INTEL® ADVISOR AND ROOFLINE MODEL

Part of Intel® Parallel Studio XE

Contacts

Advisor Support Mail List vector.advisor@intel.com

Zakhar Matveev zakhar.a.matveev@intel.com

Intel Advisor Product Architect

Kirill Rogozhin kirill.rogozhin@intel.com

Intel Advisor Project Manager

Egor Kazachkov egor.kazachkov@intel.com

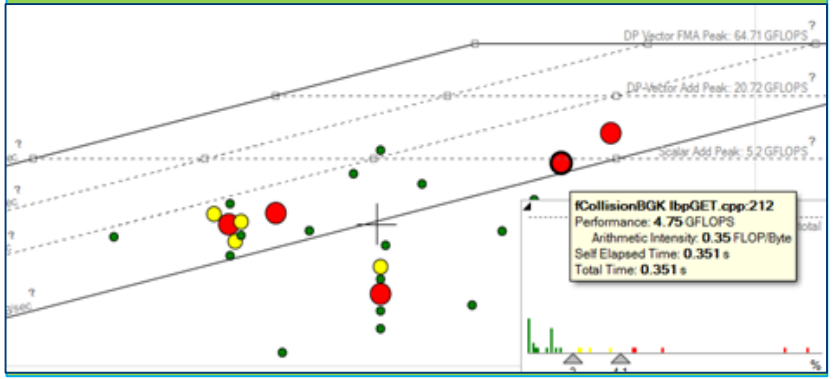
Intel Advisor Senior Developer

What is Intel® Advisor

Vectorization analysis

Function Call Sites and Loops	Why No Vectorization?	Self Time	Total Time	Type	Vectorized Loops		
					Vec...	Efficiency	Gain...
_tmainCRTStartup		0.000s	1.669s	Function			
main		0.000s	1.669s	Function			
[loop in main at 3loop	inner loop wa...	0.000s	1.373s	Scalar			
[loop in main at 3loops.c		1.139s	1.373s	Vectori...	AVX2	~60%	4.79x
[loop in f at 3loops.cpp:	inner loop was ...	0.000s	0.296s	Scalar			
[loop in f at 3loops.cpp:	inner loop was ...	0.000s	0.296s	Scalar			
[loop in f at 3loops.cpp:		0.296s	0.296s	Vectori...	AVX2	~38%	6.12x
f		0.000s	0.296s	Inlined...			
[loop in main at 3loops.c		0.234s	0.234s	Inside ...			

Roofline



Cache Simulator and MAP

Site Location	Strides Distribution	Access Pattern
[loop in ComputeTimeStep..	80% / 0% / 20%	Mixed strides
[loop in pricePath_Core at ...	92% / 0% / 8%	Mixed strides

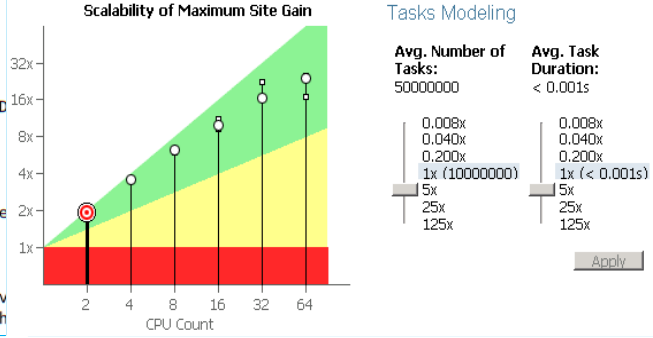
Memory Access Patterns Report | Dependencies Report | Recommenda

ID	Stride	Type	Source
P53	1	Unit stride	ch_4_v253.cpp:76
P1		Gather stride	ch_4_v253.cpp:76
P52	1	Unit stride	ch_4_v253.cpp:218

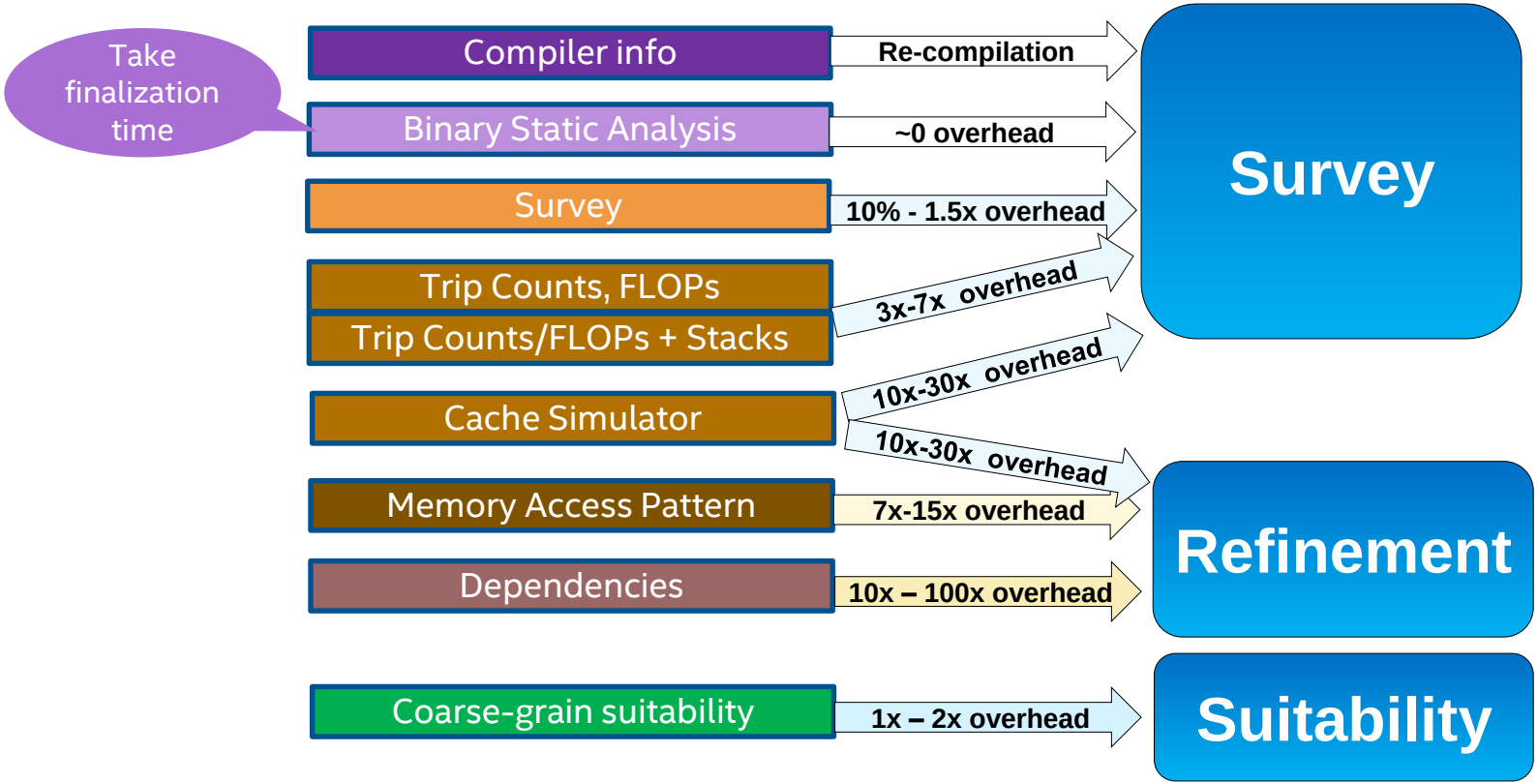
Python API

```
import advisor

project = advisor.open_project(sys.argv[1])
data = project.load(advisor.SURVEY)

roofs = data.get_roofs(4, advisor.RoofsStrategy.MULTI_THREAD

for roof in roofs:
    # memory roofs
    if 'bandwidth' in roof.name.lower():
        bandwidth = roof.bandwidth / math.pow(10, 9) # conve
        print '{} {:.0f} GB/s'.format(roof.name, bandwidth)
    # compute roofs
    else:
        bandwidth = roof.bandwidth / math.pow(10, 9)  # conv
        print '{} {:.0f} GFLOPS'.format(roof.name, bandwidth
```

Threading prototyping



What is Intel® Advisor - Collectors



Collections vs Analysis

Analysis	Collections
Vectorization (basic)	Survey + Trip Counts
Vectorization (advanced)	As above + MAP + Dependencies
Roofline (CARM)	Survey + Trip Counts with FLOP
Roofline (Integrated)	Survey + Trip Counts with FLOP and Cache Simulator
Threading	Survey + Suitability + Dependencies
Custom Analysis (Python API)	Depends

Mix and match as you wish

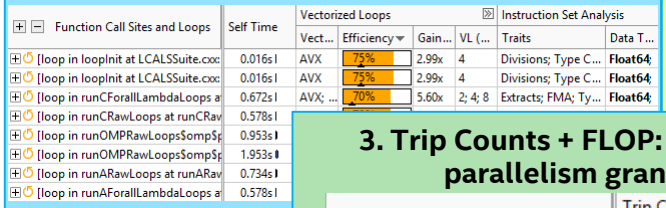
More data come with a cost

VECTORIZATION

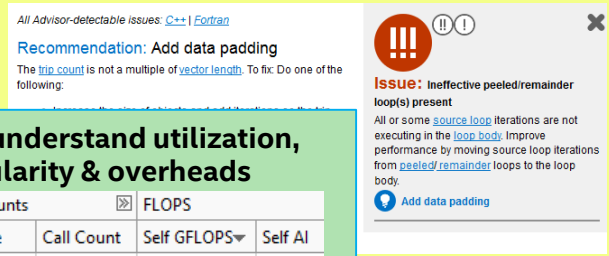
5 Steps to Efficient Vectorization

Intel® Advisor – Vectorization Advisor

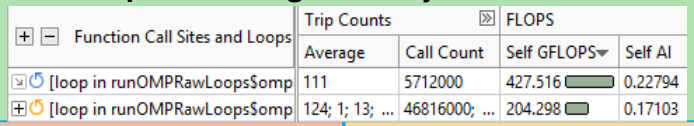
1. Compiler diagnostics + Performance Data + SIMD efficiency information



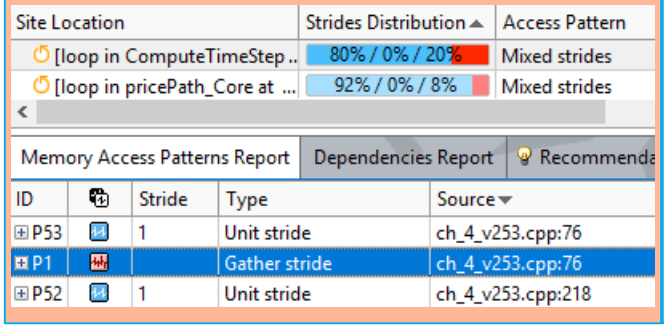
2. Guidance: detect problem and recommend how to fix it



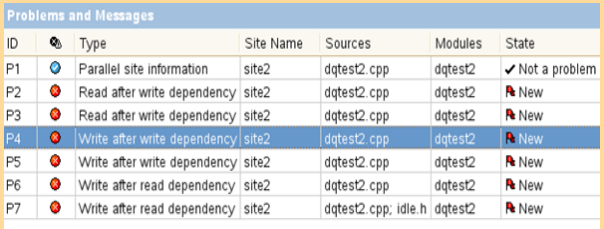
3. Trip Counts + FLOP: understand utilization, parallelism granularity & overheads



4. Memory Access Patterns Analysis



5. Loop-Carried Dependency Analysis



1. Compiler diagnostics + Performance Data + SIMD efficiency information

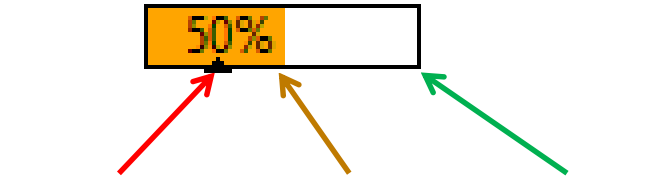
Function Call Sites and Loops	Self Time	Vectorized Loops				Instruction Set Analysis	
		Vect...	Efficiency	Gain...	VL (...	Traits	Data T...
[loop in loopInit at LCALSSuite.cxx:	0.016s	AVX	75%	2.99x	4	Divisions; Type C...	**Float64**;
[loop in loopInit at LCALSSuite.cxx:	0.016s	AVX	75%	2.99x	4	Divisions; Type C...	**Float64**;
[loop in runCForallLambdaLoops a	0.672s	AVX; ...	70%	5.60x	2; 4; 8	Extracts; FMA; Ty...	**Float64**;
[loop in runCRawLoops at runCRaw	0.578s	AVX; ...	70%	5.60x	2; 4; 8	Extracts; FMA; Ty...	Float64...
[loop in runOMPRawLoopsompp	0.953s	AVX	69%	2.75x	4	FMA	**Float64**
[loop in runOMPRawLoopsompp	1.953s	AVX	68%	2.74x	4		**Float64**;
[loop in runARawLoops at runARaw	0.734s	AVX2	67%	2.67x	4	Blends; Divisions; ...	Float32; F
[loop in runAForallLambdaLoops a	0.578s	AVX2	67%	2.67x	4	Blends; Divisions; ...	Float32; F

+ Binary Analysis

Vector Efficiency: All The Data In One Place

My "performance thermometer"

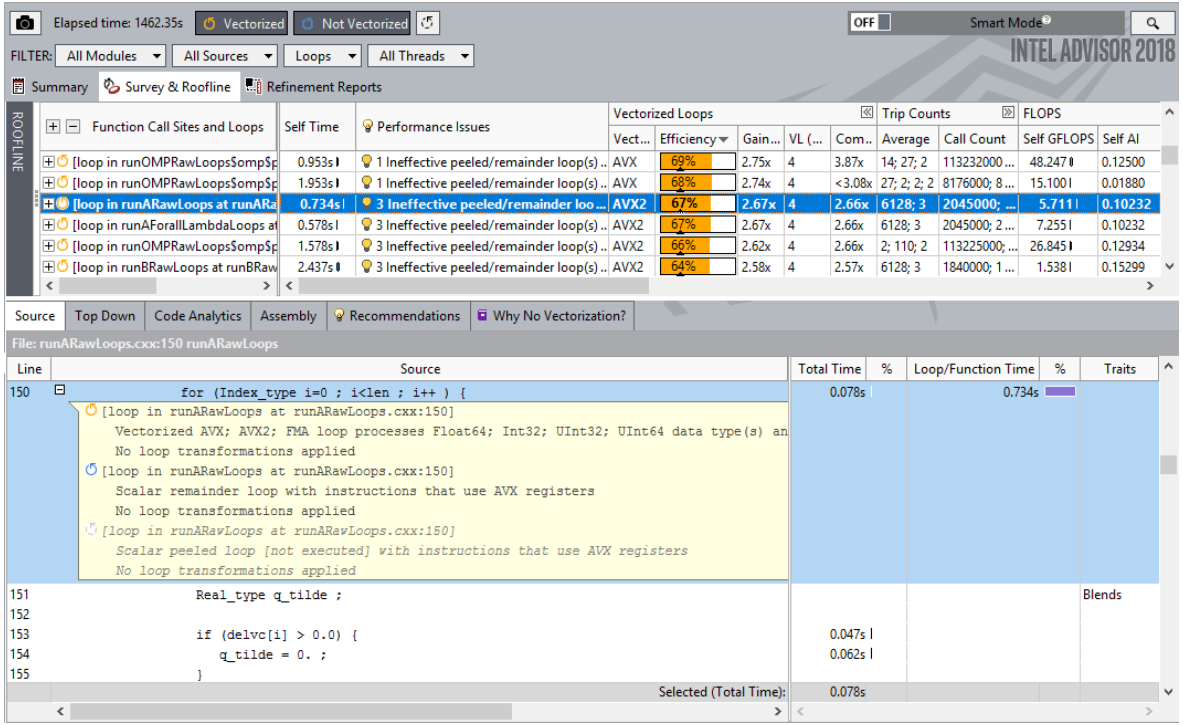
Function Call Sites and Loops	Self Time	Vectorized Loops				Instruction Set Analysis
		Vect...	Efficiency	Gain...	VL (...	Traits
[loop in runCForallLambdaLoops at runCForallLar	0.734s	AVX; ...	26%	2.11x	4; 8	Extracts; Inserts; Type Conversions
[loop in runCRawLoops at runCRawLoops.cxx:704	0.625s	AVX; ...	26%	2.11x	4; 8	Extracts; Inserts; Type Conversions
[loop in runCForallLambdaLoops at runCForallLar	2.703s	AVX2	31%	2.50x	4; 8	FMA; Inserts; Permutes; Unpacks
[loop in runCRawLoops at runCRawLoops.cxx:117	2.609s	AVX2	31%	2.50x	4; 8	FMA; Inserts; Permutes; Unpacks
[loop in runOMPRawLoopsompparallel@135 at	0.453s	AVX2	45%	1.80x	4	Blends; Divisions; FMA; Masked Stores; Square Roots
[loop in runAForallLambdaLoops at runAForallLar	0.234s	AVX2	45%	1.82x	4	Blends; Divisions; FMA; Masked Stores; Square Roots



- **Auto-vectorization**: affected <3% of code
  - With moderate speed-ups
- First attempt to **simply put #pragma omp simd**:
  - Introduced slow-down
- Look at Vector Issues and Traits to find out why
  - All kinds of "memory manipulations"
  - Usually an indication of "bad" access pattern

Survey: Find out if your code is "under vectorized" and why

Details for any piece of code



1. Compiler diagnostics + Performance Data + SIMD efficiency information

Function Call Sites and Loops	Self Time	Vectorized Loops				Instruction Set Analysis	
		Vect...	Efficiency	Gain...	VL (...	Traits	Data T...
[loop in loopInit at LCALSSuite.cxx:	0.016s	AVX	75%	2.99x	4	Divisions; Type C...	Float64;
[loop in loopInit at LCALSSuite.cxx:	0.016s	AVX	75%	2.99x	4	Divisions; Type C...	Float64;
[loop in runCForallLambdaLoops a	0.672s	AVX; ...	70%	5.60x	2; 4; 8	Extracts; FMA; Ty...	Float64;
[loop in runCRawLoops at runCRav	0.578s	AVX; ...	70%	5.60x	2; 4; 8	Extracts; FMA; Ty...	Float64...
[loop in runOMPRawLoopsompp	0.953s	AVX	69%	2.75x	4	FMA	Float64
[loop in runOMPRawLoopsompp	1.953s	AVX	68%	2.74x	4		Float64;
[loop in runARawLoops at runARav	0.734s	AVX2	67%	2.67x	4	Blends; Divisions; ...	Float32; F
[loop in runAForallLambdaLoops a	0.578s	AVX2	67%	2.67x	4	Blends; Divisions; ...	Float32; F

2. Guidance: detect problem and recommend how to fix it

All Advisor-detectable issues: C++ | Fortran

Recommendation: Add data padding

The trip count is not a multiple of vector length. To fix: Do one of the following:

- Increase the size of objects and add iterations so the trip count is a multiple of vector length.
- Increase the size of static and automatic objects, and use a compiler option to add data padding.

Windows* OS	Linux* OS
/Qopt-assume-safe-padding	-qopt-assume-safe-padding

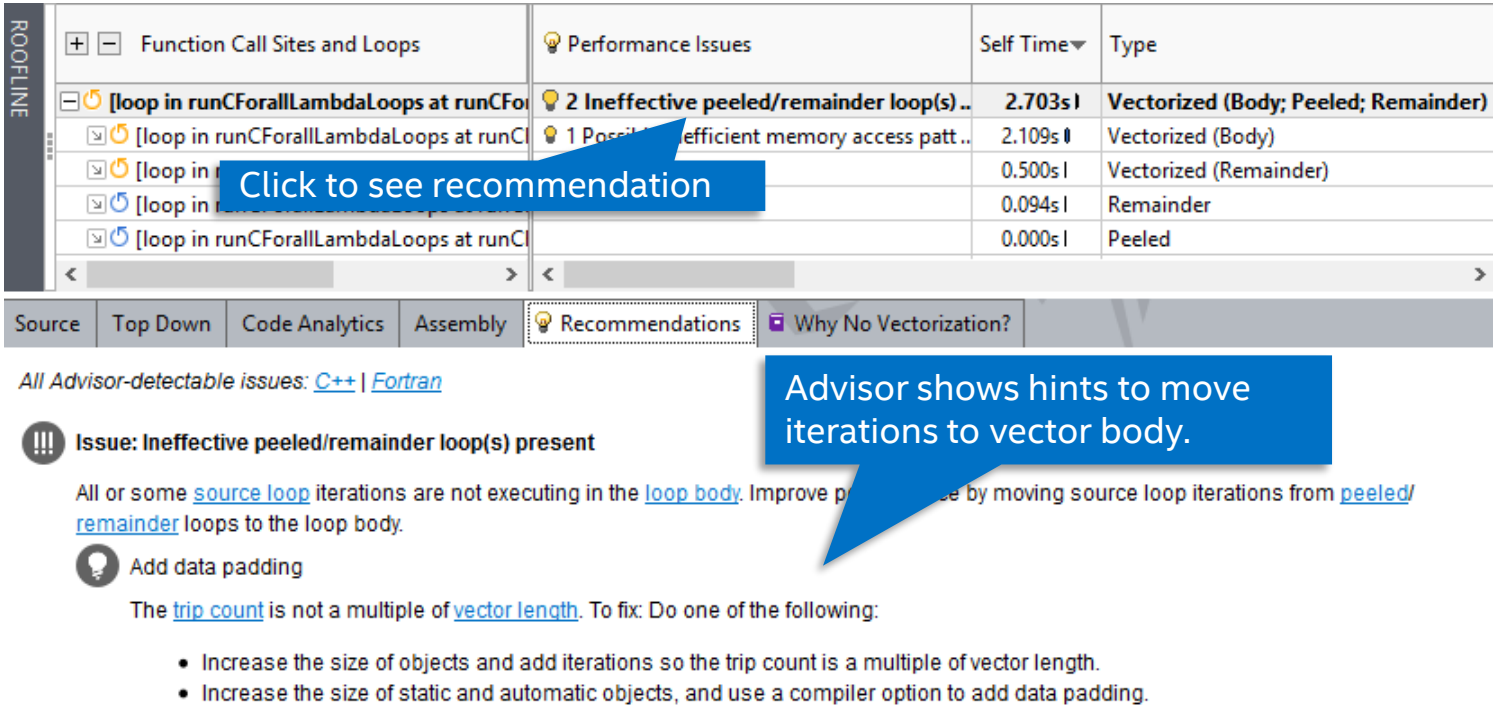
Issue: **Ineffective peeled/remainder loop(s) present**

All or some source loop iterations are not executing in the loop body. Improve performance by moving source loop iterations from peeled/ remainder loops to the loop body.

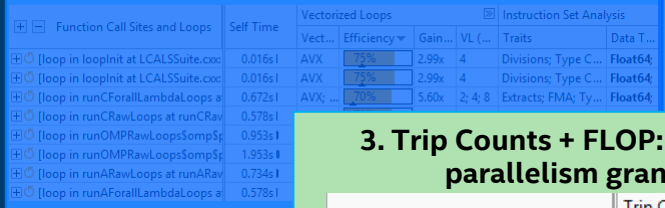
Add data padding

Get Specific Advice For Improving Vectorization

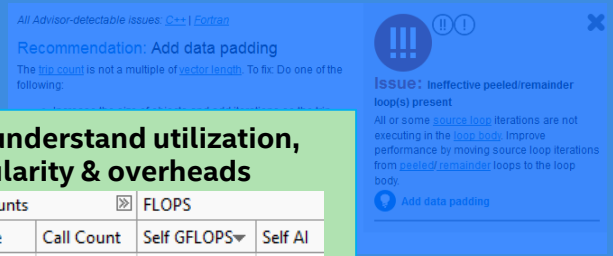
Intel® Advisor – Vectorization Advisor



1. Compiler diagnostics + Performance Data + SIMD efficiency information



2. Guidance: detect problem and recommend how to fix it



3. Trip Counts + FLOP: understand utilization, parallelism granularity & overheads

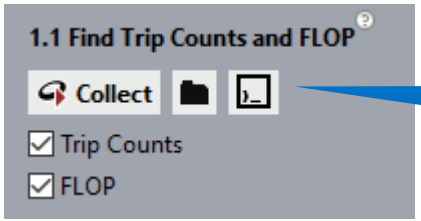
Function Call Sites and Loops	Trip Counts		FLOPS	
	Average	Call Count	Self GFLOPS	Self AI
[loop in runOMPRawLoops$omp	111	5712000	427.516	0.22794
[loop in runOMPRawLoops$omp	124; 1; 13; ...	46816000; ...	204.298	0.17103

Critical Data Made Easy

Loop Trip Counts

Knowing the time spent in a loop is not enough!

Function Call Sites and Loops	Self Time	Type	Trip Counts			
			Average	Min	Max	Call Count
[loop in runOMPRawLoopsompp	4.190s	Vectorized+Threaded (Body; Peeled; Remainder)	2; 110; 2	1; 17; 1	3; 111; 3	112590000 ...
[loop in runOMPRawLoops$om	3.768s	Remainder+Threaded (OpenMP)	2	1	3	1125900000
[loop in runOMPRawLoops$om	0.406s	Vectorized (Body)+Threaded (OpenMP)	110	17	111	12320000
[loop in runOMPRawLoops$om	0.016s	Peeled+Threaded (OpenMP)	2	1	3	880000



Check actual trip counts

Find trip counts for each part of a loop

Precise Repeatable FLOP Metrics

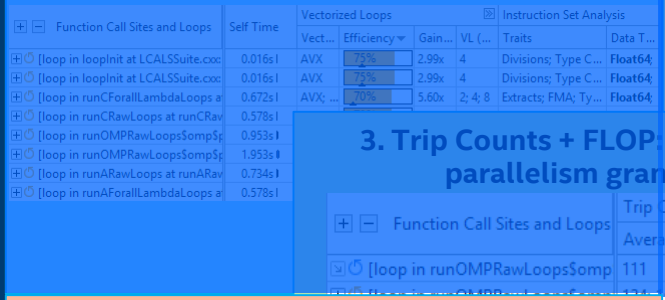
Intel® Advisor – Vectorization Optimization

- FLOPS by loop and function
- All recent Intel processors

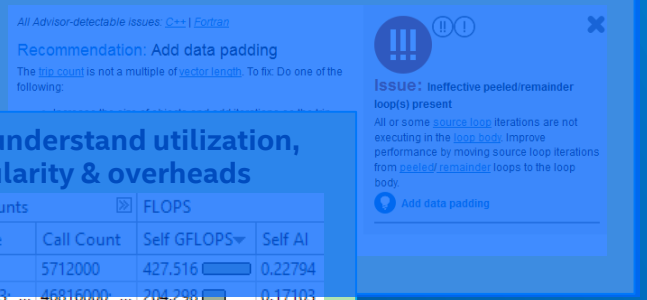
- Instrumentation (count FLOP) plus sampling (time with low overhead)
- Adjusted for masking with AVX-512 processors

Function Call Sites and Loops	Self Time	Vectorized Loops				FLOPS	
		Vect...	Efficiency	Gain...	VL (...	Self GFLOPS	Self AI
[loop in runOMPRawLoopsompp	1.984s	AVX; ...	100%	4.30x	4	204.298	0.17103
[loop in runOMPRawLoops$om	1.469s	AVX2			4	398.921	0.17574
[loop in runOMPRawLoops$om	0.078s	AVX			4	20.633	0.06250
[loop in runOMPRawLoops$om	0.141s					13.152	0.06250
[loop in runOMPRawLoops$om	0.234s					12.797	0.14315
[loop in runOMPRawLoops$om	0.063s					0.104	0.06250
[loop in runOMPRawLoopsompp	1.406s	AVX2	52%	1.05x	2	107.057	0.22428
[loop in runOMPRawLoopsompp	1.172s	AVX	81%	3.22x	4	63.354	0.07500

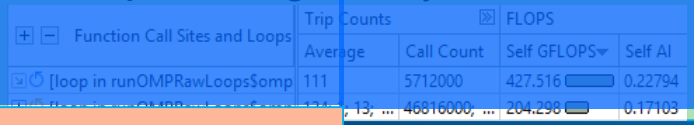
1. Compiler diagnostics + Performance Data + SIMD efficiency information



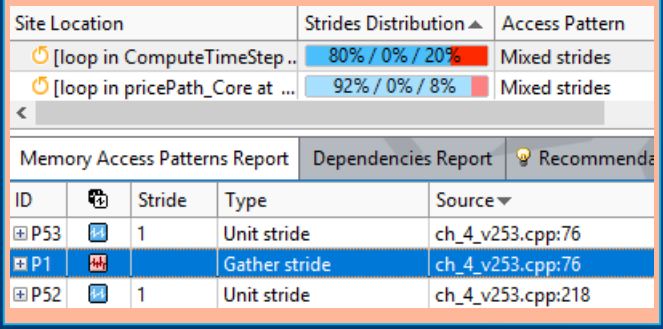
2. Guidance: detect problem and recommend how to fix it



3. Trip Counts + FLOPS: understand utilization, parallelism granularity & overheads

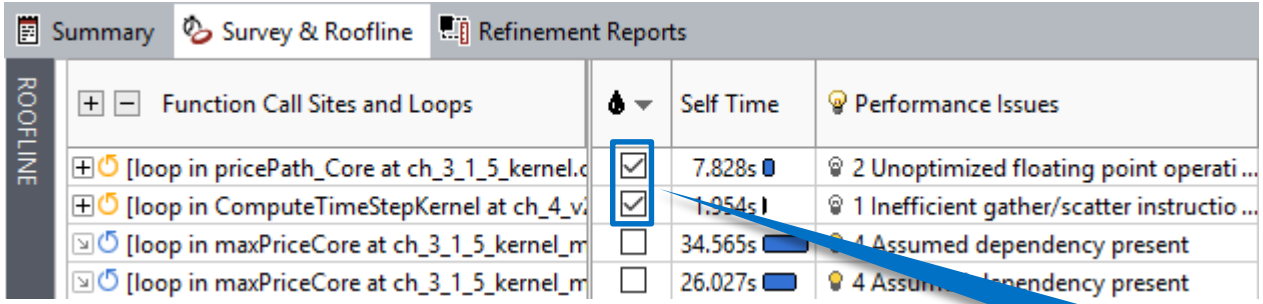


3. Memory Access Patterns Analysis



Improve Vectorization

Memory Access pattern analysis



Run Memory Access Patterns analysis, just to check how memory is used in the loop and the called function

Advisor Memory Access Pattern (MAP):
know your access pattern

Unit-Stride access

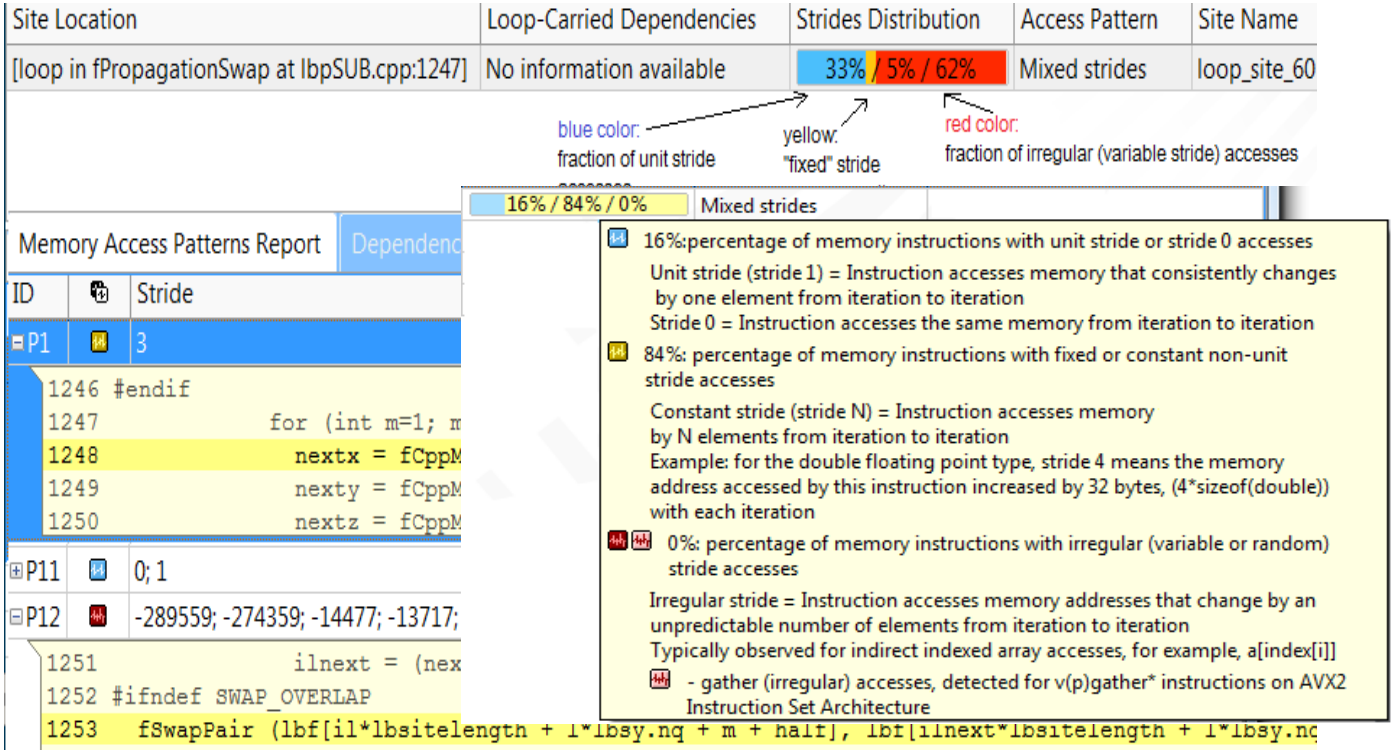
```
for (i=0; i<N; i++)
A[i] = C[i]*D[i]
```

Constant stride access

```
for (i=0; i<N; i++)
point[i].x = x[i]
```

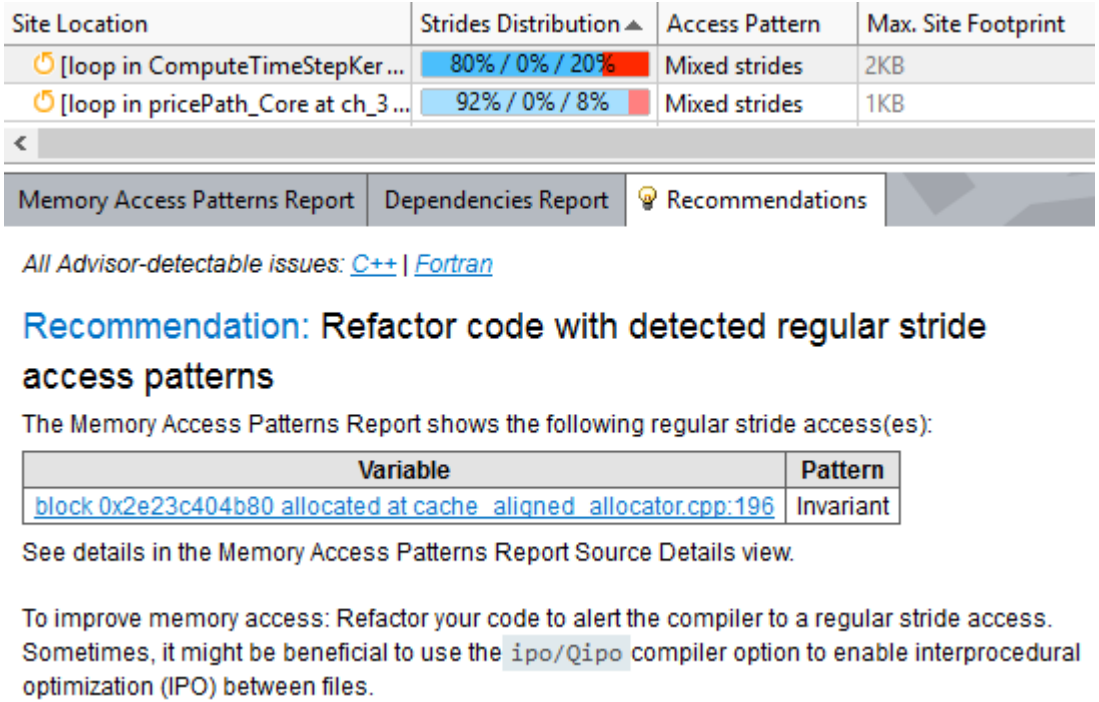
Variable stride access

```
for (i=0; i<N; i++)
A[B[i]] = C[i]*D[i]
```



Find vector optimization opportunities

Memory Access pattern analysis



Site Location	Strides Distribution	Access Pattern	Max. Site Footprint
[loop in ComputeTimeStepKer ...	80% / 0% / 20%	Mixed strides	2KB
[loop in pricePath_Core at ch_3 ...	92% / 0% / 8%	Mixed strides	1KB

Memory Access Patterns Report | Dependencies Report | Recommendations

All Advisor-detectable issues: C++ | Fortran

Recommendation: Refactor code with detected regular stride access patterns

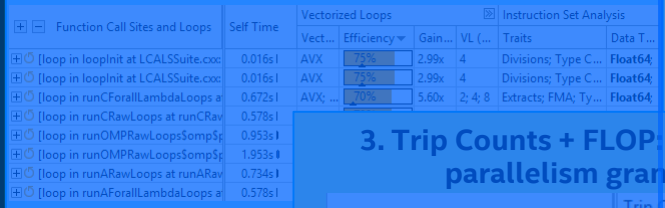
The Memory Access Patterns Report shows the following regular stride access(es):

Variable	Pattern
block 0x2e23c404b80 allocated at cache_aligned_allocator.cpp:196	Invariant

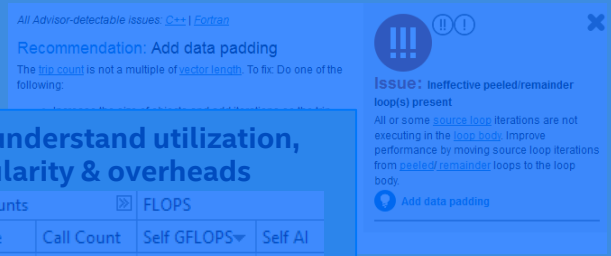
See details in the Memory Access Patterns Report Source Details view.

To improve memory access: Refactor your code to alert the compiler to a regular stride access. Sometimes, it might be beneficial to use the `ipo/Qipo` compiler option to enable interprocedural optimization (IPO) between files.

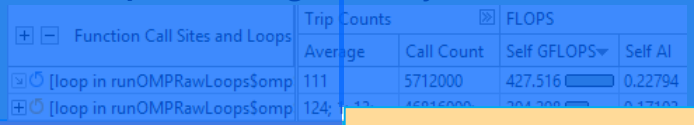
1. Compiler diagnostics + Performance Data + SIMD efficiency information



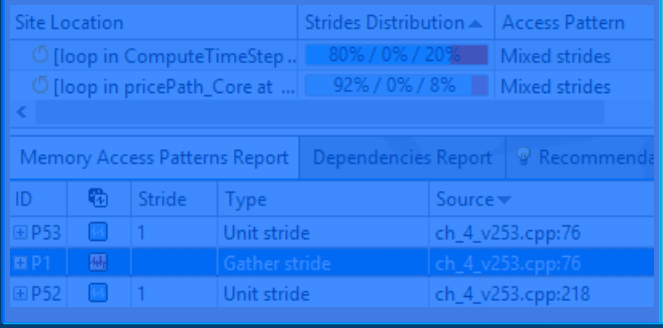
2. Guidance: detect problem and recommend how to fix it



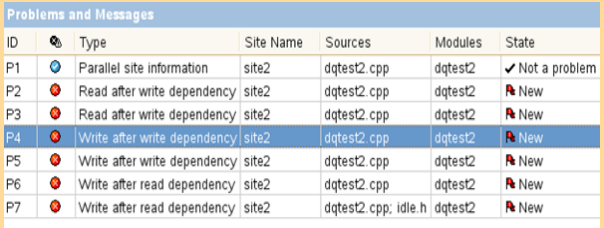
3. Trip Counts + FLOPS: understand utilization, parallelism granularity & overheads



4. Memory Access Patterns Analysis



5. Loop-Carried Dependency Analysis



ENABLING VECTORIZATION

Vector Issues	Self Time	Total Time	Type	W
2 Assumed dependency present	20.030s	20.030s	Scalar Versions	
	13.508s	13.508s	Scalar	
	6.895s	27.750s	Scalar	

Check dependencies

Refinement Reports

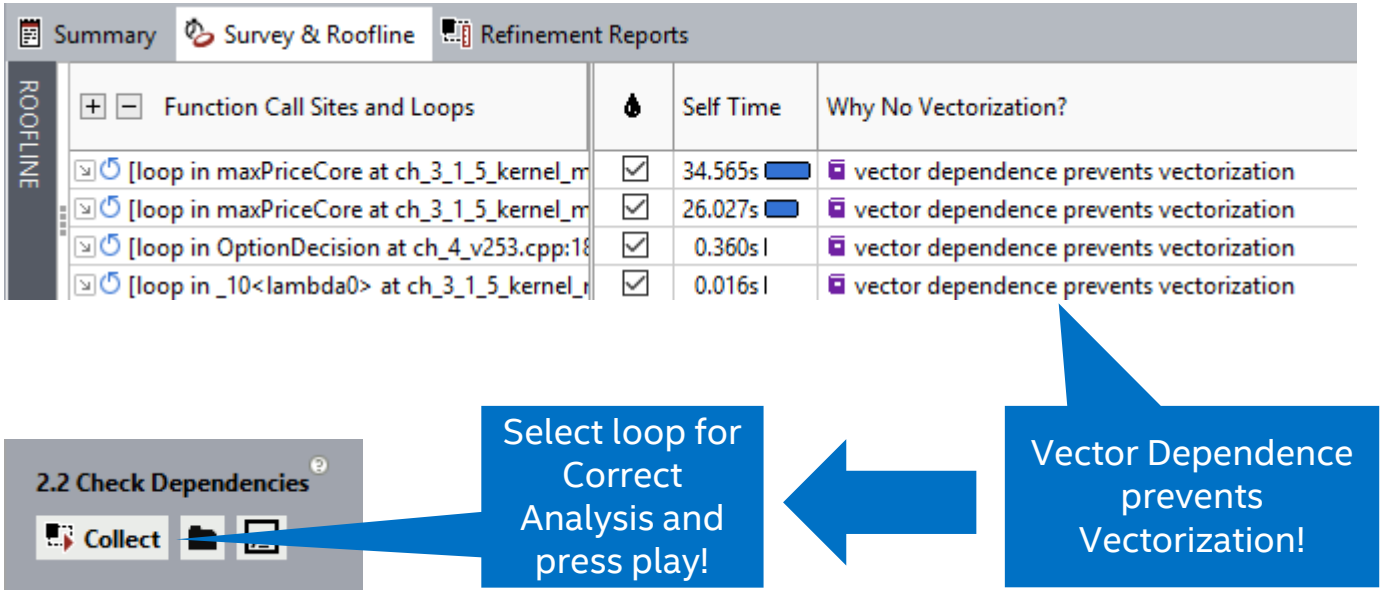
	Loop-Carried Dependencies
h_4_v253.cpp:183]	No dependencies found
_3_1_5_kernel_max.cpp:80]	No dependencies found

Use #pragma simd

Vector Issues	Self Time	Total Time	Type	Vectorized Loops			
				Vector ISA	Efficiency	Gain ...	VL (V...
	10.507s	22.989s	Scalar				
2 Possible inef ...	1.762s	3.190s	Vectorized Ver...	AVX512	73%	5.84x	8

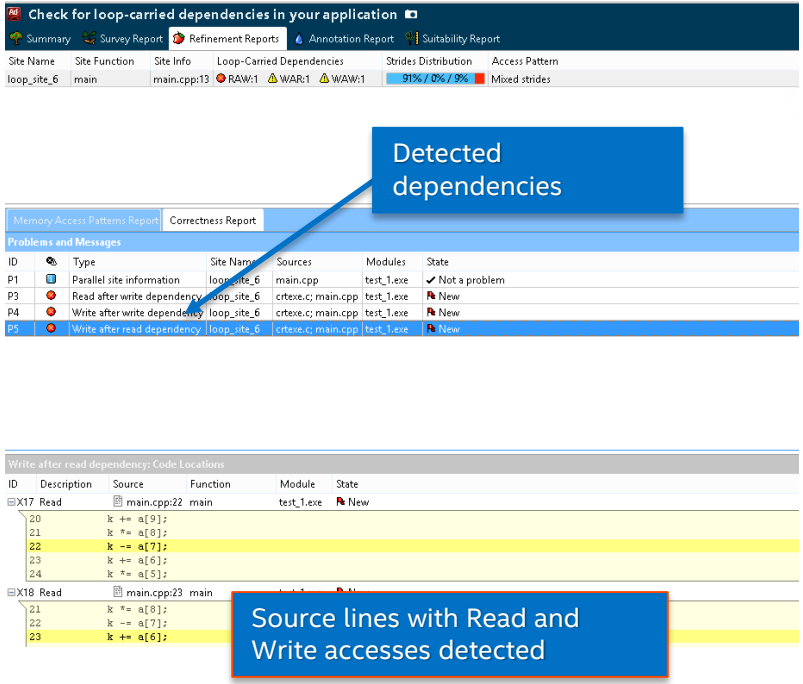
Is It Safe to Vectorize?

Loop-carried dependencies analysis verifies correctness



Correctness – Is It Safe to Vectorize?

Loop-carried dependencies analysis



Received recommendations to force vectorization of a loop:

1. Mark-up loop and check for REAL dependencies
2. Explore dependencies with code snippets

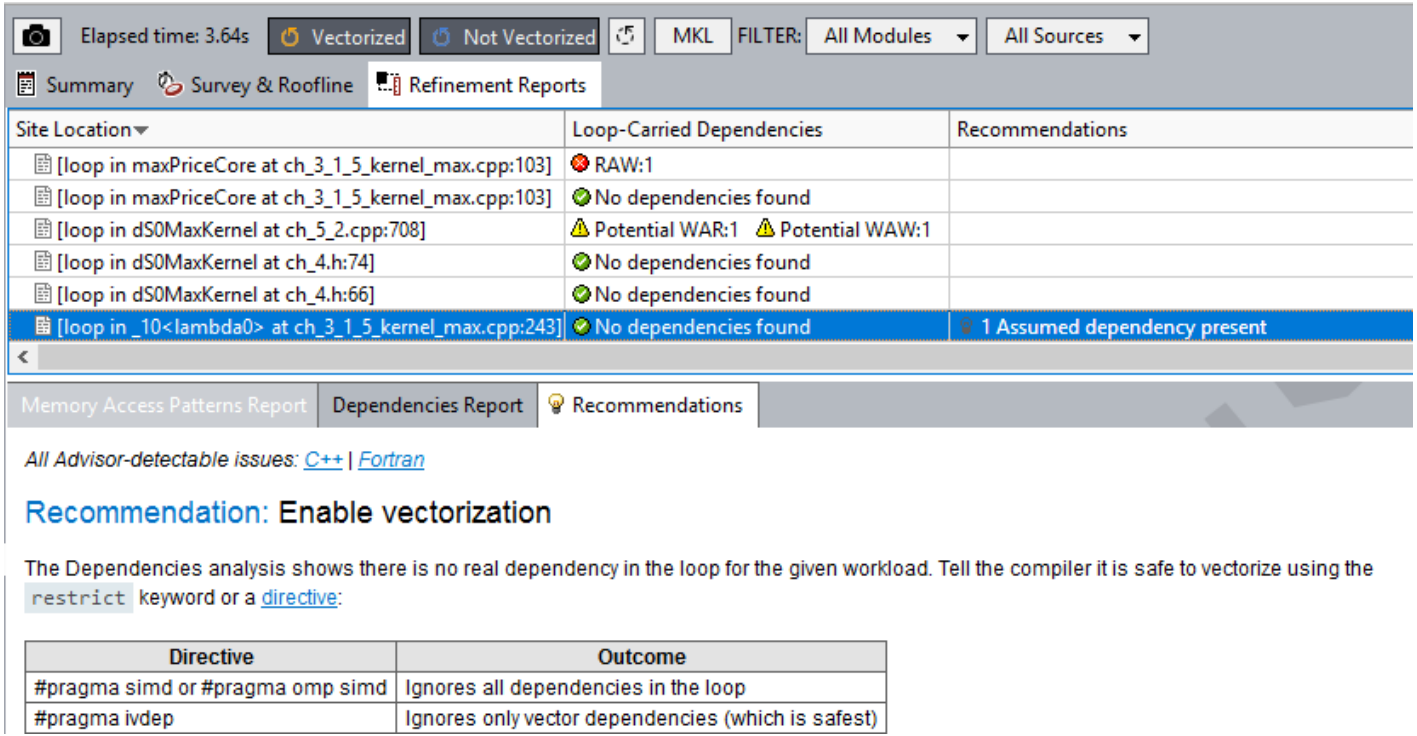
In this example 3 dependencies were detected:

- RAW – Read After Write
- WAR – Write After Read
- WAW – Write After Write

This is NOT a good candidate to force vectorization!

Data Dependencies – Tough Problem

Is it safe to force the compiler to vectorize?



ROOFLINE

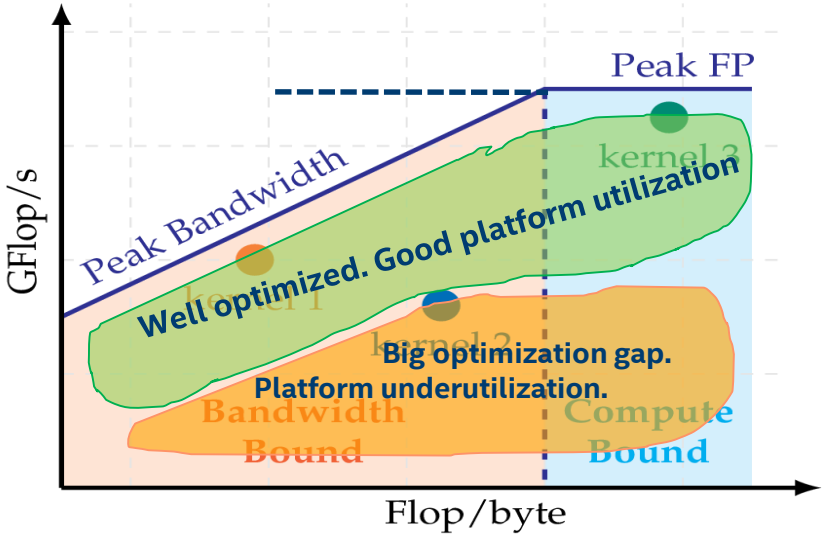
Questions to answer with Roofline: for your loops / functions

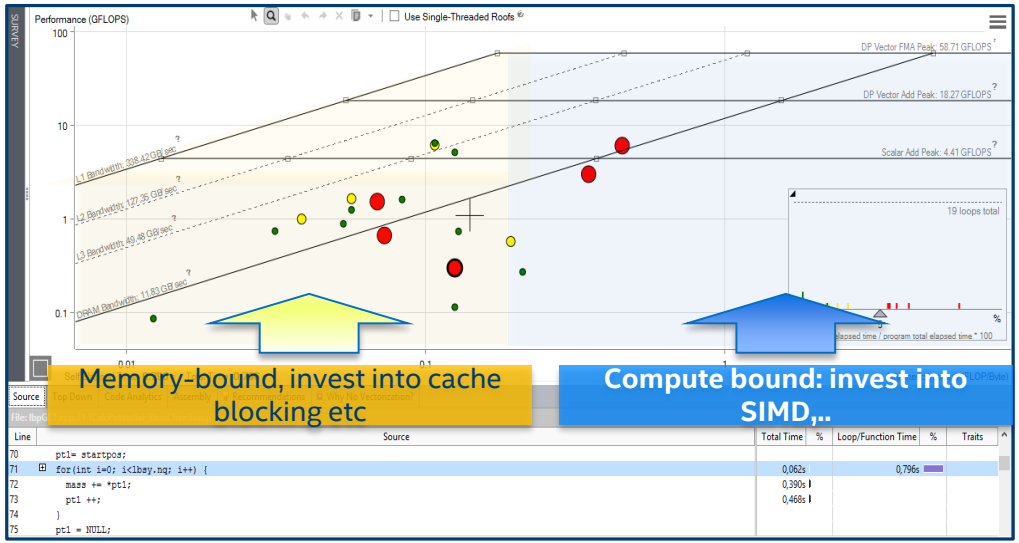
Am I doing well? How far am I from the peak?

(do I utilize hardware well or not?)

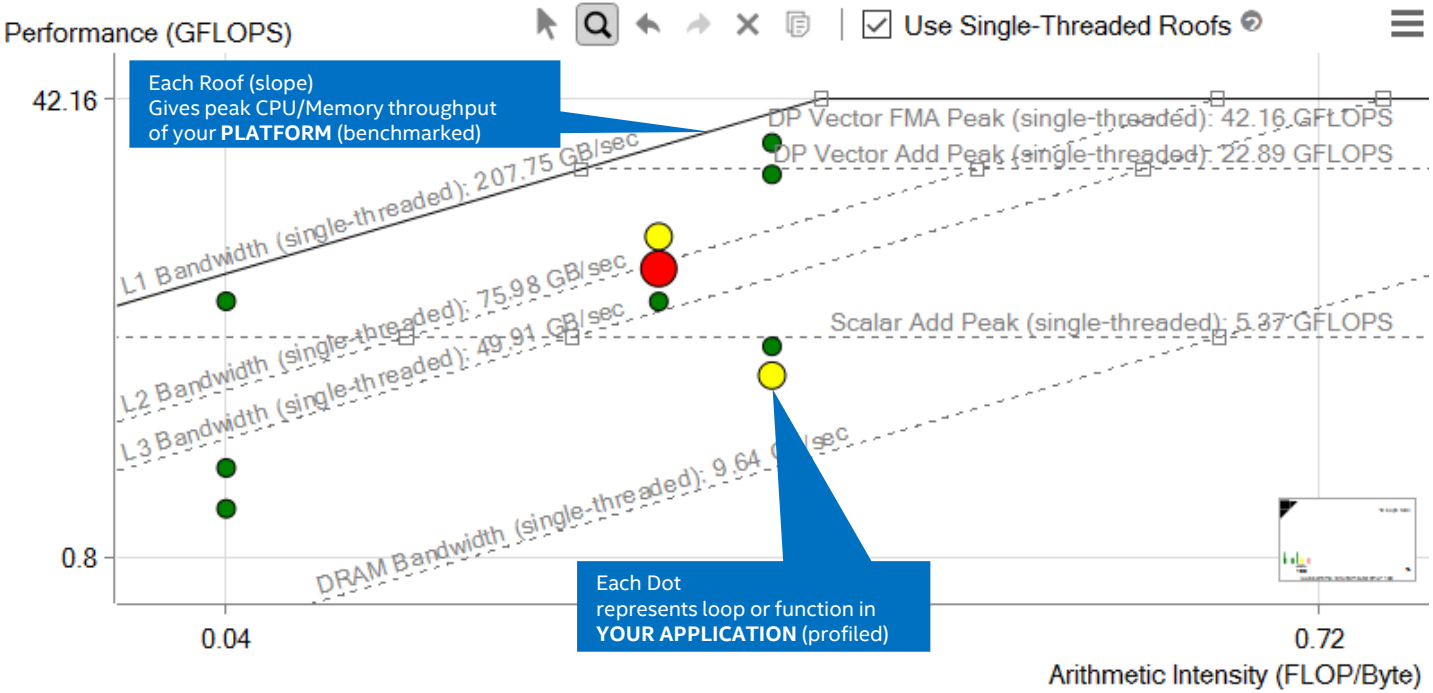
Where is the final bottleneck?

(where will be my limit after all optimizations?)
Long-term ROI, optimization strategy





Automated Roofline Chart Generation in Advisor - CARM

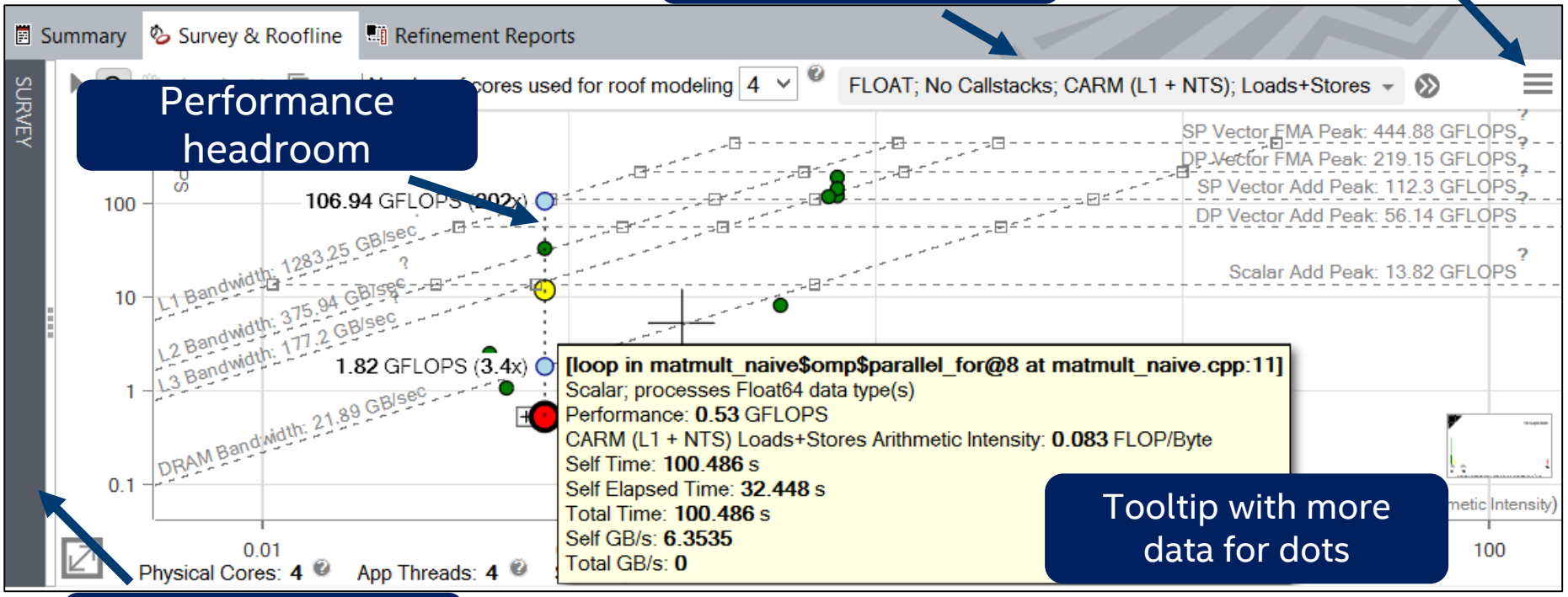


Legend:

- Takes less time
- Takes considerable time
- Takes much time

Summarized memory-compute efficiency picture for the application

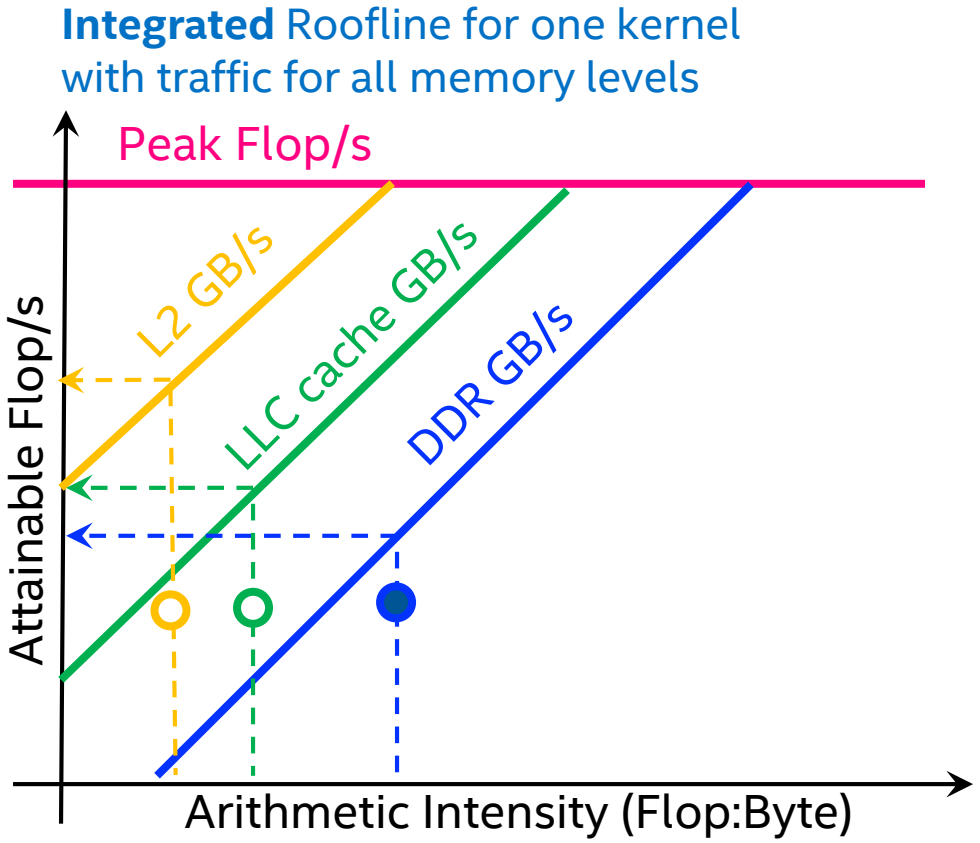
Roofline picture



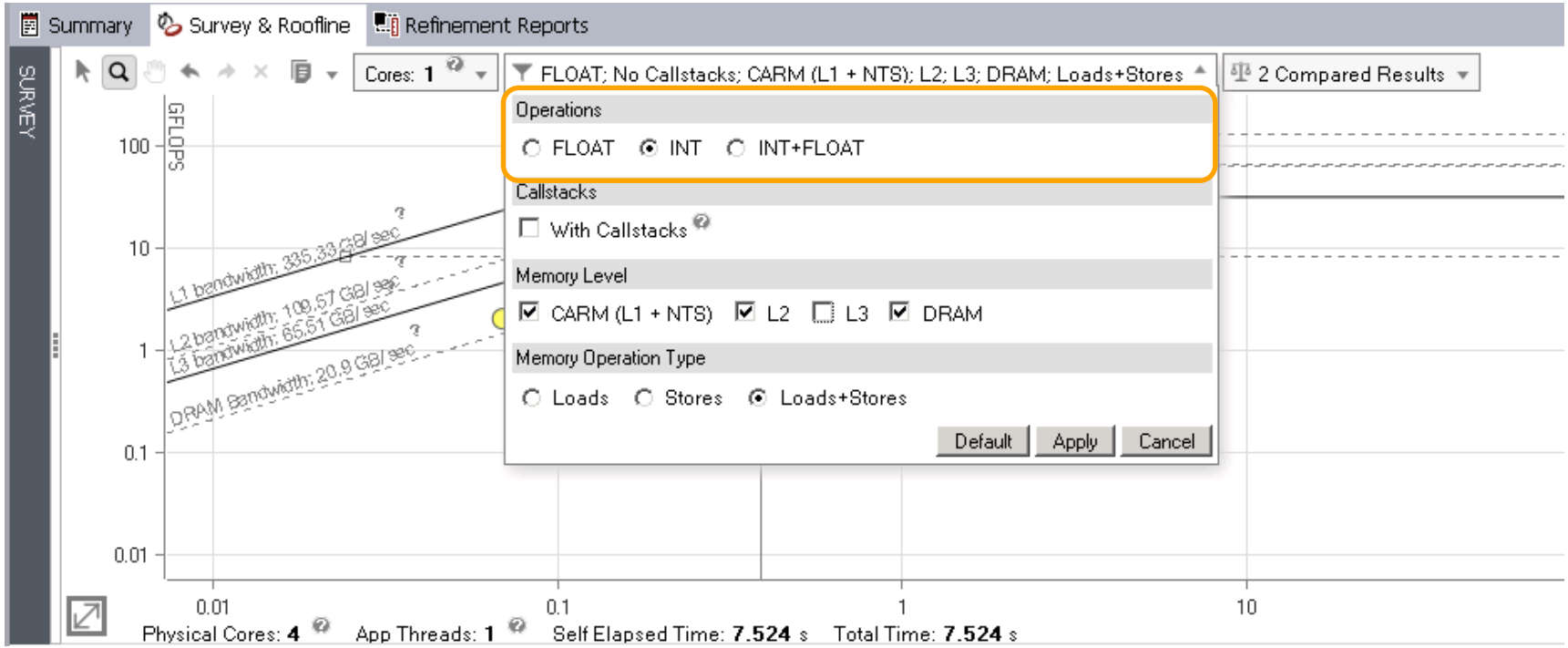
Integrated Roofline. What is my current limit?

Performance is limited by minimum of intercepts (L2, LLC, DRAM, CPU)

In this case: ***by DRAM***

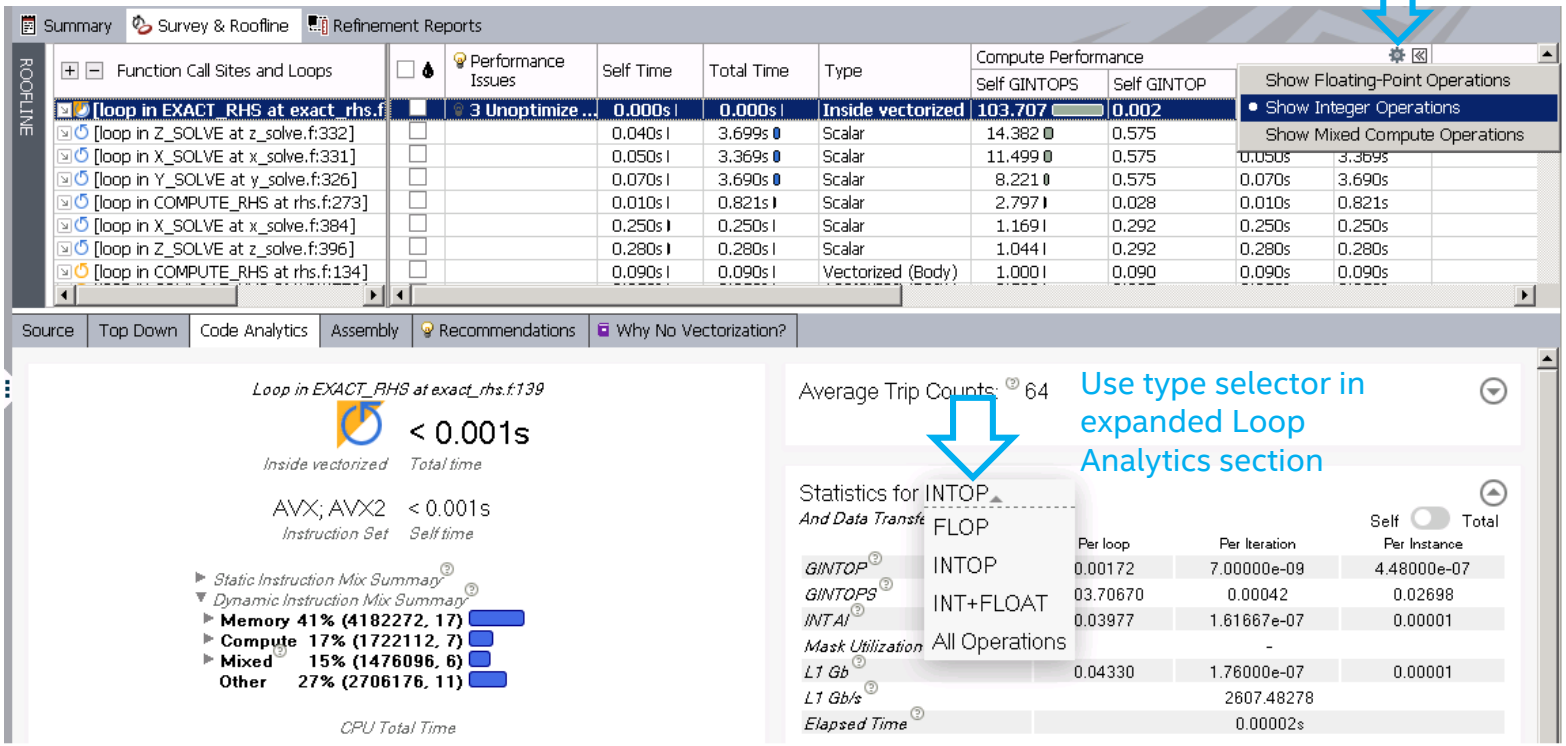


NEW*: Selecting Integer, Float or Mixed operations

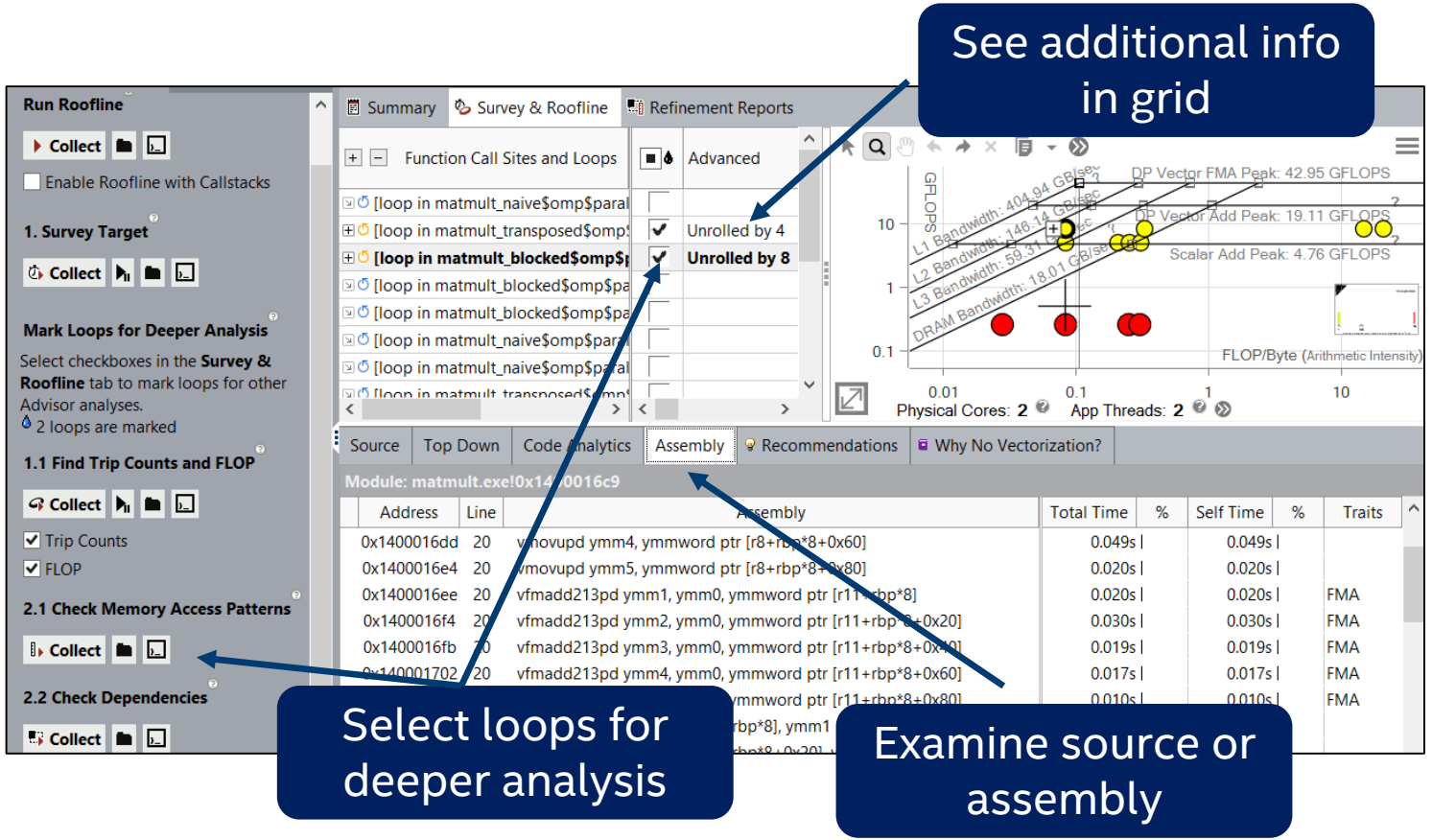


Integer Operations in Survey Grid and Loop Analytics

Use settings button on the “Compute Performance” column



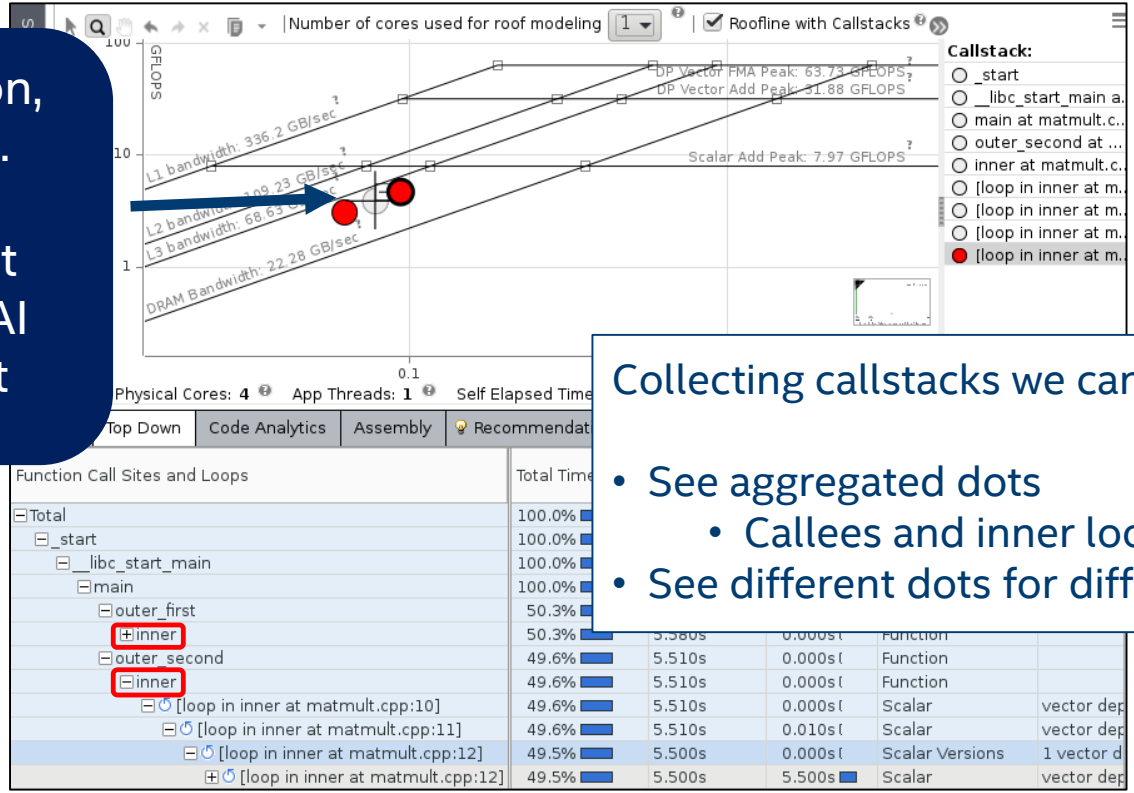
Use the rest of the Advisor



A few words about callstacks

Same function, same loop…

But different FLOPS and AI on different callpaths!



Collecting callstacks we can

- See aggregated dots
  - Callees and inner loops included
- See different dots for different callchains

Available in current product version, does not work with hierarchical roofline yet

In a few words

Using Intel® Advisor, you can

- Collect the data for the Hierarchical and Integrated Roofline
- Analyze the roofline picture
- Focus on data you are interested in
- Compare roofline for different runs
- Share roofline results
- and more

PYTHON API

Python API

- Small library and Python module to access all Advisor-collected data. Actively used internally at Intel.
- About 200 metrics for each loop/function + about 500 instruction mix metrics.
- Supports:
  - Python 2.7
  - Linux: external and internal Python
  - Windows: internal Python only (advixe-python)
- See examples at <advisor_install_dir>/pythonapi/examples

Python API - example

```
import advisor

project = advisor.open_project(sys.argv[1])
data = project.load(advisor.SURVEY)

roofs = data.get_roofs(4, advisor.RoofsStrategy.MULTI_THREAD)

for roof in roofs:
    # memory roofs
    if 'bandwidth' in roof.name.lower():
        bandwidth = roof.bandwidth / math.pow(10, 9) # converting to GByte/s
        print '{} {:.0f} GB/s'.format(roof.name, bandwidth)
    # compute roofs
    else:
        bandwidth = roof.bandwidth / math.pow(10, 9)  # converting to GFlOPS
        print '{} {:.0f} GFLOPS'.format(roof.name, bandwidth)
```

QUESTIONS?

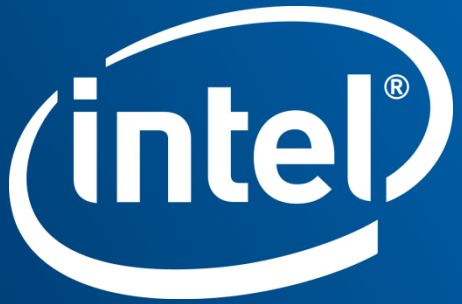
Read more

Intro on [integer] roofline usage:

https://software.intel.com/en-us/articles/a-brief-overview-of-integer-roofline-modeling-in-intel-advisor

Short intro on Python API usage:

https://software.seek.intel.com/LP=18275


intel®

Legal Disclaimer & Optimization Notice

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.

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.

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