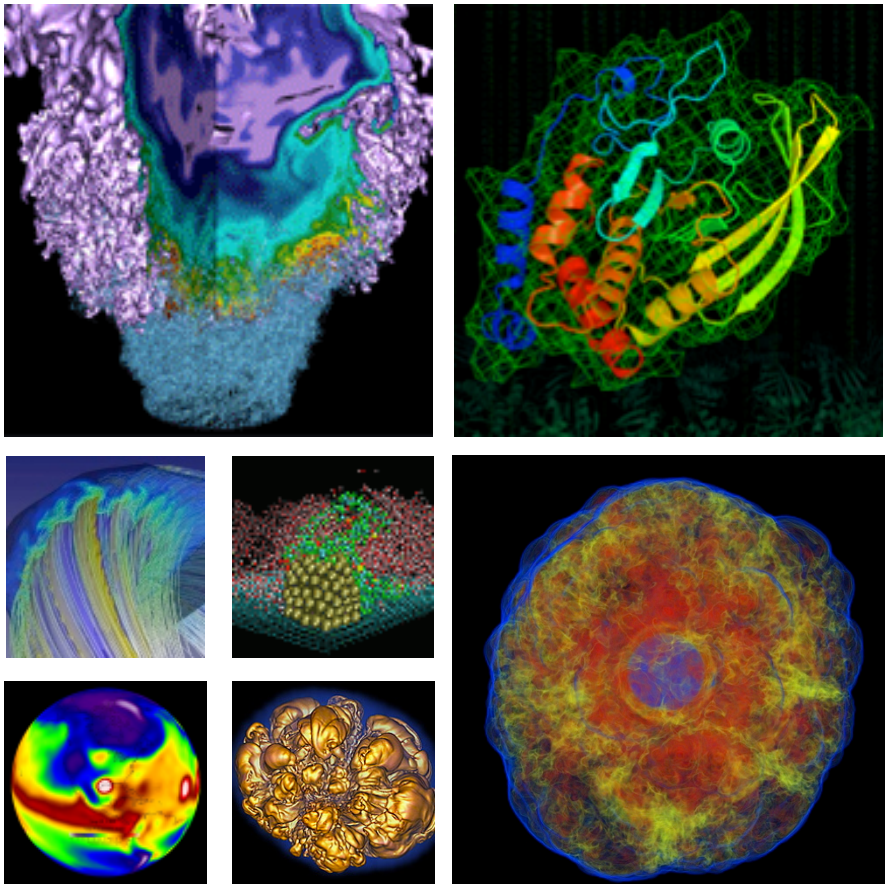
Intel Advisor and Roofline Automation



Tuomas Koskela (NERSC),
Zakhar Matveev (Intel)

June 14, 2017

5 Steps to Efficient Vectorization - Vector Advisor

1. Compiler diagnostics + Performance Data + SIMD efficiency information

Function Call Sites and Loops	Self Time	Total Time	Compiler Vectorization: Loop Type	Compiler Vectorization: Why No Vectorization?
[loop in runCForallLambdaLoop:]	0.094s	0.094s	Scalar	vector dependence prevents vector...
[loop in runCForallLambdaLoop:]	0.140s	3.744s	Scalar	inner loop was already vectorized
[loop in std::_Complex_base<double,struct _C_double_complex>::i...	0.031s	0.031s	Vectorized (Body)	
Vectorized SSE; SSE2 loop processing Float32; Float64 data typ				
Peeled loop; loop stmts were reordered				
[loop in std::basic_string<char,struct std::char_traits<char>,class std::allo...	0.000s			
[loop in std::basic_string<char,struct std::char_traits<char>,class std::allo...	0.000s			
[loop in std::num_put<char,class std::ostreambuf_iterator<char,struct st...	0.000s			

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

2 Issue: Peeled/Remainder loop(s) present

8 All or some source loop iterations are not executing in the kernel loop. Improve performance by moving source loop iterations from peeled/remainder loops to the kernel loop. Read more at Vector Essentials, Utilizing Full Vectors...

Recommendation: Align memory access

Projected maximum performance gain: High

Projection confidence: Medium

...se one of the memory accesses in the source loop does not ...ry access and tell the compiler your memory access is aligned. ...byte boundary:

```
...SIZE*sizeof(float), 32);
```

3. "Accurate" Trip Counts + FLOPs: understand utilization, parallelism granularity & overheads

Total Time	Trip Counts: Median	Min	Max	Iteration Duration	Call Count
3,151s	1	1	1	3,1509s	1
0,440s	1	1	1	< 0,0001s	2408000
0,010s	1	1	2	< 0,0001s	207506
0,010s	2	1	9	< 0,0001s	1173619
0,010s	3	1	5	< 0,0001s	1312315

4. 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

5. Memory Access Patterns Analysis

Site Name	Site Function	Site Info	Loop-Carried Dependencies	Strides Distribution	Access Pattern
loop_site_203	runCRawLoops	runCRawLoops.cxx1063	RAW:1	No information available	No information available
loop_site_139	runCRawLoops	runCRawLoops.cxx622	No information available	39% / 36% / 25%	Mixed strides
loop_site_160	runCRawLoops	runCRawLoops.cxx925	No information available	100% / 0% / 0%	All unit strides

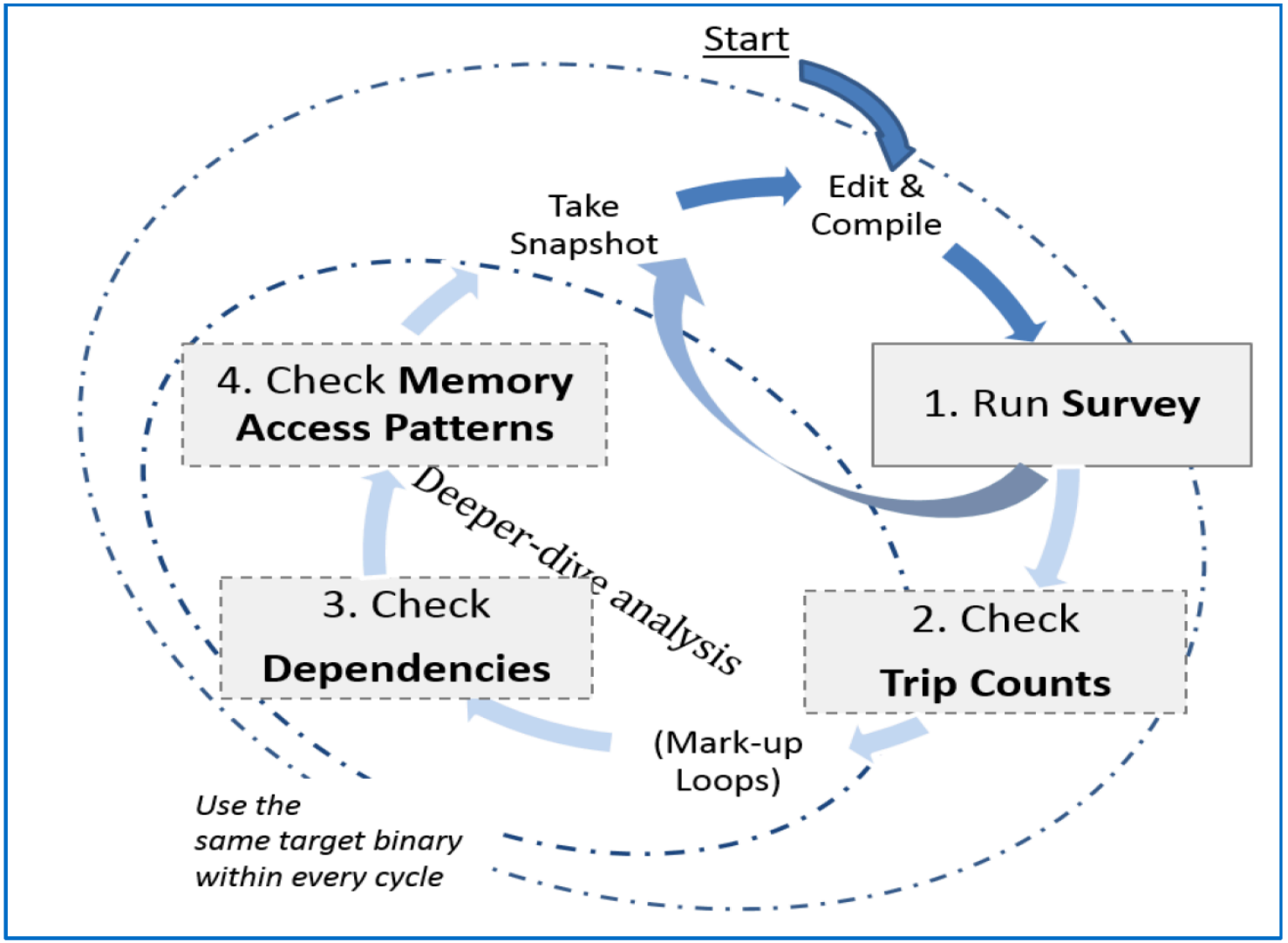
Memory Access Patterns | Correctness Report

ID	Stride	Type	Source	Modules	Alignment
P22	0; 0; 1	Unit stride	runCRawLoops.cxx637	lcals.exe	
P23	0; 0	Unit stride	runCRawLoops.cxx638	lcals.exe	
P30	-1575; -63; -26; -25; -1; 0; 1; 25; 26; 63; 2164801	Variable stride	runCRawLoops.cxx628	lcals.exe	

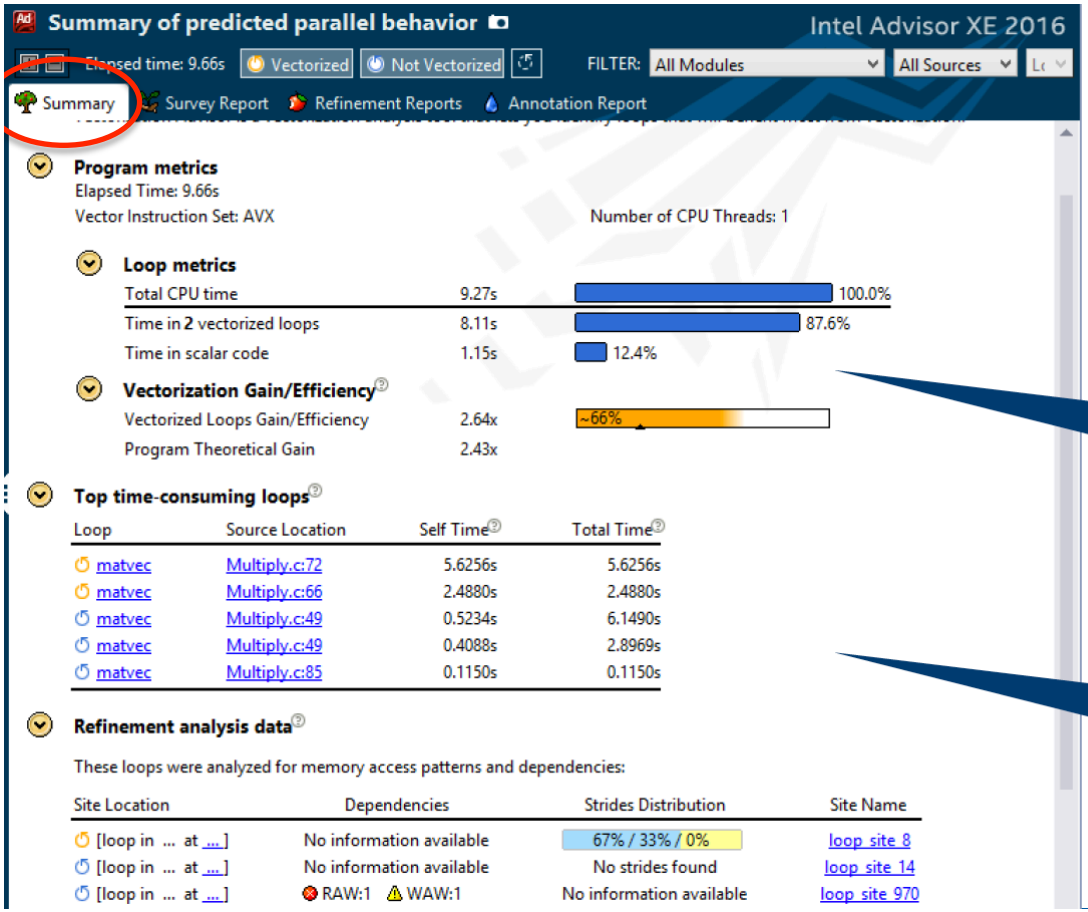
```
635                 j2 = ( j2 & 64-1 ) ;
636                 p[ip][0] += y[i2+32];
637                 p[ip][1] += z[j2+32];
638                 i2 += e[i2+32];
639                 j2 += f[j2+32];
```

```
626                 i1 &= 64-1;
627                 j1 &= 64-1;
628                 p[ip][2] += b[j1][i1];
```

Vectorization Analysis Workflow



Quickly Characterize the Vectorization Efficiency of Your Code: Advisor Summary

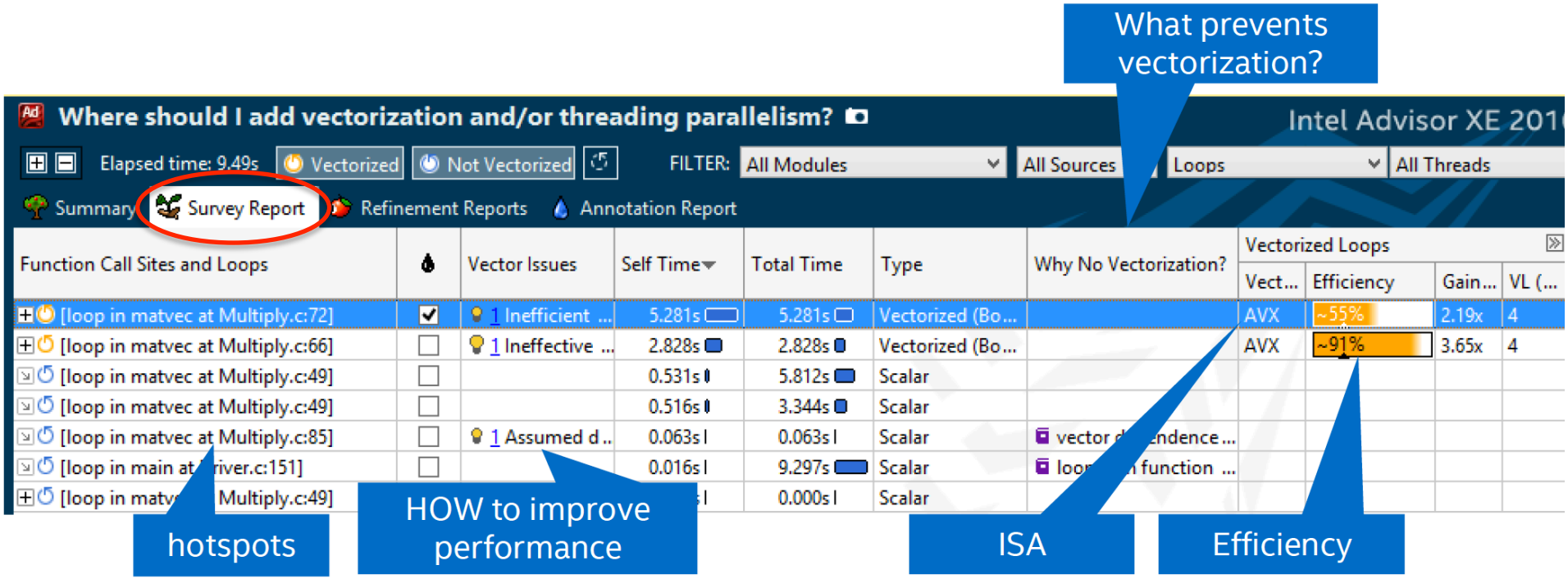


Use the summary view to quickly characterize your program

Time in **Scalar vs. Vector** loops. SIMD Efficiency.

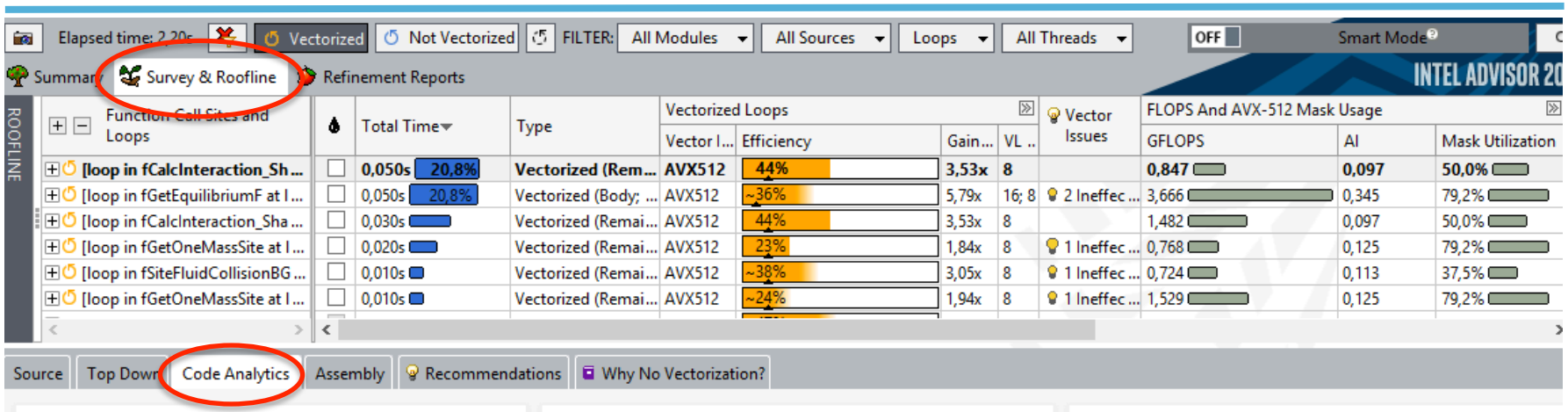
Focus on Hottest kernels

Advisor Survey: Focus and Characterize



All the data you need for effective vectorization

Code Analytics



Loop in fCalcInteraction_ShanChen_Boundary at lbpFORCE.cpp:188

0,050s
Vectorized (Remainder) *Total time*

AVX2; AVX512F_512 0,050s
Instruction Set *Self time*

- Memory 41% (7)
- Compute 35% (6)
- Other 24% (4)

Instruction Mix Summary

3,53x
44% Vectorization Efficiency
Vectorization Gain

Average Trip Counts: 1

Traits

FMA

Mask Manipulations

Code Optimizations

Compiler: Intel(R) C++ Intel(R) 64 Compiler for applications running on Intel(R) 64,

Version: 16.0.2.181 Build 20160204

Compiler estimated gain: 4,85x

Code Optimizations Applied By Compiler During Vectorization:

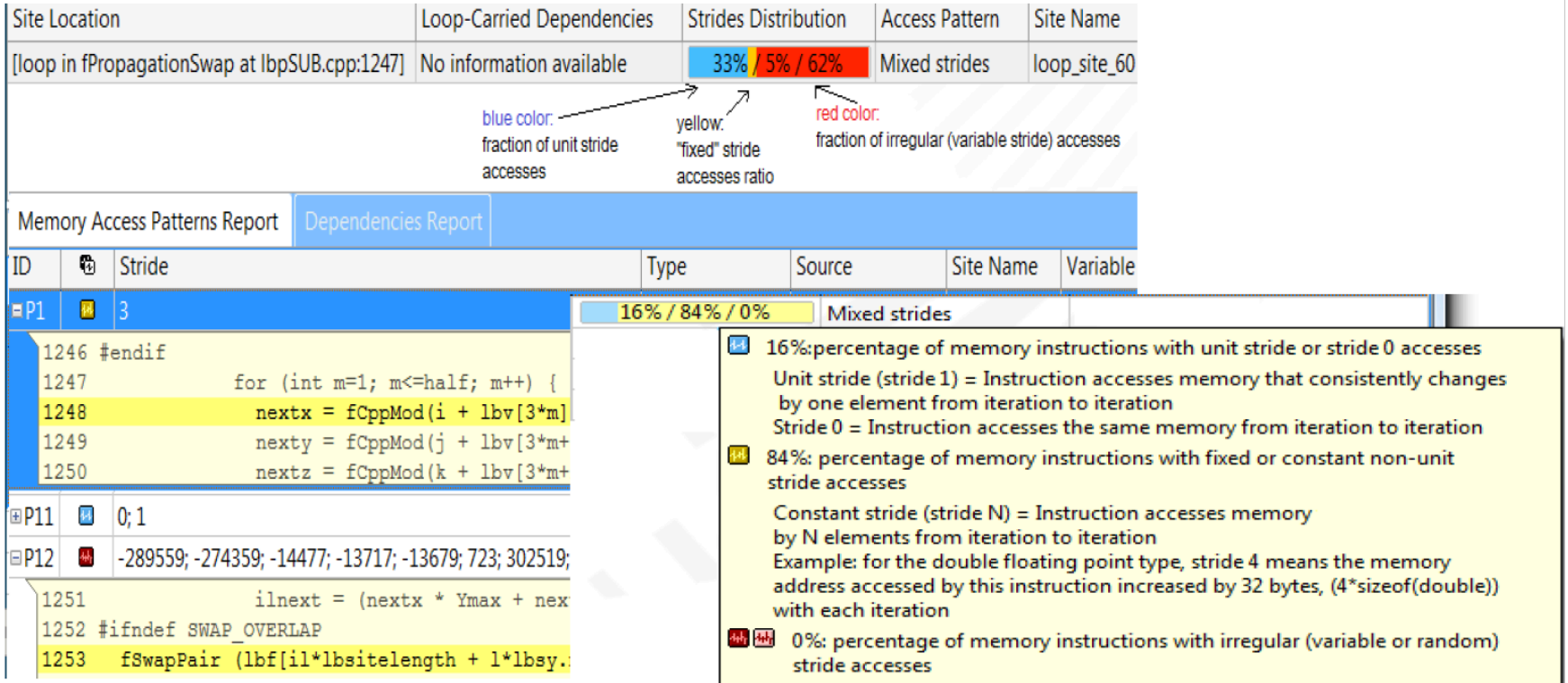
- Masked Loop Vectorization
- Unaligned Access in Vector Loop

GFLOPS: 0.8474
AVX-512 Mask Usage: 50

Instruction Mix

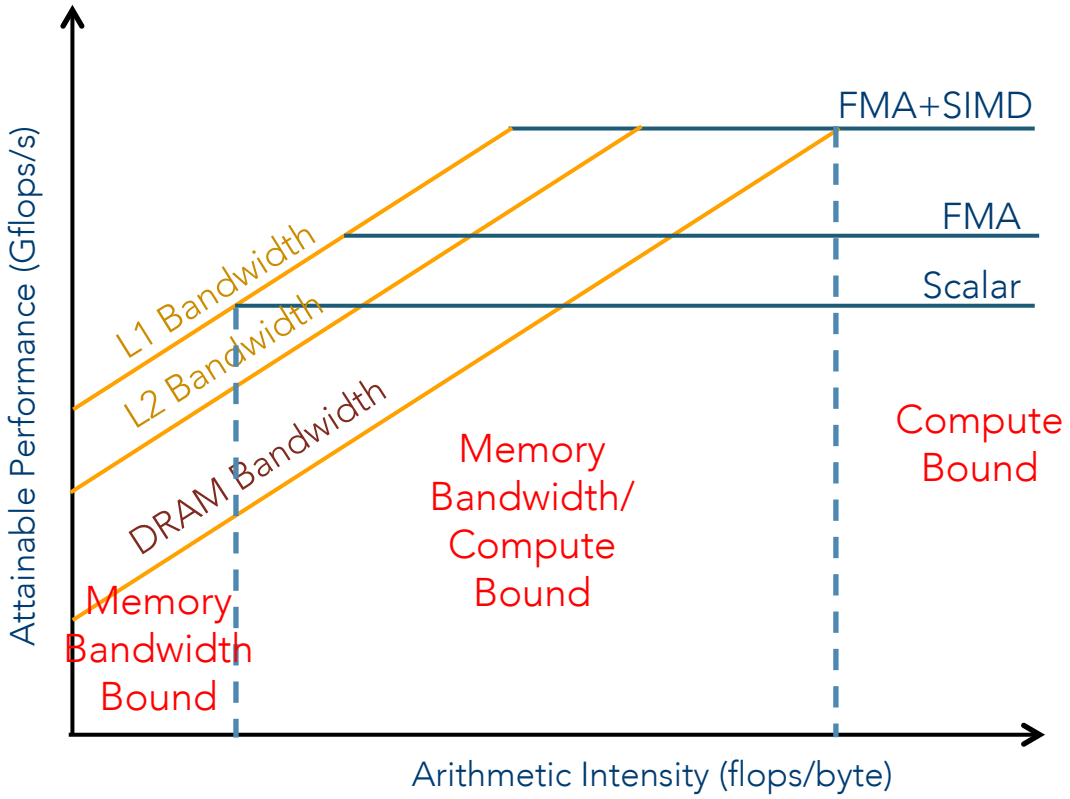
Memory: 7 *Compute:* 6 *Other:* 4 *Number of Vector Registers:* 7

Advisor Memory Access Pattern

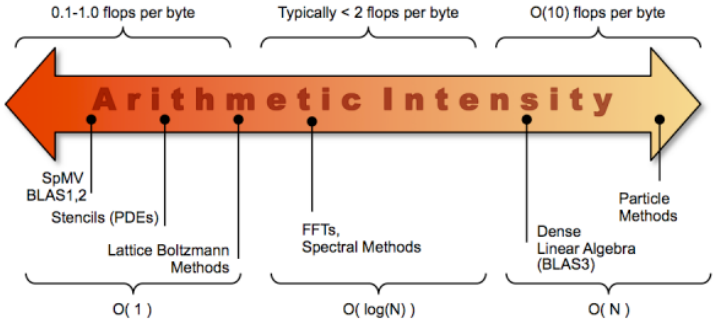


Roofline Performance Model: Am I bound by VPU/CPU or Memory

Roofline reflects an absolute performance bound (Gflops/s) of the system as a function of Arithmetic Intensity (flops/byte) of the application.



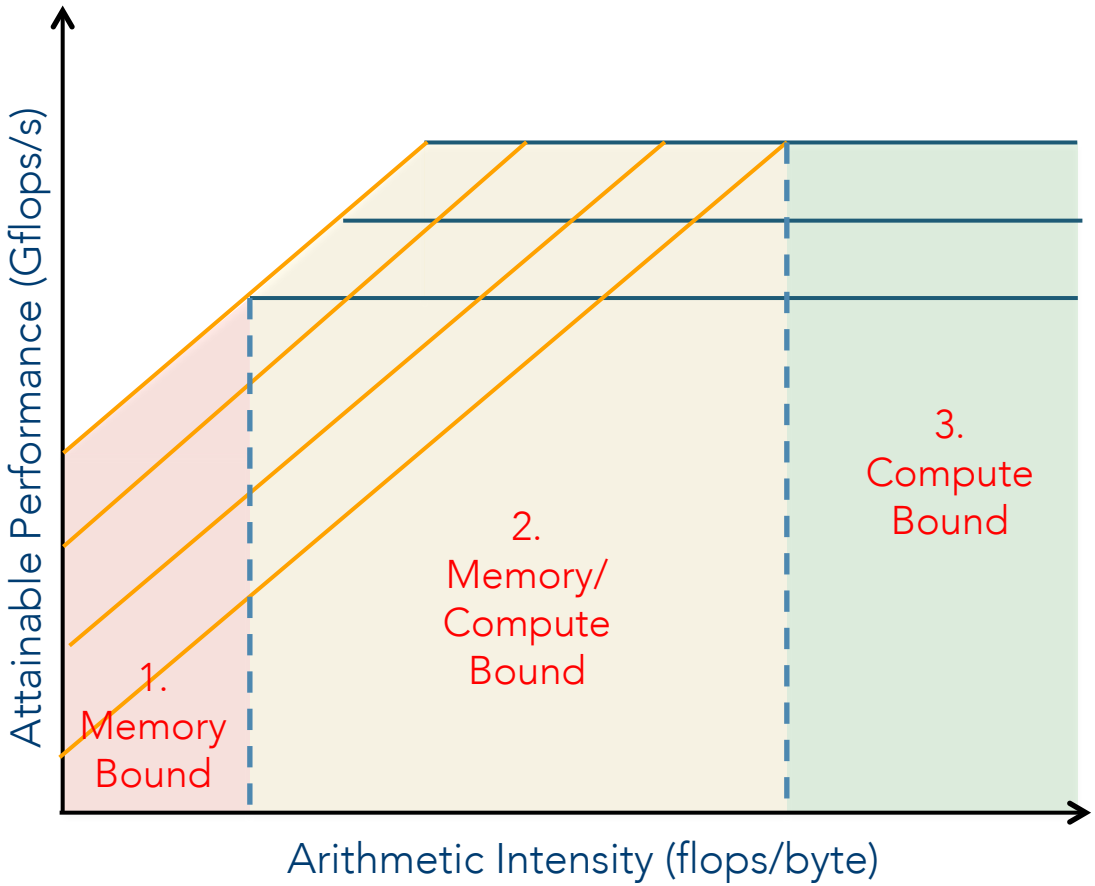
$$\text{Arithmetic Intensity} = \frac{\text{Total Flops computed}}{\text{Total Bytes transferred}}$$



Why Do We Need the Roofline Model?

- Need a sense of absolute performance when optimizing applications
  - How do I know if my performance is good?
  - Why am I not getting peak performance of the platform?
- Many potential optimization directions
  - How do I know which one to apply?
  - What is the limiting factor in my app's performance?
  - How do I know when to stop?

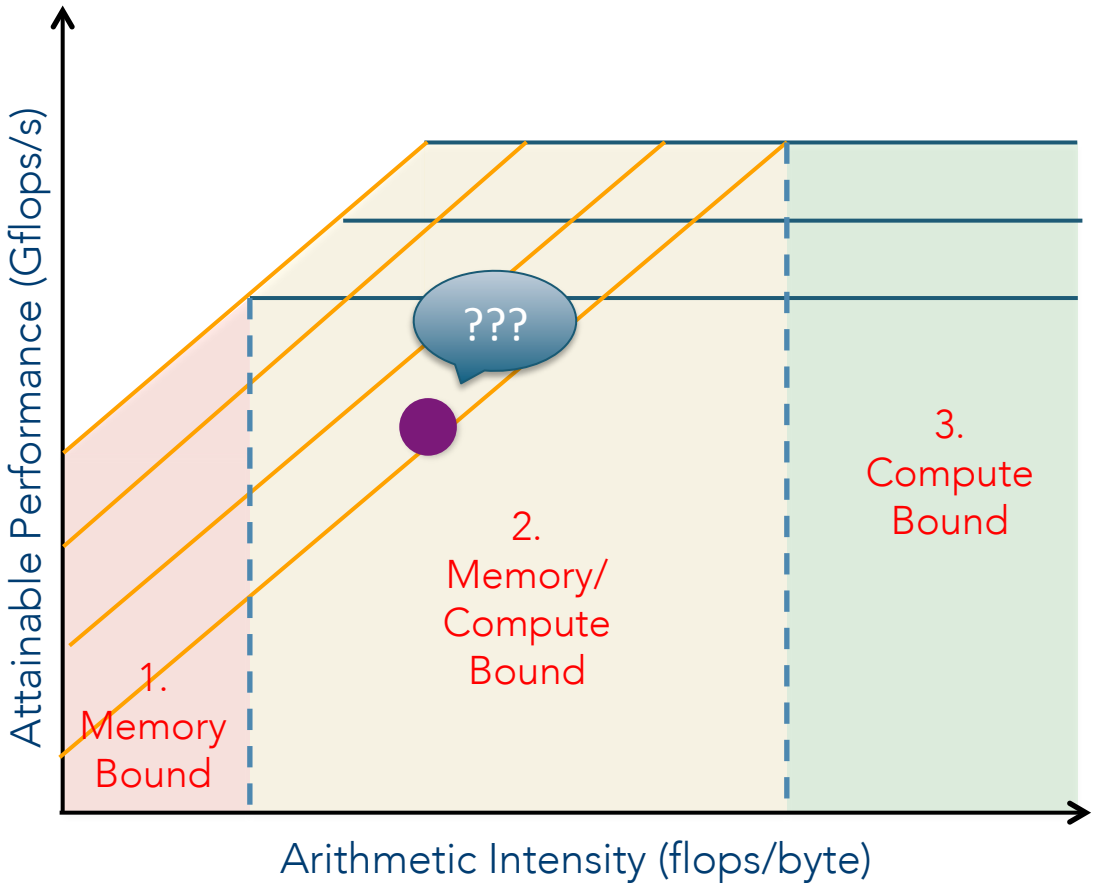
Is My Application Bound by a Memory Bandwidth Peak or a Compute Peak?



Often it's a combination of the two

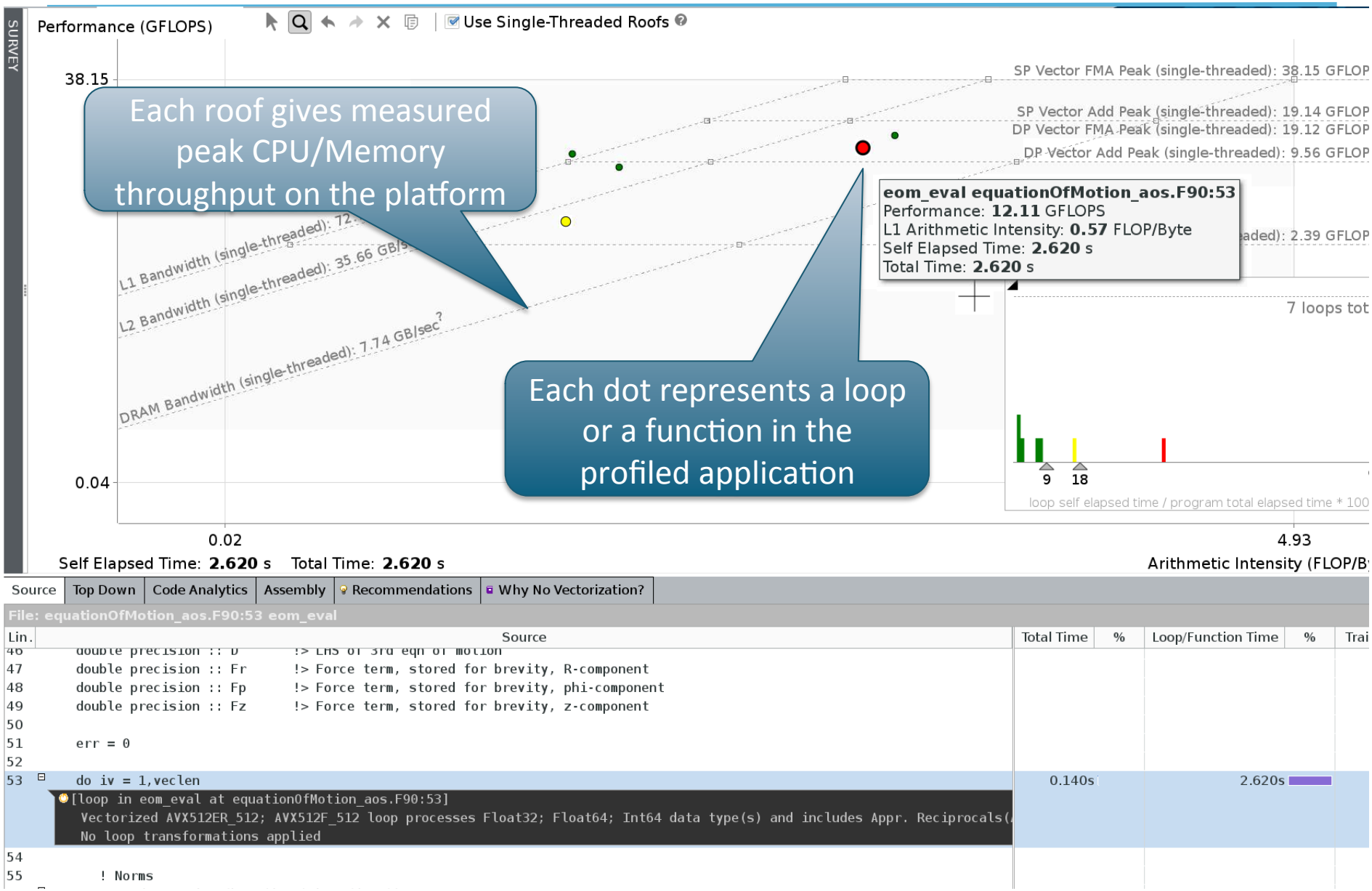
- Applications in area 1 are purely memory bandwidth bound
- Applications in area 3 are purely compute bound
- In area 2 we need more information

Place your application on the roofline, then ask yourself "Why am I Here?"



Usually, the analysis is not straightforward… You won't be on any ceiling. Or if you are, it is a coincidence. BUT - asking the questions "why am I not on a higher ceiling?" and "what should I do to reach it?" is always productive.

Roofline Automation in Advisor 2017



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

File: equationOfMotion_aos.F90:53 eom_eval

Lin.	Source	Total Time	%	Loop/Function Time	%	Trai
46	double precision :: D !> LHS of 3rd eqn of motion					
47	double precision :: Fr !> Force term, stored for brevity, R-component					
48	double precision :: Fp !> Force term, stored for brevity, phi-component					
49	double precision :: Fz !> Force term, stored for brevity, z-component					
50						
51	err = 0					
52						
53	do iv = 1,veclen	0.140s		2.620s		
	[loop in eom_eval at equationOfMotion_aos.F90:53] Vectorized AVX512ER_512; AVX512F_512 loop processes Float32; Float64; Int64 data type(s) and includes Appr. Reciprocals(No loop transformations applied					
54						
55	! Norms					

Advisor Roofline Under the Hood

Roofline application profile:

Axis Y: **FLOP/S** = **#FLOP** (mask aware) / **#Seconds**

Axis X: **AI** = **#FLOP** / **#Bytes**

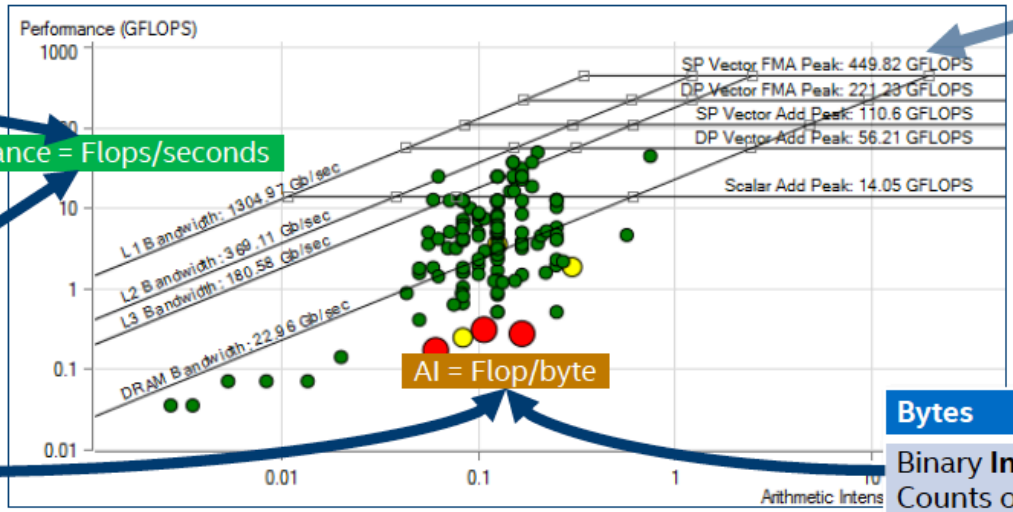
Seconds

User-mode **sampling**

Root access not needed

Roofs

Microbenchmarks
Actual peak for the current configuration



Performance = Flops/seconds

AI = Flop/byte

#FLOP

Binary **Instrumentation**
Does not rely on CPU counters

Bytes

Binary **Instrumentation**
Counts operands size (not cachelines)

What to Do With This Information?

- Compute bound applications
  - Make sure you have good OpenMP scalability. Look to see thread activity for major OpenMP regions.
  - Make sure your code is vectorizing. Look at Cycles per Instruction (CPI) and VPU utilization.
- Memory bandwidth bound applications
  - Try to improve memory locality, cache reuse
  - Identify the key arrays leading to high memory bandwidth usage and make sure they are/will-be allocated in HBM on KNL
    →Profit by getting ~ 5x more bandwidth GB/s.

Advisor Roofline access and how-to command line example

```
> module load advisor

> advixe-cl --collect survey
--project-dir ./your_project
-- <your-executable-with-parameters>

> advixe-cl --collect tripcounts
-flops-and-masks --project-dir
./your_project
-- <your-executable-with-parameters>

> advixe-gui ./your_project

> advixe-cl --report survey
--project-dir <your-project-dir>
--show-all-columns --format=csv
--report-output <output-file.csv>
```

1st step
Obtain "Seconds"
1.1x overhead

2nd step
Obtain #FLOP count:
3x-5x overhead

Launch GUI

Write report into csv file

Command line MPI example (slurm)

1st step:

```
> srun -n <num-of-ranks> -c <num_of_cores_per_rank>
advixe-cl -v -collect survey
-project-dir=<same_dir_name>
-data-limit=0 <your_executable>
```

2nd step:

```
> srun -n <num-of-ranks> -c <num_of_cores_per_rank>
advixe-cl -v -collect tripcounts
-flops-and-masks -project-dir=<same_dir_name>
-data-limit=0 <your_executable>
```

Further Reading and Viewing

- http://www.nersc.gov/users/software/performance-and-debugging-tools/advisor/
- S. Williams et al. *CACM* (2009) **http://crd.lbl.gov/departments/computer-science/PAR/research/roofline**
- A. Ilic et al., IEEE Computer Architecture Letters (2014)
- **http://www.intel.com/content/www/us/en/events/hpcdevcon/parallel-programming-track.html#utilizing**
- **https://www.youtube.com/watch?v=h2QEM1HpFgg**
- **https://software.intel.com/en-us/articles/getting-started-with-intel-advisor-roofline-feature**
- **http://www.nersc.gov/users/application-performance/measuring-arithmetic-intensity/**
- **https://www.codeproject.com/Articles/1169323/Intel-Advisor-Roofline-Analysis**
- **https://software.intel.com/sites/default/files/managed/1e/19/roofing-a-house.pdf**
- **https://github.com/tkoskela/pyAdvisor**

National Energy Research Scientific Computing Center