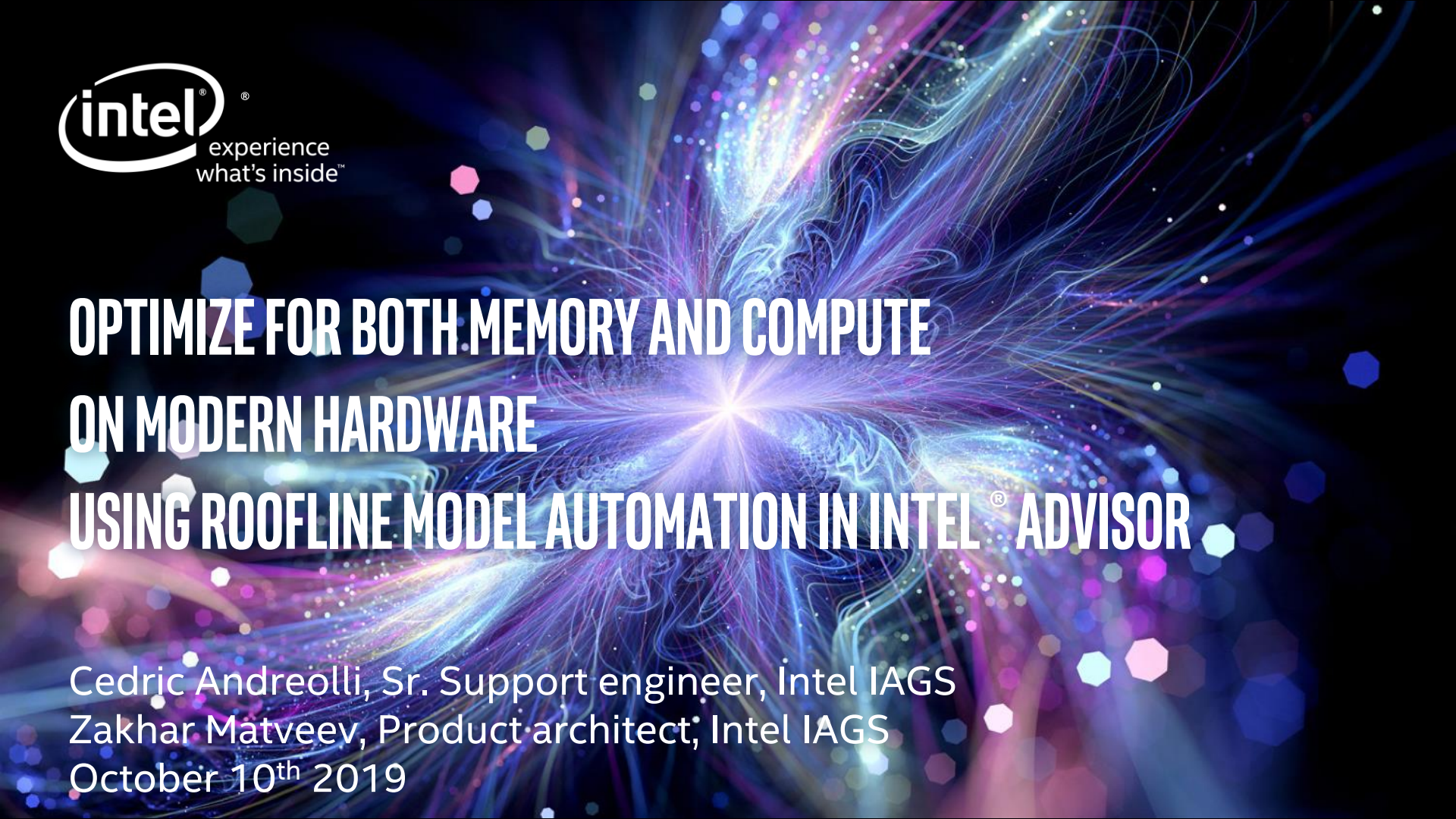


OPTIMIZE FOR BOTH MEMORY AND COMPUTE ON MODERN HARDWARE USING ROOFLINE MODEL AUTOMATION IN INTEL® ADVISOR

Cedric Andreolli, Sr. Support engineer, Intel IAGS
Zakhar Matveev, Product architect, Intel IAGS
October 10th 2019

Acknowledgments

Cedric Andreoli (Intel),

Sam Williams (LBNL), Aleksandar Ilic (INESC),

Kate Antakova (Intel), Philippe Thierry (Intel)

The Roofline Model

Flop: How many FP operations is your kernel performing

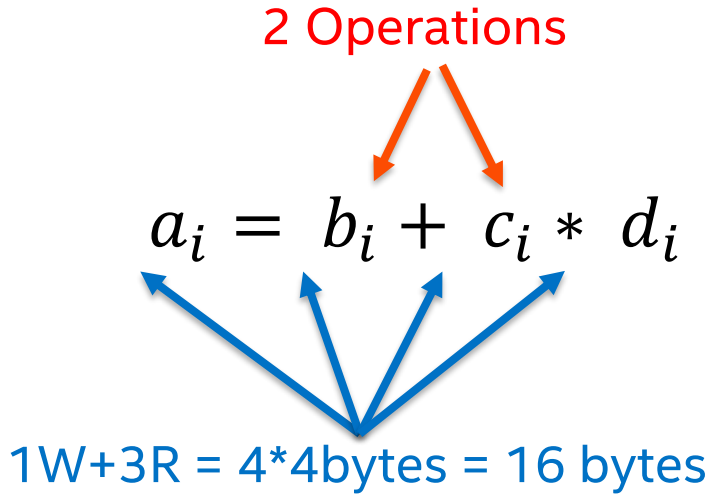
Byte transferred: How many bytes need to be transferred

Performance is expressed in Flop/s

- Number of operations per second
- We usually want to maximize this number

Arithmetic intensity

- Ratio between flops and bytes
- Directly impacts the performance (memory bound/compute bound)



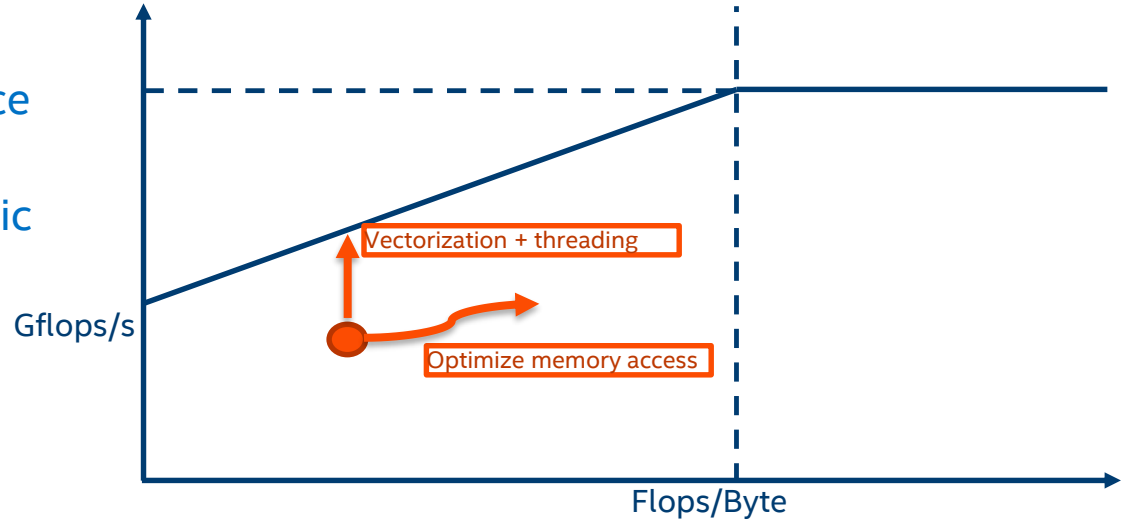
Building a Roofline Model

Roofline model is based on the formula

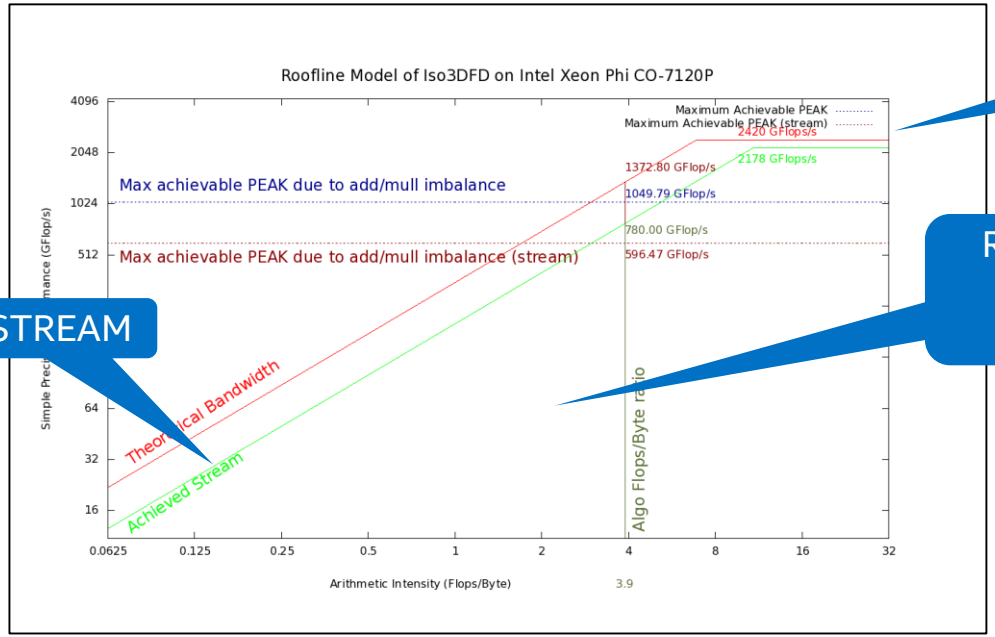
$$\text{Gflop/s} = \min \begin{cases} \textit{Platform PEAK} \\ \textit{Platform BW} * \textit{AI} \end{cases}$$

Roofline Axis

- Vertical axis is the performance in Gflop/s
- Horizontal axis is the arithmetic intensity (Flop/Byte)
- Using log scales



Old approach – pen and paper to get Roofline data



Run DGEMM

Run STREAM

Read the source, count FP ops, loads&stores

```
for(int bz=HALF_LENGTH; bz<n3; bz+=n3_Tblock)
    ALF_LENGTH; by<n2; by+=n2_Tblock)
    LF_LENGTH; bx<n1; bx+=n1_Tblock) {
    MIN(bz+n3_Tblock, n3);
    MIN(by+n2_Tblock, n2);
    MIN(n1_Tblock, n1-bx);

  bz; iz<izEnd; iz++) {
  by; iy<iyEnd; iy++) {
  xt = ptr_next_base + iz*n1n2 + iy*n1 + bx;
  prev = ptr_prev_base + iz*n1n2 + iy*n1 + bx;
float* vel = ptr_vel_base + iz*n1n2 + iy*n1 + bx;
for(int ix=0; ix<ixEnd; ix++) {
  value = 0.0;
  += prev[ix]*coeff[0];
  ir=1; ir<=HALF_LENGTH; ir++) {
  e += coeff[ir] * (prev[ix + ir] + prev[ix - ir])
  ;
value += coeff[ir] * (prev[ix + ir*n1] + prev[ix -
    ir*n1]);
value += coeff[
    ir*n1n2]);
}
next[ix] = 2.0f* prev[ix] - next[ix] + value*vel[ix];
}
}}}
```

51 adds

4 loads

27 muls

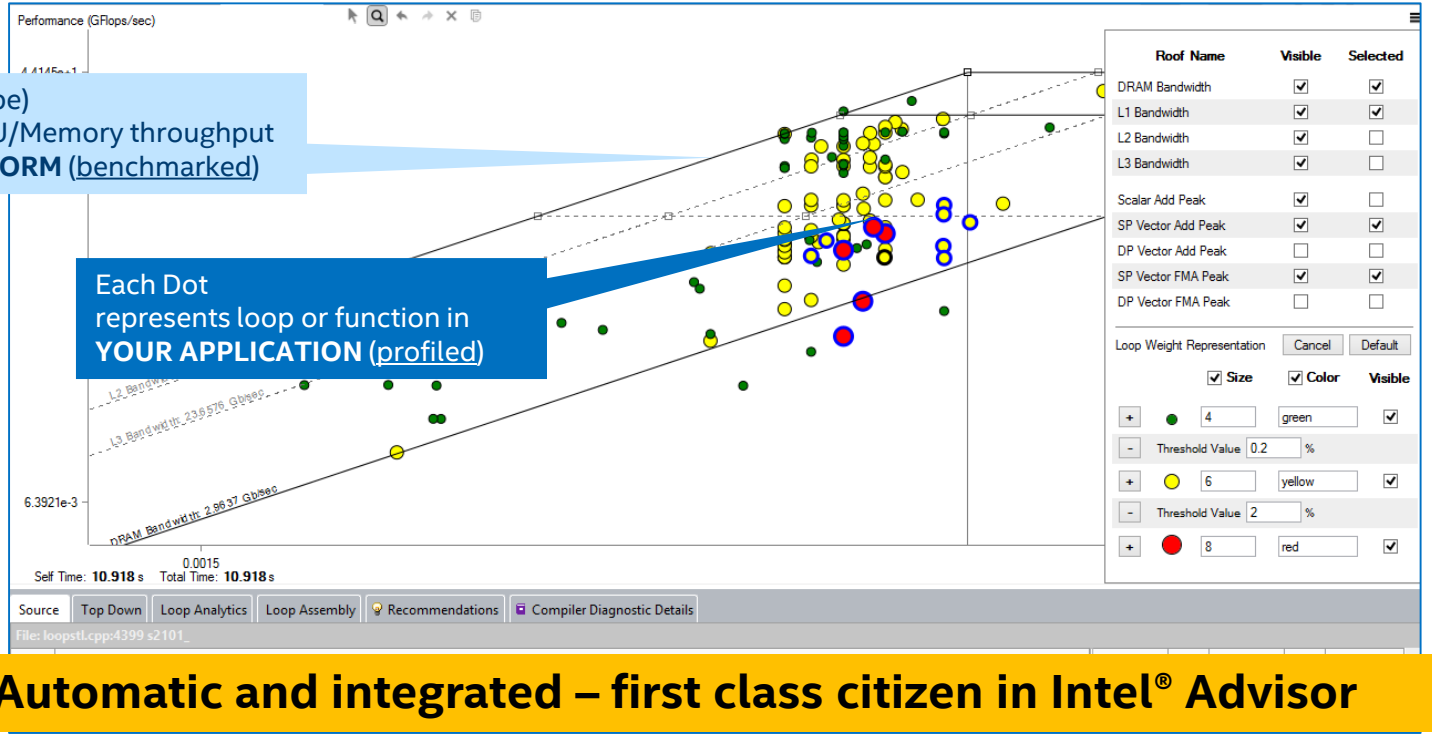
1 store

"3D stencil performance evaluation and auto-tuning on multi and many-core computers", C.Andreolli et.al.

Cumbersome – but people still did it!

(“Cache-aware”) Roofline Automation , since 2016

Each Roof (slope)
Gives peak CPU/Memory throughput of your **PLATFORM** (benchmarked)



Automatic and integrated – first class citizen in Intel® Advisor

Questions to answer with Roofline: for your loops / functions

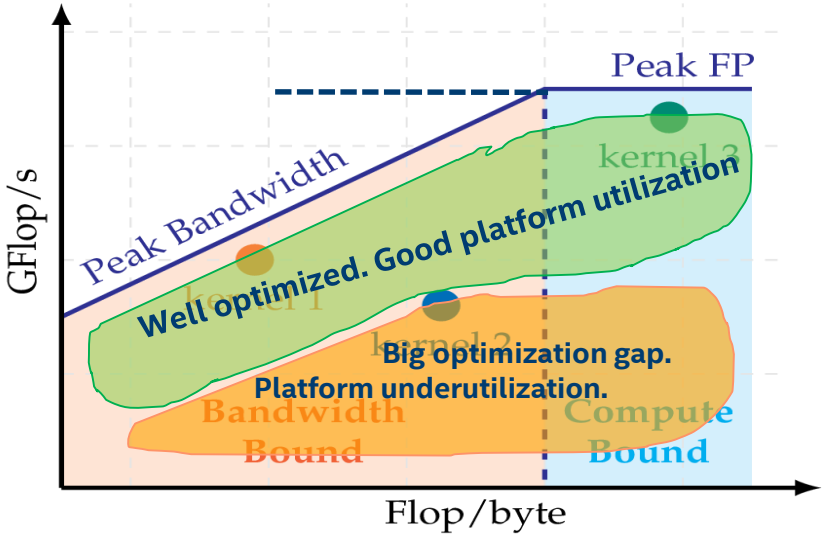
1 **Am I doing well? How far am I from the pick?**

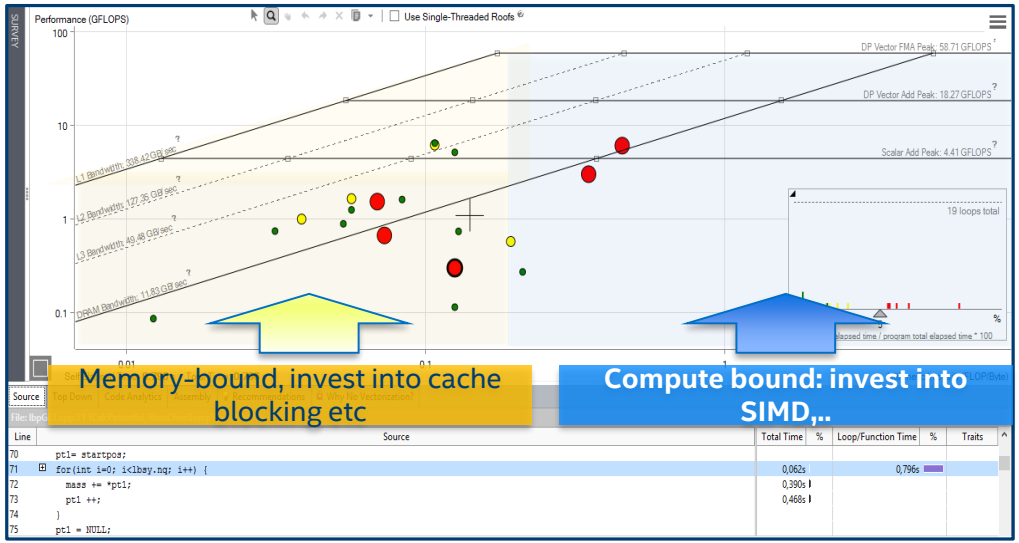
(do I utilize hardware well or not?)

2 **Final** Bottleneck?

(where will be my limit after I done all optimizations?)

Long-term ROI, optimization strategy





The Roofline Model in Intel® Advisor

First Implementation: Cache-Aware Roofline Model (CARM)

- Based on instrumentation
- 2 runs: one for sampling and timing loops and functions (low overhead); second one for instrumentation
- Algorithmic version of the Roofline Model, optimization usually does not impact AI
- ☺ **Really powerful to characterize an algorithm**
- ☹ **Not always easy to find your current #1 bottleneck**

Currently exposed as a technical preview feature, need:
export ADVIXE_EXPERIMENTAL=int_roofline

Additional Implementation: Integrated or « mutli-level » Roofline Model (IRM)

- Based on cache simulation, evaluate the traffic between each memory subsystem (L1/L2/LLC/DRAM)
- ☺ **Incorporates the « original Roofline model », provides meaningful information for the #1 bottleneck improvement**
- ☹ **Requires more time to run and more visual aids to comprehend**

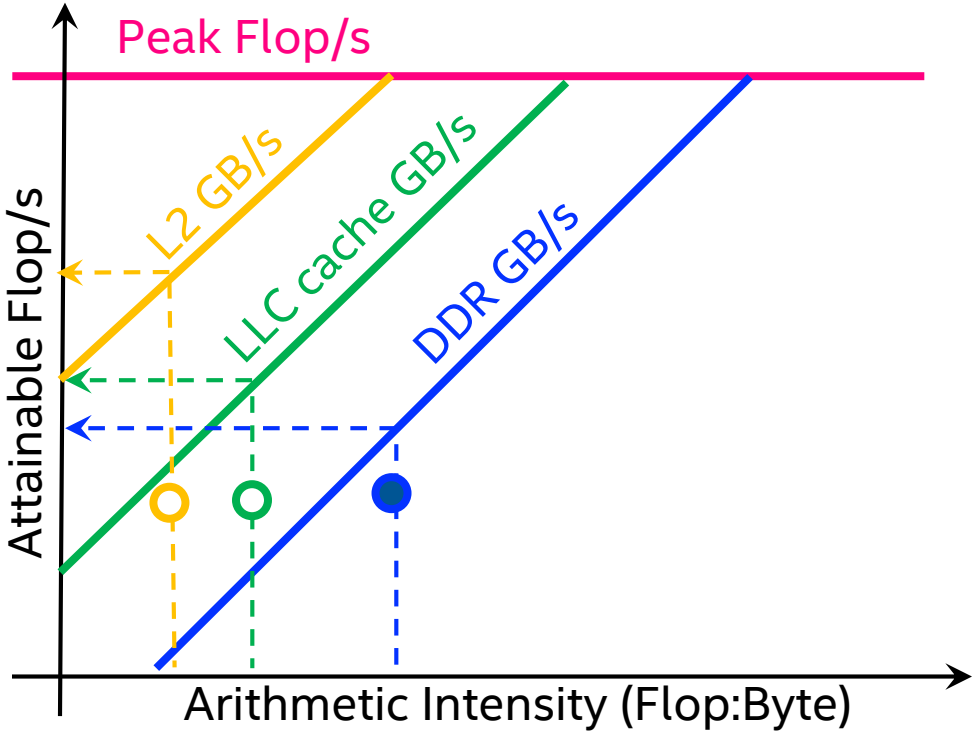
3 Integrated Roofline. What is my current limit?

Performance is limited by minimum of intercepts:

- L1,
- L2,
- LLC,
- DRAM,
- CPU

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

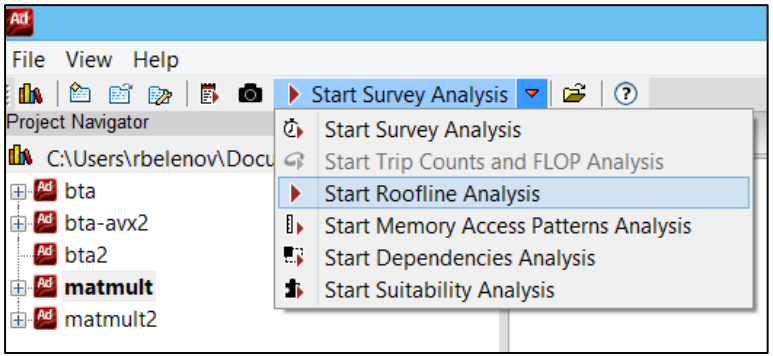
Hierarchical Roofline for one kernel




intel
experience
what's inside™
GETTING ROOFLINE PROFILE

Running roofline collection

GUI



Command line

```
> source advixe-vars.sh
> advixe-cl -collect roofline -project-dir <advisor project>
-- <your app>
```

- Simple click or single command
- **Two collections** are launched one after another
  - Low-overhead sampling to get execution times
  - Binary instrumentation for FLOPs and traffic
- Roofs are obtained via benchmarks

Roofline collection – advanced

Launch two collections explicitly – may be required for MPI

```
> advixe-cl -project-dir <your project> -collect survey -- <your app>
> advixe-cl -project-dir <your project> -collect tripcounts -- <your app>
```

Customize cache configuration

```
> advixe-cl -project-dir <your project> -collect roofline
                                --cache-config 4:1024k:32w -- <your app>
```

- “What if” analysis/projection
- More info in backup slides

How to generate **IRM** Roofline profile?*

```
$ source advixe-vars.sh
```

```
$ export ADVIXE_EXPERIMENTAL=int_roofline
```

1st method. Not compatible with MPI applications :

```
$ advixe-cl -collect roofline -enable-cache-simulation --project-dir
./your_project -- <your-executable-with-parameters>
```

2nd method (compatible with MPI applications and more flexible):

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

```
$ advixe-cl -collect tripcounts -enable-cache-simulation -flop --project-dir
./your_project -- <your-executable-with-parameters>
```

(optional) copy data to your UI desktop system

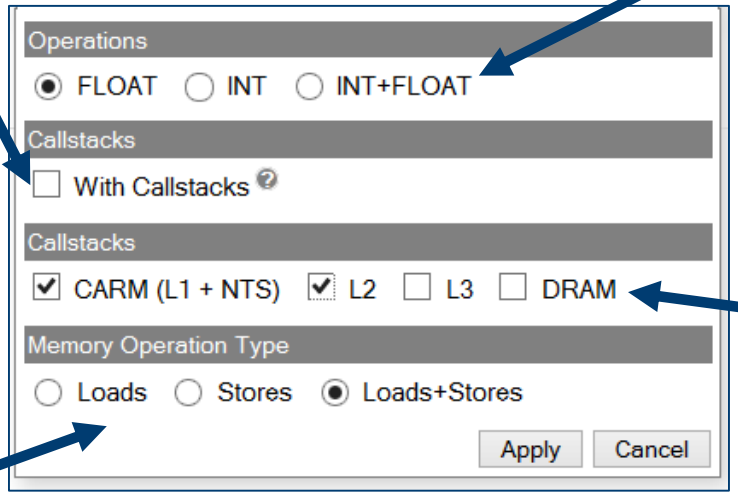
```
$ advixe-gui ./your_project
```

IRM How-to: https://software.intel.com/en-us/articles/integrated-roofline-model-with-intel-advisor

Chart configuration

Aggregate data over calltree (currently works only for CARM, not integrated roofline!)

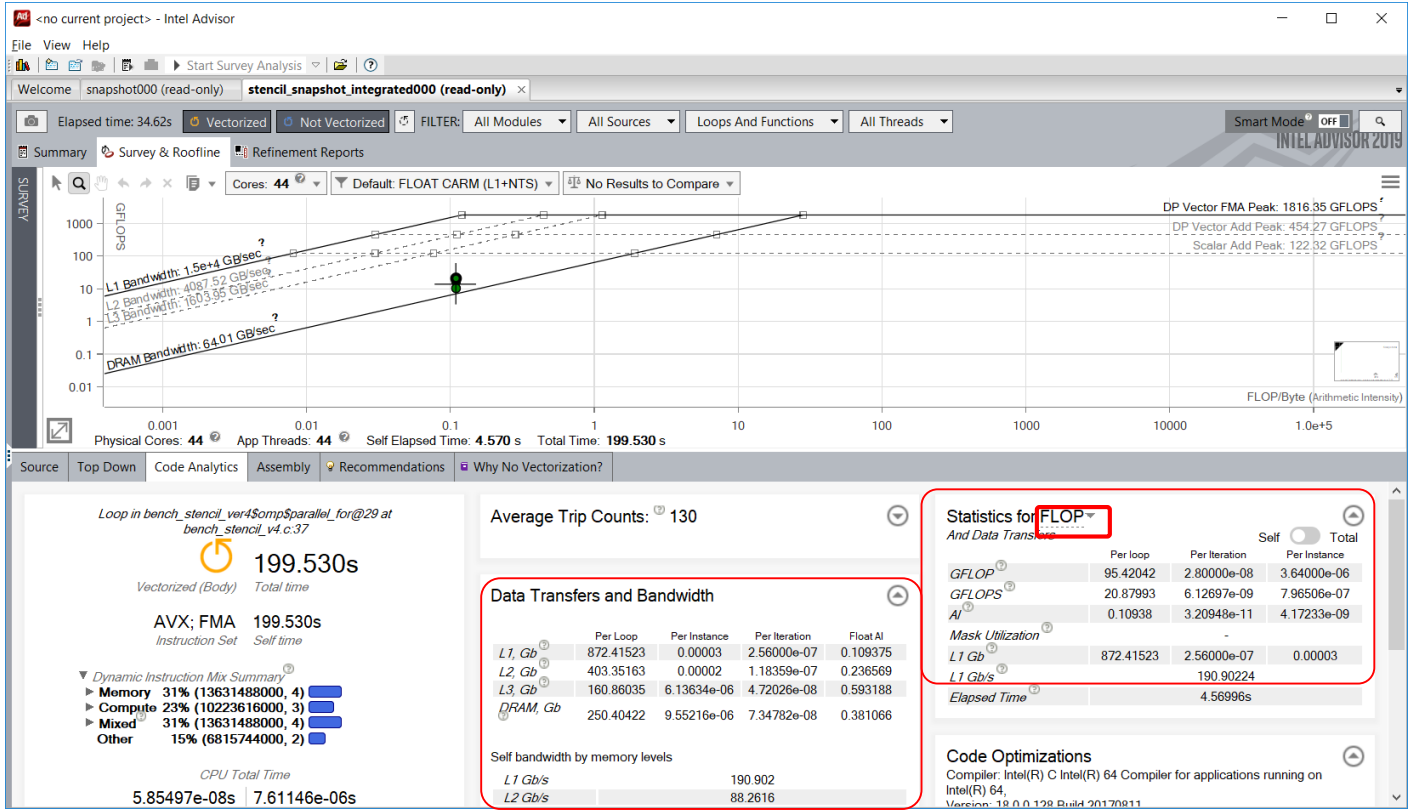
Select which operations are counted



Select memory levels

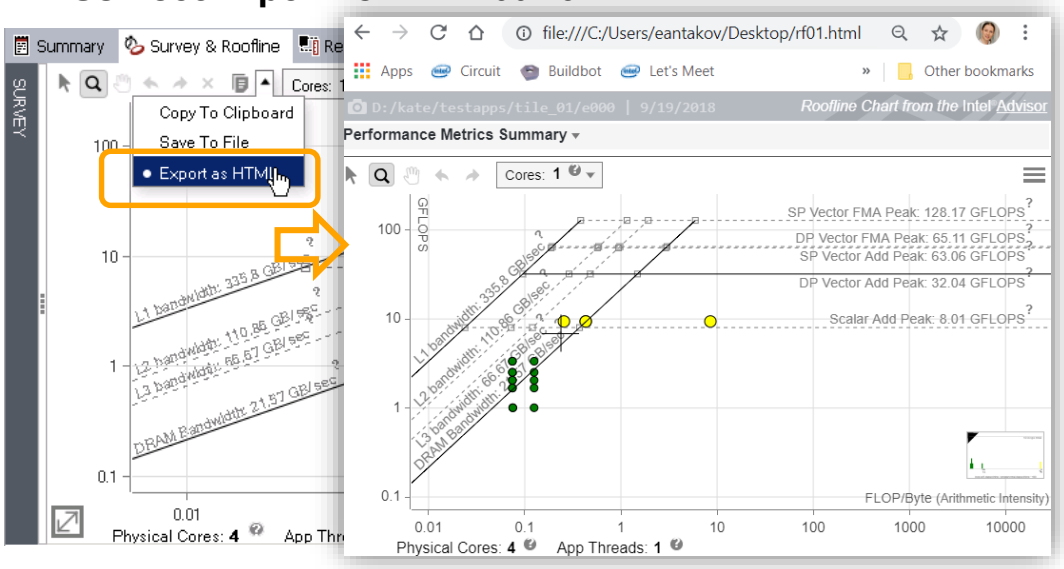
Select only loads or stores

Use “Code Analytics” in conjunction with Roofline



Exporting Integer and Integrated Roofline as HTML

GUI: Use **Export as HTML** button



Command line:

export ADVIXE_EXPERIMENTAL=int_roofline

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

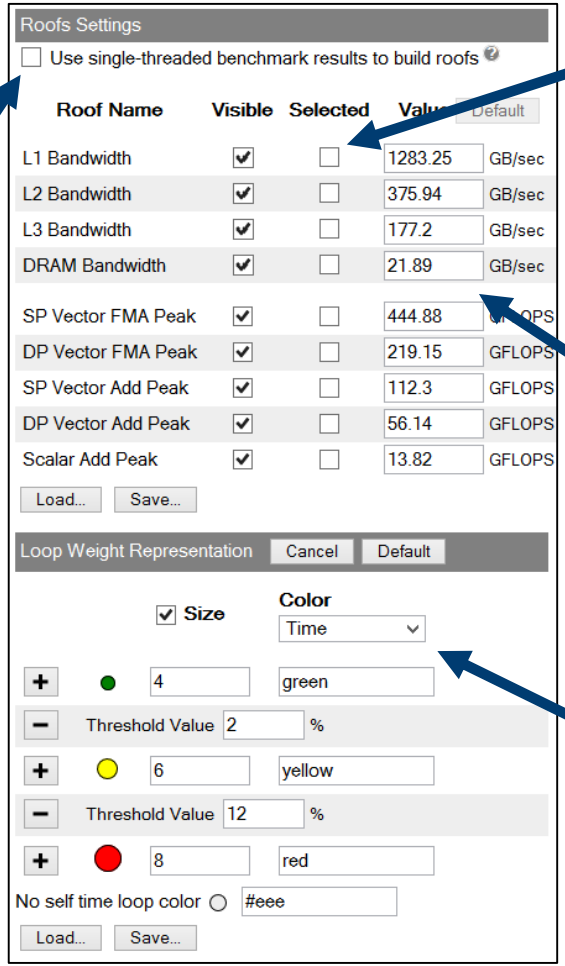
Possible
- data types: float, int, mixed
- memory levels: L1, L2, L3, DRAM
- memory operation types: load, store, all

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

Roofs configuration (and more…)

Customize roof calculation:
- Fully loaded system assumed by default
- Tick if most cores are idle

Change hotspot classification

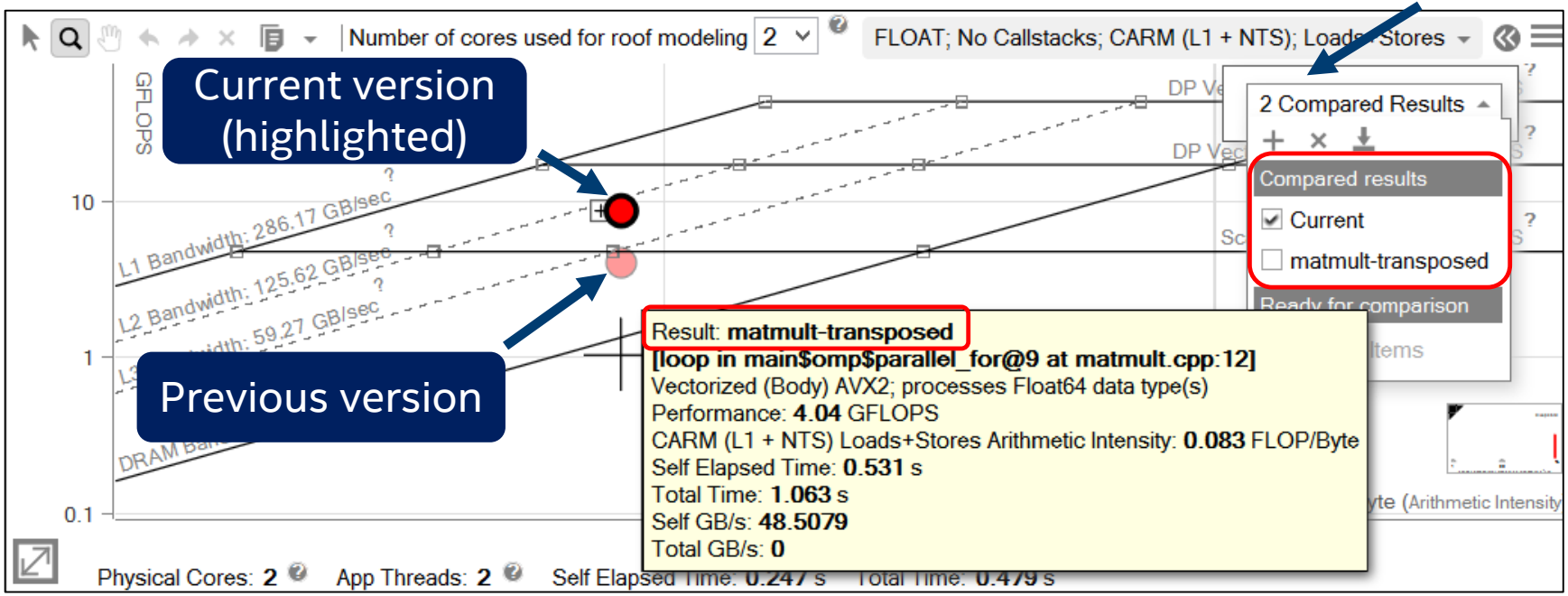


Hide or highlight the roofs

Edit values manually (if needed)

Choose dots coloring strategy (vector-vs-scalar or slow-vs-fast)

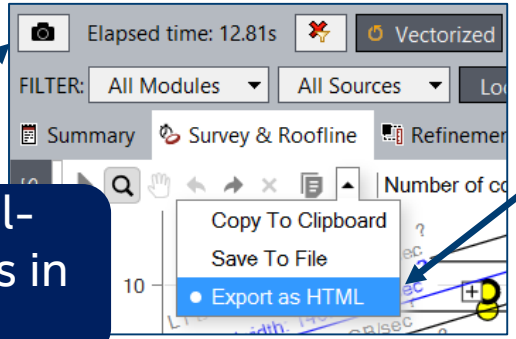
Compare results



Easy to check optimization progress

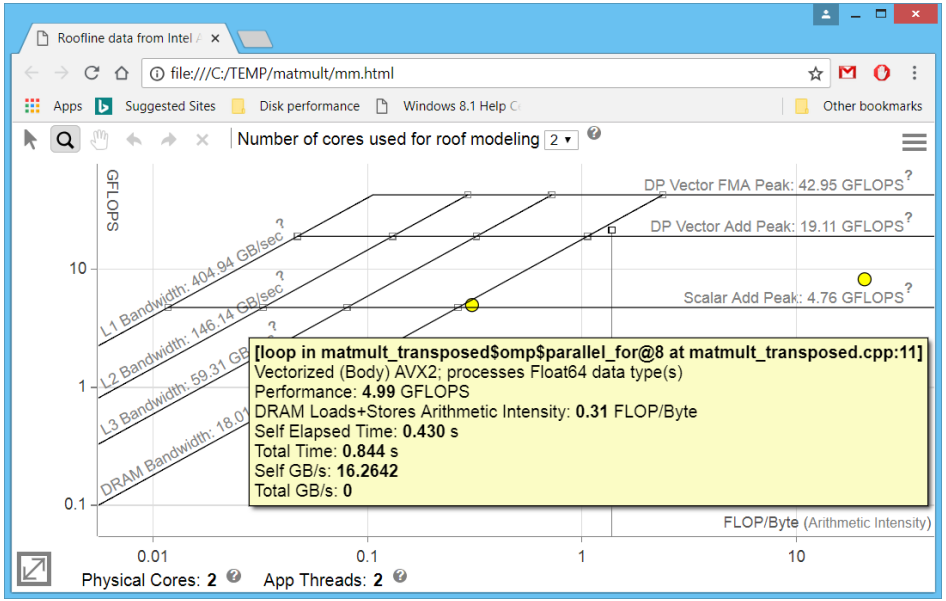
Share with others

Snapshot (full-featured, opens in Advisor)



Standalone ***interactive*** HTML (limited functionality)

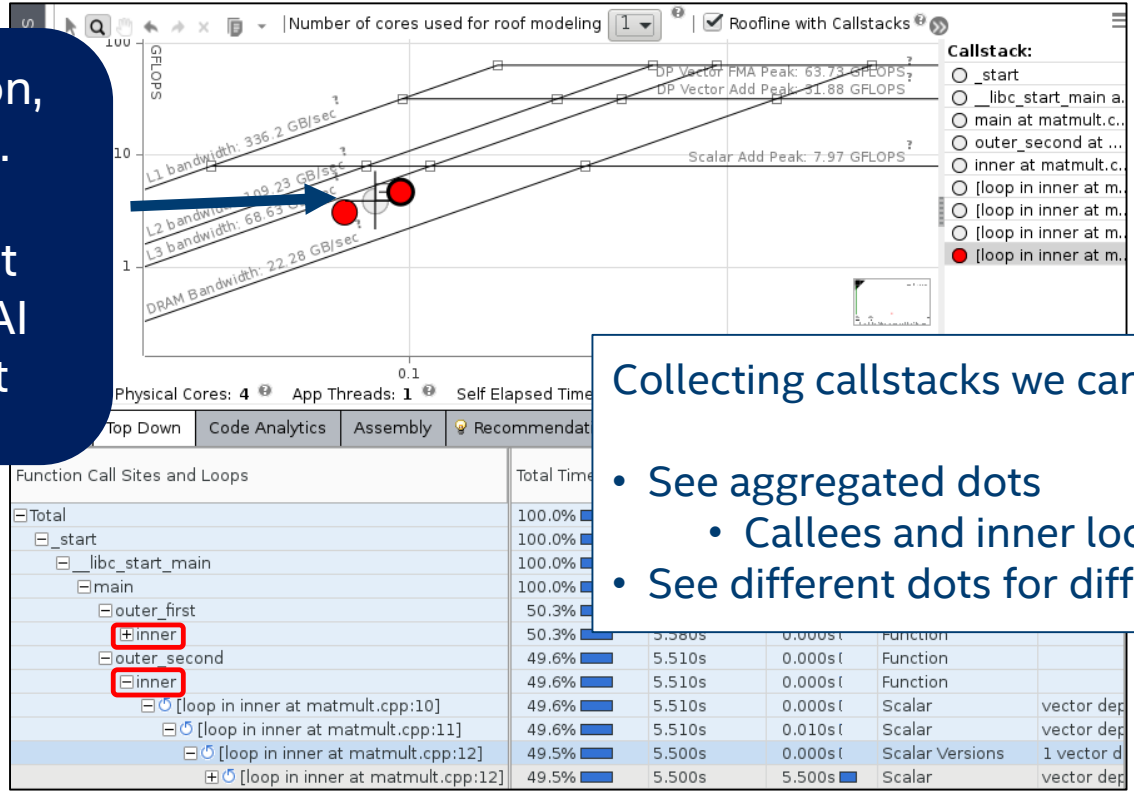
Share roofline by email! - with colleagues or your manager



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

Python API

- Fairly new Advisor Extensibility/customization mechanism. Actively used internally in Intel
- Up to 500 metrics for each loop/function. Really easy to use:
  - <advisor_install_dir>/pythonapi/examples
  - $python ***survey_bottomup.py*** *<project_dir>*
  - Generate your own customized roofline charts

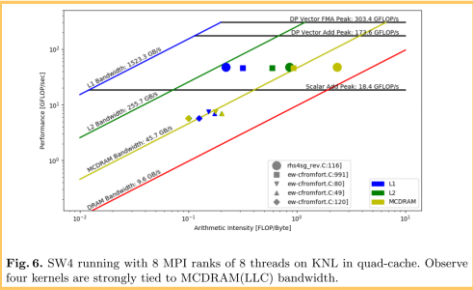


Fig. 6. SW4 running with 8 MPI ranks of 8 threads on KNL in quad-cache. Observe four kernels are strongly tied to MCDRAM(LLC) bandwidth.

From ISC'18 paper (cudos Tuomas)

```
import sys

try:
    import advisor

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

survey = project.load(advisor.SURVEY)

for row in survey.bottomup:
    # row as string
    print(row)
    # row as iterator
    for key in row:
        # row as dictionary
        print('{}: {}'.format(key, row[key]))
```