The background features several bright blue, glowing light trails that sweep across the frame from left to right, converging towards the right side. The trails vary in thickness and intensity, creating a sense of motion and energy. In the background, there is a faint, semi-transparent pattern of binary code (0s and 1s) that appears to be part of the light trails or a separate layer of digital data.

INTEL[®] ADVISOR AND ROOFLINE MODEL

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

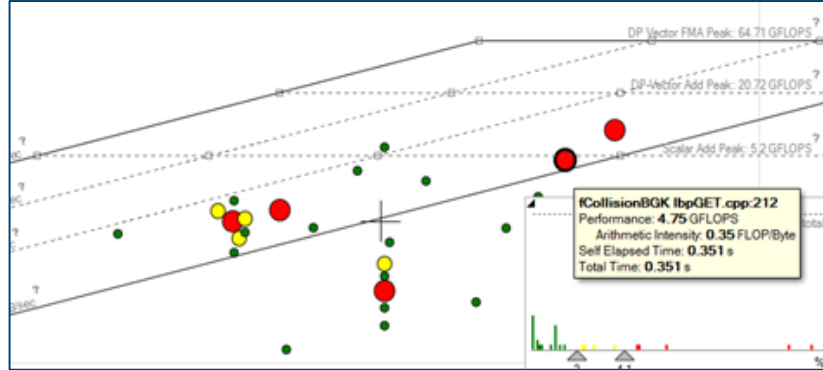
Intel Advisor Senior Developer

What is Intel® Advisor

Vectorization analysis

Function Call Sites and Loops	Why No Vectorization?	Self Time	Total Time	Type	Vectorized Loops		
					Vec...	Efficiency	Gain...
f _tmainCRTStartup		0.000s	1.669s	Function			
f main		0.000s	1.669s	Function			
[loop in main at 3loops.c]	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 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

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

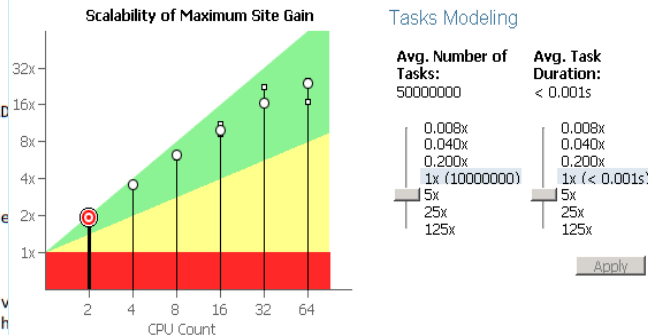
```
import advisor

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) # convert to GB/s
        print '{} {:.0f} GB/s'.format(roof.name, bandwidth)
    # compute roofs
    else:
        bandwidth = roof.bandwidth / math.pow(10, 9) # convert to GFLOPS
        print '{} {:.0f} GFLOPS'.format(roof.name, bandwidth)
```

Threading prototyping





VECTORIZATION

Get Faster Code Faster! Intel® Advisor

Vectorization Optimization

Have you:

- Recompiled for AVX2 with little gain
- Wondered where to vectorize?
- Recoded intrinsics for new arch.?
- Struggled with compiler reports?

Data Driven Vectorization:

- What vectorization will pay off most?
- What's blocking vectorization? Why?
- Are my loops vector friendly?
- Will reorganizing data increase performance?
- Is it safe to just use #pragma omp simd?

The screenshot shows the Intel Advisor 2018 interface. At the top, there are filters for 'Elapsed time: 1462.35s', 'Vectorized' (selected), 'Not Vectorized', and 'FILTER: All Modules, All Sources, Loops, All Threads'. Below the filters are tabs for 'Summary', 'Survey & Roofline', and 'Refinement Reports'. The main table displays the following data:

Function Call Sites and Loops	Performance Issues	Self Time	Total Time	Type	Why No Vectorization?	Vectorized Loops					FLOPS	
						Vect...	Efficiency	Gain...	VL (...)	Com...	Self GFLOPS	Self AI
[loop in runOMPRowLoops\$omp\$parallel_f...	2 Assumed d...	15.484s	578.046s	Threaded (Op...	vector dependence...						2.235	0.02459
[loop in runCRowLoops at runCRowLoops.cxx]	2 Assumed d...	11.766s	11.766s	Scalar	vector dependence...						0.995	0.08333
[loop in runCForAllLambdaLoops at runCForal...]	2 Assumed d...	11.766s	11.766s	Scalar	vector dependence...						0.995	0.08333
[loop in runCRowLoops at runCRowLoops.cxx]	2 Assumed d...	5.156s	5.156s	Scalar	vector dependence...						1.512	0.11458
[loop in runCForAllLambdaLoops at runCForal...]	2 Assumed d...	5.125s	5.125s	Scalar	vector dependence...						1.521	0.11458
[loop in runOMPRowLoops\$omp\$parallel@64]	1 Ineffective ...	4.190s	4.190s	Vectorized+Thr...		AVX	100%	5.10x	4	5.28x	6.767	0.02486
[loop in runOMPRowLoops\$omp\$parallel@...		3.768s	3.768s	Remainder+Th...							4.138	0.02083
[loop in runOMPRowLoops\$omp\$parallel@...		0.406s	0.406s	Vectorized (Bo...		AVX			4	5.28x	27.368	0.03125
[loop in runOMPRowLoops\$omp\$parallel@...		0.016s	0.016s	Peeled+Thread...							0.113	0.02083

Optimization Notice

Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



The Right Data At Your Fingertips

Get all the data you need for high impact vectorization

The screenshot shows the Intel Advisor 2018 interface with several callouts pointing to specific data points:

- Filter by which loops are vectorized!**: Points to the 'Vectorized' and 'Not Vectorized' filters.
- Trip Counts**: Points to the 'Trip Counts' column header.
- What prevents vectorization?**: Points to the 'Why No Vectorization?' column.
- Focus on hot loops**: Points to the top row of the table, which has the highest self time.
- What vectorization issues do I have?**: Points to the 'Performance Issues' column.
- Which Vector instructions are being used?**: Points to the 'Vectorized Loops' section, specifically the 'AVX' instruction.
- How efficient is the code?**: Points to the 'Efficiency' column, which shows a value of 100% for the AVX instruction.

Function Call Sites and Loops	Performance Issues	Self Time	Trip Counts		Why No Vectorization?	Vectorized Loops				
			Average	Call Count		Vect...	Efficiency	Gain...	VL (...)	Com..
[loop in runOMPRawLoopsSomp...	2 Assumed dependency present	15.484s	446	101976000	vector dependence prevents vectoriz...					
[loop in runCRawLoops at runCRa...	2 Assumed dependency present	11.766s	12511	75120000	vector dependence prevents vectorization					
[loop in runCForAllLambdaLoops a...	2 Assumed dependency present	11.766s	12511	75120000	vector dependence prevents vectorization					
[loop in runCRawLoops at runCRa...	2 Assumed dependency present	5.156s	19387	3075000	vector dependence prevents vectorization					
[loop in runCForAllLambdaLoops a...	2 Assumed dependency present	5.125s	19387	3075000	vector dependence prevents vectorization					
[loop in runOMPRawLoopsSompS...	1 Ineffective peeled/remainder loop(s)...	4.190s	2; 110; 2	112590000...		AVX	100%	5.10x	4	5.28x
[loop in runOMPRawLoopsSomp...		3.768s	2	112590000						
[loop in runOMPRawLoopsSomp...		0.406s	110	12320000		AVX			4	5.28x
[loop in runOMPRawLoopsSomp...		0.016s	2	880000						

Get Faster Code Faster!

5 Steps to Efficient Vectorization

Intel® Advisor – Vectorization Advisor

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 loopInnit at LCALLSuite.coc...	0.016s	AVX	75%	2.99x	4	Divisions; Type C...	Float64
[loop in loopInnit at LCALLSuite.coc...	0.016s	AVX	75%	2.99x	4	Divisions; Type C...	Float64
[loop in runCForallLambdaLoops at ...	0.672s	AVX; ...	70%	5.60x	2; 4; 8	Extracts; FMA; Ty...	Float64
[loop in runCRawLoops at runCRa...	0.578s						
[loop in runOMPRawLoopsSompS...	0.953s						
[loop in runOMPRawLoopsSompS...	1.953s						
[loop in runARawLoops at runARa...	0.734s						
[loop in runAForallLambdaLoops a...	0.578s						

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:

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](#)

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 runOMPRawLoopsSomp...	111	5712000	427.516	0.22794
[loop in runOMPRawLoopsSomp...	124; 1; 13; ...	46816000; ...	204.298	0.17103

4. Memory Access Patterns Analysis

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

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

5. Loop-Carried Dependency Analysis

ID	Type	Site Name	Sources	Modules	State
P1	Parallel site information	site2	dqtest2.cpp	dqtest2	✓ Not a problem
P2	Read after write dependency	site2	dqtest2.cpp	dqtest2	🔴 New
P3	Read after write dependency	site2	dqtest2.cpp	dqtest2	🔴 New
P4	Write after write dependency	site2	dqtest2.cpp	dqtest2	🔴 New
P5	Write after write dependency	site2	dqtest2.cpp	dqtest2	🔴 New
P6	Write after read dependency	site2	dqtest2.cpp	dqtest2	🔴 New
P7	Write after read dependency	site2	dqtest2.cpp; idle.h	dqtest2	🔴 New

Optimization Notice

Copyright © 2017, Intel Corporation. All rights reserved.
 *Other names and brands may be claimed as the property of others.



1. Compiler diagnostics + Performance Data + SIMD efficiency information

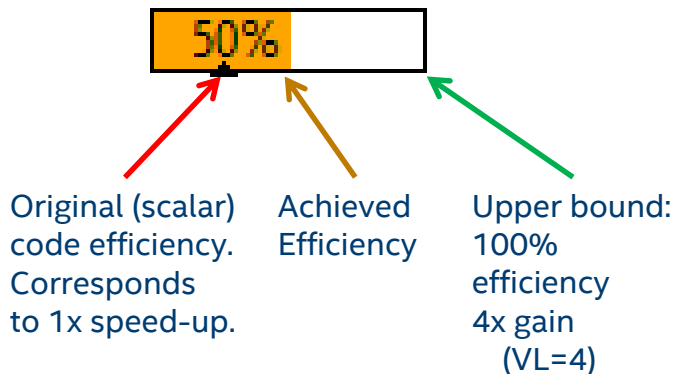
Function Call Sites and Loops	Self Time	Vectorized Loops				Instruction Set Analysis	
		Vect...	Efficiency ▾	Gain...	VL (...)	Traits	Data T...
loop in loopInit at LCALLSuite.coc	0.016s	AVX	75%	2.99x	4	Divisions; Type C...	Float64
loop in loopInit at LCALLSuite.coc	0.016s	AVX	75%	2.99x	4	Divisions; Type C...	Float64
loop in runCForallLambdaLoops a	0.672s	AVX; ...	70%	5.60x	2; 4; 8	Extracts; FMA; Ty...	Float64
loop in runCRawLoops at runCRa	0.578s	AVX; ...	70%	5.60x	2; 4; 8	Extracts; FMA; Ty...	Float64...
loop in runOMPRawLoopsSompSg	0.953s	AVX	69%	2.75x	4	FMA	Float64
loop in runOMPRawLoopsSompSg	1.953s	AVX	68%	2.74x	4		Float64
loop in runARawLoops at runARa	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 (...)	
[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 runOMPRawLoops\$omp\$parallel@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

Vectorization tied to your code

Elapsed time: 1462.35s Vectorized Not Vectorized OFF Smart Mode

FILTER: All Modules All Sources Loops All Threads

Summary Survey & Roofline Refinement Reports

Function Call Sites and Loops	Self Time	Performance Issues	Vectorized Loops				Trip Counts		FLOPS		
			Vect...	Efficiency	Gain...	VL (...)	Com..	Average	Call Count	Self GFLOPS	Self AI
[loop in runOMPRawLoopsSompSp]	0.953s	1 Ineffective peeled/remainder loop(s) ..	AVX	69%	2.75x	4	3.87x	14; 27; 2	113232000...	48.247	0.12500
[loop in runOMPRawLoopsSompSp]	1.953s	1 Ineffective peeled/remainder loop(s) ..	AVX	68%	2.74x	4	<3.08x	27; 2; 2; 2	8176000; 8...	15.100	0.01880
[loop in runARawLoops at runARawLoops.cxx:150]	0.734s	3 Ineffective peeled/remainder loo...	AVX2	67%	2.67x	4	2.66x	6128; 3	2045000; ...	5.711	0.10232
[loop in runAForAllLambdaLoops at runARawLoops.cxx:150]	0.578s	3 Ineffective peeled/remainder loop(s) ..	AVX2	67%	2.67x	4	2.66x	6128; 3	2045000; 2...	7.255	0.10232
[loop in runOMPRawLoopsSompSp]	1.578s	3 Ineffective peeled/remainder loop(s) ..	AVX2	66%	2.62x	4	2.66x	2; 110; 2	113225000; ...	26.845	0.12934
[loop in runBRawLoops at runBRawLoops.cxx:150]	2.437s	3 Ineffective peeled/remainder loop(s) ..	AVX2	64%	2.58x	4	2.57x	6128; 3	1840000; 1...	1.538	0.15299

Source Top Down Code Analytics Assembly Recommendations Why No Vectorization?

File: runARawLoops.cxx:150 runARawLoops

Line	Source	Total Time	%	Loop/Function Time	%	Traits
150	<pre> for (Index_type i=0 ; i<len ; i++) { [loop in runARawLoops at runARawLoops.cxx:150] Vectorized AVX; AVX2; FMA loop processes Float64; Int32; UInt32; UInt64 data type(s) and No loop transformations applied [loop in runARawLoops at runARawLoops.cxx:150] Scalar remainder loop with instructions that use AVX registers No loop transformations applied [loop in runARawLoops at runARawLoops.cxx:150] Scalar peeled loop [not executed] with instructions that use AVX registers No loop transformations applied </pre>	0.078s		0.734s		
151	Real_type q_tilde ;					Blends
153	if (delve[i] > 0.0) {	0.047s				
154	q_tilde = 0. ;	0.062s				
155	}					
Selected (Total Time):		0.078s				

Optimization Notice

Don't Just Vectorize, Vectorize Efficiently

See detailed times for each part of your loops. Is it worth more effort?

Function Call Sites and Loops	Self Time	Type	Vectorized Loops			
			Vector ISA	Efficiency	Gain ...	VL ...
[loop in runCRawLoops at runCRawLoops.cxx:117]	2.609s	Vectorized (Body; Peeled; Remainder)	AVX2	31%	2.50x	4; 8
[loop in runCRawLoops at runCRawLoops.cxx:117]	2.031s	Vectorized (Body)	AVX2			8
[loop in runCRawLoops at runCRawLoops.cxx:117]	0.516s	Vectorized (Remainder)	AVX2			4
[loop in runCRawLoops at runCRawLoops.cxx:117]	0.062s	Remainder				
[loop in runCRawLoops at runCRawLoops.cxx:117]	0.000s	Peeled				
[loop in runBRawLoops at runBRawLoops.cxx:55]	2.562s	Vectorized (Body; Remainder)	AVX	80%	3.19x	4
[loop in runBRawLoops at runBRawLoops.cxx:55]	2.500s	Vectorized (Body)	AVX			4
[loop in runBRawLoops at runBRawLoops.cxx:55]	0.062s	Remainder				
[loop in runBRawLoops at runBRawLoops.cxx:55]	0.000s	Vectorized (Remainder)	AVX			4

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 LCALLSuite.coc	0.016s	AVX	75%	2.99x	4	Divisions; Type C...	Float64
[loop in loopInit at LCALLSuite.coc	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 runCRa	0.578s	AVX; ...	70%	5.60x	2; 4; 8	Extracts; FMA; Ty...	Float64...
[loop in runOMPRawLoopsSomp5	0.953s	AVX	69%	2.75x	4	FMA	Float64
[loop in runOMPRawLoopsSomp5	1.953s	AVX	68%	2.74x	4		Float64
[loop in runARawLoops at runARa	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
/Oopt-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

Function Call Sites and Loops		Performance Issues	Self Time	Type
[-]	[loop in runCForallLambdaLoops at runC...	💡 2 Ineffective peeled/remainder loop(s) ..	2.703s	Vectorized (Body; Peeled; Remainder)
[v]	[loop in runCForallLambdaLoops at runC...	💡 1 Possible inefficient memory access patt..	2.109s	Vectorized (Body)
[v]	[loop in runCForallLambdaLoops at runC...		0.500s	Vectorized (Remainder)
[v]	[loop in runCForallLambdaLoops at runC...		0.094s	Remainder
[v]	[loop in runCForallLambdaLoops at runC...		0.000s	Peeled

Click to see recommendation

Source Top Down Code Analytics Assembly Recommendations Why No Vectorization?

All Advisor-detectable issues: [C++](#) | [Fortran](#)

!!! 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

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.

Advisor shows hints to move iterations to vector body.

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 LCALLSSuite.coc	0.016s	AVX	75%	2.99x	4	Divisions; Type C...	Float64
[loop in loopInit at LCALLSSuite.coc	0.016s	AVX	75%	2.99x	4	Divisions; Type C...	Float64
[loop in runCForallLambdaLoops at	0.672s	AVX...	70%	5.60x	2; 4; 8	Extracts; FMA; Ty...	Float64
[loop in runCRawLoops at runCRaw	0.578s						
[loop in runOMPRawLoopsSompS	0.953s						
[loop in runOMPRawLoopsSompS	1.953s						
[loop in runARawLoops at runARaw	0.734s						
[loop in runAForallLambdaLoops at	0.578s						

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:



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](#)

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 runOMPRawLoopsSomp	111	5712000	427.516	0.22794
[loop in runOMPRawLoopsSomp	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 runOMPRawLoops\$omp\$...]	4.190s	Vectorized+Threaded (Body; Peeled; Remainder)	2; 110; 2	1; 17; 1	3; 111; 3	112590000 ...
[loop in runOMPRawLoops\$omp\$...]	3.768s	Remainder+Threaded (OpenMP)	2	1	3	1125900000
[loop in runOMPRawLoops\$omp\$...]	0.406s	Vectorized (Body)+Threaded (OpenMP)	110	17	111	12320000
[loop in runOMPRawLoops\$omp\$...]	0.016s	Peeled+Threaded (OpenMP)	2	1	3	880000

1.1 Find Trip Counts and FLOP

Collect

Trip Counts

FLOP

Check actual trip counts

Find trip counts for each part of a loop

Precise Repeatable FLOP Metrics

- 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 runOMPRawLoopsSompSp	1.984s 0	AVX; ...	100%	4.30x	4	204.298	0.17103
[loop in runOMPRawLoopsSom	1.469s 1	AVX2			4	398.921	0.17574
[loop in runOMPRawLoopsSom	0.078s 1	AVX			4	20.633 0	0.06250
[loop in runOMPRawLoopsSom	0.141s 1					13.152 1	0.06250
[loop in runOMPRawLoopsSom	0.234s 1					12.797 1	0.14315
[loop in runOMPRawLoopsSom	0.063s 1					0.104 1	0.06250
[loop in runOMPRawLoopsSompSp	1.406s 1	AVX2	52%	1.05x	2	107.057	0.22428
[loop in runOMPRawLoopsSompSp	1.172s 1	AVX	81%	3.22x	4	63.354 0	0.07500

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 LCALLSSuite.coc	0.016s	AVX	75%	2.99x	4	Divisions; Type C...	Float64
[loop in loopInIt at LCALLSSuite.coc	0.016s	AVX	75%	2.99x	4	Divisions; Type C...	Float64
[loop in runCForallLambdaLoops at	0.672s	AVX...	70%	5.60x	2; 4; 8	Extracts; FMA; Ty...	Float64
[loop in runCRawLoops at runCRaw	0.578s						
[loop in runOMPRawLoopsSomp5g	0.953s						
[loop in runOMPRawLoopsSomp5g	1.953s						
[loop in runARawLoops at runARaw	0.734s						
[loop in runAForallLambdaLoops at	0.578s						

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:



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](#)

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

Function Call Sites and Loops	Trip Counts		FLOPS	
	Average	Call Count	Self GFLOPS	Self AI
[loop in runOMPRawLoopsSomp	111	5712000	427.516	0.22794
[loop in runCForallLambdaLoops	134	46816000	204.298	0.17103

3. Memory Access Patterns Analysis

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		Recommendations
ID	Icon	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

Improve Vectorization

Memory Access pattern analysis

Summary			Survey & Roofline		Refinement Reports	
ROOFLINE	+ -	Function Call Sites and Loops	🔥	Self Time	💡	Performance Issues
	+ 🔁	[loop in pricePath_Core at ch_3_1_5_kernel.c	<input checked="" type="checkbox"/>	7.828s	💡	2 Unoptimized floating point operati ...
	+ 🔁	[loop in ComputeTimeStepKernel at ch_4_v	<input checked="" type="checkbox"/>	1.954s	💡	1 Inefficient gather/scatter instructio ...
	⌵ 🔁	[loop in maxPriceCore at ch_3_1_5_kernel_m	<input type="checkbox"/>	34.565s	💡	4 Assumed dependency present
	⌵ 🔁	[loop in maxPriceCore at ch_3_1_5_kernel_m	<input type="checkbox"/>	26.027s	💡	4 Assumed dependency present

Select loops of interest

2.1 Check Memory Access Patterns

Collect

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]
```

Site Location	Loop-Carried Dependencies	Strides Distribution	Access Pattern	Site Name
[loop in fPropagationSwap at lbpSUB.cpp:1247]	No information available	33% / 5% / 62%	Mixed strides	loop_site_60

blue color: fraction of unit stride
 yellow: "fixed" stride
 red color: fraction of irregular (variable stride) accesses

Memory Access Patterns Report

ID	Stride
P1	3

```

1246 #endif
1247         for (int m=1; m
1248             nextx = fCppM
1249             nexty = fCppM
1250             nextz = fCppM
  
```

Strides Distribution	Access Pattern
16% / 84% / 0%	Mixed strides

- 16%: percentage of memory instructions with unit stride or stride 0 accesses
 Unit stride (stride 1) = Instruction accesses memory that consistently changes by one element from iteration to iteration
 Stride 0 = Instruction accesses the same memory from iteration to iteration
- 84%: percentage of memory instructions with fixed or constant non-unit stride accesses
 Constant stride (stride N) = Instruction accesses memory by N elements from iteration to iteration
 Example: for the double floating point type, stride 4 means the memory address accessed by this instruction increased by 32 bytes, (4*sizeof(double)) with each iteration
- 0%: percentage of memory instructions with irregular (variable or random) stride accesses
 Irregular stride = Instruction accesses memory addresses that change by an unpredictable number of elements from iteration to iteration
 Typically observed for indirect indexed array accesses, for example, a[index[i]]

■ - gather (irregular) accesses, detected for v(p)gather* instructions on AVX2 Instruction Set Architecture

Optimization Notice

Copyright © 2017, Intel Corporation. All rights reserved.
 *Other names and brands may be claimed as the property of others.



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

Function Call Sites and Loops	Self Time	Vectorized Loops				Instruction Set Analysis	
		Vect...	Efficiency	Gain...	VL (...)	Traits	Data T...
[loop in loopInnit at LCALLSSuite.coc	0.016s	AVX	75%	2.99x	4	Divisions; Type C...	Float64
[loop in loopInnit at LCALLSSuite.coc	0.016s	AVX	75%	2.99x	4	Divisions; Type C...	Float64
[loop in runCForallLambdaLoops at	0.672s	AVX;	70%	5.60x	2; 4; 8	Extracts; FMA; Ty...	Float64
[loop in runCRawLoops at runCRAV	0.578s						
[loop in runOMPRawLoopsSomp5g	0.953s						
[loop in runOMPRawLoopsSomp5g	1.953s						
[loop in runARawLoops at runARav	0.734s						
[loop in runAForallLambdaLoops at	0.578s						

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:



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](#)

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

Function Call Sites and Loops	Trip Counts		FLOPS	
	Average	Call Count	Self GFLOPS	Self AI
[loop in runOMPRawLoopsSomp	111	5712000	427.516	0.22794
[loop in runOMPRawLoopsSomp	124	15816000	304.300	0.17103

4. Memory Access Patterns Analysis

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		Recommendations
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	

5. Loop-Carried Dependency Analysis

Problems and Messages

ID	Type	Site Name	Sources	Modules	State
P1	Parallel site information	site2	dqtest2.cpp	dqtest2	✓ Not a problem
P2	Read after write dependency	site2	dqtest2.cpp	dqtest2	🔴 New
P3	Read after write dependency	site2	dqtest2.cpp	dqtest2	🔴 New
P4	Write after write dependency	site2	dqtest2.cpp	dqtest2	🔴 New
P5	Write after write dependency	site2	dqtest2.cpp	dqtest2	🔴 New
P6	Write after read dependency	site2	dqtest2.cpp	dqtest2	🔴 New
P7	Write after read dependency	site2	dqtest2.cpp; idle.h	dqtest2	🔴 New

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

Use #pragma simd

Refinement Reports	
	Loop-Carried Dependencies
[h_4_v253.cpp:183]	✓ No dependencies found
[_3_1_5_kernel_max.cpp:80]	✓ No dependencies found

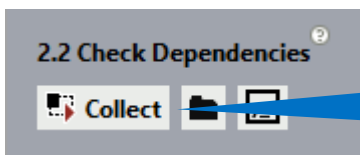
Vector Issues	Self Time▼	Total Time	Type	Vectorized Loops			
				Vector ISA	Efficiency	Gain ...	VL (V...
	10.507s	22.989s	Scalar				
• 2 Possible inef...	1.762s	3.190s	Vectorized Ver...	AVX512	73%	5.84x	8

Optimization Notice

Is It Safe to Vectorize?

Loop-carried dependencies analysis verifies correctness

Summary			Survey & Roofline		Refinement Reports	
ROOFLINE	Function Call Sites and Loops		🔥	Self Time	Why No Vectorization?	
	[-] [loop in maxPriceCore at ch_3_1_5_kernel_m	<input checked="" type="checkbox"/>		34.565s	vector dependence prevents vectorization	
	[-] [loop in maxPriceCore at ch_3_1_5_kernel_m	<input checked="" type="checkbox"/>		26.027s	vector dependence prevents vectorization	
	[-] [loop in OptionDecision at ch_4_v253.cpp:18	<input checked="" type="checkbox"/>		0.360s	vector dependence prevents vectorization	
	[-] [loop in _10<lambda0> at ch_3_1_5_kernel_m	<input checked="" type="checkbox"/>		0.016s	vector dependence prevents vectorization	



Select loop for
Correct
Analysis and
press play!

Vector Dependence
prevents
Vectorization!

Correctness – Is It Safe to Vectorize?

Loop-carried dependencies analysis

Check for loop-carried dependencies in your application

Summary Survey Report Refinement Reports Annotation Report Suitability Report

Site Name Site Function Site Info Loop-Carried Dependencies Strides Distribution Access Pattern

loop_site_6 main main.cpp:13 RAW:1 WAR:1 WAW:1 91% / 0% / 9% Mixed strides

Detected dependencies

ID	Type	Site Name	Sources	Modules	State
P1	Parallel site information	loop_site_6	main.cpp	test_1.exe	✓ Not a problem
P3	Read after write dependency	loop_site_6	critex.c; main.cpp	test_1.exe	New
P4	Write after write dependency	loop_site_6	critex.c; main.cpp	test_1.exe	New
P5	Write after read dependency	loop_site_6	critex.c; main.cpp	test_1.exe	New

ID	Description	Source	Function	Module	State
X17	Read	main.cpp:22	main	test_1.exe	New
X18	Read	main.cpp:23	main	test_1.exe	New

```
20 k += a[9];
21 k -= a[8];
22 k -= a[7];
23 k += a[6];
24 k -= a[5];
```

Source lines with Read and Write accesses detected

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 #1

Is it safe to force the compiler to vectorize?

Elapsed time: 3.64s Vectorized Not Vectorized MKL FILTER: All Modules All Sources

Summary Survey & Roofline Refinement Reports

Site Location	Loop-Carried Dependencies	Recommendations
[loop in maxPriceCore at ch_3_1_5_kernel_max.cpp:103]	RAW:1	
[loop in maxPriceCore at ch_3_1_5_kernel_max.cpp:103]	No dependencies found	
[loop in dS0MaxKernel at ch_5_2.cpp:708]	Potential WAR:1 Potential WAW:1	
[loop in dS0MaxKernel at ch_4.h:74]	No dependencies found	
[loop in dS0MaxKernel at ch_4.h:66]	No dependencies found	
[loop in _10<lambda0> at ch_3_1_5_kernel_max.cpp:243]	No dependencies found	1 Assumed dependency present

Memory Access Patterns Report Dependencies Report Recommendations

All Advisor-detectable issues: [C++](#) | [Fortran](#)

Recommendation: Enable vectorization

The Dependencies analysis shows there is no real dependency in the loop for the given workload. Tell the compiler it is safe to vectorize using the `restrict` keyword or a [directive](#):

Directive	Outcome
<code>#pragma simd</code> or <code>#pragma omp simd</code>	Ignores all dependencies in the loop
<code>#pragma ivdep</code>	Ignores only vector dependencies (which is safest)

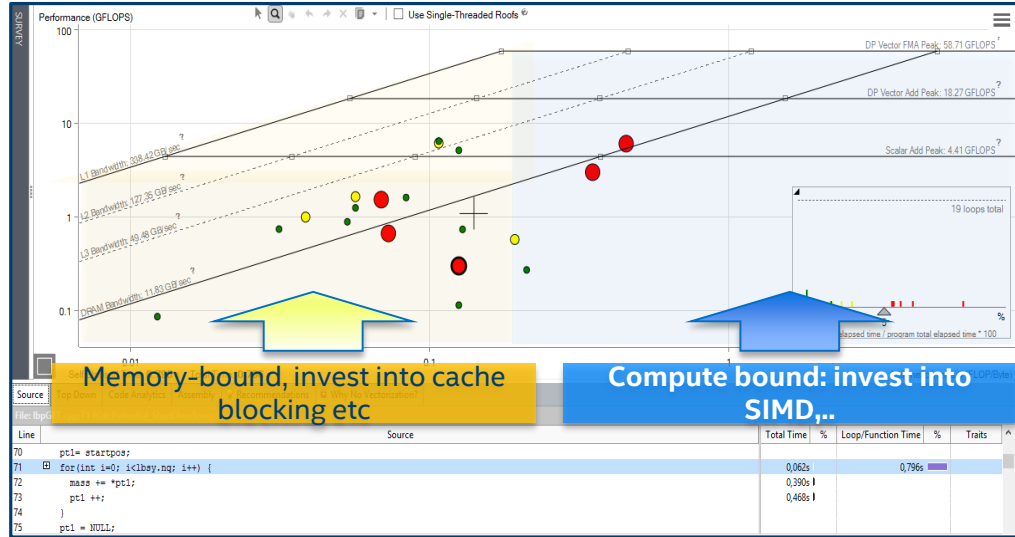
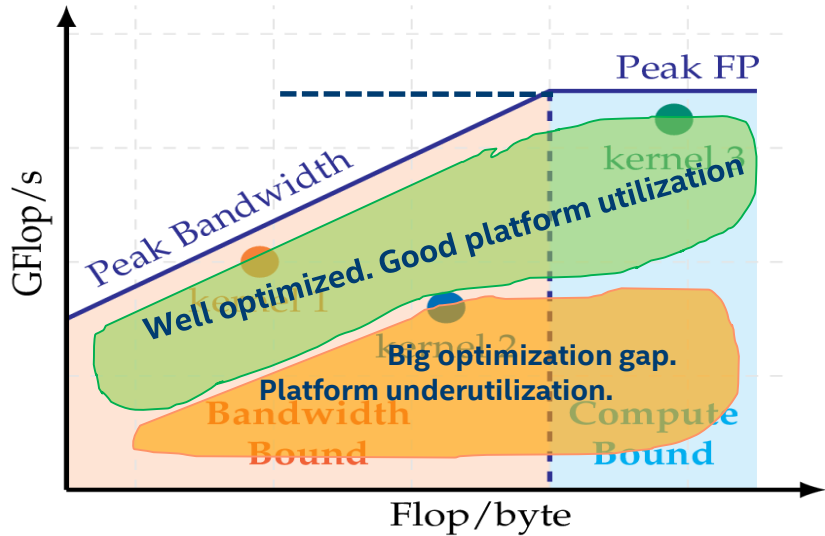
Optimization Notice

ROOFLINE

Questions to answer with Roofline: for your loops / functions

1 Am I doing well? How far am I from the peak?
(do I utilize hardware well or not?)

2 Where is the final bottleneck?
(where will be my limit after all optimizations?)
Long-term ROI, optimization strategy



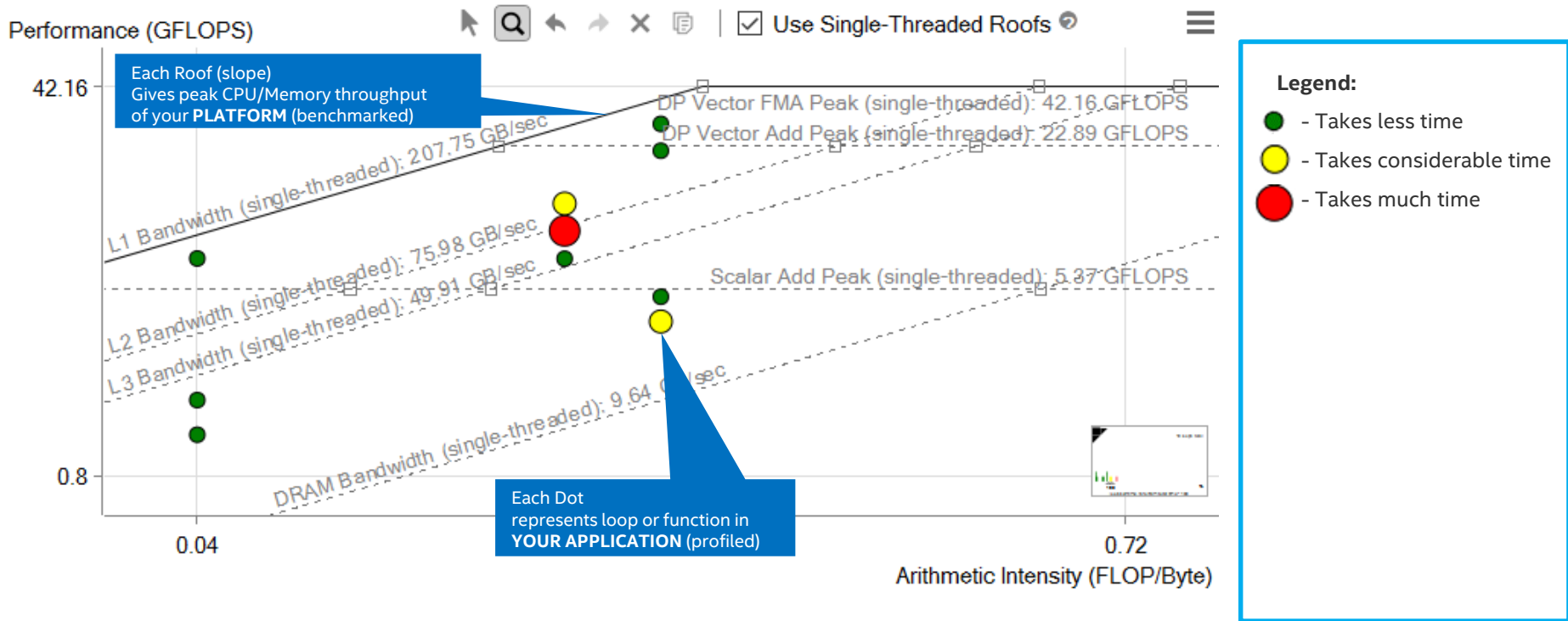
Optimization Notice

Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Intel Confidential



Automated Roofline Chart Generation in Advisor - CARM



Summarized memory-compute efficiency picture for the application

Optimization Notice

Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Intel Confidential

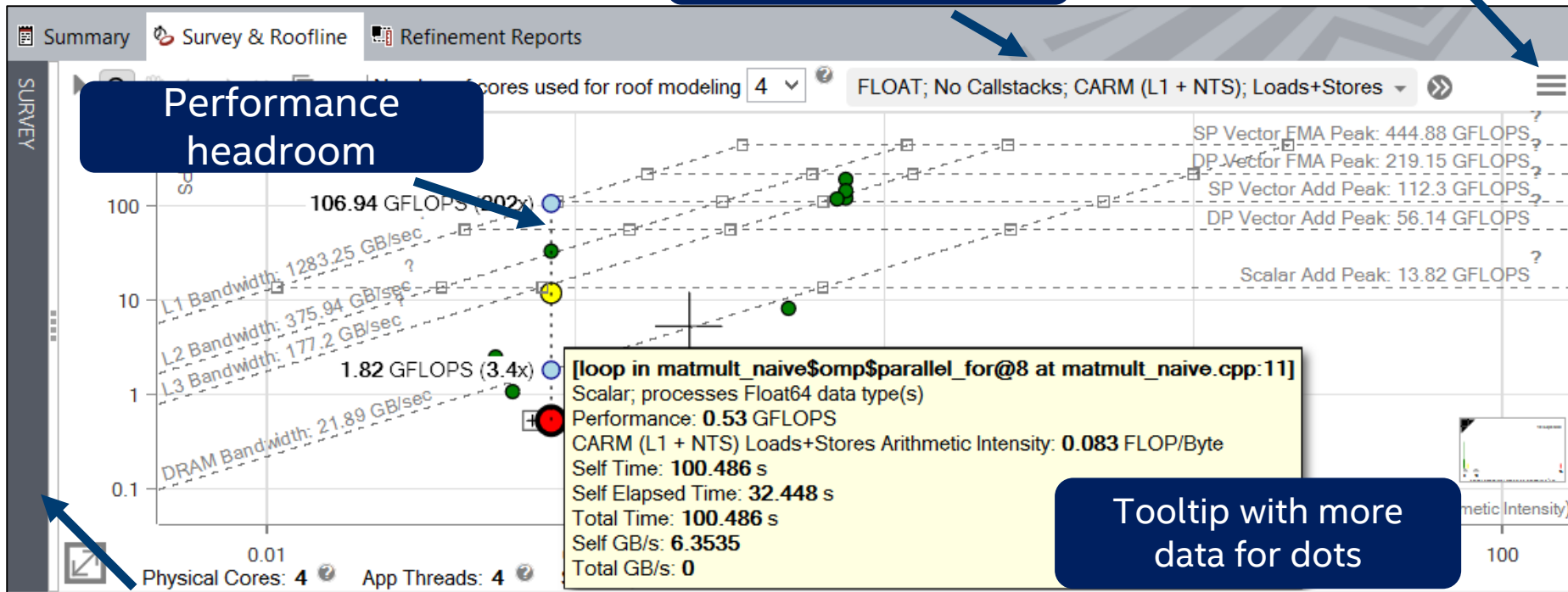


Roofline picture

Chart configuration

Roof configuration

Performance headroom



Tooltip with more data for dots

Switch to grid representation

Optimization Notice

Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Chart configuration

Aggregate data over calltree

Select which operations are counted

The screenshot shows a configuration dialog box with the following sections and options:

- Operations:** Radio buttons for FLOAT, INT, and INT+FLOAT.
- Callstacks:** A checkbox for With Callstacks.
- Callstacks:** Checkboxes for CARM (L1 + NTS), L2, L3, and DRAM.
- Memory Operation Type:** Radio buttons for Loads, Stores, and Loads+Stores.
- Buttons:** Apply and Cancel.

Callout arrows point from the text boxes to the following elements in the dialog:

- From "Aggregate data over calltree" to the "With Callstacks" checkbox.
- From "Select which operations are counted" to the "Operations" radio buttons.
- From "Select memory levels" to the "Callstacks" checkboxes (L2, L3, DRAM).
- From "Select only loads or stores" to the "Memory Operation Type" radio buttons.

Select only loads or stores

Select memory levels

Optimization Notice

Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Integrated Roofline Memory Traffic Data in Survey Grid

Review memory level and loads/stores distribution to see memory traffic for specific memory level

The screenshot displays the Intel Advisor 2019 interface. The top section shows a 'Survey & Roofline' view with a table of memory traffic data. A context menu is open over the table, showing options like 'Show L2 Memory Metrics' and 'Show All Memory Operations (Loads and Stores)'. The bottom section shows a detailed view of a specific loop, including 'Data Transfers and Bandwidth' and 'Statistics for FLOP And Data Transfers'.

Function Call Sites and Loops	Performance Issues	Self Time	Total Time	Type	Memory
[loop in stencil_2d at tiling.cpp:41]	3 Possible ...	1.030s	1.030s	Inside vecto.	Self L2 Loaded GB: 21.626, Self L2 Stored: 0.833
[loop in stencil_2d at tiling.cpp:41]	3 Possible in ...	0.200s	0.200s	Scalar	2.546, 0.138
[loop in main at tiling.cpp:61]	1 Data type...	0.060s	0.060s	Scalar	1.283, 0.071
[loop in main at tiling.cpp:101]	1 Data type...	0.049s	0.059s	Scalar	1.283, 0.071
[loop in main at tiling.cpp:82]	1 Data type...	0.040s	0.050s	Scalar	1.283, 0.071
[loop in main at tiling.cpp:120]	1 Data type...	0.030s	0.030s	Scalar	1.283, 0.071
[loop in main at tiling.cpp:101]		0.010s	0.010s	Vectorized (B...	0, 0
[loop in main at tiling.cpp:82]		0.010s	0.010s	Vectorized (B...	0, 0
f _tmainCRTStartup		0.000s	1.469s	Function	< 0.001, 0
f stencil_2d		0.000s	1.240s	Inlined Functio	< 0.001, 0
f main		0.000s	1.469s	Function	0.040, 0.040
[loop in stencil_2d at tiling.cpp:41]		0.000s	1.030s	Inside vectoriz	0.154, 0.006
[loop in stencil_2d at tiling.cpp:38]		0.000s	1.230s	Scalar	0, 0
[loop in stencil_2d at tiling.cpp:176]	1 Data type...	0.000s	1.240s	Scalar	< 0.001, 0
[loop in stencil_2d at tiling.cpp:41]		0.000s	0.200s	Scalar	0.020, 0.001
f printf		0.000s	0.040s	Function	0, 0
[loop in output_1 at output.c:1073]		0.000s	0.040s	Scalar	0, 0
f output_1		0.000s	0.040s	Function	0.002, 0

Memory Level	Per Loop	Per Instance	Per Iteration	Float AI
L1 Gb/s	37.84478	0.00002	1.88000e-07	0.255319
L2 Gb/s	22.45959	0.00001	1.11572e-07	0.430217
L3 Gb/s	22.45991	0.00001	1.11573e-07	0.430211
DRAM Gb/s	1.13457	7.21340e-07	5.63616e-09	8.51644

Statistic	Per loop	Per Iteration	Self	Total
GFLOP	9.66250	4.80000e-08	6.14325e-06	
GFLOPS	9.38121	4.66027e-08	5.96441e-06	
AI	0.25532	1.26834e-09	1.62328e-07	
Mask Utilization				
L1 Gb/s	37.84478	1.88000e-07	0.00002	
L2 Gb/s			36.74306	
Elapsed Time		1.02998s		

Optimization Notice

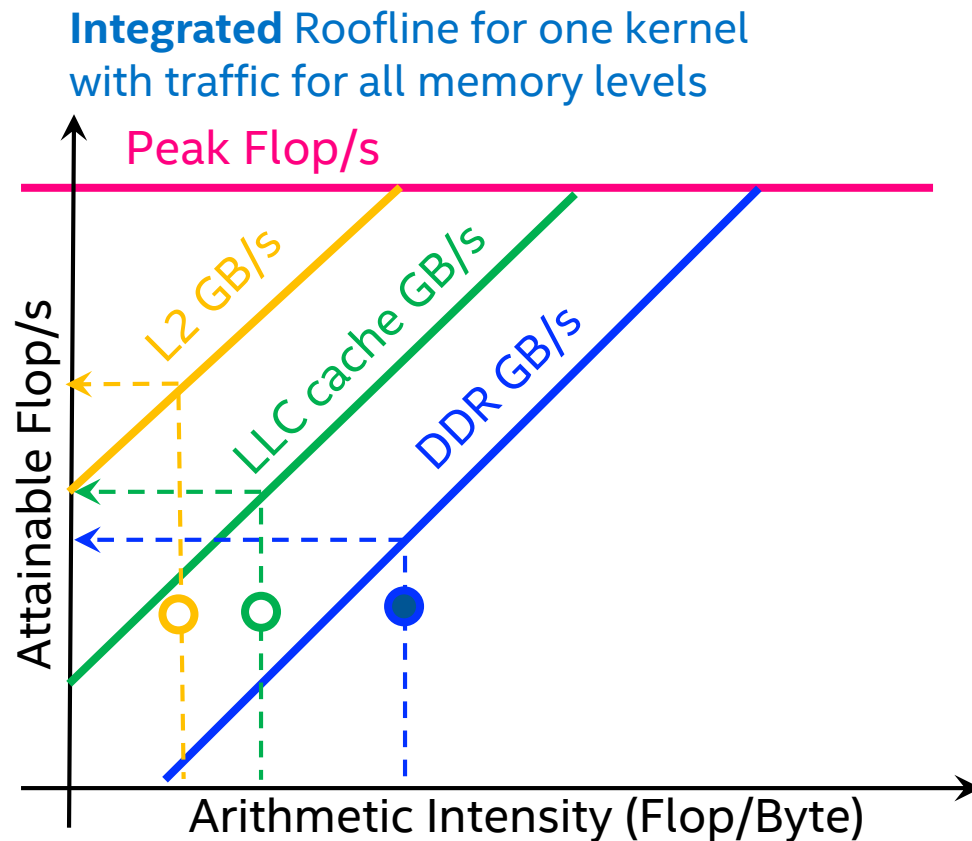
Copyright © 2017, Intel Corporation. All rights reserved.
 *Other names and brands may be claimed as the property of others.



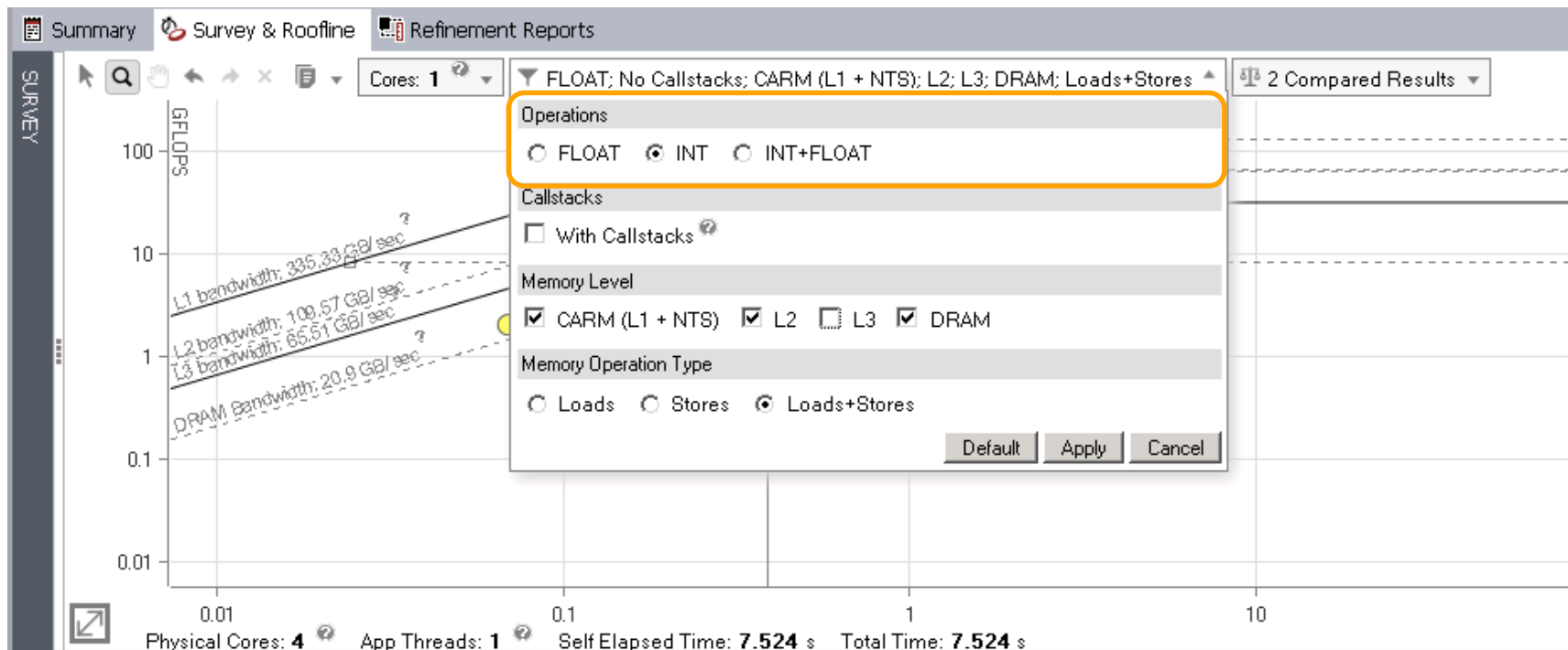
4 Integrated Roofline. What is my current limit?

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

The screenshot displays the Intel VTune Performance Analyzer interface. The top section is the Survey Grid, which lists various function call sites and loops. A blue arrow points to a gear icon in the 'Compute Performance' column header, which opens a settings menu with options: 'Show Floating-Point Operations', 'Show Integer Operations', and 'Show Mixed Compute Operations'. The 'Show Integer Operations' option is selected.

Function Call Sites and Loops	Performance Issues	Self Time	Total Time	Type	Compute Performance		Settings	
					Self GINTOPS	Self GINTOP	Self	Total
[loop in EXACT_RHS at exact_rhs.f:139]	3 Unoptimize...	0.000s	0.000s	Inside vectorized	103.707	0.002	<input checked="" type="checkbox"/>	
[loop in Z_SOLVE at z_solve.f:332]		0.040s	3.699s	Scalar	14.382	0.575	<input type="checkbox"/>	
[loop in X_SOLVE at x_solve.f:331]		0.050s	3.369s	Scalar	11.499	0.575	<input type="checkbox"/>	
[loop in Y_SOLVE at y_solve.f:326]		0.070s	3.690s	Scalar	8.221	0.575	<input type="checkbox"/>	0.070s 3.690s
[loop in COMPUTE_RHS at rhs.f:273]		0.010s	0.821s	Scalar	2.797	0.028	<input type="checkbox"/>	0.010s 0.821s
[loop in X_SOLVE at x_solve.f:384]		0.250s	0.250s	Scalar	1.169	0.292	<input type="checkbox"/>	0.250s 0.250s
[loop in Z_SOLVE at z_solve.f:396]		0.280s	0.280s	Scalar	1.044	0.292	<input type="checkbox"/>	0.280s 0.280s
[loop in COMPUTE_RHS at rhs.f:134]		0.090s	0.090s	Vectorized (Body)	1.000	0.090	<input type="checkbox"/>	0.090s 0.090s

The bottom section is the Loop Analytics section, showing details for the selected loop: 'Loop in EXACT_RHS at exact_rhs.f:139'. It indicates the loop is 'Inside vectorized' with a total time of '< 0.001s'. Below this, it shows 'AVX; AVX2 < 0.001s' for the instruction set self time. A bar chart shows the instruction mix: Memory (41%), Compute (17%), Mixed (15%), and Other (27%). To the right, the 'Statistics for INTOP' table is displayed, with a blue arrow pointing to the 'Type' selector dropdown menu. The table shows statistics for various integer operations.

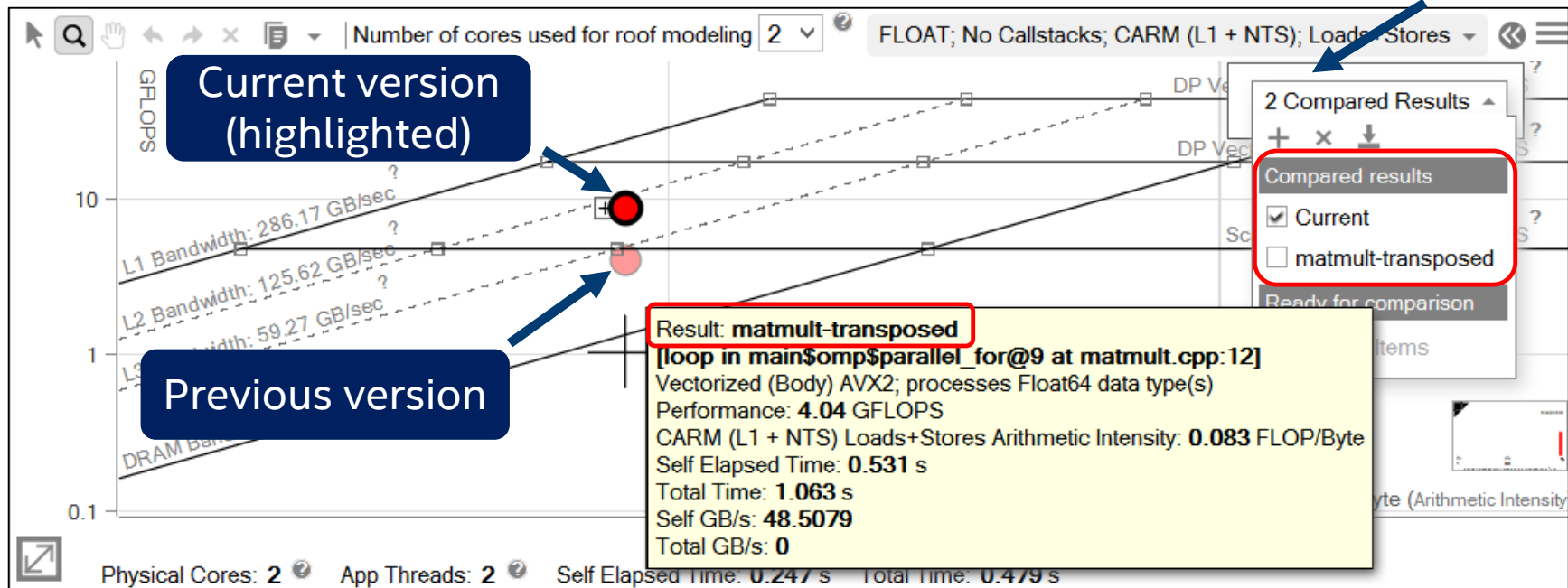
And Data Transf	Type	Per loop	Per Iteration	Self Per Instance	Total
FLOP					
GINTOP	INTOP	0.00172	7.00000e-09	4.48000e-07	
GINTOPS	INT+FLOAT	03.70670	0.00042	0.02698	
INTAI	All Operations	0.03977	1.61667e-07	0.00001	
Mask Utilization					
L1 Gb		0.04330	1.76000e-07	0.00001	
L1 Gb/s			2607.48278		
Elapsed Time			0.00002s		

Optimization Notice



Compare results

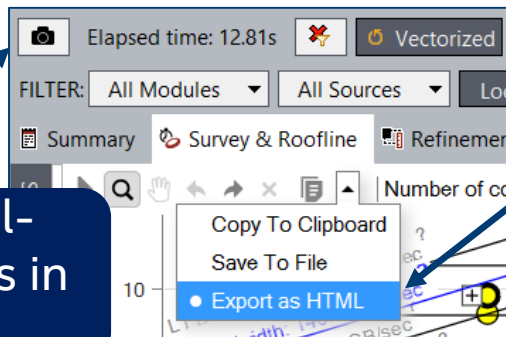
Loaded results for two versions



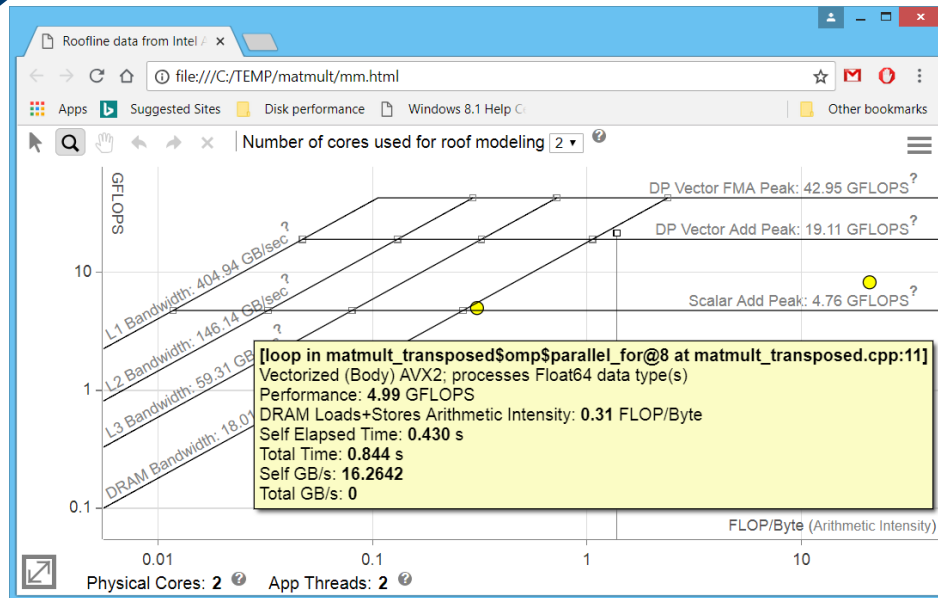
Easy to check optimization progress

Share with others

Snapshot (full-featured, opens in Advisor)



Standalone *interactive* HTML
(limited functionality)
Share roofline by email! - with colleagues or your manager



Optimization Notice

Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Use the rest of the Advisor

Run Roofline

Collect [Icons]

Enable Roofline with Callstacks

1. Survey Target

Collect [Icons]

Mark Loops for Deeper Analysis

Select checkboxes in the **Survey & Roofline** tab to mark loops for other Advisor analyses.

2 loops are marked

1.1 Find Trip Counts and FLOP

Collect [Icons]

✓ Trip Counts

✓ FLOP

2.1 Check Memory Access Patterns

Collect [Icons]

2.2 Check Dependencies

Collect [Icons]

Summary Survey & Roofline Refinement Reports

Function Call Sites and Loops Advanced

Function Call Sites and Loops	Advanced
[loop in matmult_naive\$omp\$paral	
[loop in matmult_transposed\$omp\$	Unrolled by 4
[loop in matmult_blocked\$omp\$pa	Unrolled by 8
[loop in matmult_blocked\$omp\$pa	
[loop in matmult_naive\$omp\$paral	
[loop in matmult_naive\$omp\$paral	
[loop in matmult_transposed\$omp\$	

Source Top Down Code Analytics Assembly Recommendations Why No Vectorization?

Module: matmult.exe!0x1400016c9

Address	Line	Assembly	Total Time	%	Self Time	%	Traits
0x1400016dd	20	vmovupd ymm4, ymmword ptr [r8+rbp*8+0x60]	0.049s		0.049s		
0x1400016e4	20	vmovupd ymm5, ymmword ptr [r8+rbp*8+0x80]	0.020s		0.020s		
0x1400016ee	20	vfmadd213pd ymm1, ymm0, ymmword ptr [r11+rbp*8]	0.020s		0.020s		FMA
0x1400016f4	20	vfmadd213pd ymm2, ymm0, ymmword ptr [r11+rbp*8+0x20]	0.030s		0.030s		FMA
0x1400016fb	20	vfmadd213pd ymm3, ymm0, ymmword ptr [r11+rbp*8+0x40]	0.019s		0.019s		FMA
0x140001702	20	vfmadd213pd ymm4, ymm0, ymmword ptr [r11+rbp*8+0x60]	0.017s		0.017s		FMA
		vmovupd ymmword ptr [r11+rbp*8+0x80]	0.010s		0.010s		FMA

Scalability Graph: ScFLOP vs FLOP/Byte (Arithmetic Intensity)

- L1 Bandwidth: 404.94 GB/s
- L2 Bandwidth: 146.14 GB/s
- L3 Bandwidth: 59.31 GB/s
- DRAM Bandwidth: 18.01 GB/s
- DP Vector FMA Peak: 42.95 GFLOPS
- DP Vector Add Peak: 19.11 GFLOPS
- Scalar Add Peak: 4.76 GFLOPS

Physical Cores: 2 App Threads: 2

See additional info in grid

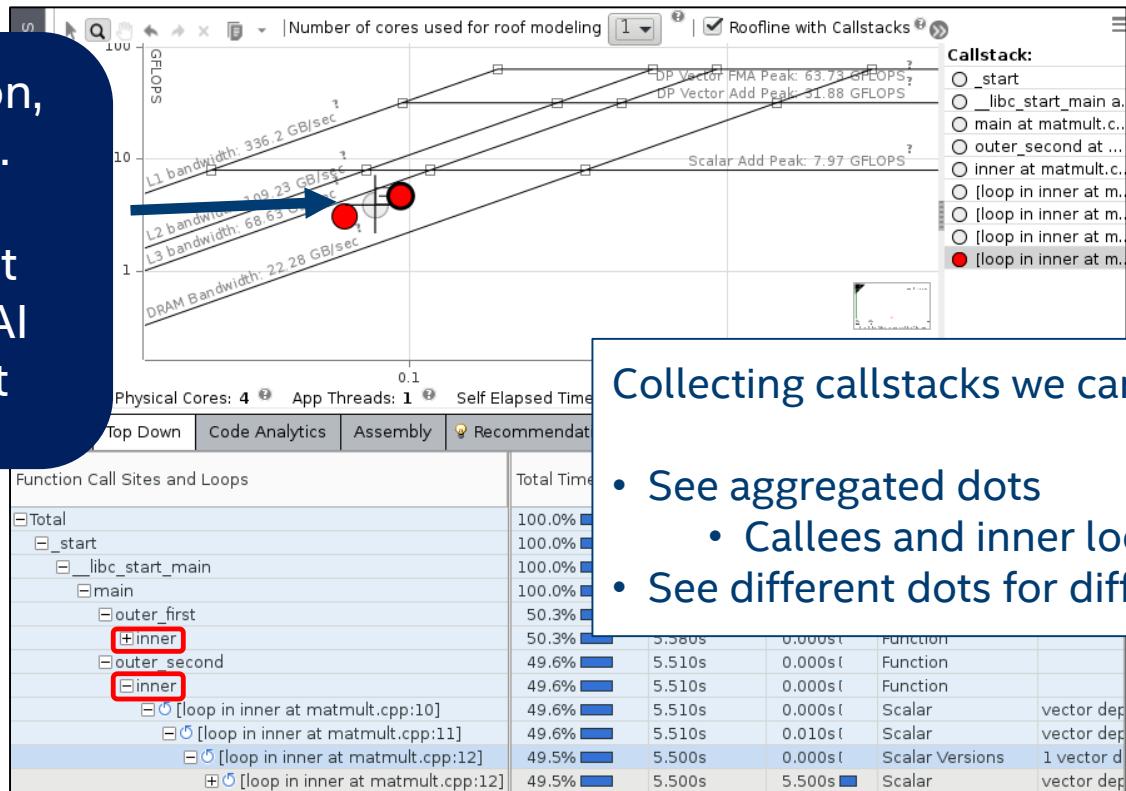
Select loops for deeper analysis

Examine source or assembly

A few words about callstacks

Same function,
same loop...

But different
FLOPS and AI
on different
callpaths!

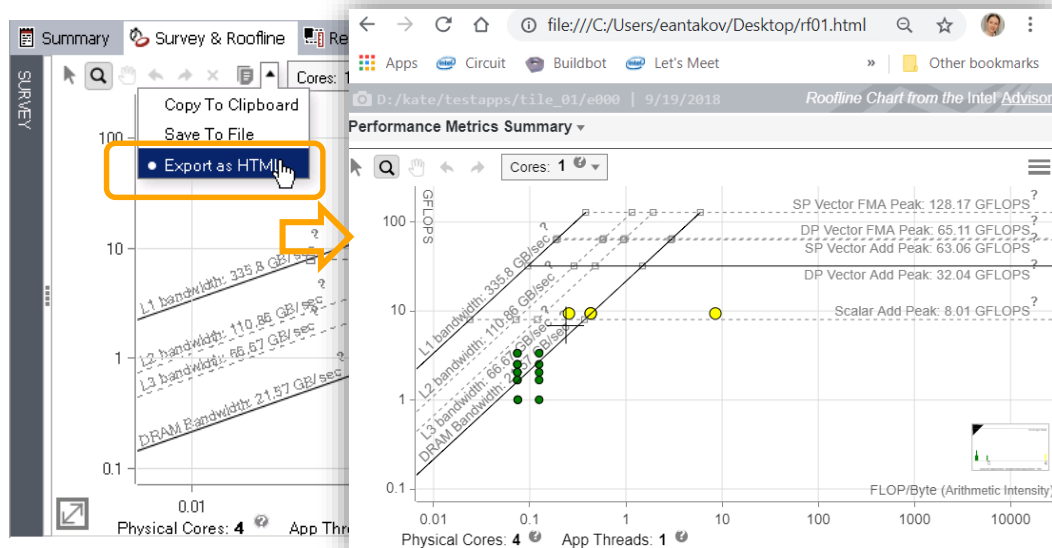


Collecting callstacks we can

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

Exporting Integer and Integrated Roofline as HTML

GUI: Use Export as HTML button



Command line:

```
advixe-cl --report roofline  
-data-type=float  
-memory-level=L2  
-memory-operation-type=load  
-project-dir /path/to/project/dir
```

Possible

data types: float, int, mixed
memory levels: L1, L2, L3, DRAM
memory operation types: load, store, all

- Export Roofline from command line does not need GUI sub-system on clusters
- Useful for rooflines quick exchange

Optimization Notice

Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



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

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

HANDS-ON EXERCISE

Activities

- Activity 0: Building Stencil
- Activity 1: Doing Survey
- Activity 2: Dealing with **data type conversions**
- Activity 3: Checking for dependencies
- Activity 4: Adding **threading and** trying to enable **vectorization**
- Activity 5: Checking Memory Access Patterns
- Activity 6: Making **unit stride** explicit
- Activity 7: Doing Roofline analysis
- Activity 8: Splitting task **to tiles**
- Activity 9: Enabling **AVX512**
- Activity 10: Comparing roofline charts

STENCIL

STENCIL CODE EXAMPLE

Consider solving differential equation with finite-difference method on 3-dimensional grid

Example: calculating Laplace operator of some field

```
float * in = (float *)
    malloc(DIM*DIM*DIM * sizeof(float));
float * out = (float *)
    malloc(DIM*DIM*DIM * sizeof(float));

uint64_t iStride = 1;
uint64_t jStride = DIM;
uint64_t kStride = DIM * DIM;
```

```
for(k=1;k<DIM-1;k++){
    for(j=1;j<DIM-1;j++){
        for(i=1;i<DIM-1;i++){
            int ijk = i*iStride + j*jStride + k*kStride;
            out[ijk] = -6.0 * in[ijk
                ]
                + in[ijk-iStride]
                + in[ijk+iStride]
                + in[ijk-jStride]
                + in[ijk+jStride]
                + in[ijk-kStride]
                + in[ijk+kStride];
        }
    }
}
```

We encourage you to try the following steps on your own code, using the slides as a guide

Activity 0: Building STENCIL

Build & Run

Setup environment:

```
$ source /soft/compilers/intel-2019/compilers_and_libraries/linux/bin/compilervars.sh intel64
```

Copy and unpack stencil sources:

```
$ cp /projects/ATPESC19_Instructors/advisor/advisor_lab.tar.gz ~ && cd ~ && tar xzf advisor_lab.tar.gz
```

Go to working directory

```
$ cd ~/advisor_lab/src && git checkout ver0
```

Build application

```
$ make
```

Run application

```
$ ./stencil
```


Activity 0. Screenshot

```
[day1@clx-2 src]$ source /opt/intel/compilers_and_libraries/linux/bin/compilervars.sh intel64
[day1@clx-2 src]$ cd ~/advisor_lab/src && git checkout ver0
HEAD is now at 92efb0f... Initial commit
[day1@clx-2 src]$ make
icc -Ofast -qopenmp -no-ipo -fno-inline-functions -g main.c bench_stencil.c -o stencil
[day1@clx-2 src]$ ./stencil
      Naive: Dim= 512, nIterations= 10, Time= 4.102s, Useful GB/s= 5.297
[day1@clx-2 src]$ █
```

Activity 1: Doing Survey

Launch Advisor

Purpose: Run Survey analysis in Advisor to get the baseline version

Setup environment:

```
$ source /soft/compilers/intel-2019/advisor_2019/advixe_vars.sh
```

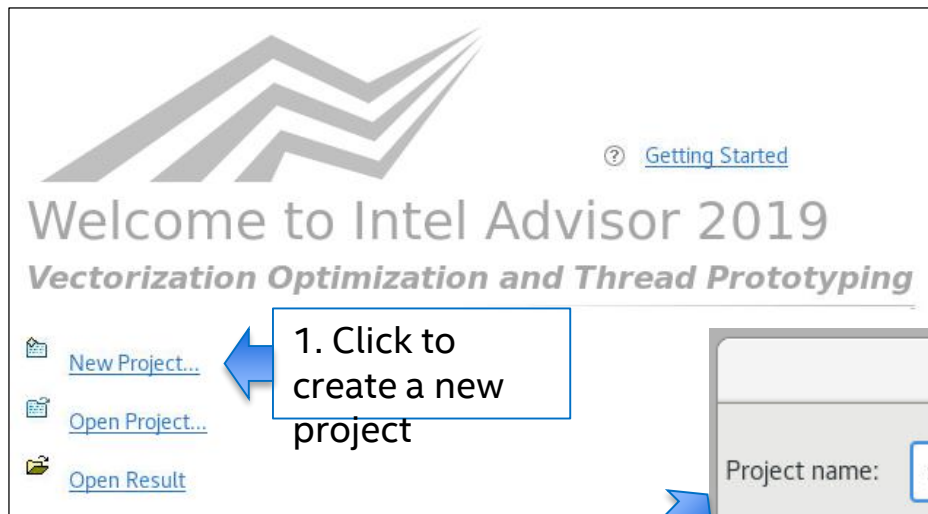
```
$ export ADVIXE_EXPERIMENTAL=int_roofline,roofline_guidance
```

Launch Advisor GUI:

```
$ advixe-gui &
```

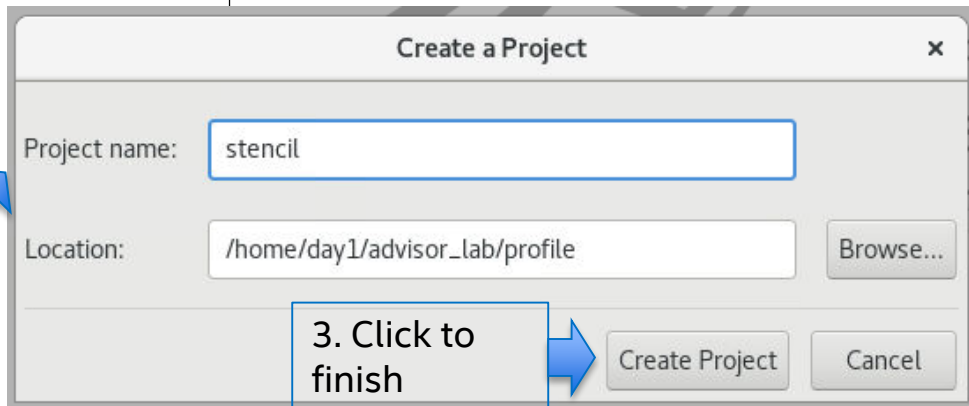
```
[day1@clx-2 src]$ source /opt/intel/advisor_2019/advixe-vars.sh
Copyright (C) 2009-2019 Intel Corporation. All rights reserved.
Intel(R) Advisor 2019 (build 591490)
[day1@clx-2 src]$ export ADVIXE_EXPERIMENTAL=int_roofline,roofline_guidance
[day1@clx-2 src]$ advixe-gui &
[1] 5336
```

Create Advisor Project



1. Click to create a new project

2. Type name of the project

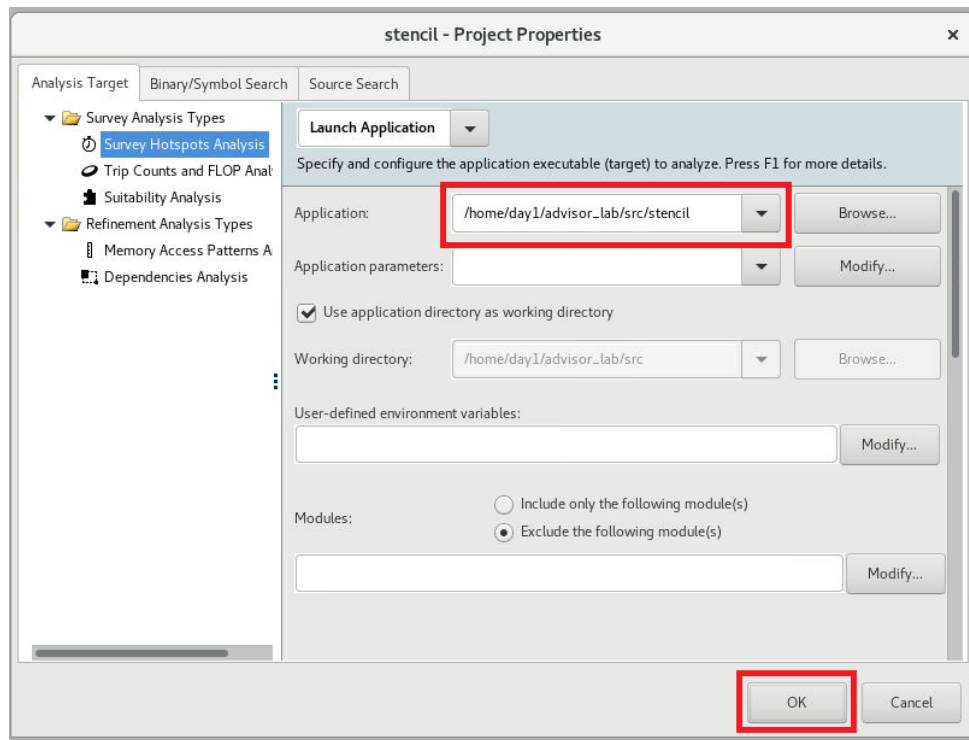


3. Click to finish creation

Set UP Project

Set the application to launch:
`~/advisor_lab/src/stencil`

Press OK button



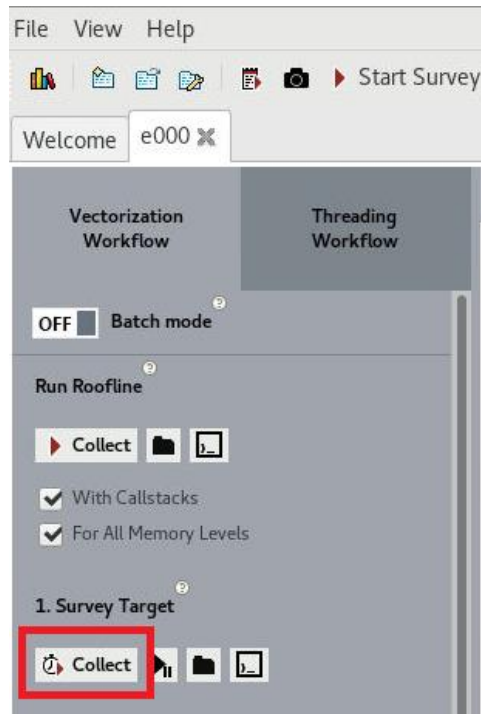
Optimization Notice

Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Start Survey Analysis

Press “Collect” button in “1. Survey Target” section



Optimization Notice

Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Activity 1. Screenshot

The screenshot displays the Intel Advisor 2019 interface. The top menu bar includes 'File', 'View', and 'Help'. Below the menu bar, there are icons for 'Start Survey Analysis' and a 'Welcome' message with 'e000'.

The main window is divided into several sections:

- Vectorization Workflow:** Includes a 'Batch mode' toggle (OFF) and a 'Run Routine' section with 'Collect' and 'With Callstacks' options.
- Threading Workflow:** Includes a 'Mark Loops for Deeper Analysis' section with checkboxes for 'Trip Counts', 'FLOP', and 'Analyze all loops'.
- Performance Analysis:** A table titled 'ROUNDOFFLINE' showing performance metrics for various function call sites and loops.
- Source Window:** Displays C++ code for a stencil computation, with a highlighted loop at line 25.

The 'ROUNDOFFLINE' table shows the following data:

Function Call Sites and Loops	Performance Issues	CPU Time	Type	Why No Vectorization?	Vectorized Loops	Instruction Set Analysis
		Total Time	Self Time		Vecto... Efficiency	Gain E... VL (V... Traits
[+] [loop in bench_stencil at bench_stencil.c:25]	0 Assumed dep...	5.610s	5.610s	Scalar	vector dependence pre...	Type Conversions
[+] [loop in main at main.c:20]	1 Misaligned b...	0.420s	0.420s	Vectorized (Body)	SSE	NT-stores
[+] [loop in bench_stencil at bench_stencil.c:24]	1 Assumed dep...	5.620s	0.010s	Scalar	vector dependence pre...	
[+] f_start		6.040s	0.000s	Function		
[+] f_main		6.040s	0.000s	Function		NT-stores
[+] [loop in bench_stencil at bench_stencil.c:23]	1 Assumed dep...	5.620s	0.000s	Function		
[+] [loop in bench_stencil at bench_stencil.c:22]	1 Assumed dep...	5.620s	0.000s	Scalar	vector dependence pre...	Float6

The 'Source' window shows the following C++ code:

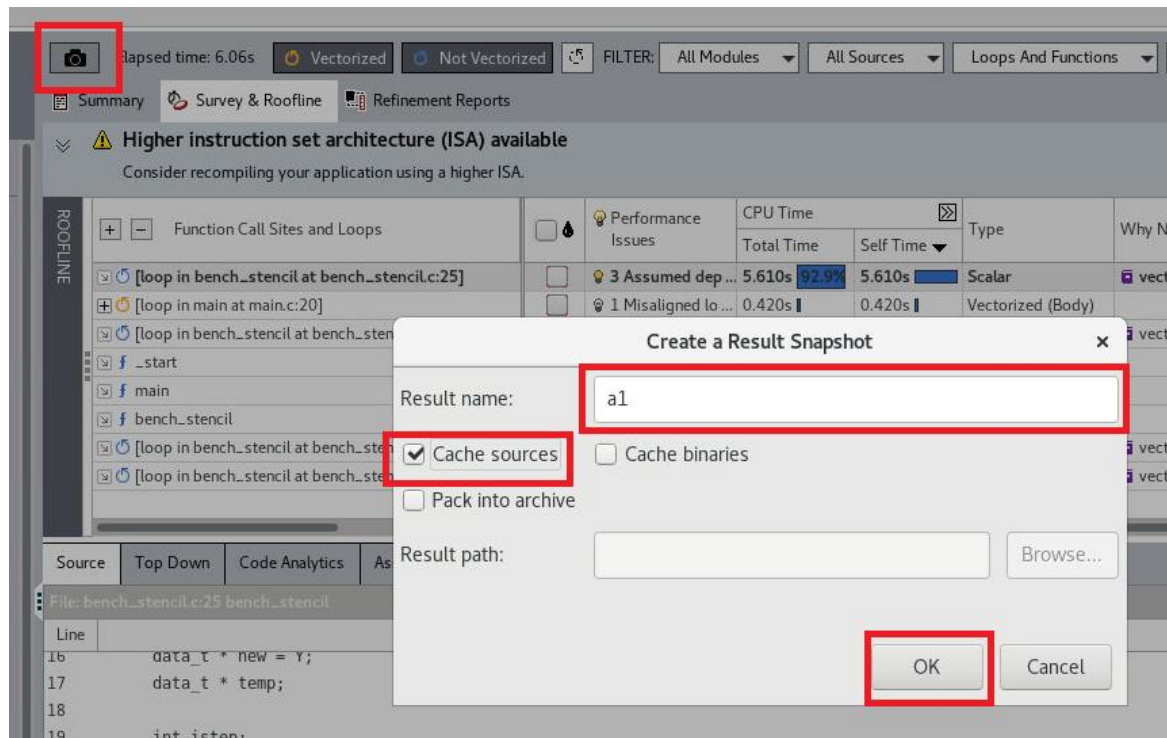
```
Line Source Total Time % Loop/Function Time % Traits
16 data_t * new = r;
17 data_t * temp;
18
19 int istep;
20
21 StartTime = omp_get_wtime();
22 for (istep = 0; istep < NSTEP; istep++) {
23     for (k = 1; k < dim + 1; k++) {
24         for (j = 1; j < dim + 1; j++) {
25             for (i = 1; i < dim + 1; i++) {
                [loop in bench_stencil at bench_stencil.c:25]
                Scalar loop. Not vectorized: vector dependence prevents vectorization
                No loop transformations applied
            }
        }
    }
}
26 int ijk = i * iStride + j * jStride + k * kStride;
27 new[ijk] = -6.0 * old[ijk]
28           + old[ijk - iStride]
29           + old[ijk + iStride]
30           + old[ijk - jStride]
```

Optimization Notice

Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Create a snapshot



Activity 2: Dealing with data type conversions

LOOK AT THE RECOMMENDATIONS

Elapsed time: 6.06s Vectorized Not Vectorized FILTER: All Modules All Sources Loops And Functions All Threads Customize View

Summary Survey & Roofline Refinement Reports INTEL ADVISOR 2019

Higher instruction set architecture (ISA) available
Consider recompiling your application using a higher ISA.

Function Call Sites and Loops	Performance Issues	CPU Time		Type	Why No Vectorization?	Vectorized Loops				Instruction Set Analysis		
		Total Time	Self Time			Vectorized	Efficiency	Gain E...	VL (V...	Traits	Data	
[loop in bench_stencil at bench_stencil.c:25]	3 Assumed dep...	5.610s	02.9%	5.610s	Scalar	vector dependence pre...					Type Conversions	Float2
[loop in main at main.c:20]	1 Misaligned to...	0.420s		0.420s	Vectorized (Body)		SSE	~100%	4.67x	4	NT-stores	Float2
[loop in bench_stencil at bench_stencil.c:24]	1 Assumed dep...	5.620s	03.0%	0.010s	Scalar	vector dependence pre...						
f _start		6.040s		0.000s	Function							
f main		6.040s		0.000s	Function						NT-stores	Float3
f bench_stencil		5.620s	03.0%	0.000s	Function							
[loop in bench_stencil at bench_stencil.c:23]	1 Assumed dep...	5.620s	03.0%	0.000s	Scalar	vector dependence pre...						
[loop in bench_stencil at bench_stencil.c:22]	1 Assumed dep...	5.620s	03.0%	0.000s	Scalar	vector dependence pre...						Float6

Source Top Down Code Analytics Assembly Recommendations Why No Vectorization?

Although static analysis presumes the loop may benefit from [FMA](#) instructions available with the AVX2 or higher ISA, no FMA instructions executed for this loop. To fix: Use the following compiler options:

- xCORE-AVX2 to compile for machines with and without AVX2 support
- axCORE-AVX2 to compile for machines with AVX2 support only
- xCOMMON-AVX512 to compile for machines with AVX-512 support only
- axCOMMON-AVX512 to compile for machines with and without AVX-512 support

Note: the compiler options may vary depending on the CPU microarchitecture.

Assumed dependency present
Confirm dependency is real

Potential underutilization of FMA instructions
[Target the higher ISA](#)

Data type conversions present
Use the smallest data type

Data type conversions present
There are multiple data types within loops. Utilize hardware vectorization support more effectively by avoiding data type conversion.

Use the smallest data type
The source loop contains data types of different widths. To fix: Use the smallest data type that gives the needed precision to use the entire vector register width.

Optimization Notice

Activity 2

Purpose: Dealing with data type conversions

Build a version with fixed conversions

```
$ git checkout ver1
```

```
$ make
```

Re-run Survey analysis

Create a snapshot

Compare with previous activity

Activity 2. Screenshots

Elapsed time: 6.60s

Vectorized Not Vectorized FILTER: All Modules All Sources

Summary Survey & Roofline Refinement Reports

Vectorization Advisor

Vectorization Advisor is a vectorization analysis toolset that lets you identify loops that will benefit most from vectorization and characterize your memory vs. vectorization bottlenecks with Advisor Roofline model automatic

Program metrics

Elapsed Time **6.60s**

Vector Instruction Set SSE

Performance characteristics

Metrics	Total
Total CPU time	6.54s
Time in 1 vectorized loop	0.44s
Time in scalar code	6.10s

Vectorization Gain/Efficiency

Vectorized Loops Gain/Efficiency	4.67x	100%
Program Approximate Gain	1.25x	

Elapsed time: 4.11s

Vectorized Not Vectorized FILTER: All Modules All Sources

Summary Survey & Roofline Refinement Reports

Vectorization Advisor

Vectorization Advisor is a vectorization analysis toolset that lets you identify loops that will benefit most from vectorization and characterize your memory vs. vectorization bottlenecks with Advisor Roofline model automatic

Program metrics

Elapsed Time **4.11s**

Vector Instruction Set SSE

Performance characteristics

Metrics	Total
Total CPU time	4.09s
Time in 1 vectorized loop	0.43s
Time in scalar code	3.66s

Vectorization Gain/Efficiency

Vectorized Loops Gain/Efficiency	4.67x	100%
Program Approximate Gain	1.39x	

Optimization Notice

Activity 3: Doing roofline analysis

Activity 3. Collect data to GET ROOFLINE CHART

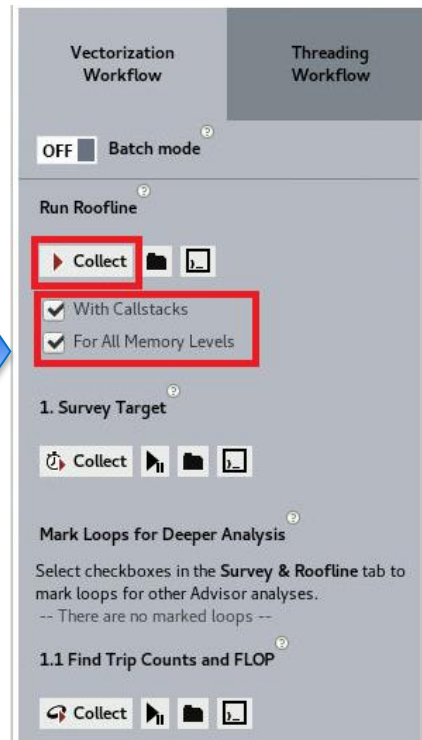
Purpose: Characterize the application using roofline model

Select “With Callstacks” and “For all memory levels”

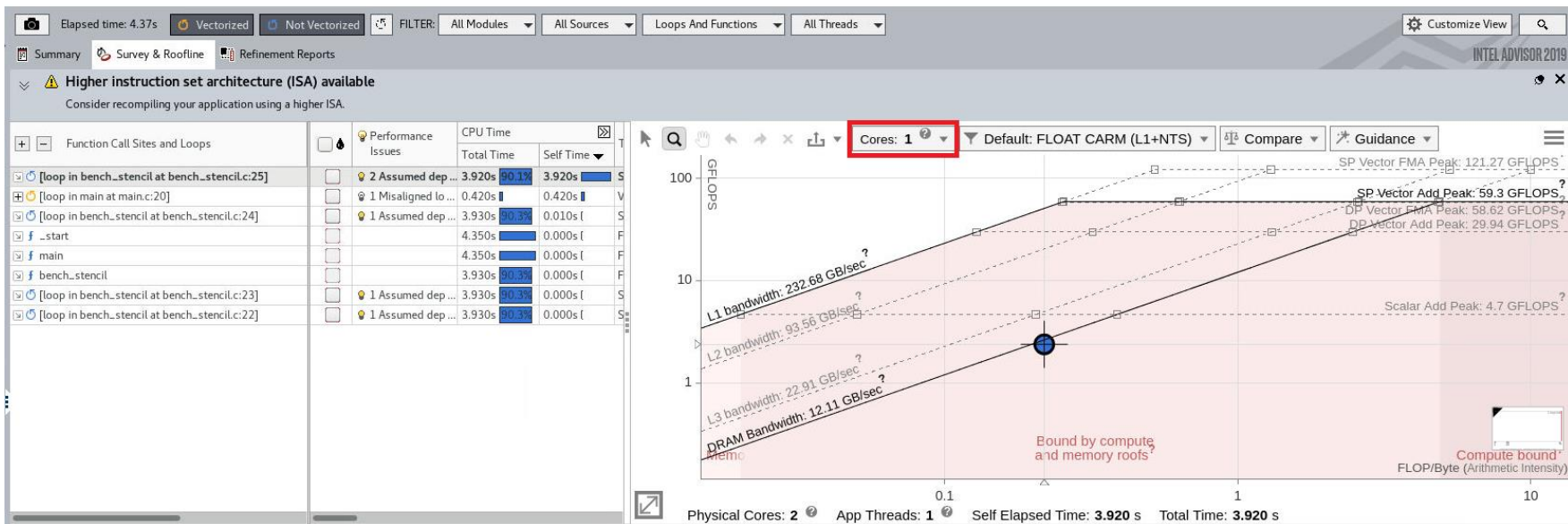
Press “Collect” button in “Run Roofline” section
~ 4 minutes

Create a snapshot

For Integrated Roofline (NEW!) →



Activity 3. Screenshot

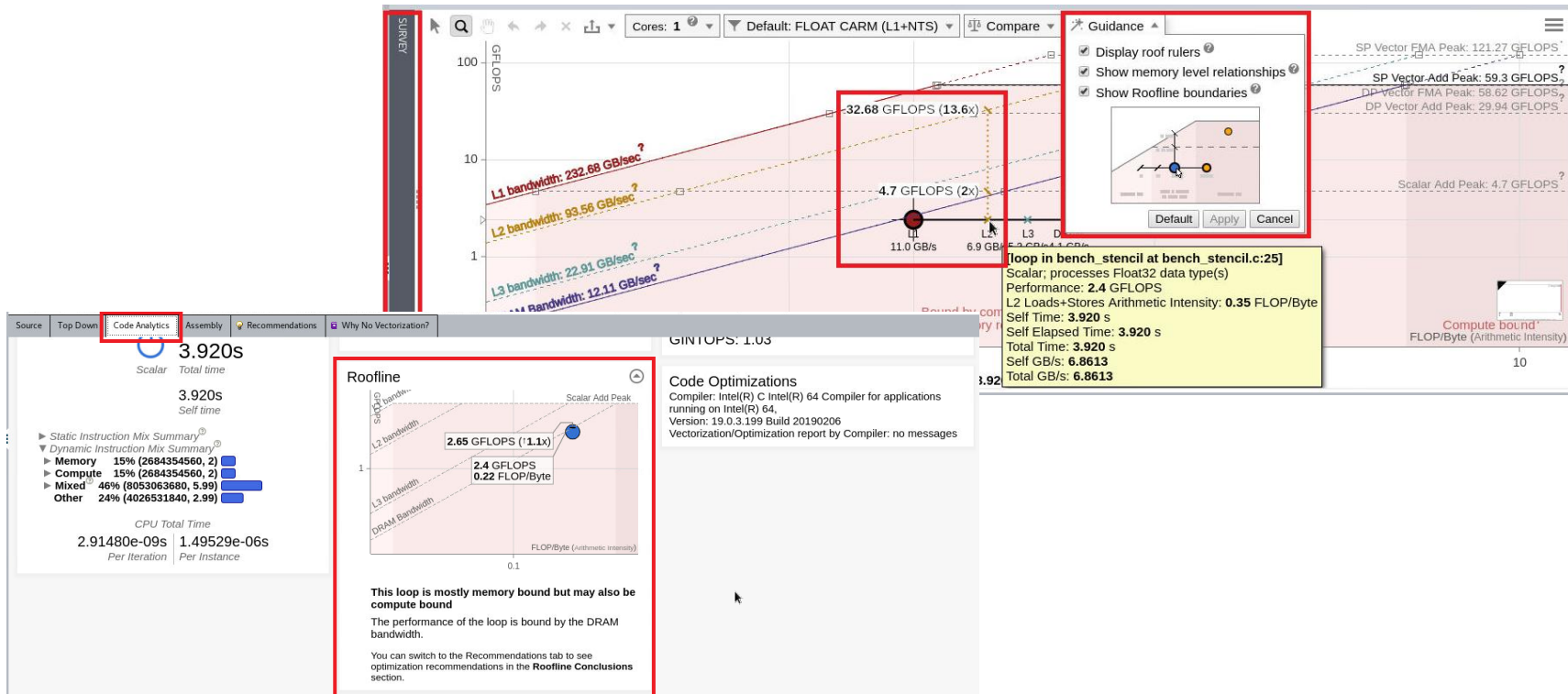


Optimization Notice

Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Activity 3. Screenshot



Optimization Notice

Activity 4: Checking for dependencies

Activity 4. Collect data to GET Dependencies

Purpose: Find loop-carried dependencies

Select “loop in bench_stencil at bench_stencil.c:23”

Press “Collect” button in “2.2 Check Dependencies” section
~ 1 minute

Create a snapshot

Vectorization Workflow | Threading Workflow

Elapsed time: 4.37s | Vectorized | Not Vectorized

Summary | Survey & Roofline | Refinement Reports

Higher instruction set architecture (ISA) available
Consider recompiling your application using a higher ISA.

OFF Batch mode

Run Roofline

Collect

With Callstacks
For All Memory Levels

1. Survey Target

Collect

Mark Loops for Deeper Analysis
Select checkboxes in the Survey & Roofline tab to mark loops for other Advisor analyses.
1 loop is marked

1.1 Find Trip Counts and FLOP

Collect

Trip Counts
FLOP

2.1 Check Memory Access Patterns

Collect

2.2 Check Dependencies

Collect

Roofline

Function Call Sites and Loops	
[loop in bench_stencil at bench_stencil.c:25]	<input type="checkbox"/>
[loop in main at main.c:20]	<input type="checkbox"/>
[loop in bench_stencil at bench_stencil.c:24]	<input type="checkbox"/>
f _start	<input type="checkbox"/>
f main	<input type="checkbox"/>
f bench_stencil	<input type="checkbox"/>
[loop in bench_stencil at bench_stencil.c:23]	<input checked="" type="checkbox"/>
[loop in bench_stencil at bench_stencil.c:22]	<input type="checkbox"/>

Source | Top Down | Code Analytics | Assembly | Recommendation

Loop in bench_stencil at bench_stencil.c:23

3.930s
Scalar Total time

0s
Self time

Static Instruction Mix Summary

Optimization Notice

Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

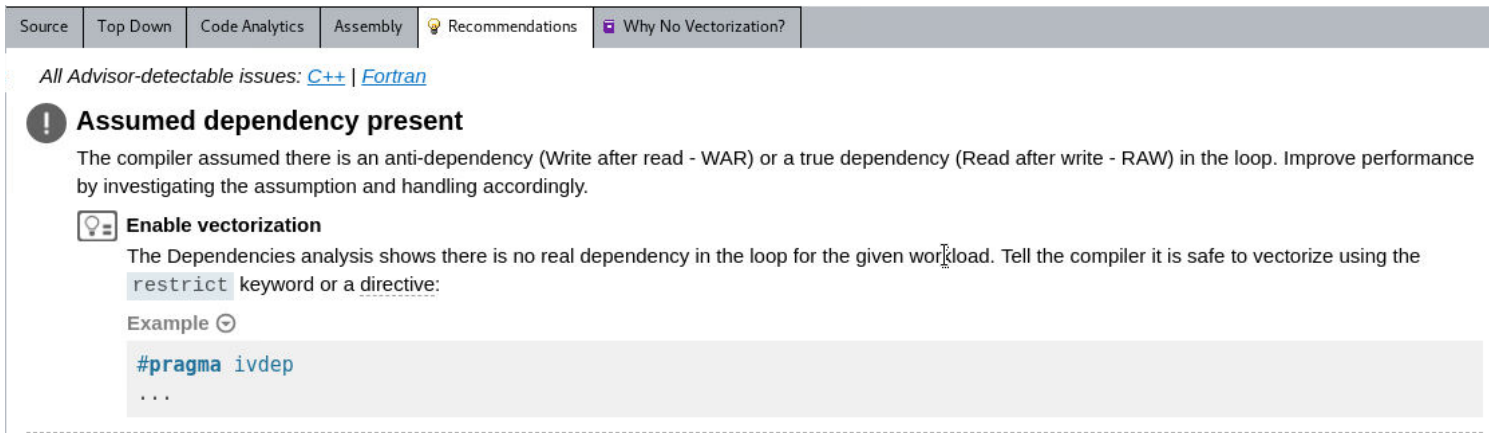


Activity 4. Screenshot



The screenshot shows the Intel Advisor 2017 interface. At the top, there are filters for 'Elapsed time: 4.37s', 'Vectorized', 'Not Vectorized', and 'FILTER: All Modules All Sources'. Below this is a navigation bar with 'Summary', 'Survey & Roofline', and 'Refinement Reports' (highlighted with a red box). The main table displays analysis results for a loop. The 'Refinement Reports' column contains a green checkmark and the text 'No Dependencies Found', which is also highlighted with a red box. Other columns include 'Site Location', 'Loop-Carried Dependencies', 'Strides Distribution', 'Access Pattern', 'Footprint Estimate' (with sub-columns for 'Max. Per-Instruction Addr. Range', 'First Instance Site Footprint', and 'Simulated Memory Footprint'), and 'Site Name'.

Site Location	Loop-Carried Dependencies	Strides Distribution	Access Pattern	Footprint Estimate			Site Name
				Max. Per-Instruction Addr. Range	First Instance Site Footprint	Simulated Memory Footprint	
▶ [loop in bench_stencil at bench_stencil.c:	✔ No Dependencies Found	No Information Available	No Information Available	No Information Available	No Information Available	No Information Available	loop_site_31



The screenshot shows the 'Why No Vectorization?' panel in Intel Advisor 2017. It lists 'All Advisor-detectable issues: C++ | Fortran'. The first issue is 'Assumed dependency present', which states: 'The compiler assumed there is an anti-dependency (Write after read - WAR) or a true dependency (Read after write - RAW) in the loop. Improve performance by investigating the assumption and handling accordingly.' Below this, there is a section for 'Enable vectorization' with the text: 'The Dependencies analysis shows there is no real dependency in the loop for the given workload. Tell the compiler it is safe to vectorize using the restrict keyword or a directive:'. An 'Example' section shows the code: '#pragma ivdep' followed by three dots.

Source | Top Down | Code Analytics | Assembly | Recommendations | Why No Vectorization?

All Advisor-detectable issues: [C++](#) | [Fortran](#)

! Assumed dependency present
The compiler assumed there is an anti-dependency (Write after read - WAR) or a true dependency (Read after write - RAW) in the loop. Improve performance by investigating the assumption and handling accordingly.

💡 Enable vectorization
The Dependencies analysis shows there is no real dependency in the loop for the given workload. Tell the compiler it is safe to vectorize using the `restrict` keyword or a directive:

Example ⌵

```
#pragma ivdep
...
```

Optimization Notice

Activity 5: Adding threading and enabling vectorization

Activity 5

Purpose: Add threading and enable vectorization

Build a version with threading and vectorization

```
$ git checkout ver3
```

```
$ make
```

Re-run Roofline analysis

Create a snapshot

Compare with previous activity

Activity 5. Screenshots

Elapsed time: 2.74s Vectorized Not Vectorized FILTER: All Modules All Sources

Summary Survey & Roofline Refinement Reports

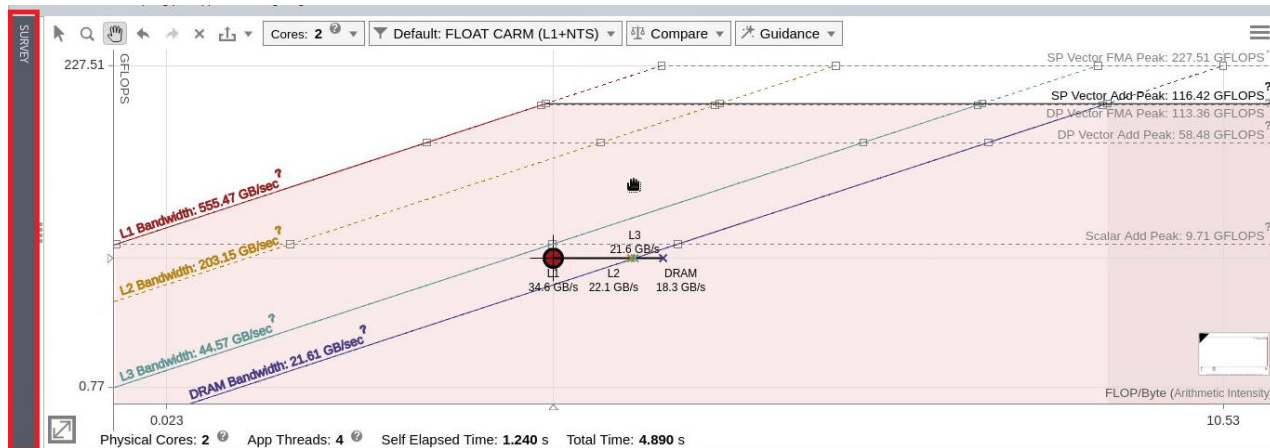
Vectorization Advisor

Vectorization Advisor is a vectorization analysis toolset that lets you identify loops that will benefit most from vector parallelism, discover performan vectorization and characterize your memory vs. vectorization bottlenecks with Advisor Roofline model automation.

Program metrics

Elapsed Time **2.74s** Number of CPU Threads 4

Vector Instruction Set SSE



Optimization Notice

Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Activity 6: Checking memory access patterns

Types OF MEMORY Access patterns

Unit-Stride access

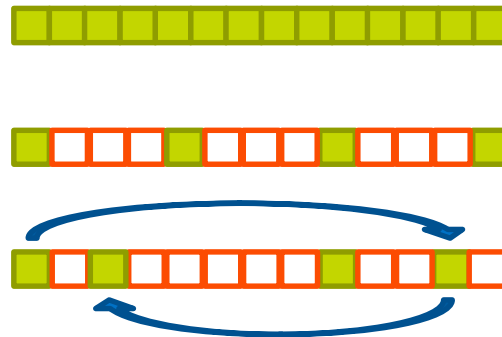
```
for (i=0; i<N; i++)  
  A[i] = C[i]*D[i]
```

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]
```



Activity 6. Collect data to GET Memory Access Patterns

Purpose: Calculate strides

Select “loop in bench_stencil\$omp\$parallel_for@23 at bench_stencil.c:24”

Press “Collect” button in “2.1 Check Memory Access Patterns” section
~ 1 minute

Create a snapshot

The screenshot shows the Intel Advisor interface with the 'Survey & Roofline' tab selected. The 'Roofline' panel on the right displays a list of function call sites and loops. The entry 'loop in bench_stencil\$omp\$parallel_for@23 at bench_stencil.c:24' is highlighted with a red box, and its checkbox is checked. The '2.1 Check Memory Access Patterns' section on the left has a 'Collect' button highlighted with a red box.

Function Call Sites and Loops	Checked
[loop in bench_stencil\$omp\$parallel_for@23 at bench_stencil.c:24]	<input checked="" type="checkbox"/>
[loop in main at main.c:20]	<input type="checkbox"/>
[loop in bench_stencil\$omp\$parallel_for@23 at bench_stencil.c:24]	<input type="checkbox"/>
f _start	<input type="checkbox"/>
f main	<input type="checkbox"/>
f bench_stencil	<input type="checkbox"/>
[loop in bench_stencil at bench_stencil.c:22]	<input type="checkbox"/>
f bench_stencil\$omp\$parallel_for@23	<input type="checkbox"/>
[loop in bench_stencil\$omp\$parallel_for@23 at bench_stencil.c:24]	<input checked="" type="checkbox"/>

Optimization Notice

Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Activity 6. Screenshot

Summary Survey & Roofline Refinement Reports

Site Location	Loop-Carried Dependencies	Strides Distribution	Access Pattern	Footprint Estimate		
				Max. Per-Instruction Addr. Range	First Instance Site Footprint	Simulated Memory F
▶ [loop in bench_stencil at bench_stencil.c:...	✔ No Dependencies Found	No Information Available	No Information Available	No Information Available	No Information Available	No Information Availi
▶ [loop in bench_stencil at bench_stencil.c:...	No Information Available	60% / 40% / 0%	fixed Strides	91MB	183MB	0B

```

22   for (istep = 0; istep < NSTEP; istep++) {
23       #pragma omp parallel for
24       for (k = 1; k < dim + 1; k++) {
25           for (j = 1; j < dim + 1; j++) {
26               for (i = 1; i < dim + 1; i++) {

```

Memory Access Patterns Report Dependencies Report Recommendations

ID	Stride	Type	Source	Nested Function	Variable references	Max. Per-Instruction Addr. Range
▶ P1	264196	Constant stride	bench_stencil.c:28		block 0x7f5f5e019010 allocated at main.c:16, block 0x7f5f7ecb7010 allocated at main.c:15	91MB
▶ P2	264196	Constant stride	bench_stencil.c:28		block 0x7f5f5e019010 allocated at main.c:16, block 0x7f5f7ecb7010 allocated at main.c:15	91MB
▶ P3	264196	Constant stride	bench_stencil.c:29		block 0x7f5f5e019010 allocated at main.c:16, block 0x7f5f7ecb7010 allocated at main.c:15	91MB
▶ P4	264196	Constant stride	bench_stencil.c:30		block 0x7f5f5e019010 allocated at main.c:16, block 0x7f5f7ecb7010 allocated at main.c:15	91MB
▶ P5	264196	Constant stride	bench_stencil.c:31		block 0x7f5f5e019010 allocated at main.c:16, block 0x7f5f7ecb7010 allocated at main.c:15	91MB
▶ P6	264196	Constant stride	bench_stencil.c:32		block 0x7f5f5e019010 allocated at main.c:16, block 0x7f5f7ecb7010 allocated at main.c:15	91MB
▶ P7	264196	Constant stride	bench_stencil.c:33		block 0x7f5f5e019010 allocated at main.c:16, block 0x7f5f7ecb7010 allocated at main.c:15	91MB
▶ P8	264196	Constant stride	bench_stencil.c:34		block 0x7f5f5e019010 allocated at main.c:16, block 0x7f5f7ecb7010 allocated at main.c:15	91MB
▶ P9		Parallel site information	bench_stencil.c:24			
▶ P11	0	Uniform stride	bench_stencil.c:24			4B
▶ P12	0	Uniform stride	bench_stencil.c:25			8B
▶ P13	0	Uniform stride	bench_stencil.c:26			4B

Optimization Notice



Activity 7: Splitting task to tiles

Activity 7

Purpose: Improve memory access pattern

Build a version with tiling

```
$ git checkout ver4
```

```
$ make
```

Re-run Roofline analysis


Create a snapshot

Compare with previous activity

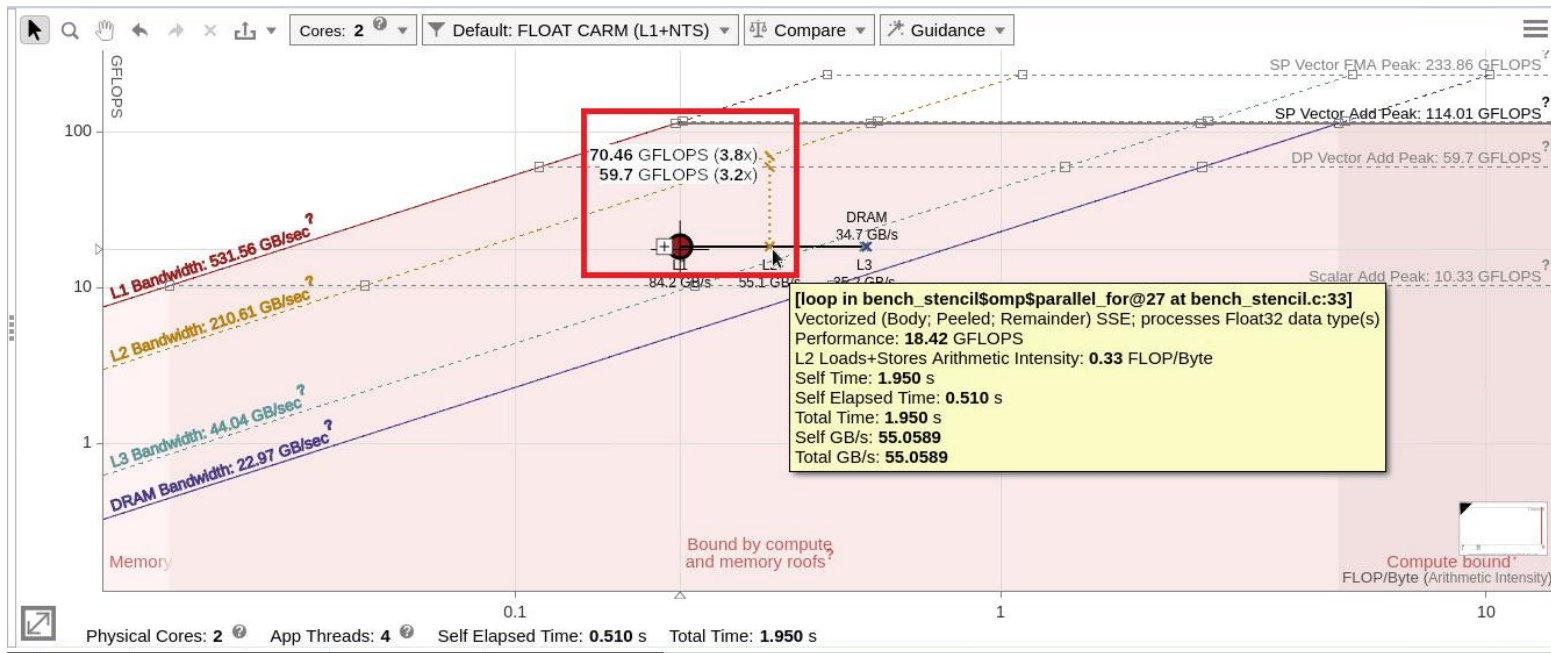
Vectorization Advisor

Vectorization Advisor is a vectorization analysis tool vectorization and characterize your memory vs. vec

Program metrics

Elapsed Time	1.32s
Vector Instruction Set	 SSE
Number of CPU Threads	4

Activity 6. Screenshot



Optimization Notice

Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Activity 8: Enabling AVX512

Activity 8

Purpose: Fix compilation options to use the highest available ISA

Build a version with new compilation flags

```
$ git checkout ver5
```

```
$ make clean && make
```

Re-run Survey analysis

Create a snapshot

Compare with previous activity


Review recommendations

Activity 8. Screenshots

Vectorization Advisor

Vectorization Advisor is a vectorization analysis tool vectorization and characterize your memory vs. vec

Program metrics

Elapsed Time	1.32s
Vector Instruction Set	 SSE
Number of CPU Threads	4

Vectorization Advisor

Vectorization Advisor is a vectorization analysis too vectorization and characterize your memory vs. vec

Program metrics

Elapsed Time	1.32s
Vector Instruction Set	AVX512
Number of CPU Threads	4

Activity 8. Screenshots

The screenshot displays the Intel VTune Performance Analyzer interface. At the top, the 'Summary' tab is active, showing a table of performance data. A red box highlights the 'Performance Issues' column, which contains the message '@ 1 Ineffective peeled/remainder loop(s) present'. Below the table, the 'Recommendations' tab is selected, showing a detailed report for the identified issue. The report includes a description of the problem, a recommendation to 'Disable dynamic alignment', and a code example showing how to use the `#pragma vector nodynamic_align` directive to improve performance.

Function Call Sites and Loops	Performance Issues	CPU Time	Type	Why No Vectorization?	Vectorized Loops
		Total Time	Self Time		Vecto ... Efficiency Gain E... VL (V ...
[loop in bench_stencil\$omp\$parallel_for@27 at bench_stencil.c:20]	@ 1 Ineffective peeled/remainder loop(s) present	1.640s	1.640s	Vectorized (Body ...	AVX512 100% 19.57x 16
[loop in main at main.c:20]	@ 2 Ineffective peeled/remainder loop(s) present	0.350s	0.350s	Vectorized (Body ...	AVX512 78% 12.45x 16
[loop in bench_stencil\$omp\$parallel_for@27 at bench_stencil.c:26]		1.760s	0.120s	Scalar	
f _start		2.370s	0.000s	Function	
f main		2.370s	0.000s	Function	
[loop in bench_stencil at bench_stencil.c:26]	@ 1 Assumed dependence present	2.020s	0.000s	Scalar	
f bench_stencil		2.020s	0.000s	Function	
f bench_stencil\$omp\$parallel_for@27		1.760s	0.000s	Function	
[loop in bench_stencil\$omp\$parallel_for@27 at bench_stencil.c:26]		1.760s	0.000s	Scalar	
[loop in bench_stencil\$omp\$parallel_for@27 at bench_stencil.c:26]		1.760s	0.000s	Threaded (OpenMP)	

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.

Disable dynamic alignment

The compiler automatically peeled iterations from the vector loop into a scalar loop to align the vector loop with a particular memory reference; however, this optimization may not be ideal. To possibly achieve better performance, disable automatic peel generation using the directive: `#pragma vector nodynamic_align`

Example (original code)

```
...
#pragma vector nodynamic_align
for (int i = 0; i < len; i++)
...
```

Optimization Notice

Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Activity 9: Disabling dynamic alignment

Activity 9

Purpose: Exclude loop peel/reminder execution

Build a version with new compilation flags

```
$ git checkout ver6
```

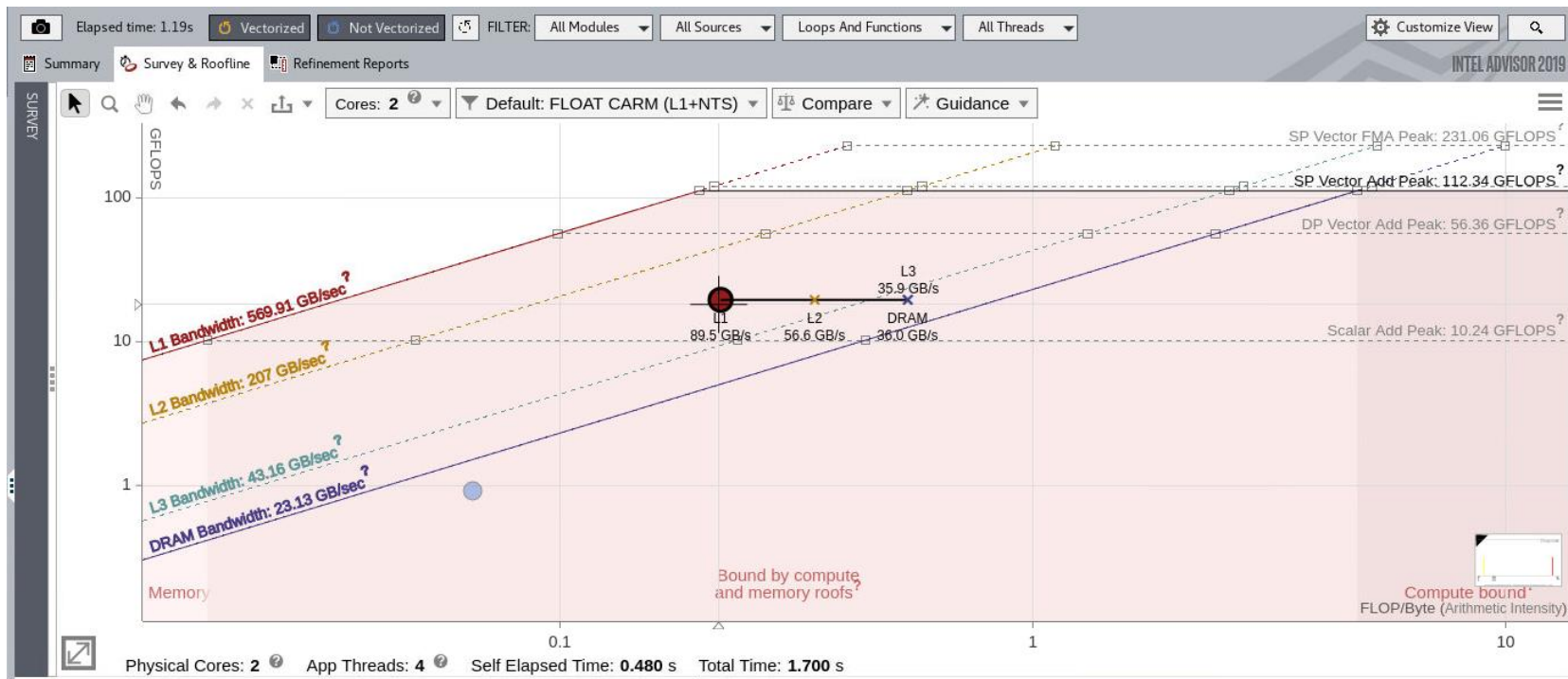
```
$ make
```

Re-run Roofline analysis

Create a snapshot

Compare with previous activity

Activity 9. Screenshots



Optimization Notice

Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



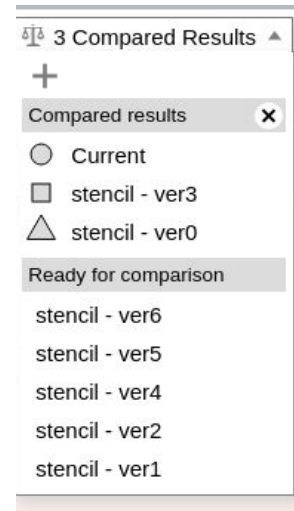
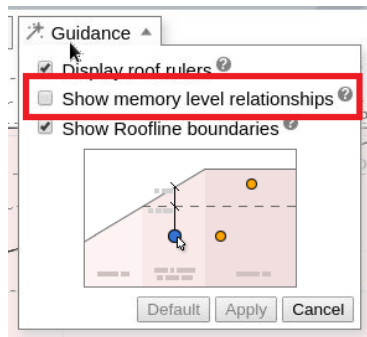
Activity 10: Comparing roofline charts

ACTIVITY 10

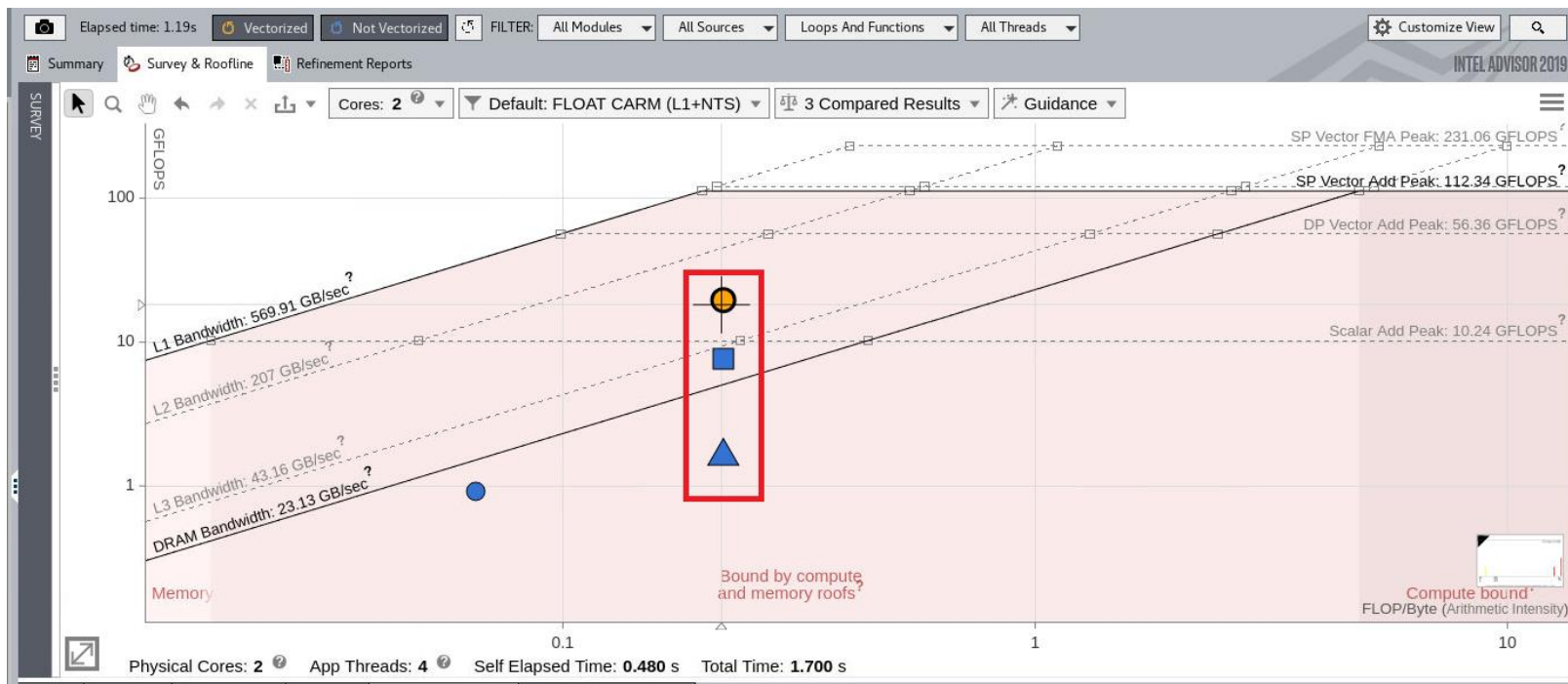
Purpose: Graph roofline chart for optimized version, and compare with initial chart

Turn off “Show different memory level relationships” at Guidance tab

Compare with results for versions of source code “ver3” and “ver0”



Activity 10. Screenshot

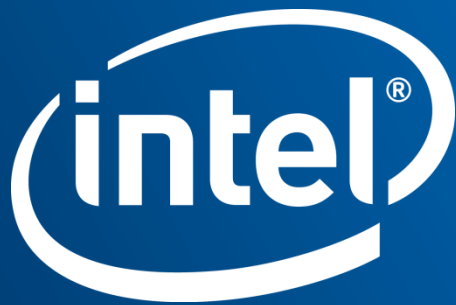


Speedup: ~12x

Optimization Notice

Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.





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.

Copyright © 2016, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

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