

Tutorial: 2pm-6pm June 16, Room Expose

Performance Optimization of Scientific Codes with the Roofline Model

Charlene Yang

National Energy Research Scientific Computing Center
Lawrence Berkeley National Laboratory
cjyang@lbl.gov

Zakhar Matveev

Intel Corporation
zakhar.a.matveev@intel.com

Aleksandar Ilic, Diogo Marques

INESC-ID, Instituto Superior Técnico
Universidade de Lisboa
{aleksandar.ilic, diogo.marques}@inesc-id.pt



Tutorial: 2pm-6pm June 16, Room Expo

Performance Optimization of Scientific Codes with the Roofline Model

- 2:00 - 2:40 Introduction to the Roofline Model - Charlene Yang
- 2:40 - 3:20 Roofline Analysis on NVIDIA GPUs - Charlene Yang
- 3:20 - 3:50 Cache-Aware Roofline Model (CARM) - Aleksandar Ilic
- 3:50 - 4:00 Installation of Intel Advisor - Zakhar Matveev
- 4:00 - 4:30 Break
- 4:30 - 4:45 Introduction to Intel Advisor - Zakhar Matveev
- 4:45 - 5:25 Roofline Analysis on Intel CPUs - Zakhar Matveev
- 5:25 - 5:55 Application Use Cases - Diogo Marques
- 5:55 - 6:00 Q&A - All

Please complete the survey afterwards!

Find this tutorial at <https://2019.isc-program.com/> and click 'Give Feedback'



Related Talks at ISC

Tutorial at Intel Developer Connect:

- **Roofline Model-Based Optimization Guidance and Tuning for Modern CPUs**
- **Zakhar A. Matveev, Diogo Marques, Aleksandar Ilic**
- **Mon Jun 17 9am - 3:30pm, Matterhorn 1, Movenpick**
- **<https://inteleventexpress.com/ISC2019/isc.htm>**

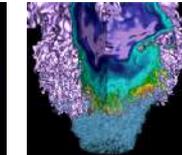
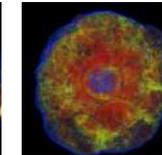
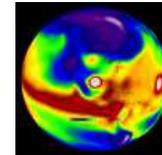
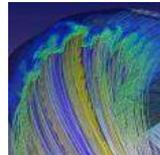
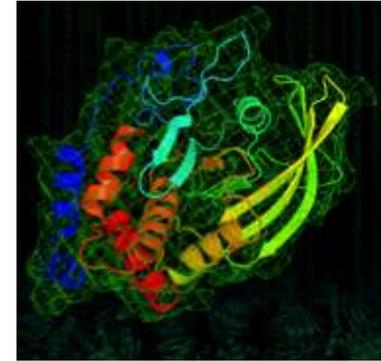
Material Download

- **Google Drive:**
 - Slides
 - Intel Advisor remote viewers (Linux/Mac/Windows)
 - Stencil code snapshot

<http://bit.ly/isc19roofline>

- **USB Sticks:**
 - Please pass them around after finishing copying

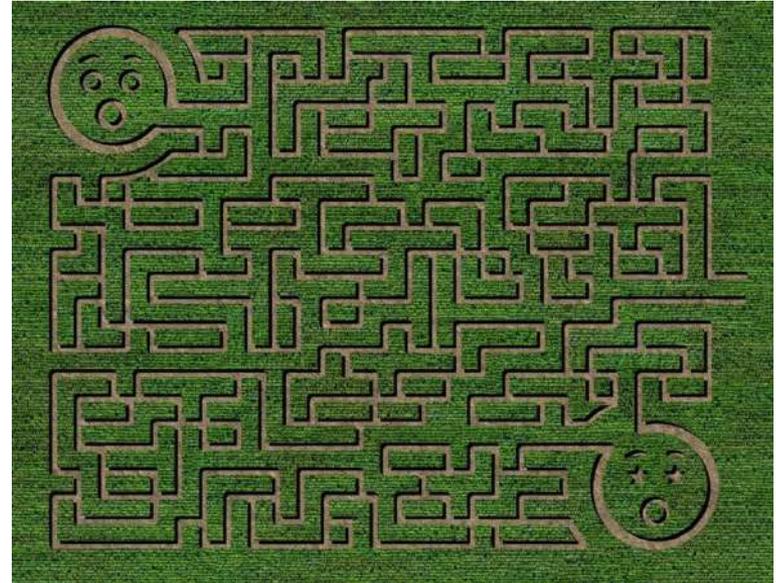
Introduction to the Roofline Model



Charlene Yang

Lawrence Berkeley National Laboratory

Jun 16 2019, Frankfurt



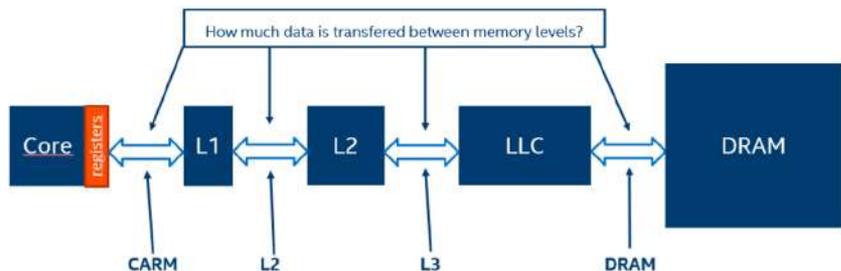
The Maze of Performance Optimization

The Map !!!

Performance Models

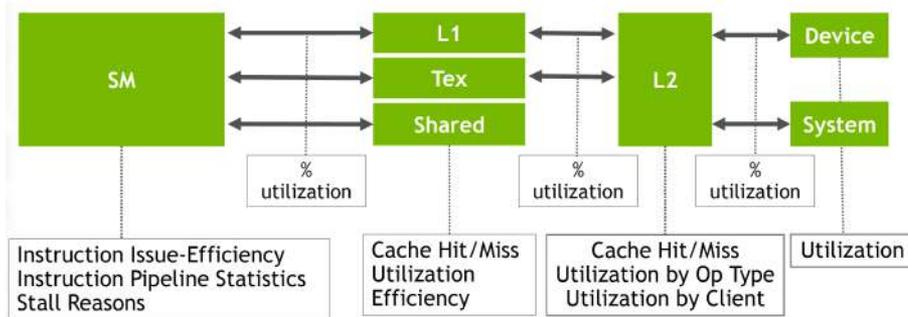


Modern architectures are complicated!



Intel Haswell CPU¹

NVIDIA Volta GPU²



Performance Models



- Many components contribute to the kernel run time
- An interplay of application characteristics and machine characteristics

#FP operations	FLOP/s
Cache data movement	Cache GB/s
DRAM data movement	DRAM GB/s
PCIe data movement	PCIe bandwidth
MPI Message Size	Network Bandwidth
MPI Send:Wait ratio	Network Gap
#MPI Wait's	Network Latency
IO	File systems

Roofline Model



Roofline Performance Model



- Sustainable performance is bound by

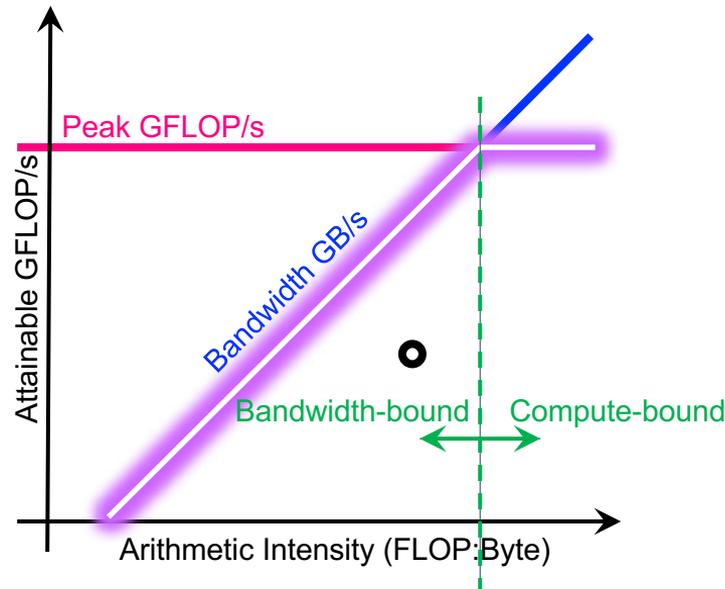
$$\text{GFLOP/s} = \min \begin{cases} \text{Peak GFLOP/s} \\ \text{AI} * \text{Peak GB/s} \end{cases}$$

- Arithmetic Intensity (AI) =

$$\text{FLOPs} / \text{Bytes}$$

- How did this come about?

→ A CPU DRAM example



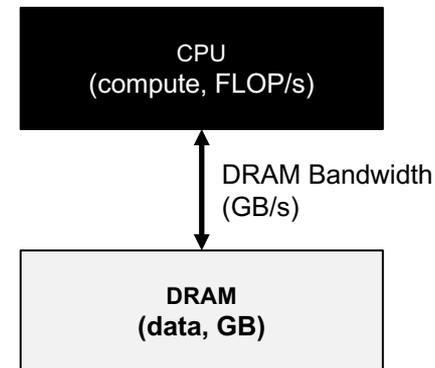
Transition @ AI ==
Peak GFLOP/s / Peak GB/s ==
'Machine Balance'

(CPU DRAM) Roofline



- One could hope to always attain peak performance (FLOP/s)
- However, finite locality (reuse) and bandwidth limit performance.
- Assume:
 - Idealized processor/caches
 - Cold start (data in DRAM)

$$\text{Time} = \max \left\{ \begin{array}{l} \#FP \text{ ops} / \text{Peak GFLOP/s} \\ \#Bytes / \text{Peak GB/s} \end{array} \right.$$

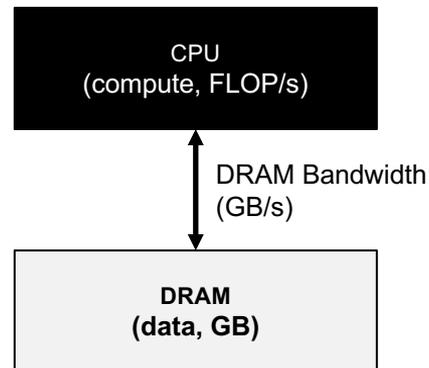


(CPU DRAM) Roofline



- One could hope to always attain peak performance (FLOP/s)
- However, finite locality (reuse) and bandwidth limit performance.
- Assume:
 - Idealized processor/caches
 - Cold start (data in DRAM)

$$\frac{\text{Time}}{\#FP\ ops} = \max \left\{ \begin{array}{l} 1 / \text{Peak GFLOP/s} \\ \#Bytes / \#FP\ ops / \text{Peak GB/s} \end{array} \right.$$

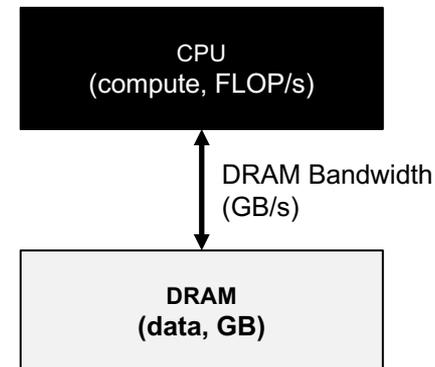


(CPU DRAM) Roofline



- One could hope to always attain peak performance (FLOP/s)
- However, finite locality (reuse) and bandwidth limit performance.
- Assume:
 - Idealized processor/caches
 - Cold start (data in DRAM)

$$\frac{\#FP\ ops}{Time} = \min \left\{ \begin{array}{l} \text{Peak GFLOP/s} \\ (\#FP\ ops / \#Bytes) * \text{Peak GB/s} \end{array} \right.$$

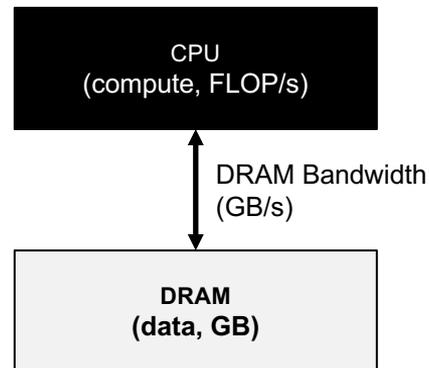


(CPU DRAM) Roofline



- One could hope to always attain peak performance (FLOP/s)
- However, finite locality (reuse) and bandwidth limit performance.
- Assume:
 - Idealized processor/caches
 - Cold start (data in DRAM)

$$\text{GFLOP/s} = \min \left\{ \begin{array}{l} \text{Peak GFLOP/s} \\ \text{AI} * \text{Peak GB/s} \end{array} \right.$$



Arithmetic Intensity (AI) = FLOPs / Bytes (as presented to DRAM)

Roofline Performance Model



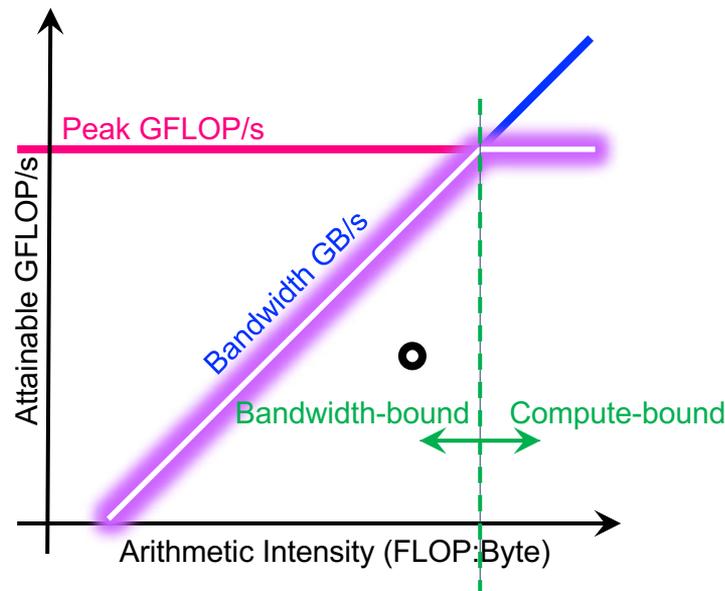
- Thus we obtain the model as

$$\text{GFLOP/s} = \min \begin{cases} \text{Peak GFLOP/s} \\ \text{AI} * \text{Peak GB/s} \end{cases}$$

where Arithmetic Intensity (AI) is

$$\text{FLOPs} / \text{Bytes}$$

- Machine Balance (FLOPs/Byte) =
8.9 (V100, DP, HBM) or **5.1** (KNL, DP, HBM)



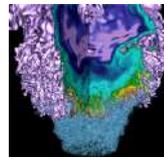
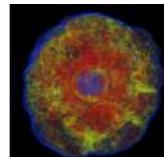
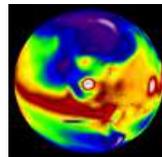
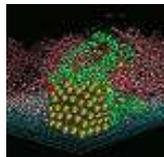
Transition @ AI ==
Peak GFLOP/s / Peak GB/s ==
'Machine Balance'

Roofline Performance Model



- **A throughput-oriented model**
 - tracks rates not times, i.e. GFLOP/s, GB/s, not seconds
- **An abstraction over**
 - architectures, ISA (CPU, GPU, Haswell, KNL, Pascal, Volta)
 - programming models, programming languages
 - numerical algorithms, problem sizes
- **In log-log scale to easily extrapolate performance along Moore's Law**

More Advanced on Roofline

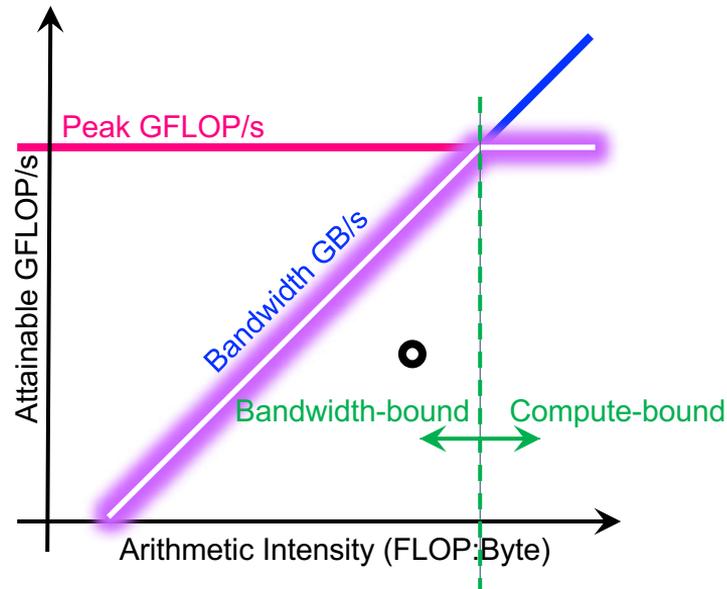


Roofline Performance Model



- This is a single Roofline
- What about the memory hierarchy, different execution configurations, and instruction mixes?

→ Hierarchical Roofline
→ Multiple compute ceilings

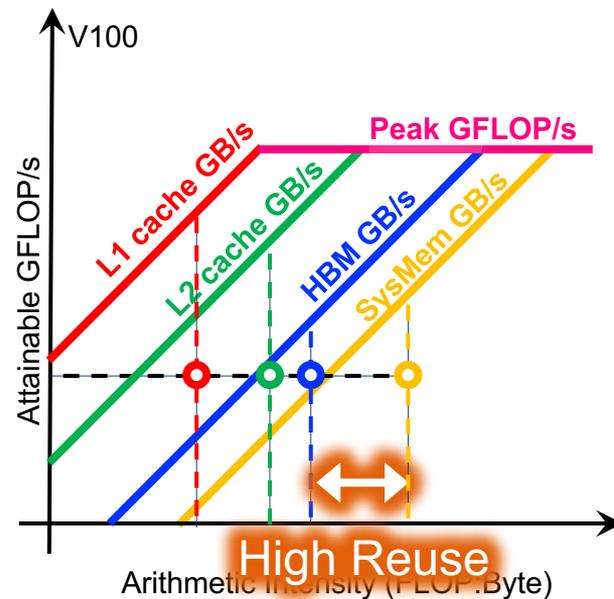


Transition @ AI ==
Peak GFLOP/s / Peak GB/s ==
'Machine Balance'

Hierarchical Roofline



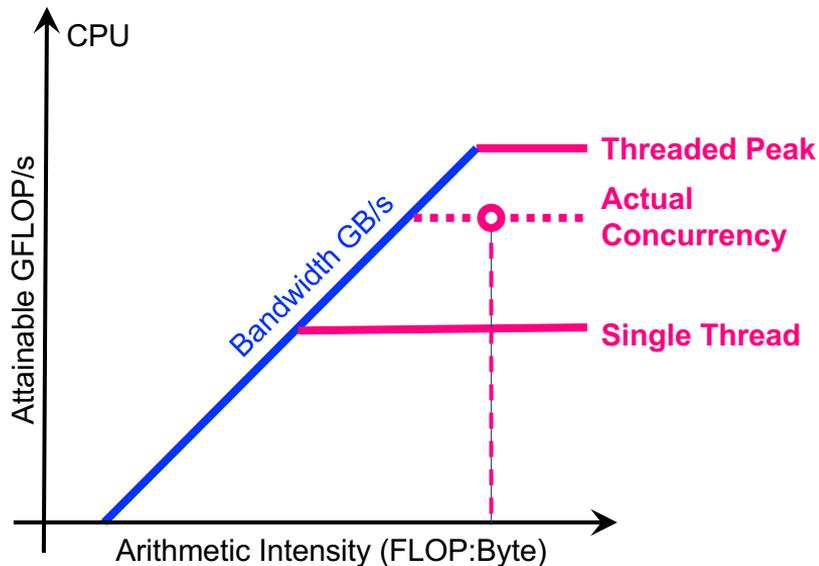
- Superposition of multiple Rooflines
 - Incorporate full memory hierarchy
 - Arithmetic Intensity =
$$\text{FLOPs} / \text{Bytes}_{L1/L2/HBM/SysMem}$$
- Each kernel will have multiple AI's but one observed GFLOP/s performance
- Hierarchical Roofline tells you about **cache locality**



Multiple Compute Ceilings



- Impact of **execution configuration**
- **Concurrency affects your peak**
 - OpenMP thread concurrency
 - SM occupancy
 - load balance
 - threadblock/thread configuration

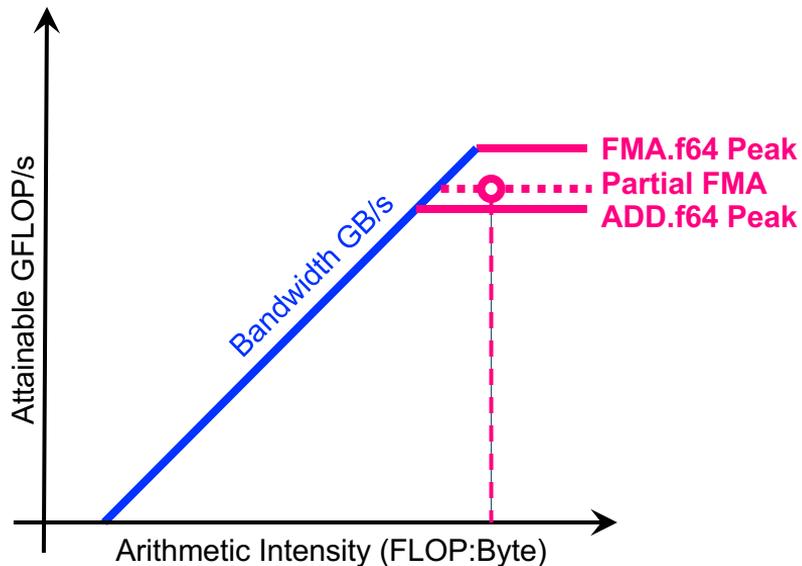


- Performance is bound by the **actual concurrency** ceiling

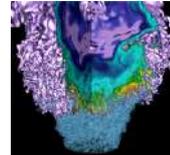
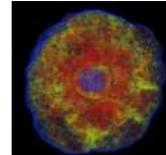
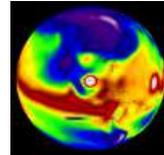
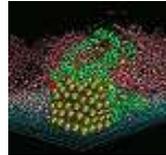
Multiple Compute Ceilings



- Impact of **instruction mix**
- Applications are usually a mix of FMA.f64, ADD.f64, MUL.f64...
- Performance is a **weighted** average ... bound by a **partial FMA** ceiling



Roofline Drives Optimization



Roofline Performance Model



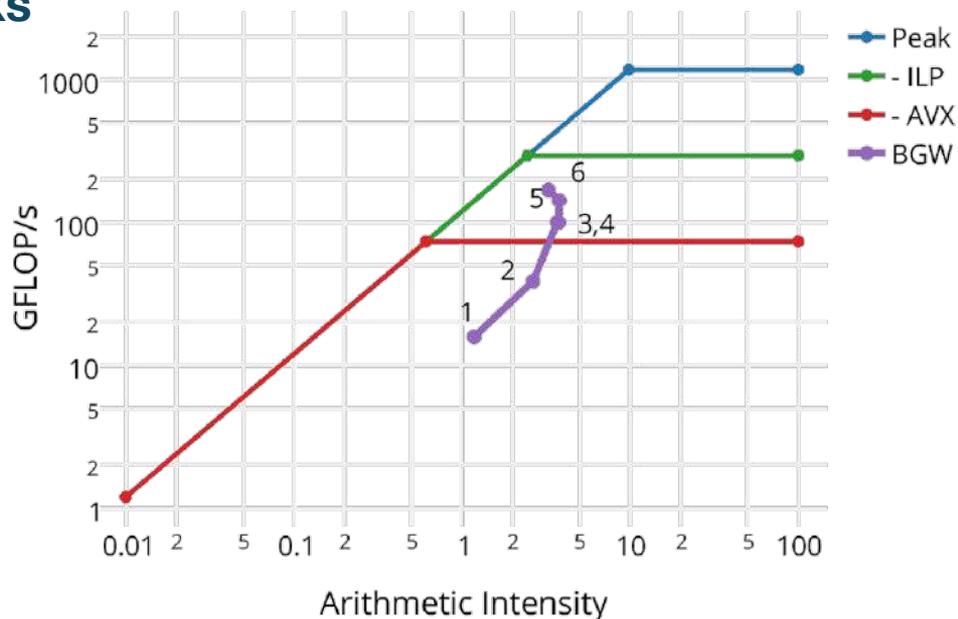
The Roofline Model

- helps you identify the bottlenecks
- guides you through optimization
- tells you when to stop

An example:

- NESAP for Cori - BerkeleyGW

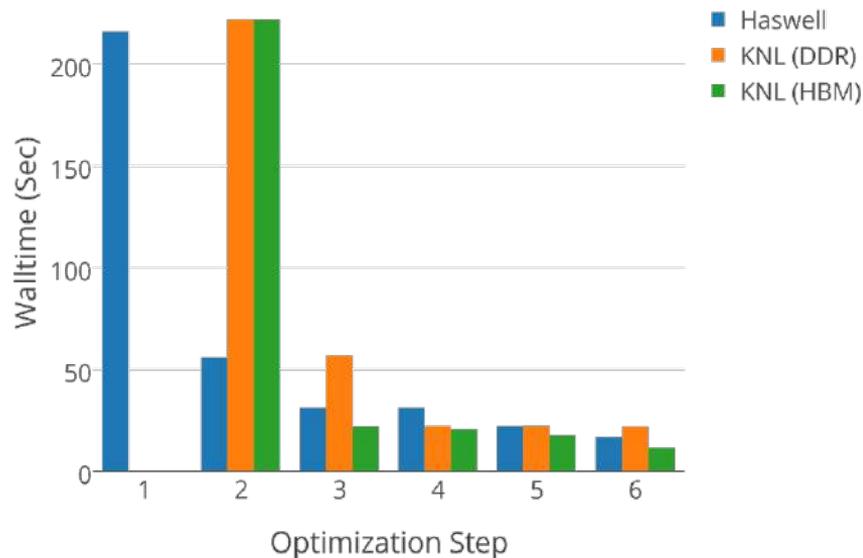
Haswell Roofline Optimization Path



Optimization Path for Kernel-C (Sigma):

1. Add OpenMP
2. Initial Vectorization
 - loop reordering
 - conditional removal
3. Cache-Blocking
4. Improved Vectorization
 - divides
5. Hyper-threading

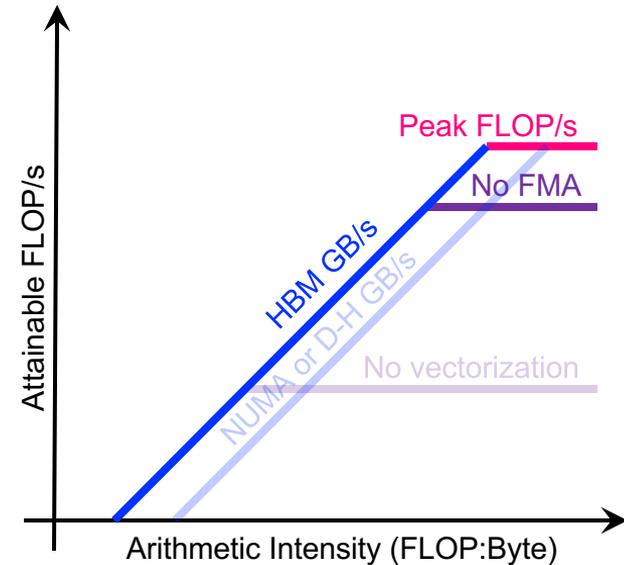
Sigma Optimization Process



General Optimization Strategy



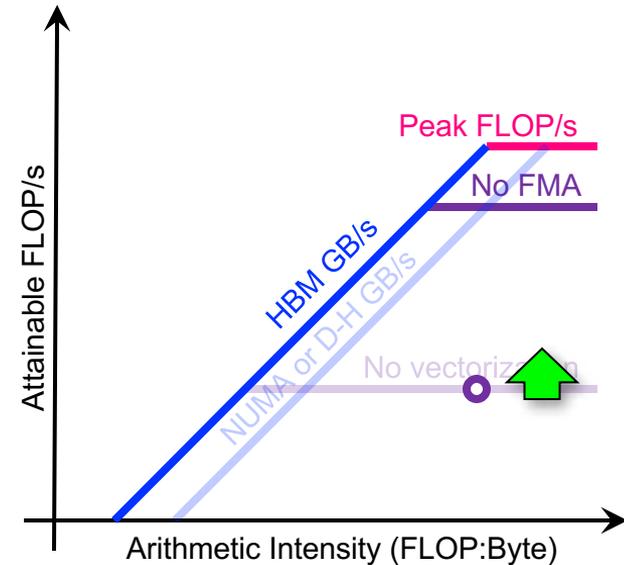
- Broadly speaking, three approaches to improving performance:



General Optimization Strategy



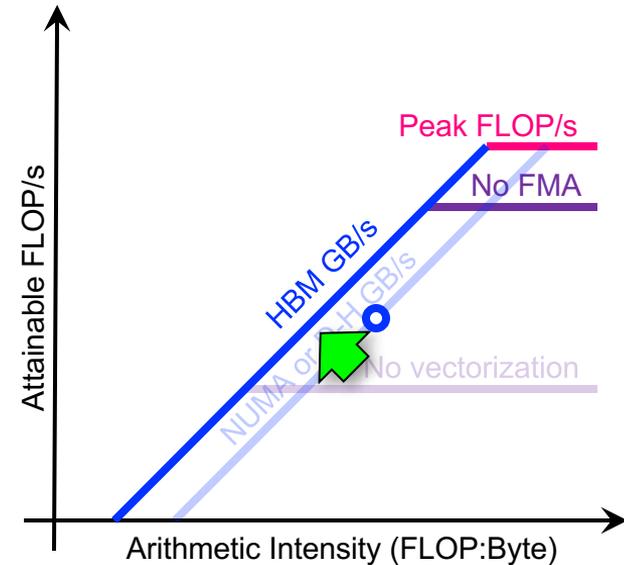
- Broadly speaking, three approaches to improving performance:
- **Maximize compute performance**
 - multithreading
 - vectorization
 - increase SM occupancy
 - utilize FMA instructions
 - minimize thread divergence



General Optimization Strategy



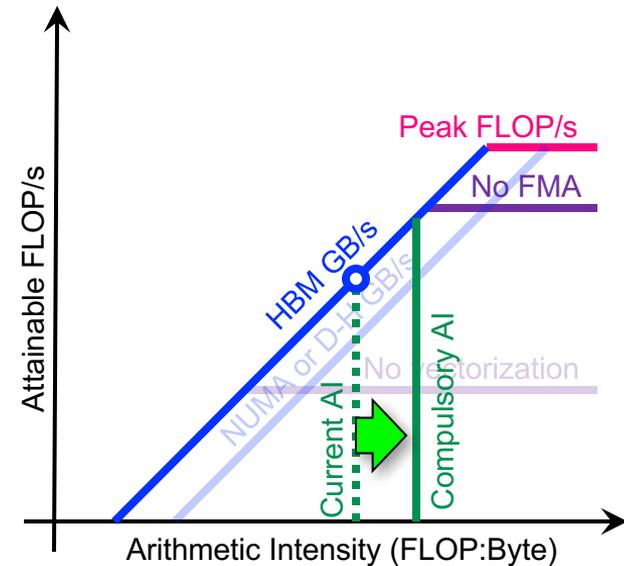
- Broadly speaking, three approaches to improving performance:
- **Maximize compute performance**
- **Maximize memory bandwidth**
 - utilize higher-level caches
 - NUMA-aware allocation
 - avoid H-D transfers
 - avoid uncoalesced memory access



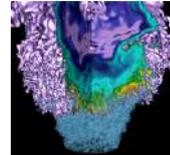
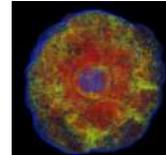
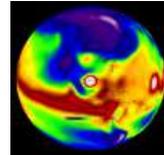
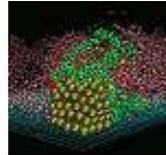
General Optimization Strategy



- Broadly speaking, three approaches to improving performance:
- **Maximize compute performance**
- **Maximize memory bandwidth**
- **Improve AI**
 - minimize data movement
 - exploit cache reuse



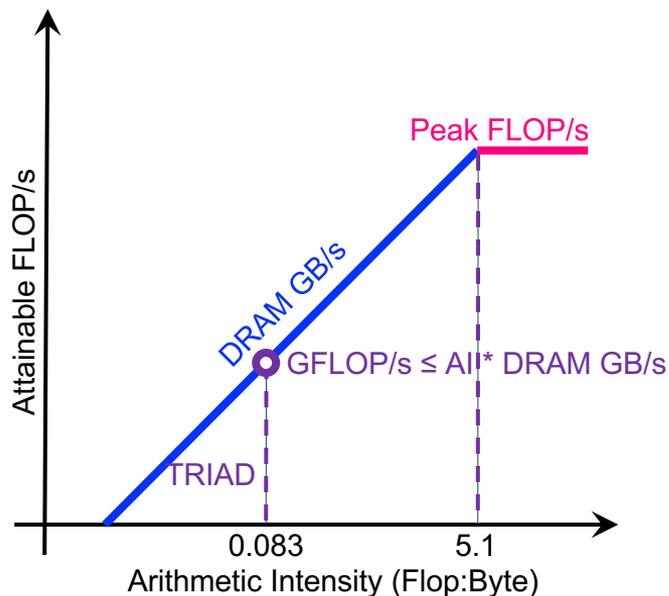
Roofline Data Collection



- Example #1: STREAM Triad

```
for(i=0;i<N;i++){  
    Z[i] = X[i] + alpha*Y[i];  
}
```

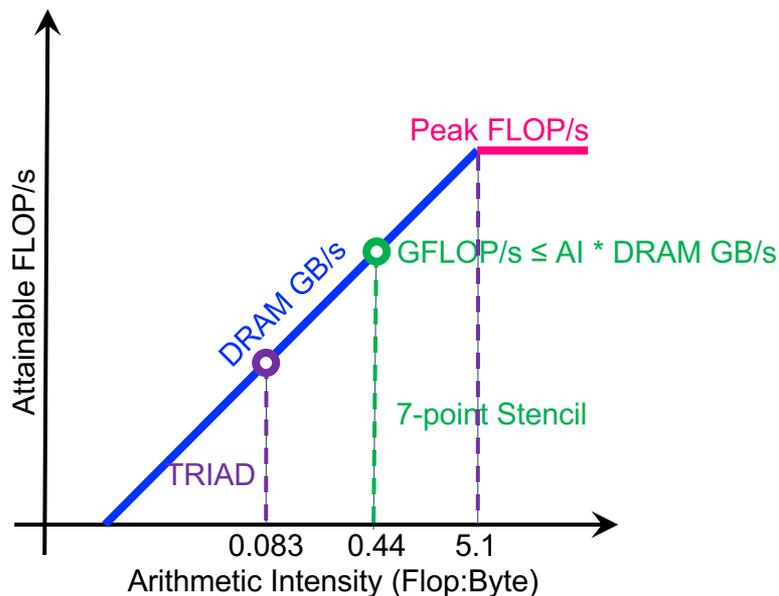
- 2 FLOPs per iteration
- Transfer 24 bytes per iteration
 - read X[i], Y[i], and write Z[i]
- **AI = 0.083 FLOPs per byte**
- **Memory bound**



- **Example #2: 7-pt stencil**
 - 7 FLOPs; 8 memory references (7 reads, 1 store) per pt
 - Cache can filter all but 1 read and 1 write per pt
 - **AI = 0.44 FLOPs per byte**
 - **Memory bound, but 5x the GFLOP/s rate**

```
for(k=1;k<dim+1;k++){
for(j=1;j<dim+1;j++){
for(i=1;i<dim+1;i++){
    new[k][j][i] = -6.0*old[k ][j ][i ]
                    + old[k ][j ][i-1]
                    + old[k ][j ][i+1]
                    + old[k ][j-1][i ]
                    + old[k ][j+1][i ]
                    + old[k-1][j ][i ]
                    + old[k+1][j ][i ];
}}}

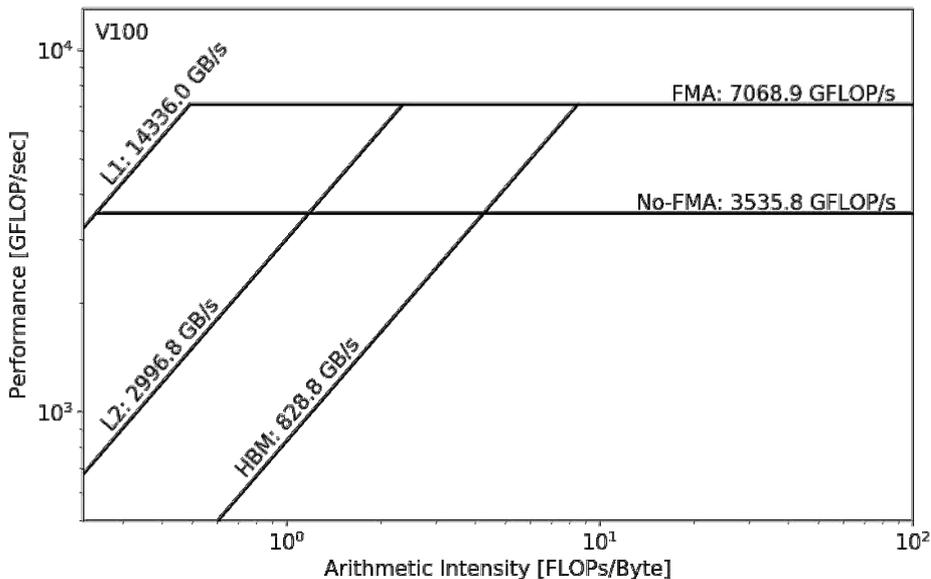
```



- Not scalable for real-life applications
- Millions of lines of code; mix of different languages
- Complicated modern architecture
 - memory hierarchy, caching effects
 - ISA
- Different problem sizes



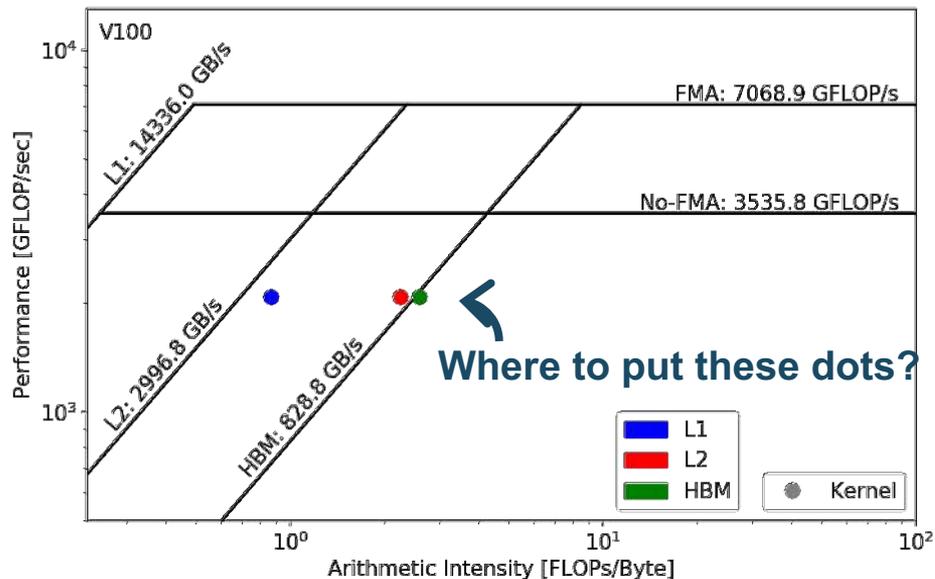
We Need Tools!



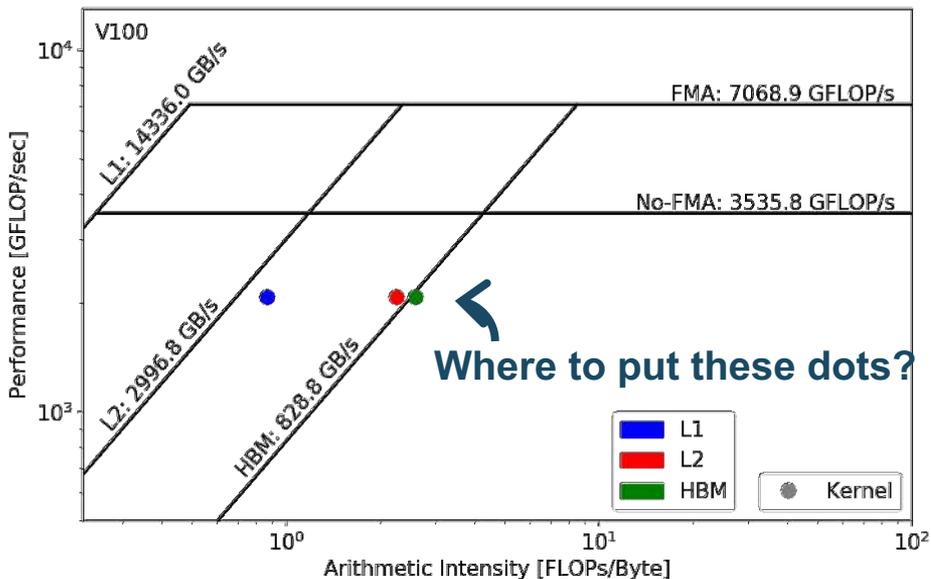
- Roofline ceilings

- vendor specifications
- empirical measurements
 - ERT
 - <https://bitbucket.org/berkeleylab/cs-roofline-toolkit>

We Need Tools!



We Need Tools!



Require three raw measurements:

- Runtime
- FLOPs
- Bytes (on each cache level)

In order to calculate AI and GFLOP/s:

$$\text{Arithmetic Intensity} = \frac{\text{FLOPs}}{\text{Data Movement}} \quad (\text{FLOPs/Byte})$$

$$\text{Performance} = \frac{\text{FLOPs}}{\text{Runtime}} \quad (\text{GFLOP/s})$$

1. Collect Roofline ceilings

- **compute** (FMA/no FMA) and **bandwidth** (DRAM, L2, ...)
- ERT: <https://bitbucket.org/berkeleylab/cs-roofline-toolkit>

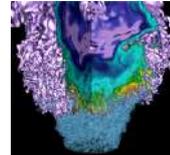
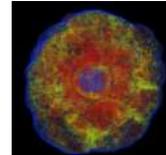
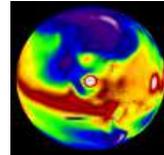
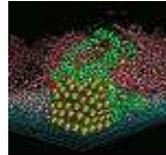
2. Collect application performance

- **FLOPs**, **bytes** (DRAM, L2, ...), **runtime**
- SDE, VTune, LIKWID, Advisor, nvprof, ...

3. Plot Roofline with Python Matplotlib (or other tools of your preference)

- **arithmetic intensity**, **GFLOP/s** performance, **ceilings**
- example scripts: <https://github.com/cyanguwa/nersc-roofline>

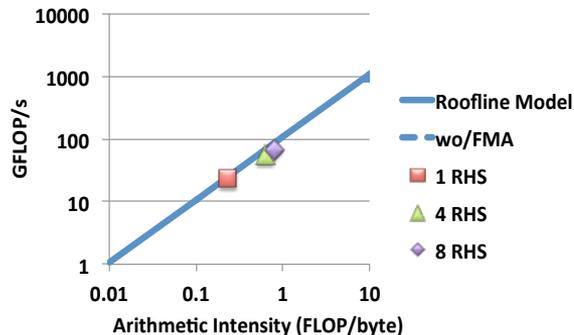
Automated Data Collection



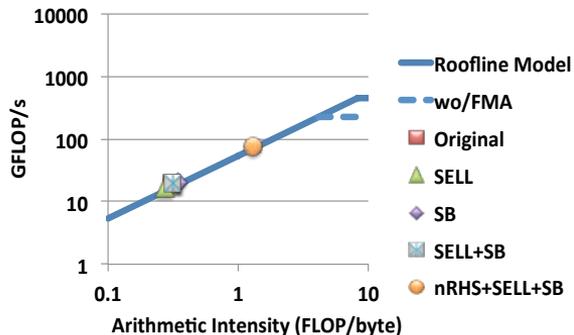
Data Collection on Intel CPUs



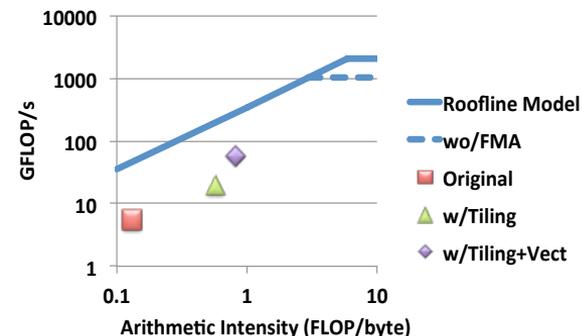
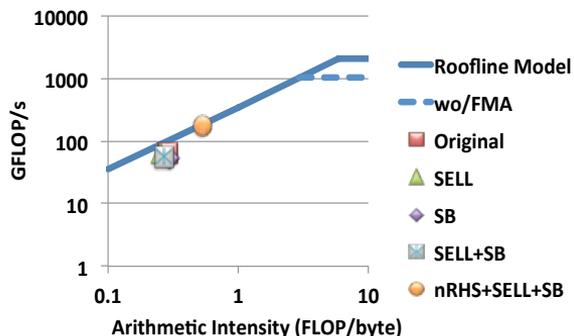
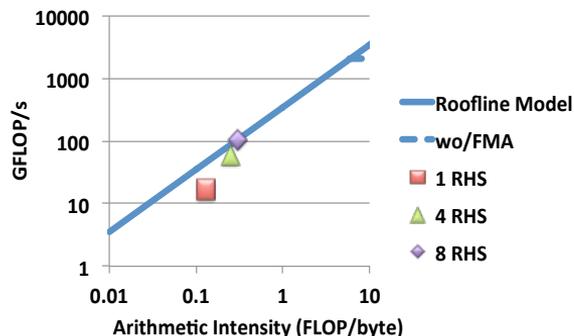
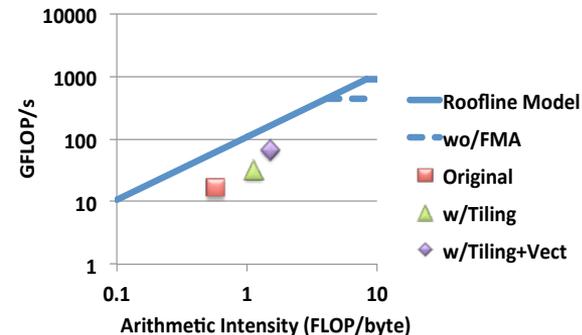
MFDn



EMGeo



PICSAR



Data Collection on Intel CPUs

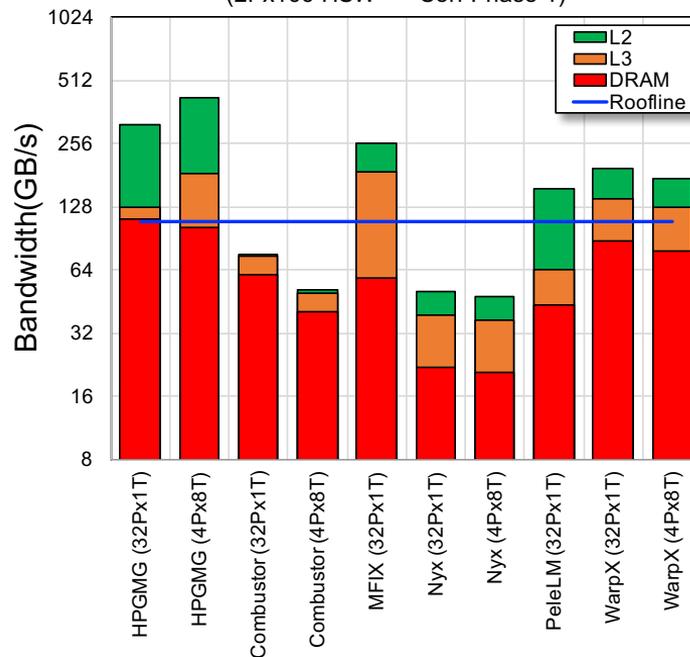


The not-so-automated way 2:

- **LIVID** for FLOPs and bytes
 - Both are based on HW counters
- Runtime
- **Hierarchical** Roofline
- Limited by quality of HW counters
- High-level characterization, no callstack

AMReX Application Characterization

(2Px16c HSW == Cori Phase 1)



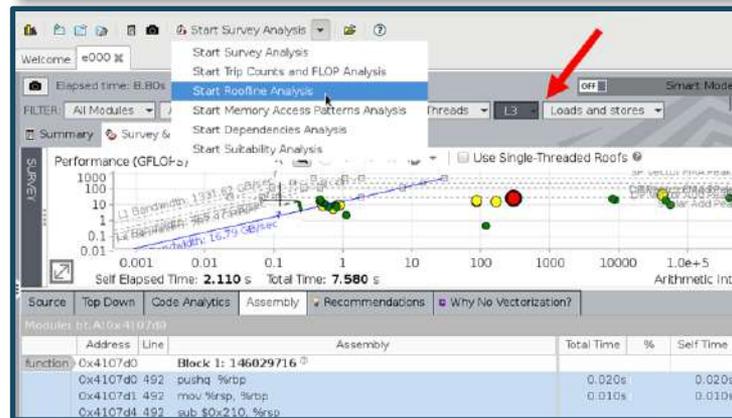
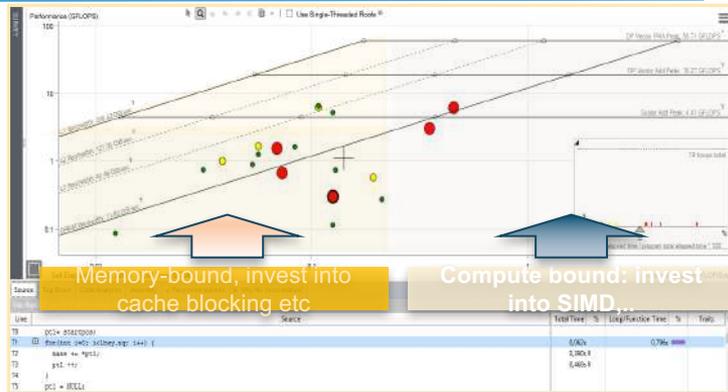
<https://github.com/RRZE-HPC/likwid>

Data Collection on Intel CPUs



The fully automated way:

- Intel Advisor, Roofline feature
- Instrument applications **automatically**
 - one dot per loop nest/function
- **FLOPs, bytes and runtime**
- **Hierarchical Roofline**
- Integrates with other Advisor capabilities
- Benchmarks target system



- Still very manual at this stage, but...
- Runtime:
 - Internal timers or `nvprof --print-gpu-trace`
- FLOPs:
 - DP/SP/HP counters and metrics, `nvprof --metrics 'flop_count_dp/sp/hp'` or `'tensor_precision_fu_utilization'`
- Bytes for different cache levels:
 - Bytes = (read transactions + write transactions) x transaction size
 - `nvprof --metrics 'metric_name'` e.g. `gld/gst_transactions`
- **Hierarchical Roofline**

- The Roofline Model formulizes the interaction between machine characteristics and application characteristics, and guides optimization
 - Peak computational throughput and bandwidth
 - Arithmetic intensity, cache locality, instruction mix...
- Automate Roofline data collection
 - Intel CPUs
 - Intel SDE + Intel VTune, Intel Advisor
 - NVIDIA GPUs
 - **nvprof**, Nsight Compute



**More in the
next few talks!**

- S. Williams, A. Waterman and D. Patterson, “Roofline: An Insightful Visual Performance Model for Multicore Architectures,” *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009
- LBNL CRD Roofline Research:
<https://crd.lbl.gov/departments/computer-science/PAR/research/roofline>
- Empirical Roofline Toolkit (ERT):
<https://bitbucket.org/berkeleylab/cs-roofline-toolkit>
- Python scripts for plotting manually-collected Roofline:
<https://github.com/cyanguwa/nersc-roofline/tree/master/Plotting>

Acknowledgement



- This material is based upon work supported by the Advanced Scientific Computing Research Program in the U.S. Department of Energy, Office of Science, under Award Number DE-AC02-05CH11231.
- This material is based upon work supported by the DOE RAPIDS SciDAC Institute.
- This research used resources of the National Energy Research Scientific Computing Center (NERSC), which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-05CH11231.



Thank You