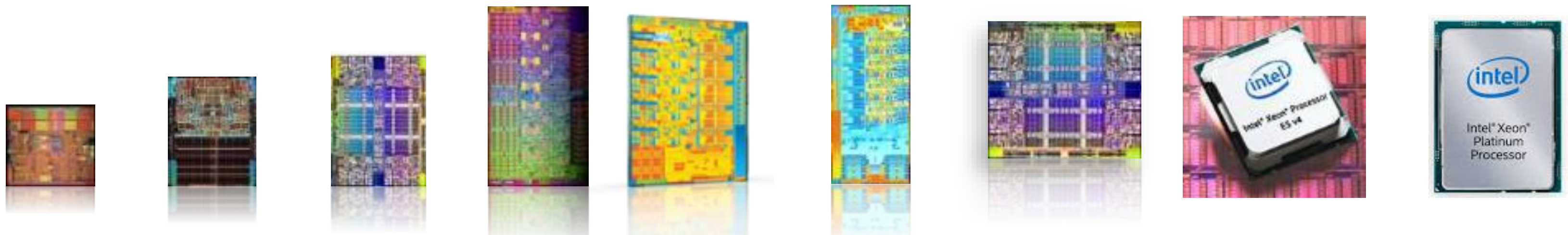


“We think we can continue Moore's Law for at least another 10 years."

Intel Senior Fellow
Mark Bohr
2015

Changing Hardware Impacts Software

More Cores → More Threads → Wider Vectors



	Intel® Xeon® Processor								
	64-bit	5100 series	5500 series	5600 series	E5-2600	E5-2600 V2	E5-2600 V3	E5-2600 V4	Platinum 8180
Core(s)	1	2	4	6	8	12	18	22	28
Threads	2	2	8	12	16	24	36	44	56
SIMD Width	128	128	128	128	256	256	256	256	512

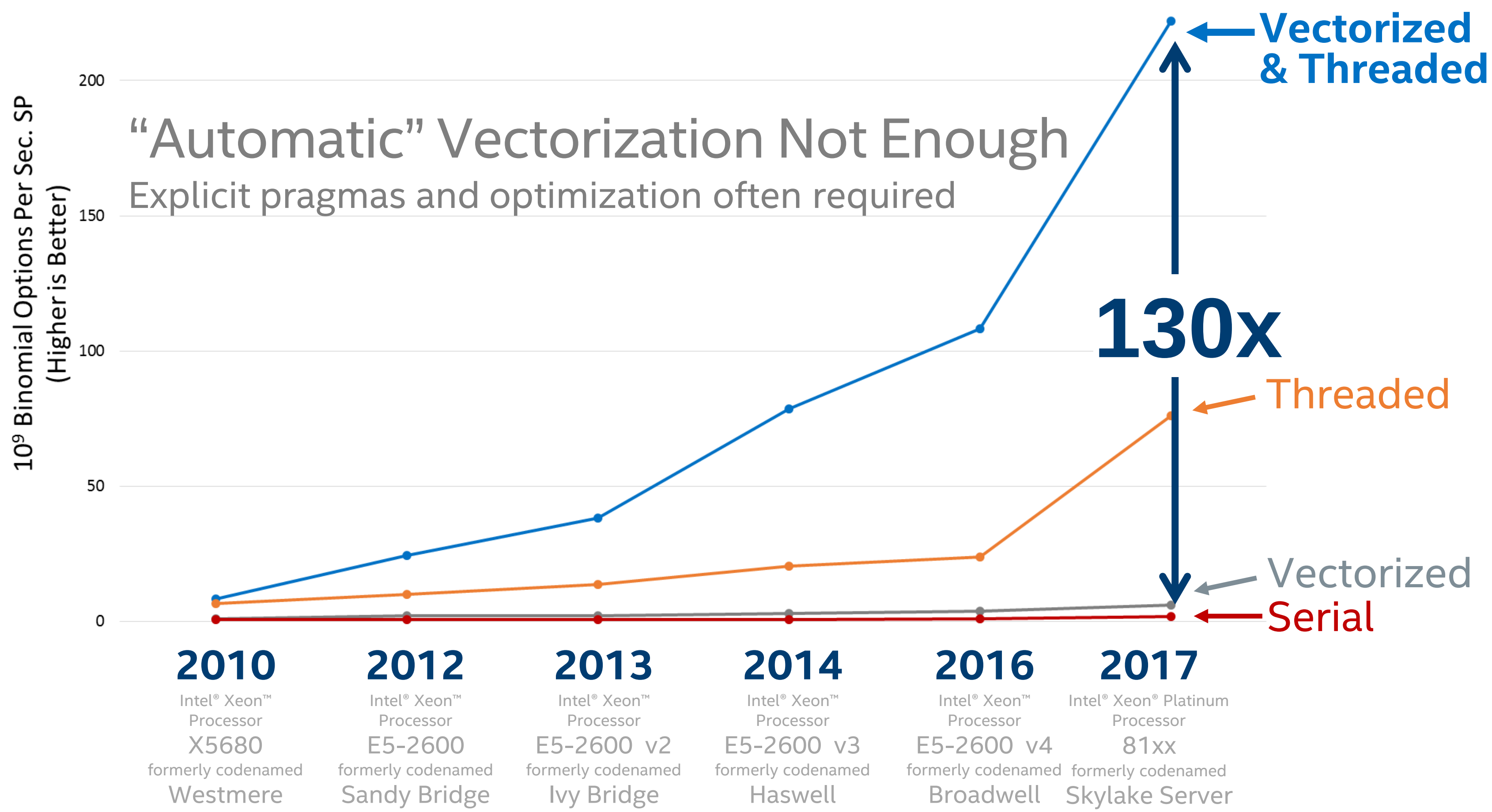
High performance software must be both

- Parallel (multi-thread, multi-process)
- Vectorized

*Product specification for launched and shipped products available on ark.intel.com.

Vectorize & Thread or Performance Dies

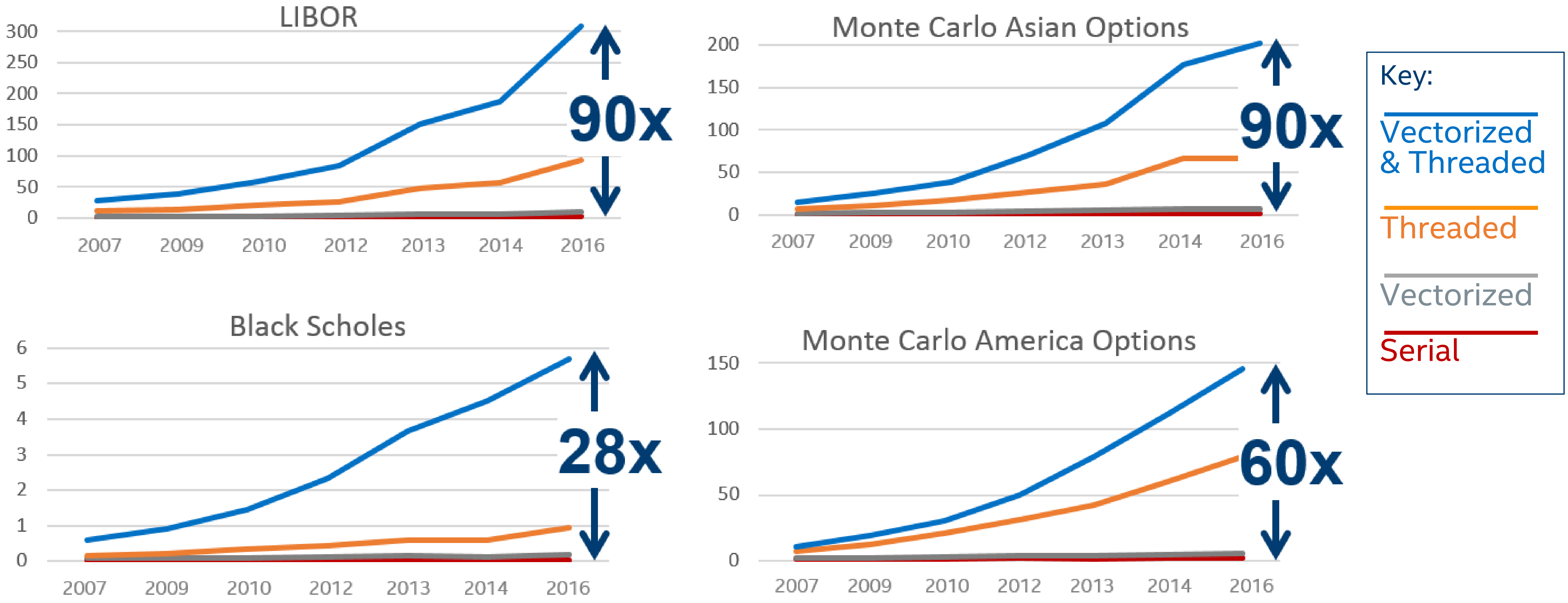
Threaded + Vectorized can be much faster than either one alone



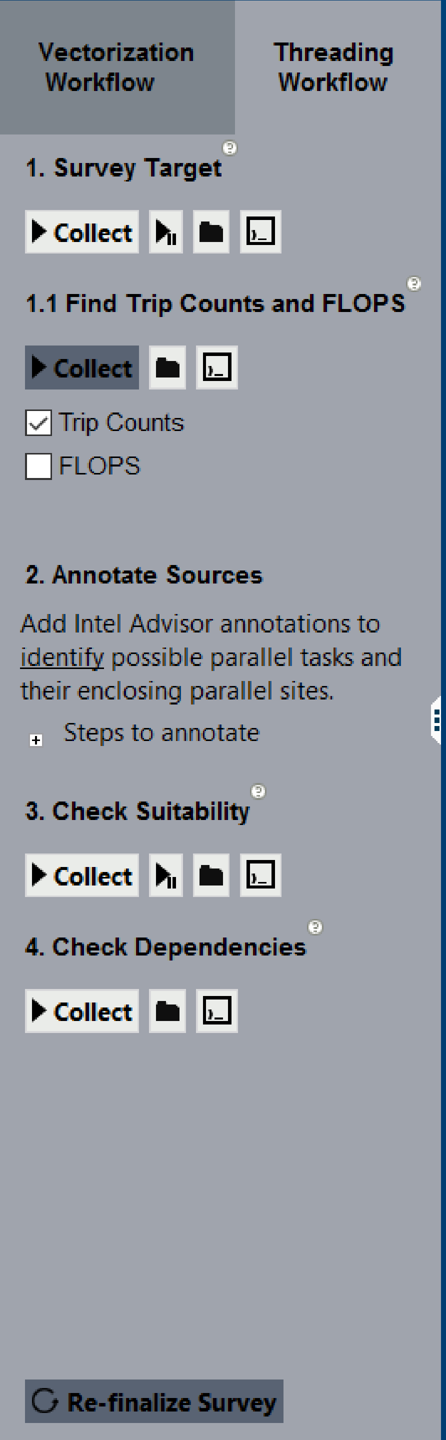
The Difference Is Growing With Each New Generation of Hardware

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.

Vectorization & Threading Critical on Modern 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

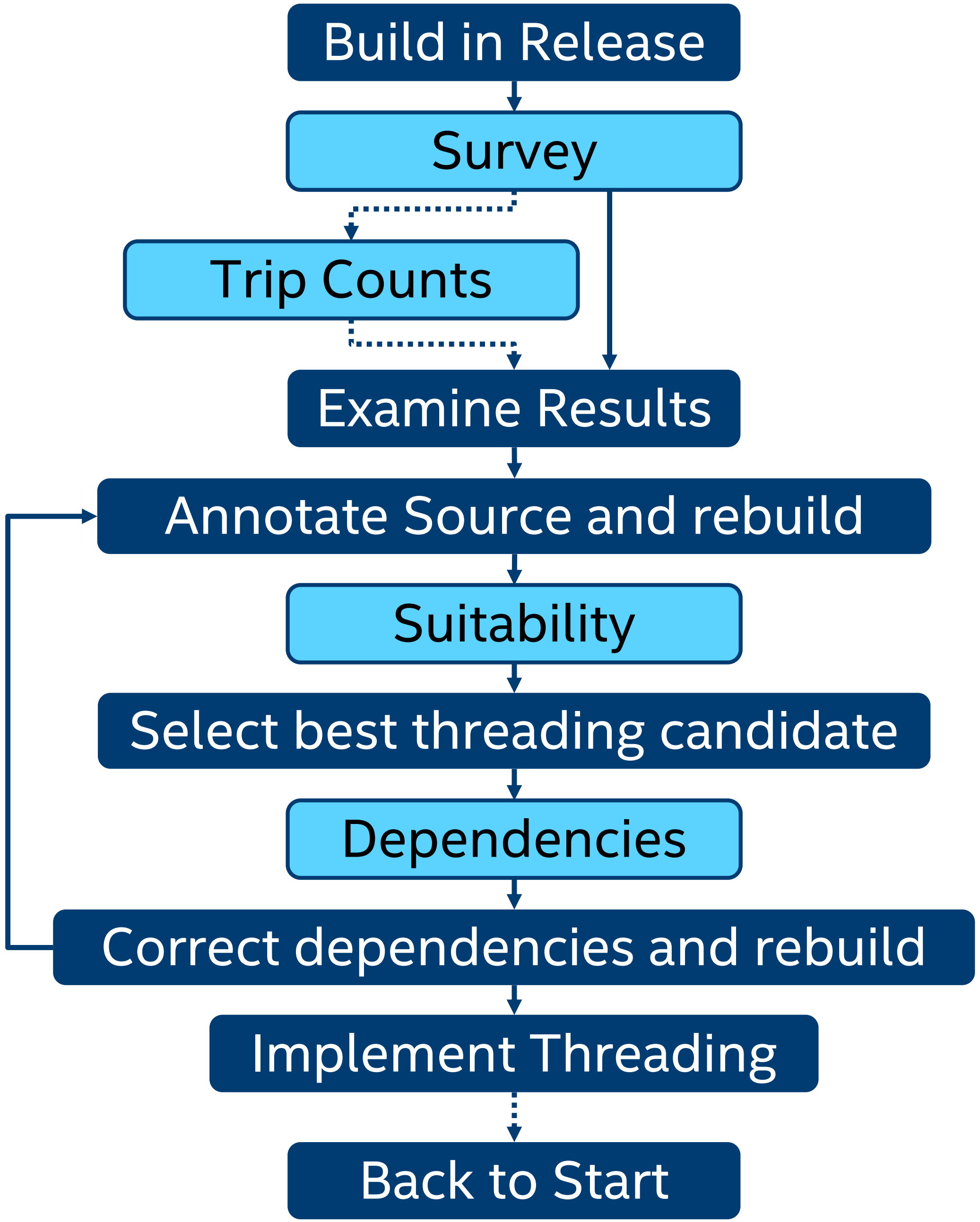
3) All of your test suites should still pass. Validate the correctness of your transformations.

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.



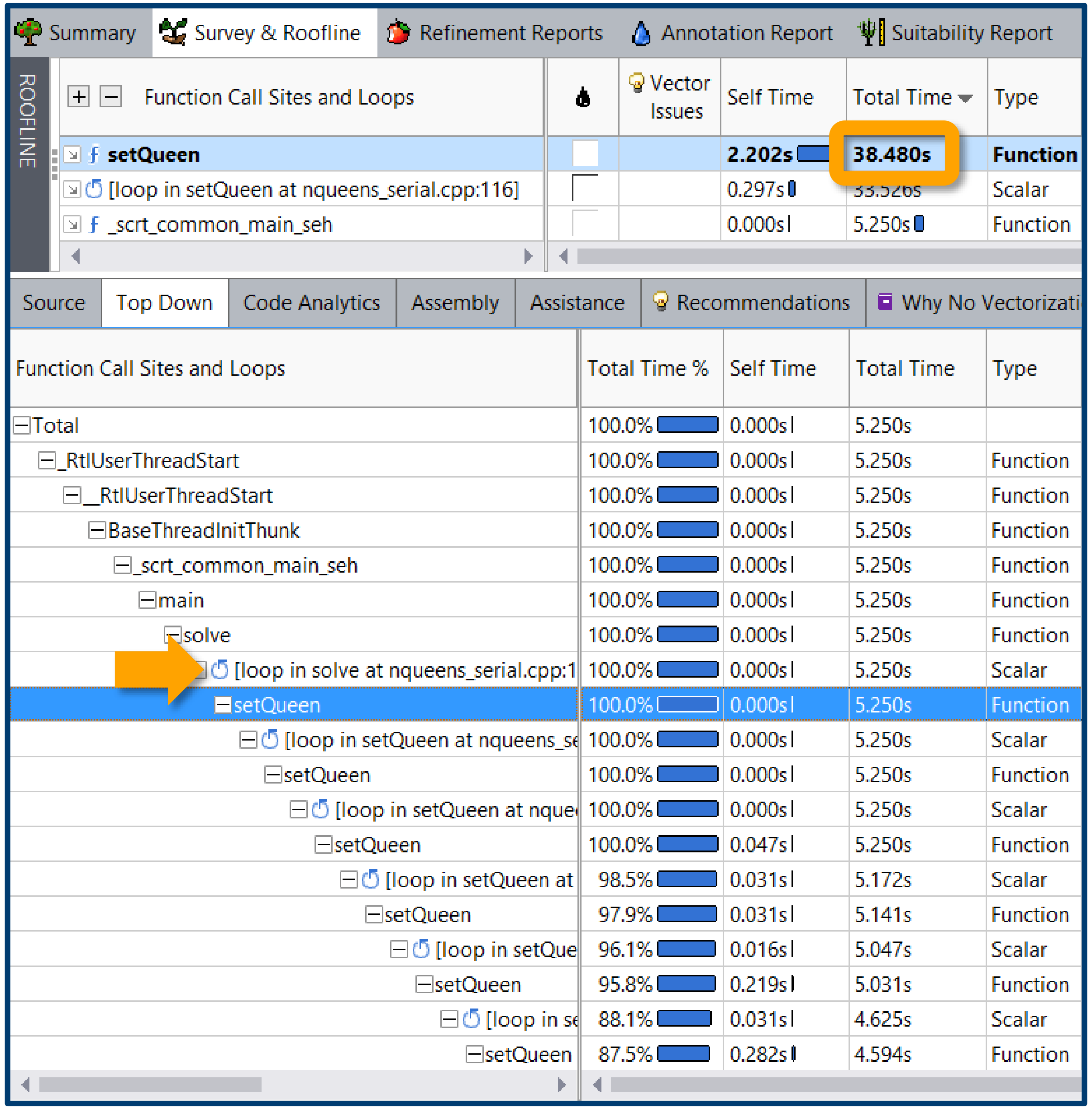
Survey Report

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.

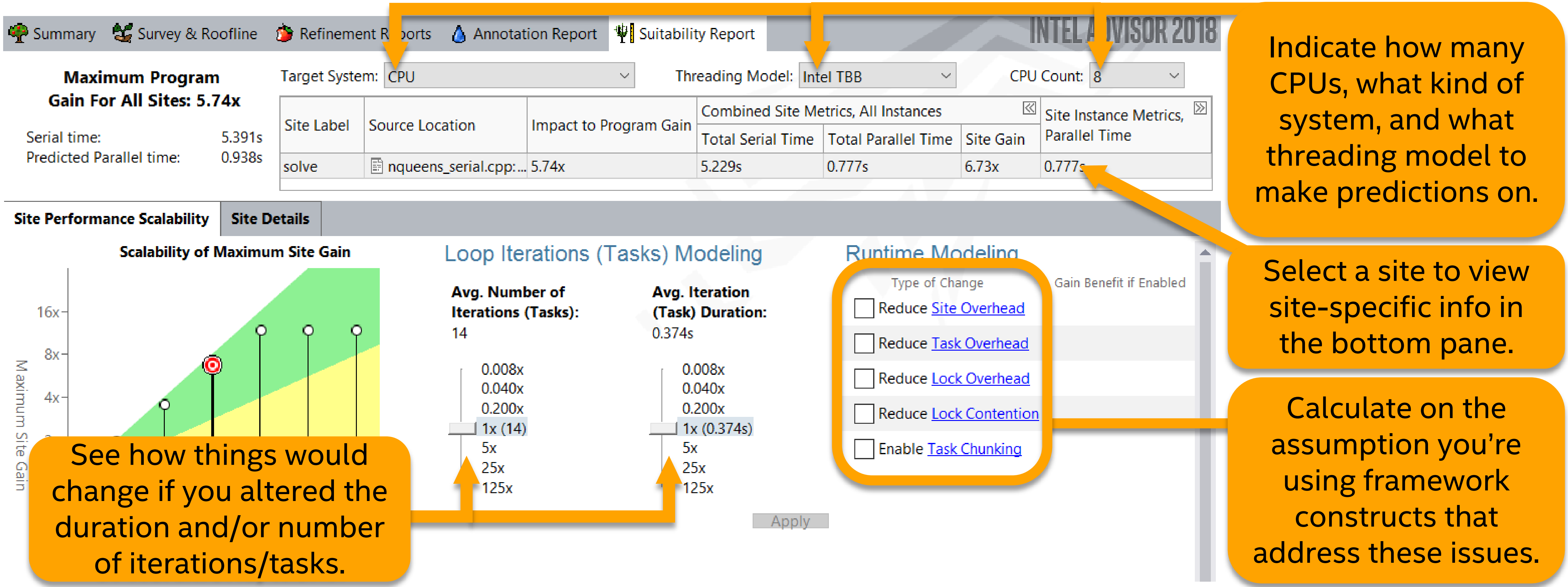
- The Advisor User's Guide contains a section on Annotations with full documentation, examples, and instructions on the above if you forget.

Common Annotations

- Site Annotations
  - Indicate locations which contain a number of tasks which can run in parallel with each other. Code within the site that is not part of a task is executed only by the entering thread, but can still run in parallel with the tasks.
  - Consist of a Site Begin and a Site End annotation.
- Task Annotations
  - Indicate chunks of code which can run in parallel with other tasks in the same site, including any additional copies of themselves which may be launched.
  - Consist of a Task Begin and a Task End annotation.
- Iteration Task Annotations
  - Indicate that the entirety of a loop's contents should be considered as a task.
  - Consist only of an Iteration Task annotation. The end is implied.
- Lock Annotations
  - Indicate where Advisor should simulate locking.
  - Consist of a Lock Acquire annotation and a Lock Release annotation.

Suitability Analysis

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

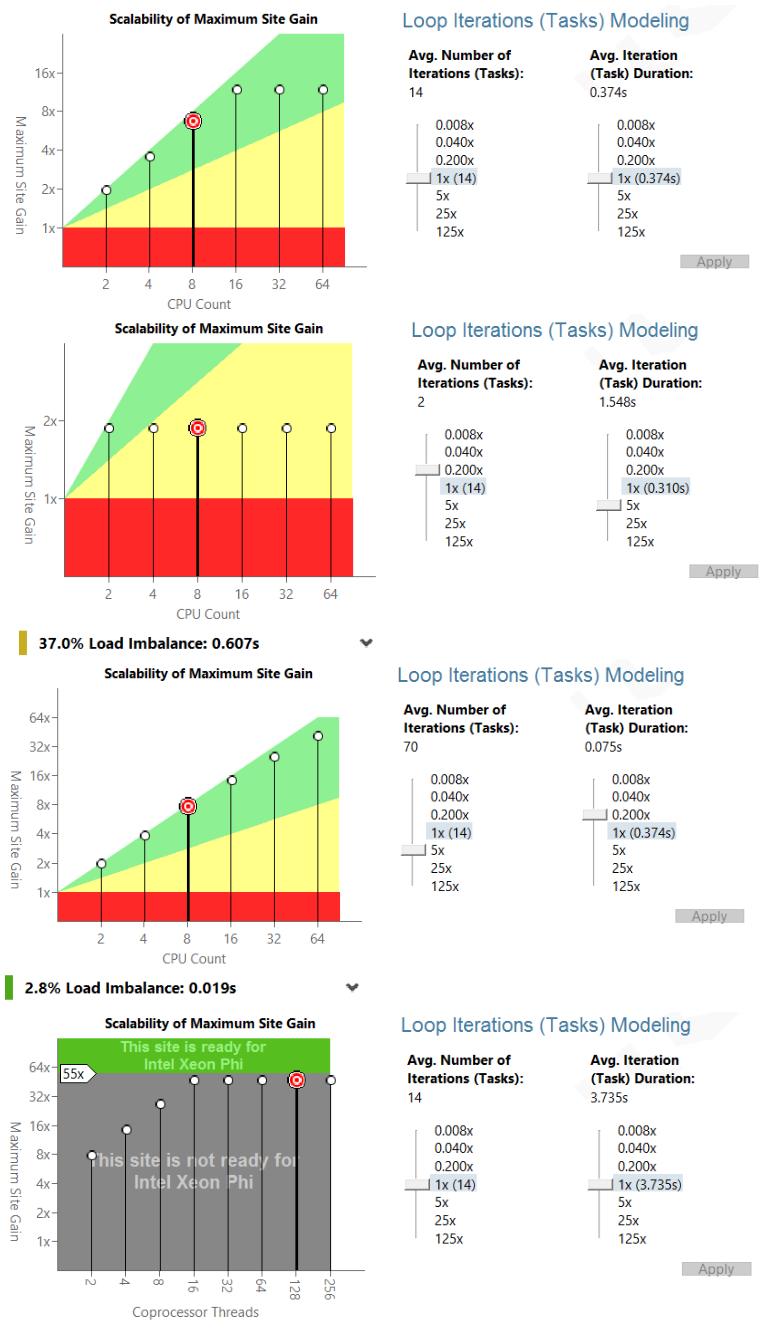


Suitability Analysis

- The white dots are the estimated number of times speedup at the given number of CPUs.
- The bullseye is placed on the one corresponding to the number of CPUs you have selected in the controls.
  - If your bullseye is in:

Red:	Yellow:	Green:
No benefit, possible slowdown.	Poor scaling, needs improvement.	Ideal scaling!

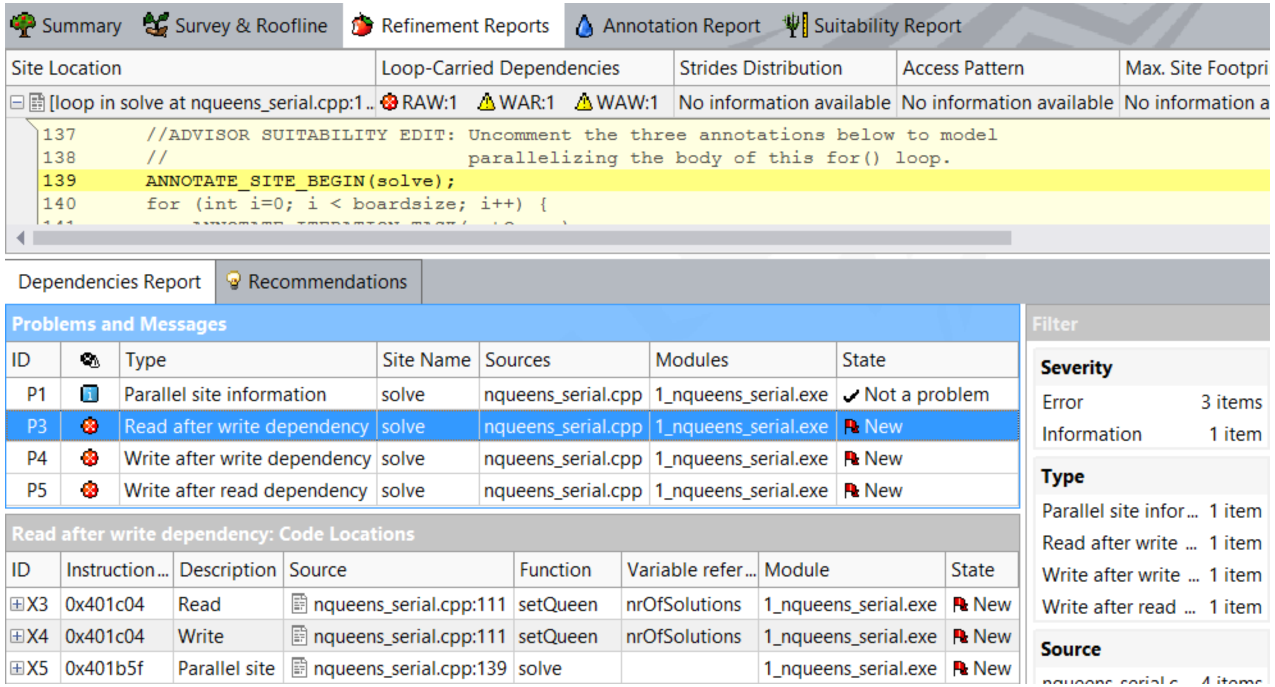
- Advisor provides warnings when it predicts that your specifications would lead to issues like load imbalance.



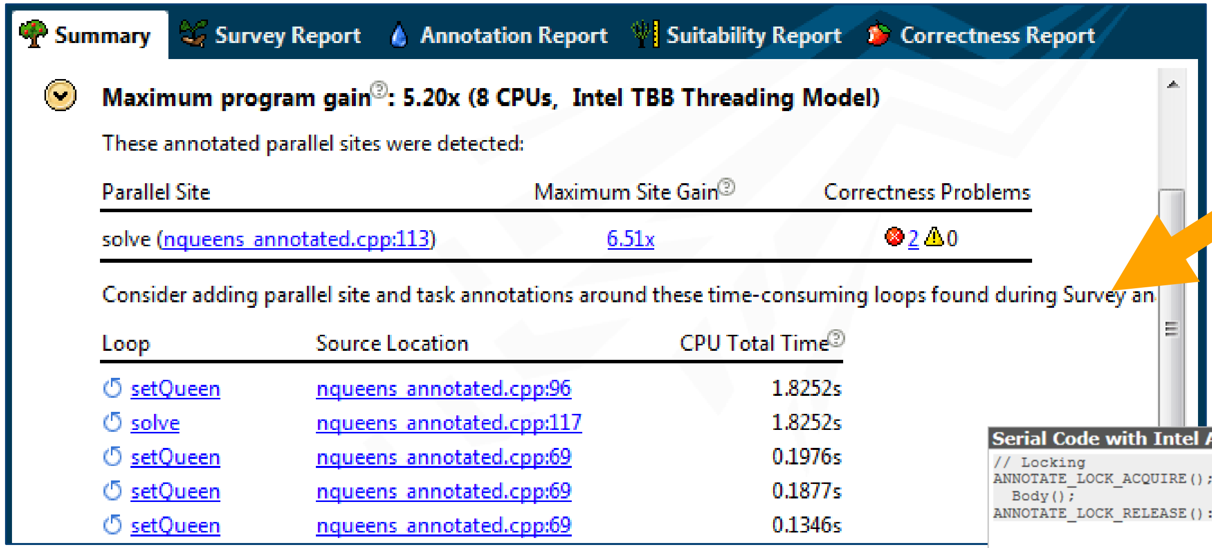
Dependencies Analysis

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.

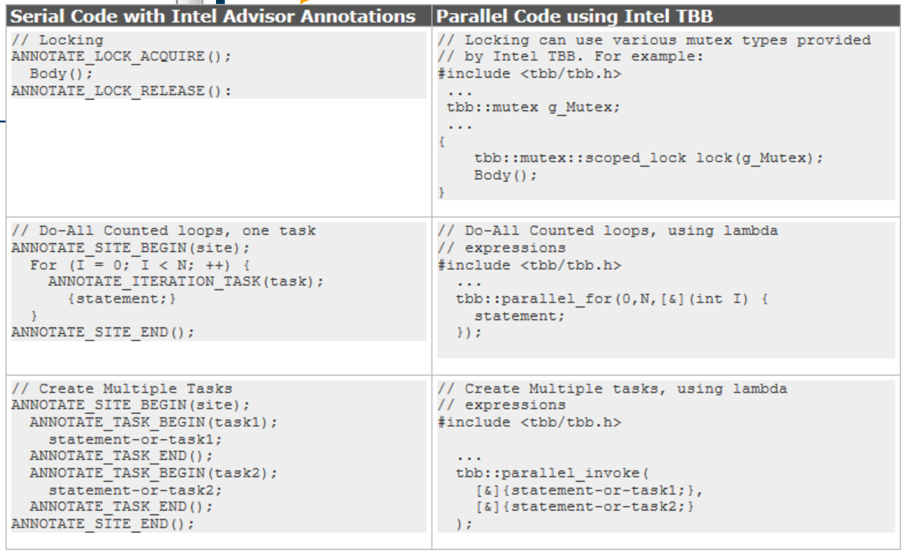


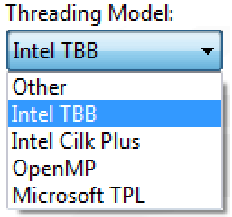
Add Parallel Framework



Here is the list of source locations

Here are templates for popular parallel frameworks

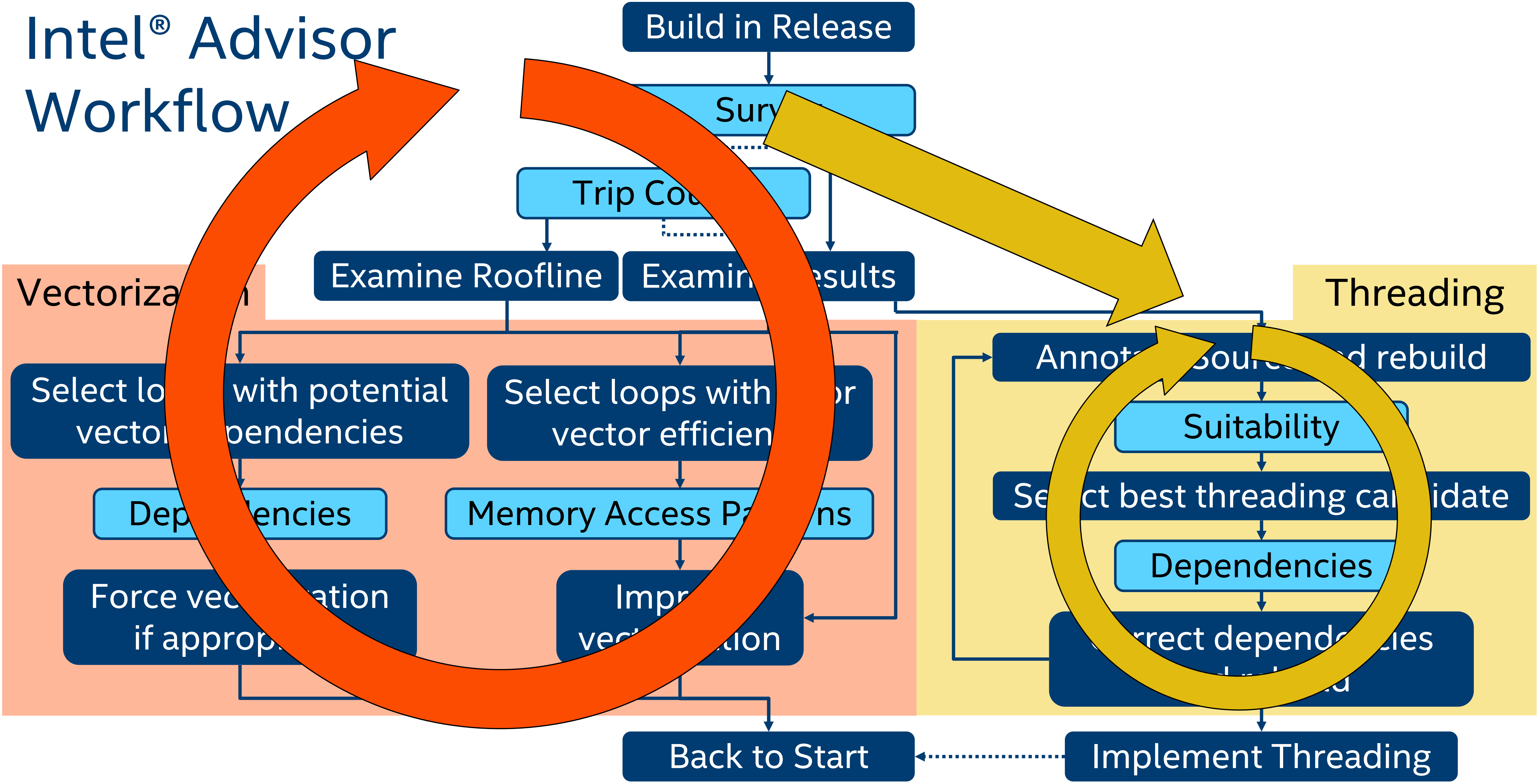


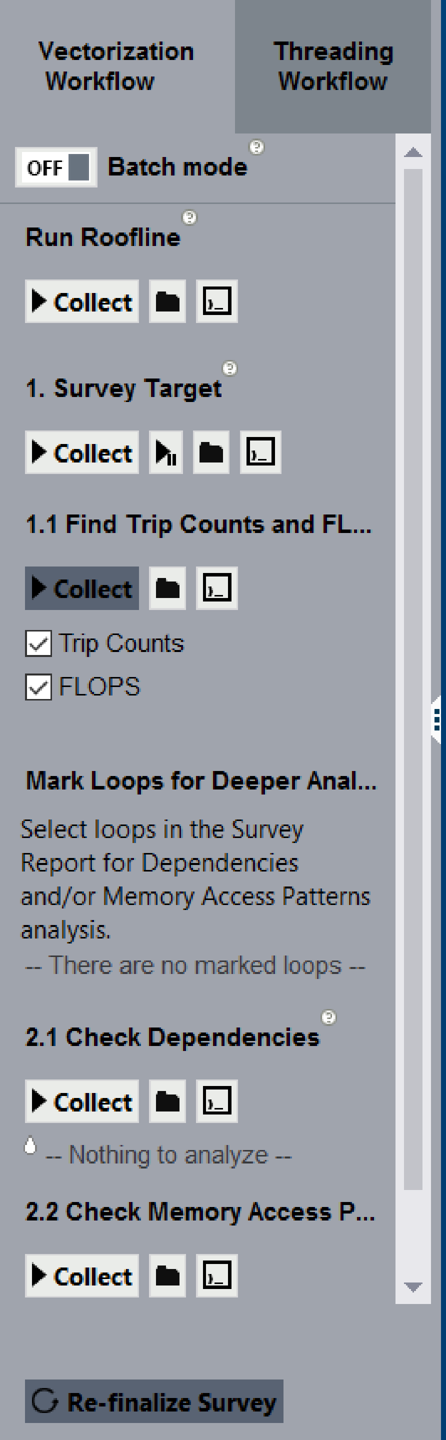


Intel® Advisor

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

Intel® Advisor Workflow

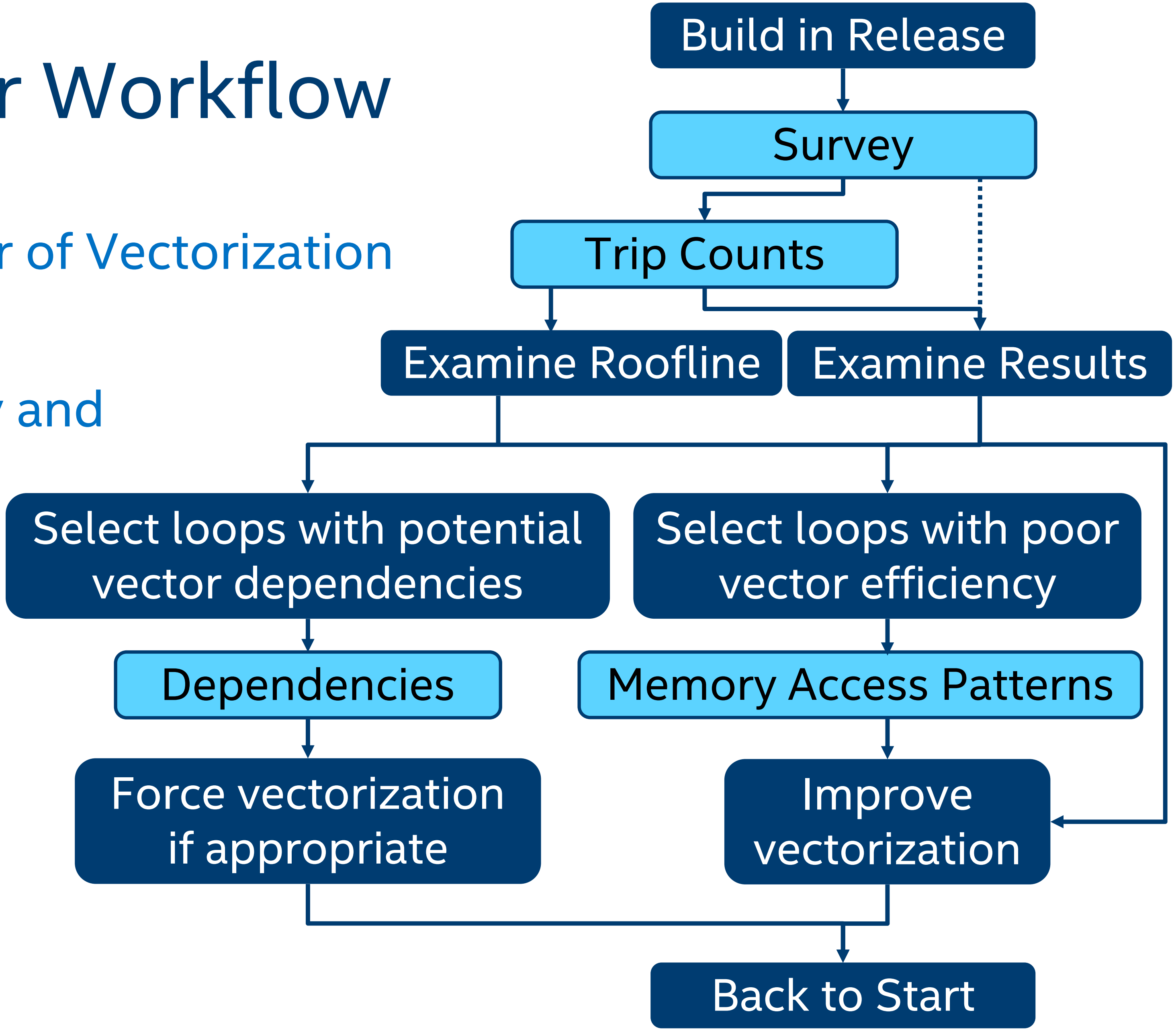




VECTORIZATION ADVISOR & ROOFLINE

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!

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

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 inefficien...	0.297s	0.297s	Vector...		AVX2	2%	0.37x	16
[loop in main at example.cpp:51]		1 Possible inefficien...	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 loo w...				

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

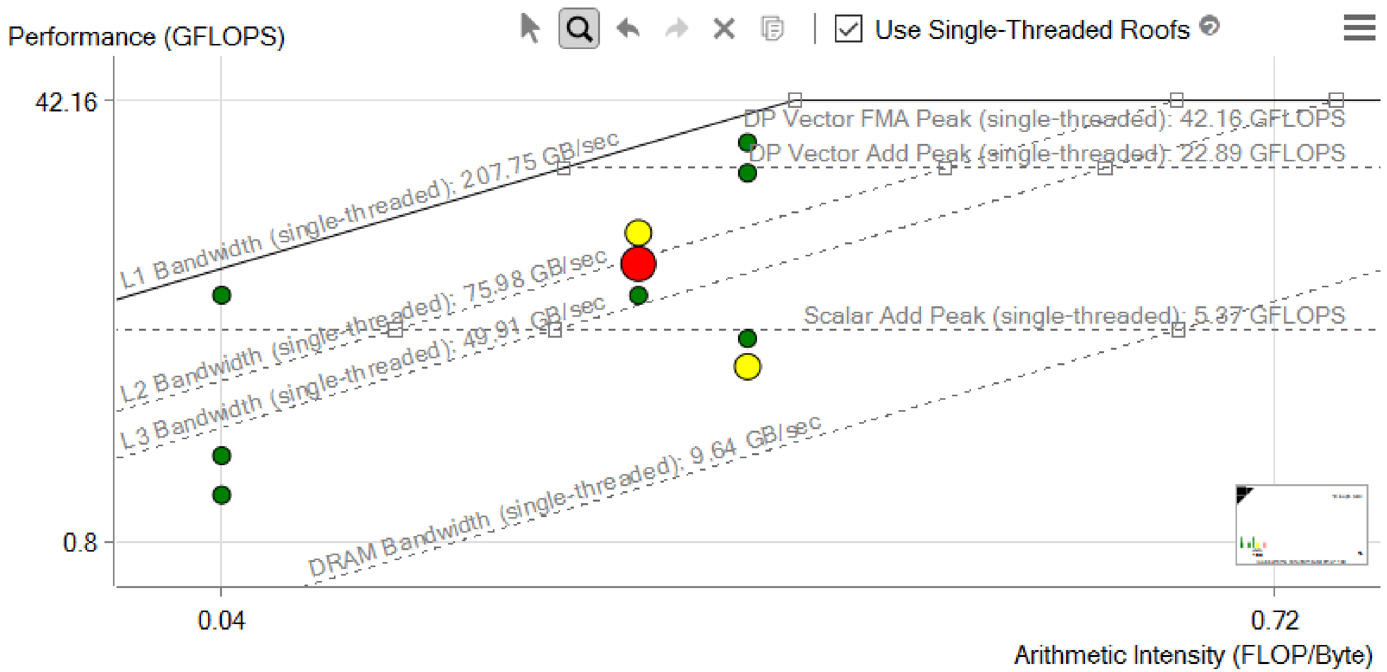
Advisor *advises* you on potential vector issues. This is often your cue to run MAP or Dependencies. Click the icon to see an explanation in the bottom pane.

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

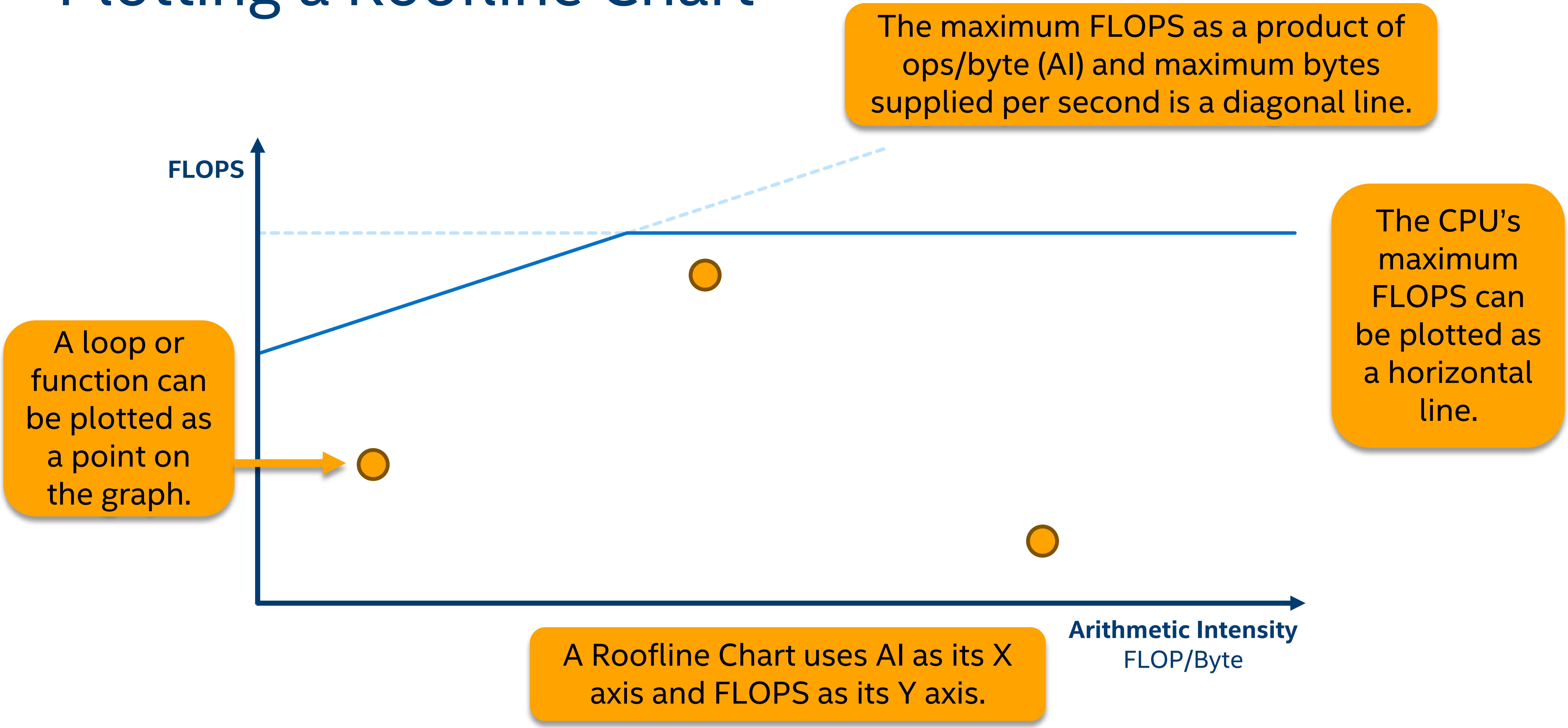
Roofline Metrics

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

Plotting a Roofline Chart



Classic vs. Cache-Aware Roofline

Intel® Advisor uses the Cache-Aware Roofline model, which has a different definition of Arithmetic Intensity than the original ("Classic") model.

Classical Roofline

- Traffic measured from one level of memory (usually DRAM)
- AI may change with data set size
- AI changes as a result of memory optimizations

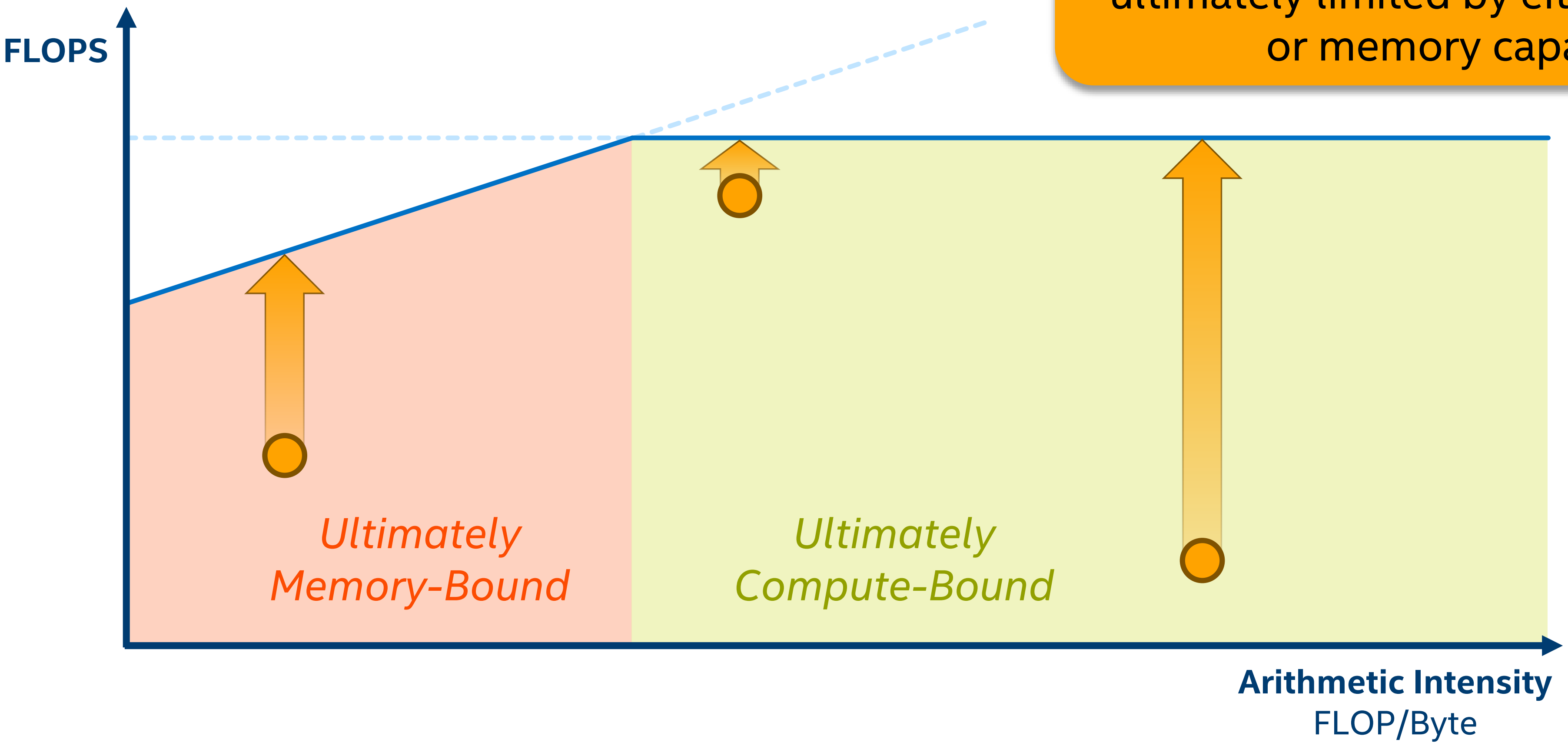
Cache-Aware Roofline

- Traffic measured from all levels of memory
- AI is tied to the algorithm and will not change with data set size
- Optimization does not change AI*, only the performance

**Compiler optimizations may modify the algorithm, which may change the AI.*

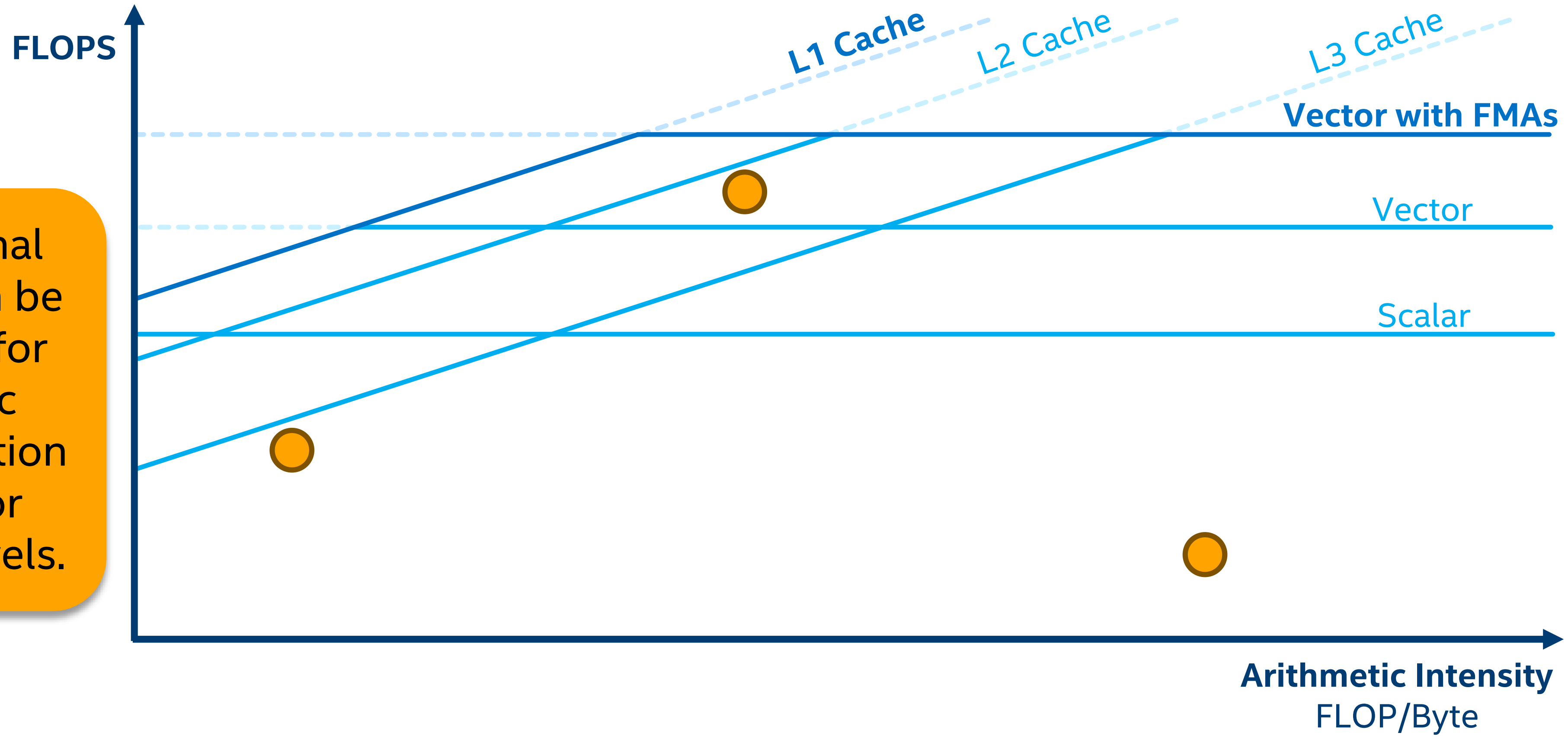
Ultimate Performance Limits

Performance cannot exceed the machine's capabilities, so each loop is ultimately limited by either compute or memory capacity.



Sub-Roofs and Current Limits

Additional roofs can be plotted for specific computation types or cache levels.



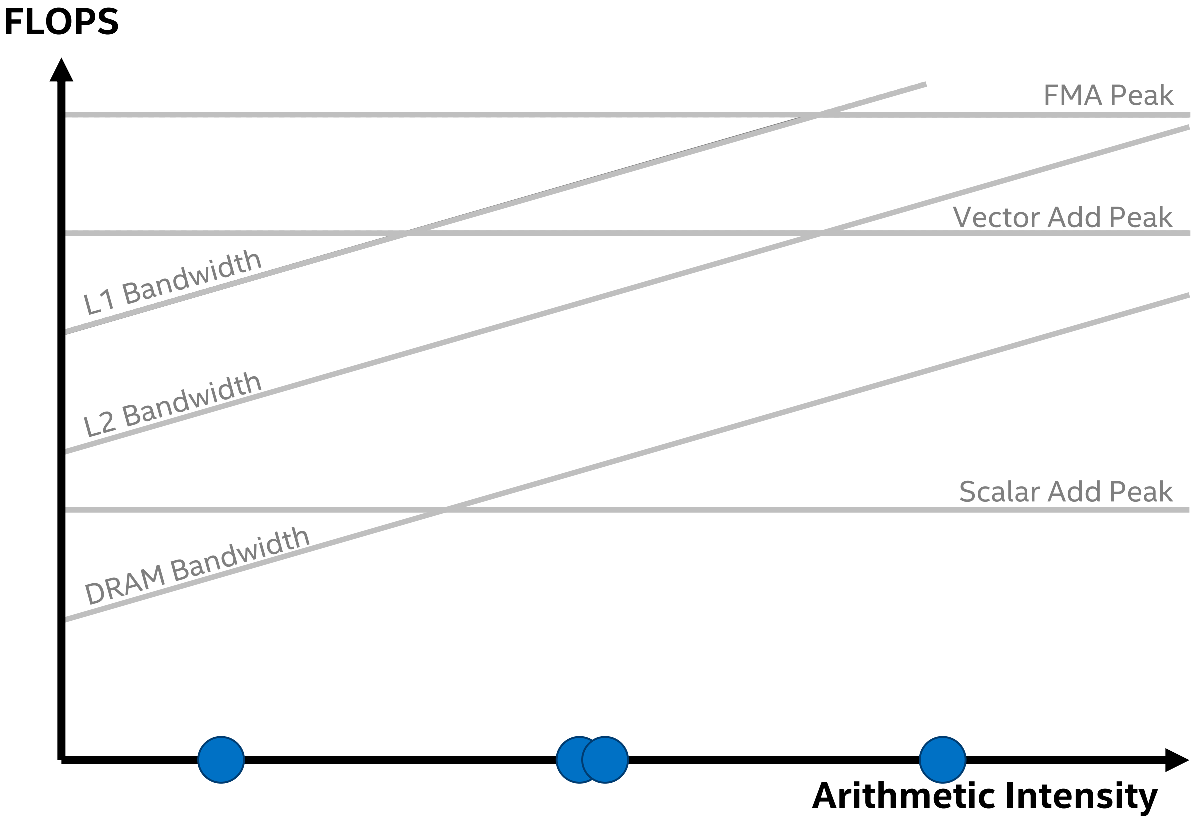
These sub-roofs can be used to help diagnose bottlenecks.

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*

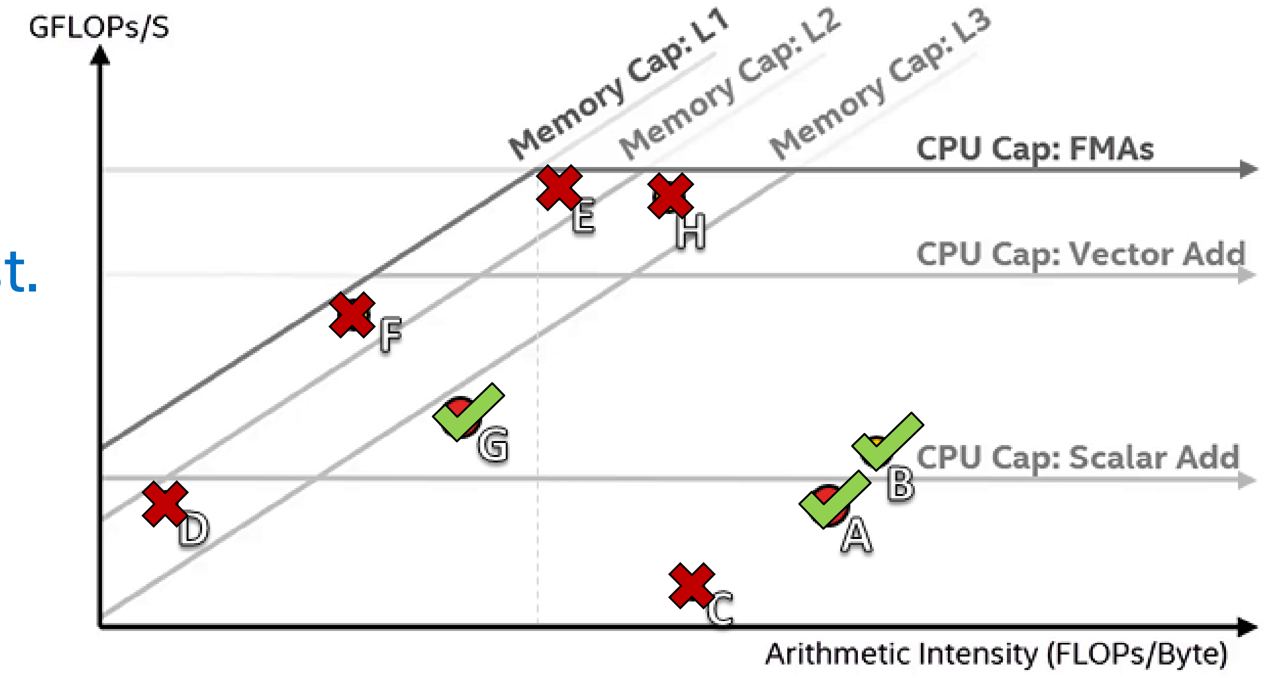


Roofline first proposed by University of California at Berkeley:
Roofline: An Insightful Visual Performance Model for Multicore Architectures, 2009
Cache aware variant proposed by Technical University of Lisbon:
Cache-Aware Roofline Model: Upgrading the Loft, 2013

Cache-Aware Roofline

Usage

- Target the bigger, redder dots that are farthest below the top roofs first.
  - Big and red = takes up a lot of time.
  - Far below roofs = lots of room for improvement.
- Roofs below: unlikely bottlenecks, unless you're *just* above them.
- Roofs above: very likely bottlenecks! The closest roofs above are the most likely causes of bottlenecking.



Note:

Not every algorithm can break every roof! Some algorithms inherently can't avoid certain bottlenecks.

Cache-Aware Roofline

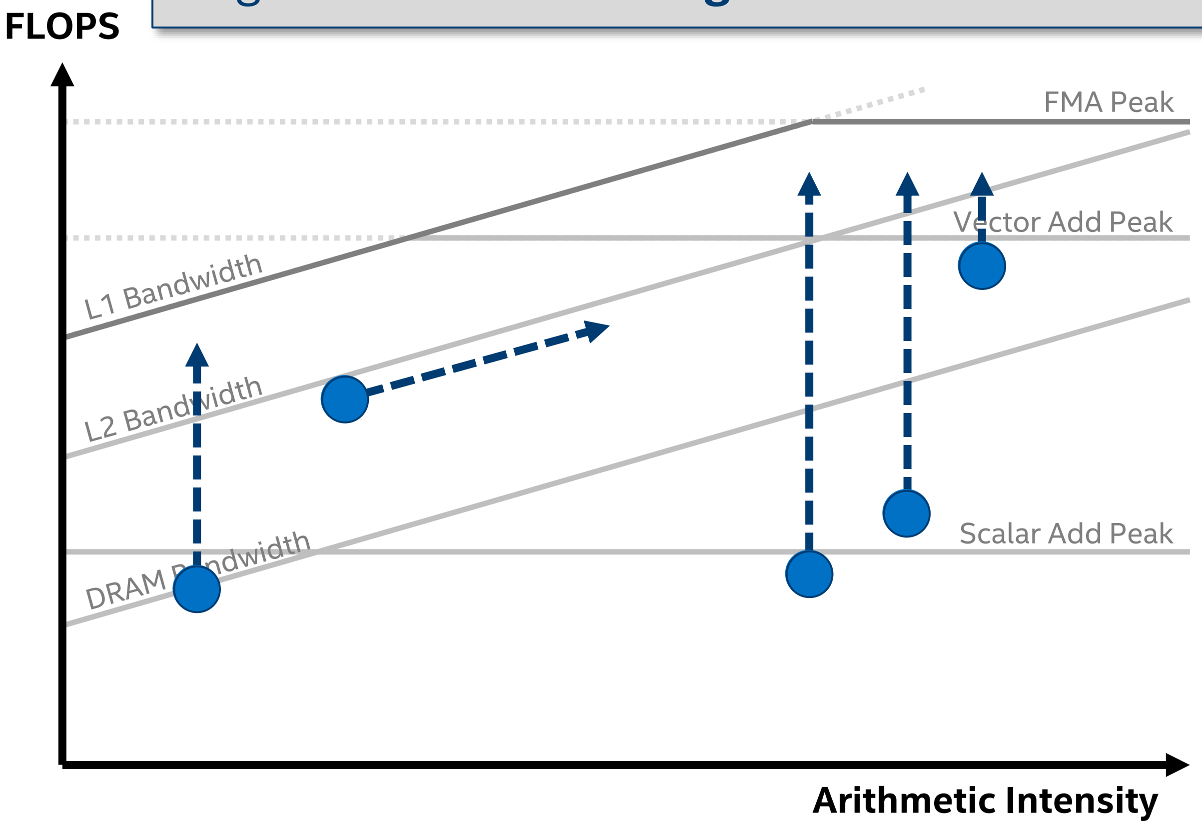
Next Steps

If under or near a memory roof…

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

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.

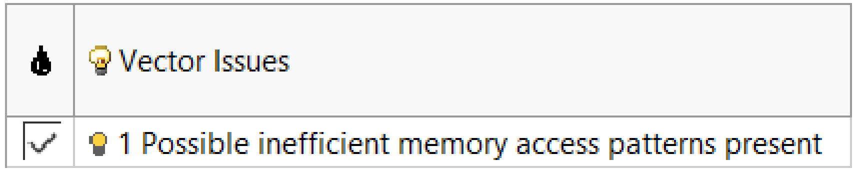
If under the Scalar Add Peak…

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.



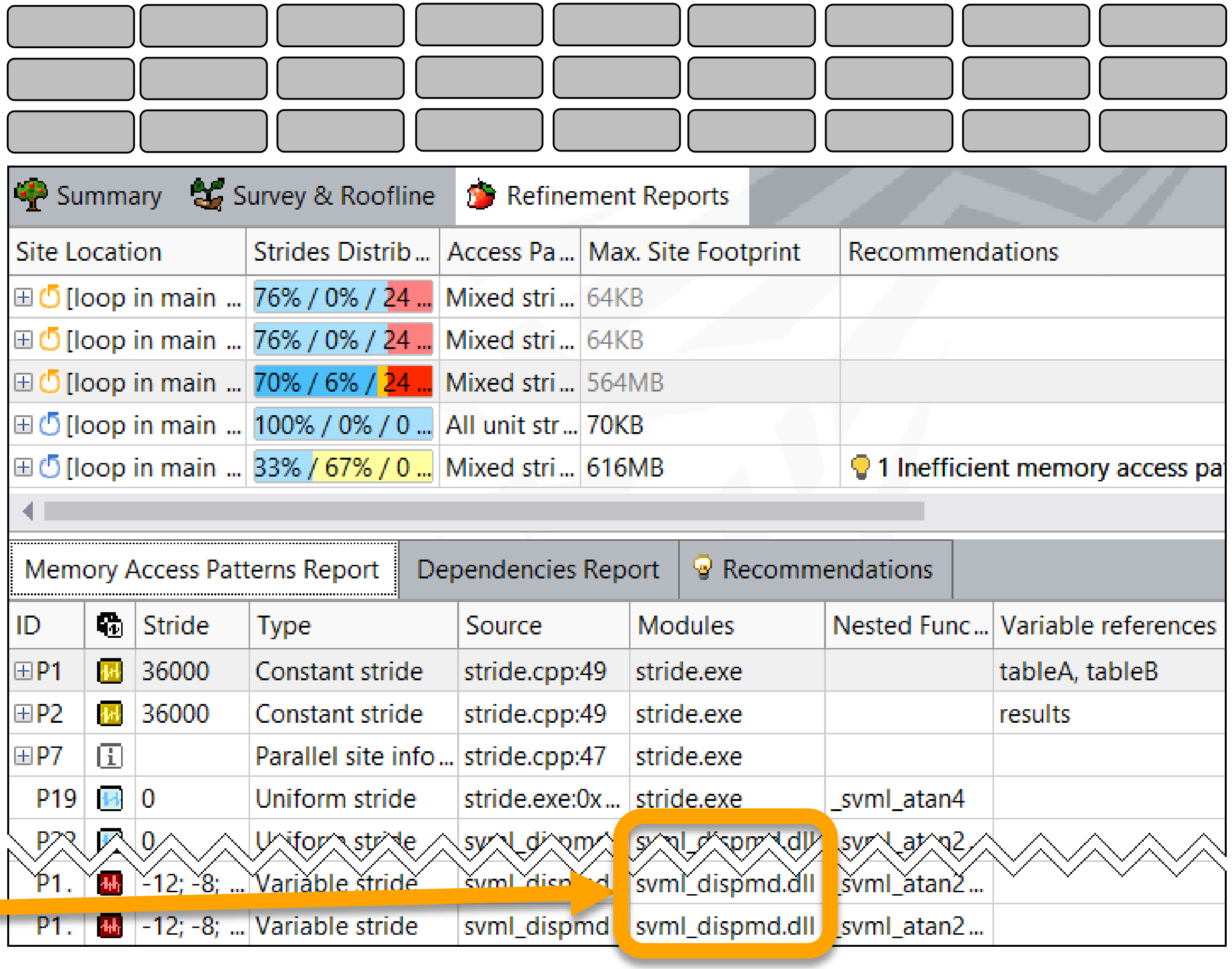
- Memory access patterns affect vectorization efficiency because they affect how data is loaded into and stored from the vector registers.
- Select the loops you want to run the MAP on using the checkboxes. It may be helpful to reduce the problem size, as MAP only needs to detect patterns, and has high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

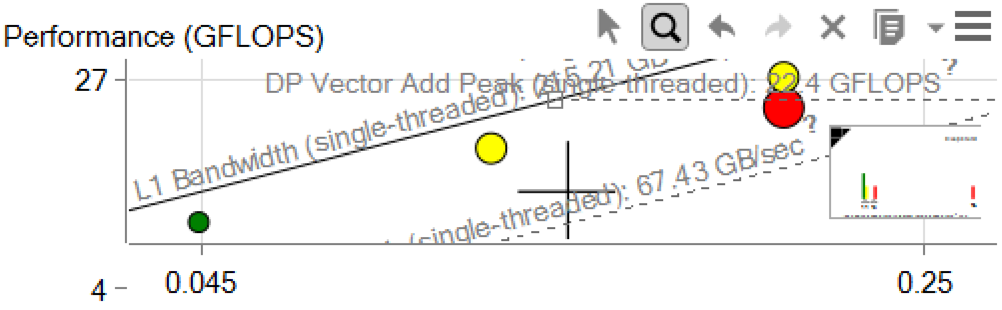
V T **Trip Counts & FLOPS** – Add to your Survey report to help fine-tune vector efficiency and capability, as well as unlock the powerful **Roofline** to visualize your bottlenecks and help direct your efforts.

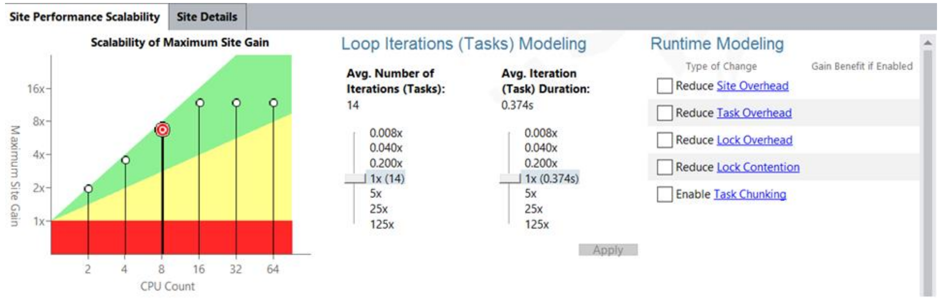
T **Suitability** – Predict how well your proposed threading model will scale under certain conditions quickly and easily.

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.

Function Call Sites and Loops		Self Time	Vectorized Loops: Vect...	Efficiency	Gai...	VL...
[loop in main at exa...	✓	0.188s				
[loop in main at exa...		0.031s	AVX2	26%	2.09x	8
[loop in main at exa...		0.000s	AVX2	100%	8.00x	8





Summary | Survey & Roofline | Refinement Reports

Site Location	Loop-Carried Dependencies	Strides Distribution
[loop in main...	No dependencies found	0% / 0% / 100%
[loop in main...	No information available	0% / 100% / 0%
[loop in main...	RAW:1	100% / 0% / 0%

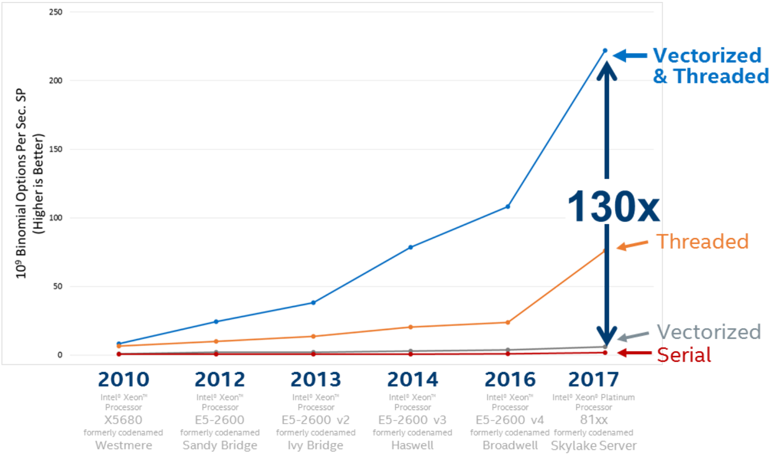
Threading Advisor Demo

intel®

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.

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

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