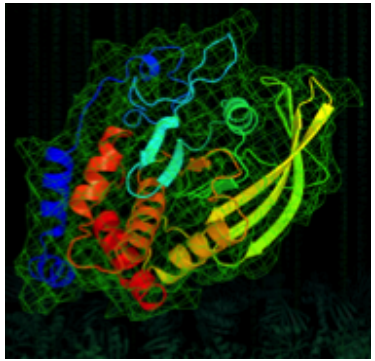
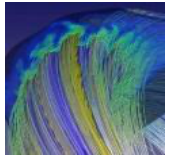
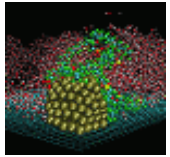
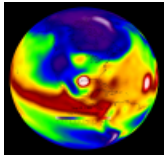
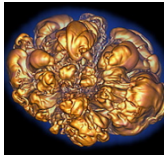
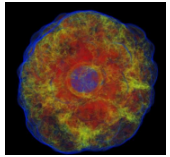
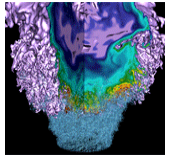
The Current and Future of Roofline



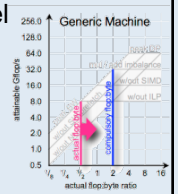
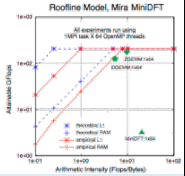
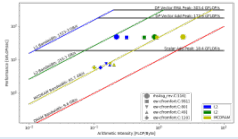
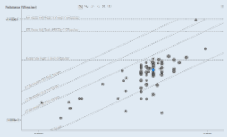
Charlene Yang

Application Performance Specialist
NERSC, LBNL

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

The Roofline Chronical

	2005 - 2011	2013 - 2016	2017 - 2019	Future
Research	▪ Developed **foundations** for the Roofline Model ▪ Applied to kernels using canonical flops and bytes 		▪ Developed performance counter Rooflines for **CPUs and GPUs** ▪ Roofline for **Simulations** and **Machine Learning** ▪ Incorporated VPU%, divides, integer operations	▪ FPGAs, CGRAs, AI processors, … ▪ Asymmetric memory hierarchies ▪ Horizontal data movement ▪ Effects of **extreme heterogeneity**
Prototype		▪ Created the **ERT** prototype for CPUs and GPUs ▪ Quantified CUDA UVM effects 	▪ Collaboration with CRD, Intel and NVIDIA on **hierarchical Roofline** 	▪ **Integer/instruction/non-FP** Rooflines ▪ Rooflines that serialize data transfers (vs. assume overlap) ▪ Integration with compilers/runtimes
Production			▪ Roofline model incorporated into **Intel Advisor** ▪ Installed at NERSC, LANL, *etc* 	▪ **Roofline for GPUs (multiple vendors)** ▪ Roofline for FPGAs/CGRAs ▪ Integer/instruction/non-FP Rooflines ▪ CISC/DL instructions

The Roofline People

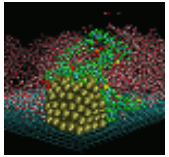
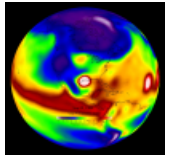
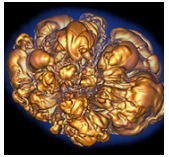
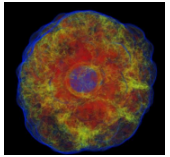
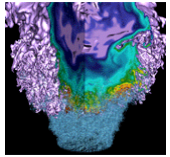
Researchers…

- Sam Williams (Roofline Lead, LBL/CRD)
- Doug Doefler (LBL/NERSC)
- Khaled Ibrahim (LBL/CRD)
- Nan Ding (LBL/CRD)
- Yunsong Wang (LBL/NERSC)
- Jack Deslippe (LBL/NERSC)
- Lenny Oliker (RAPIDS deputy, LBL/CRD)
- Terry Ligocki (LBL/CRD)
- Brian Van Straalen (LBL/CRD)
- Aleksandar Ilic (INESC, Portugal)
- Diogo Marques (INESC, Portugal)

Vendors/Industry…

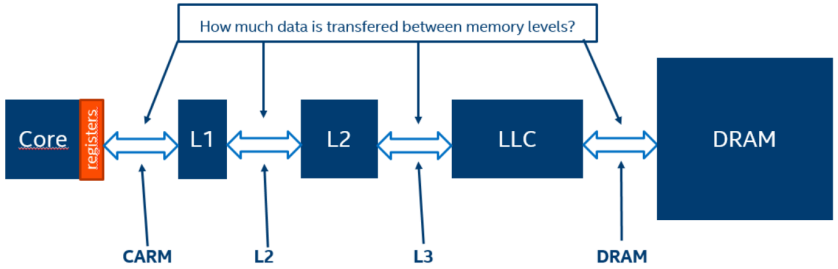
- Zakhar Matveev (Intel)
- Max Katz, Magnus Strengert (NVIDIA)
- Constantios Evangelinos (IBM)
- Protonu Basu (Facebook; formerly LBL/CRD)
- Linda Lo (Facebook; formerly U. Utah)
- David Patterson (Google, formerly UC Berkeley)

What is Roofline?

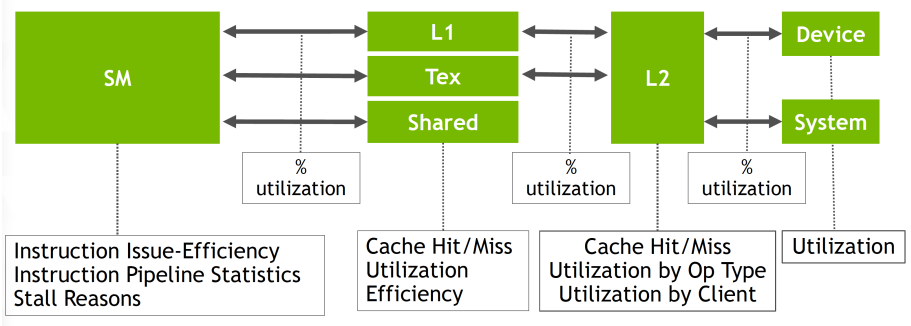
Performance Modelling

Modern architectures are complicated! A holistic view is important!



Intel Haswell CPU[1]

NVIDIA Volta GPU[2]



1. https://software.intel.com/en-us/articles/integrated-roofline-model-with-intel-advisor
2. http://on-demand.gputechconf.com/gtc/2016/presentation/s6659-avinash-baliga-perfworks.pdf

Performance Modelling

- **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

Focus on one or two dominant components!

Roofline Performance Model

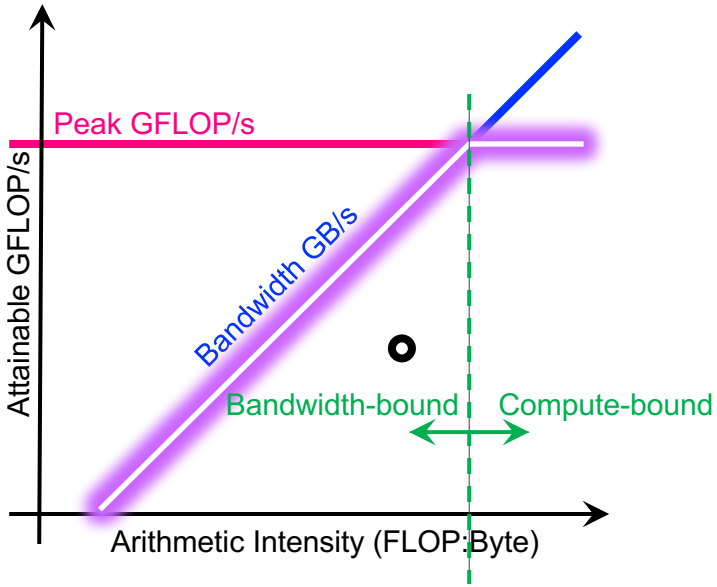
- Sustainable performance is bound by

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

- Arithmetic Intensity (AI) =

FLOPs / Bytes

- How did this come about?
  → A CPU DRAM example

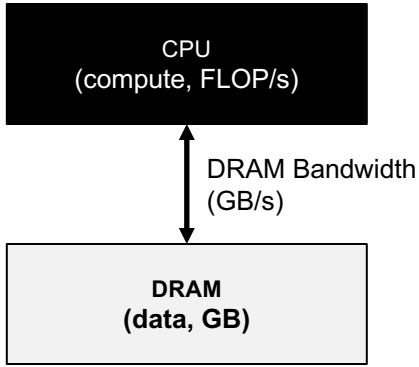


Transition @ AI ==
Peak GFLOP/s / Peak GB/s ==
'Machine Balance'

(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)

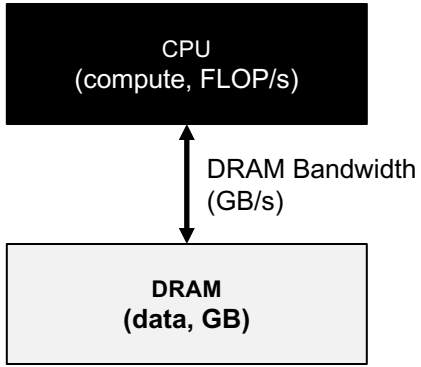
$$\text{Time} = \max \begin{cases} \#\text{FP ops} / \text{Peak GFLOP/s} \\ \#\text{Bytes} / \text{Peak GB/s} \end{cases}$$



(CPU DRAM) Roofline

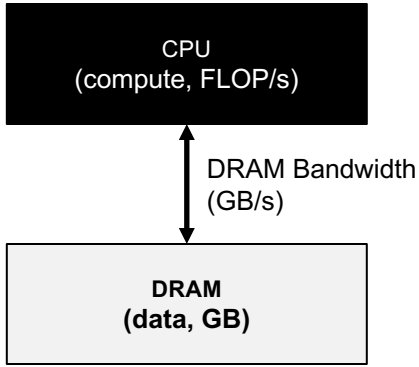
- One could hope to always attain peak performance (FLOP/s)
- However, finite locality (reuse) and bandwidth limit performance.
- Assume:
  - Idealized processor/caches
  - Cold start (data in DRAM)

$$\frac{\text{Time}}{\text{\#FP ops}} = \max \begin{cases} 1 \,/\, \text{Peak GFLOP/s} \\ \text{\#Bytes} \,/\, \text{\#FP ops} \,/\, \text{Peak GB/s} \end{cases}$$



(CPU DRAM) Roofline

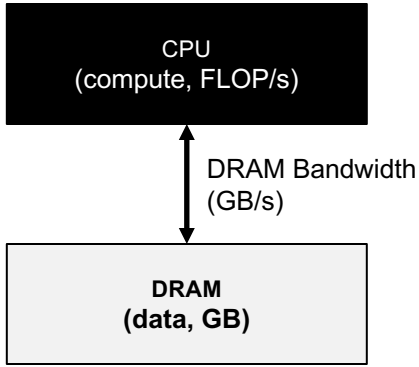
- One could hope to always attain peak performance (FLOP/s)
- However, finite locality (reuse) and bandwidth limit performance.
- Assume:
  - Idealized processor/caches
  - Cold start (data in DRAM)



$$\frac{\#\text{FP ops}}{\text{Time}} = \min \begin{cases} \text{Peak GFLOP/s} \\ (\#\text{FP ops} / \#\text{Bytes}) * \text{Peak GB/s} \end{cases}$$

(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)



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

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

Roofline Performance Model

- Thus we obtain the model as

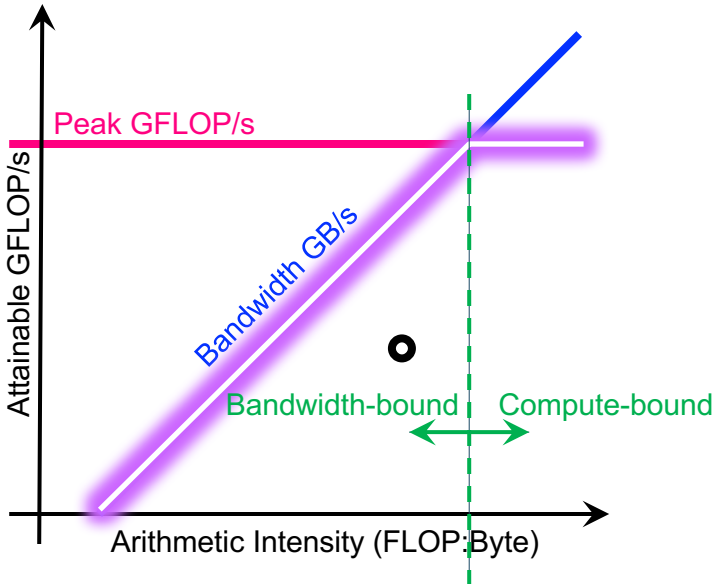
$$\text{GFLOP/s} = \min \begin{cases} \text{Peak GFLOP/s} \\ \text{AI} * \text{Peak GB/s} \end{cases}$$

where Arithmetic Intensity (AI) is

FLOPs / 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'

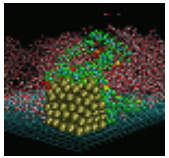
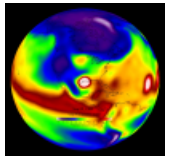
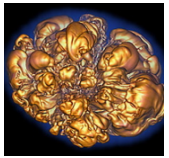
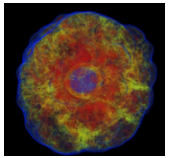
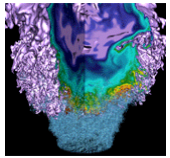
Roofline Performance Model

- 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

What can Roofline do?

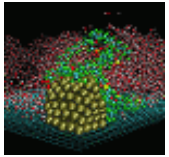
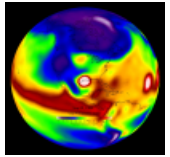
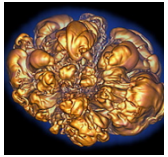
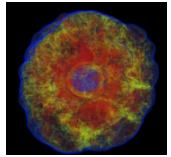
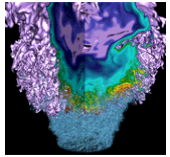






Roofline is Useful for…

- **Identifying** performance bottlenecks & **motivating** software optimizations
- **Understanding** performance differences between architectures, programming models, implementations, *etc*
- **Determining** when we're done optimizing code
  - Assess performance relative to machine capabilities
  - Motivate need for algorithmic changes
- **Predicting** performance on future machines / architectures
  - Set realistic performance expectations
  - Drive for HW/SW Co-Design

Activities on Roofline

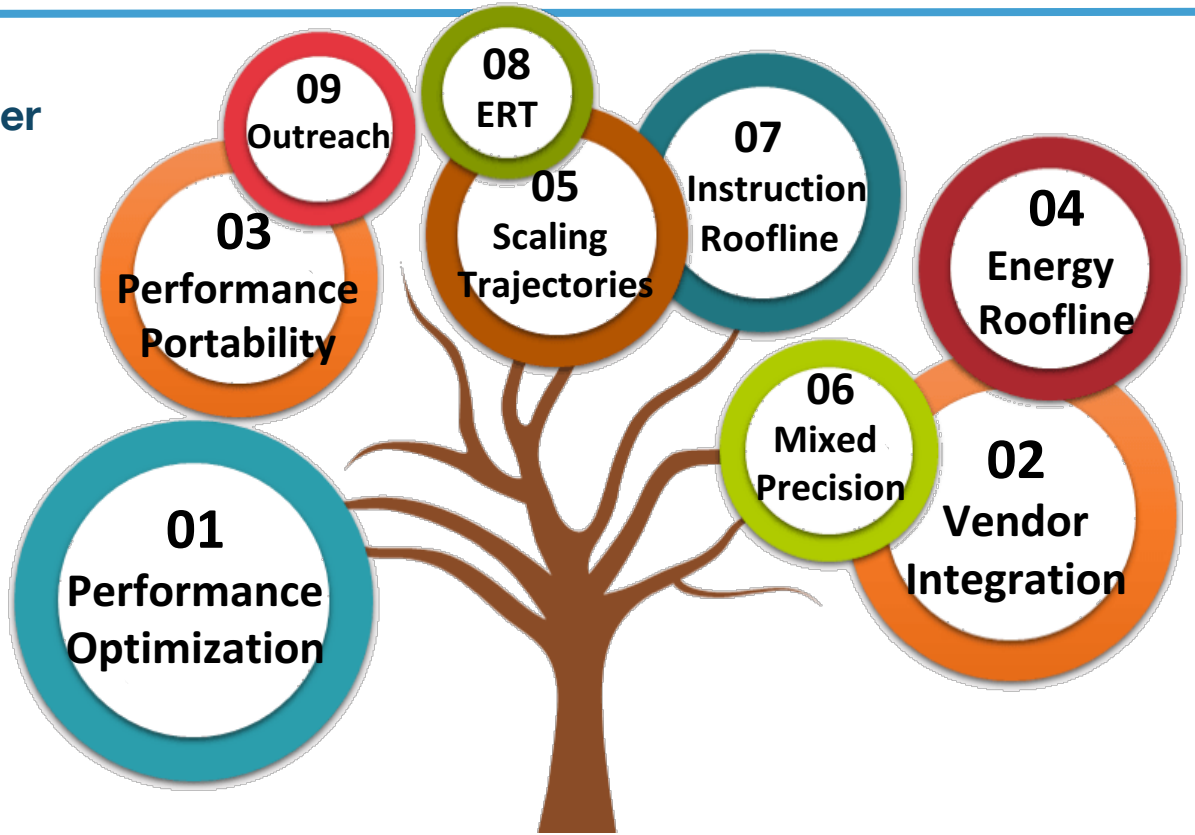
Current, Future

The Roofline Tree

Brings People Together

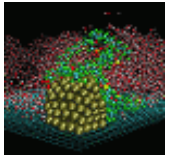
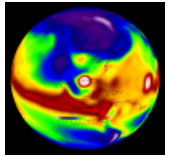
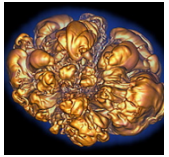
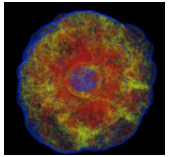
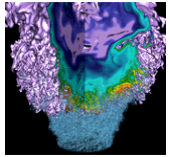
- NESAP
- CRD
- Intel
- NVIDIA
- all HPCers



Roofline Performance Model

1. Performance Optimization

NESAP, Hierarchical Roofline, Roofline drives optimization

Roofline Drives Optimization

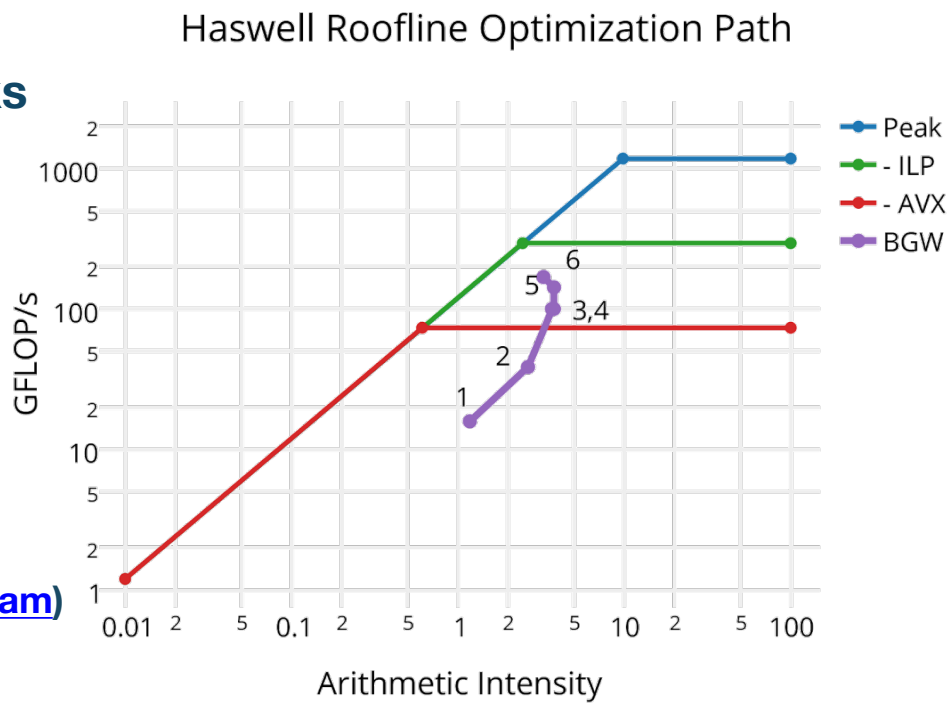
The Roofline Model

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

An example:

- NESAP for Cori - BerkeleyGW

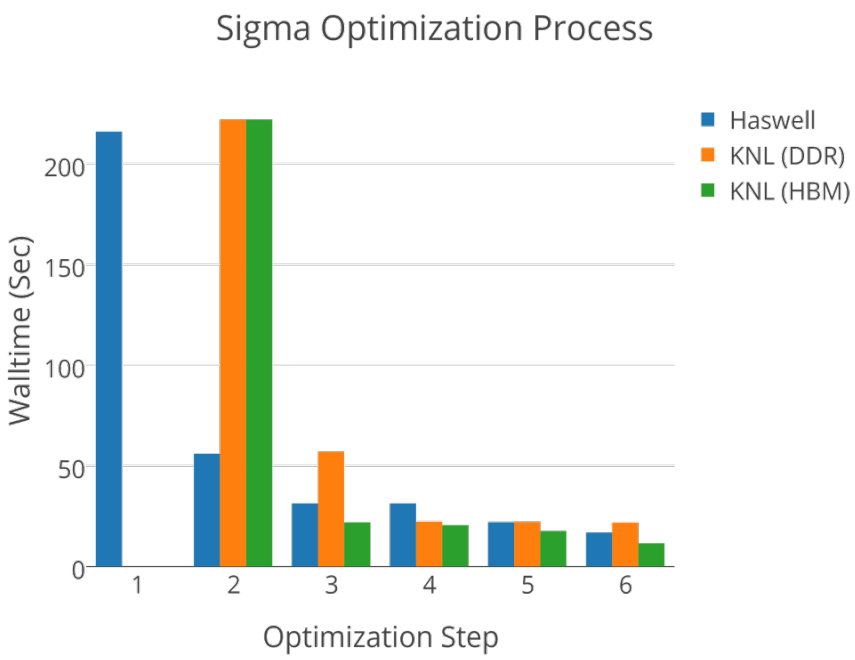
(NERSC Exascale Scientific Application Program)



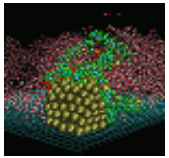
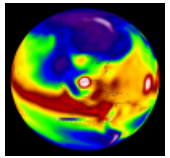
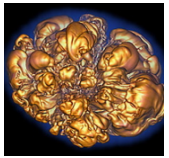
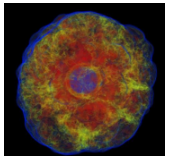
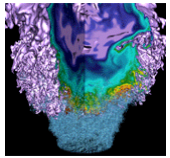
Roofline Example: BerkeleyGW

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



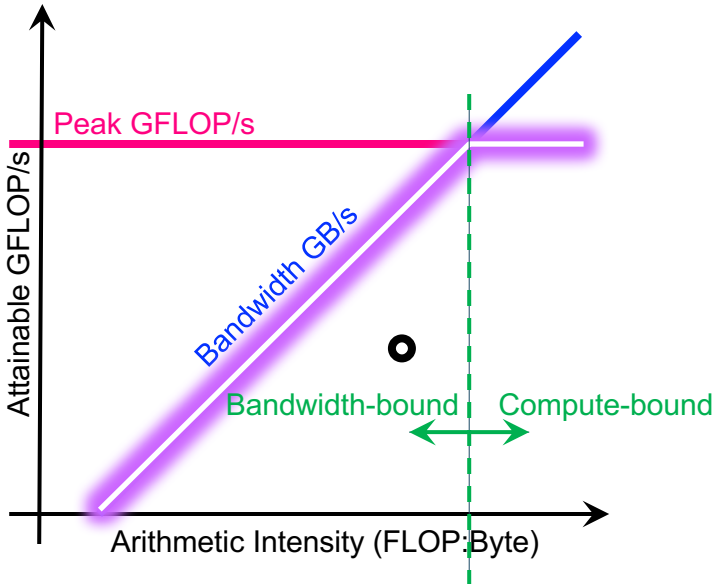
1.1 Roofline Variations

Roofline Performance Model

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

→ Hierarchical Roofline
→ Multiple compute ceilings

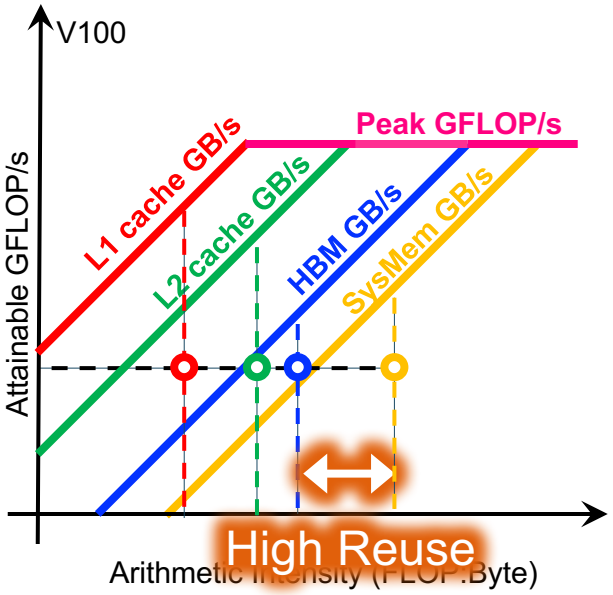


Transition @ AI ==
Peak GFLOP/s / Peak GB/s ==
'Machine Balance'

Hierarchical Roofline

- Superposition of multiple Rooflines
  - Incorporates full memory hierarchy
  - Arithmetic Intensity =

    FLOPs / Bytes$_{\text{L1/L2/HBM/SysMem}}$

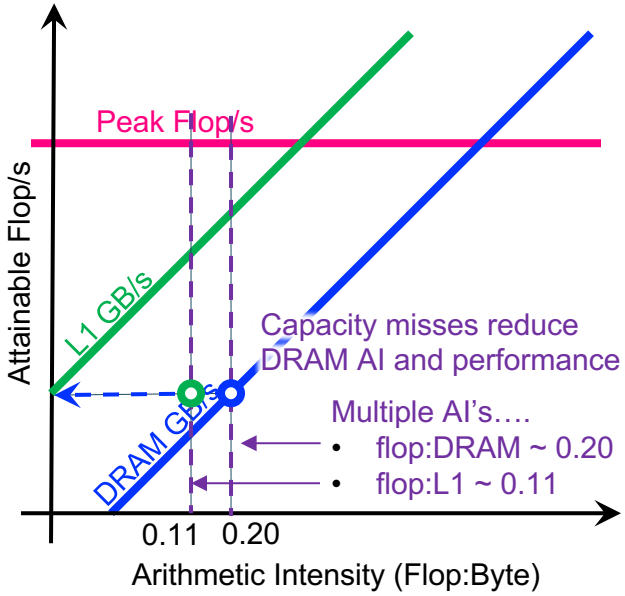


- Each kernel will have multiple AI's but one observed GFLOP/s performance

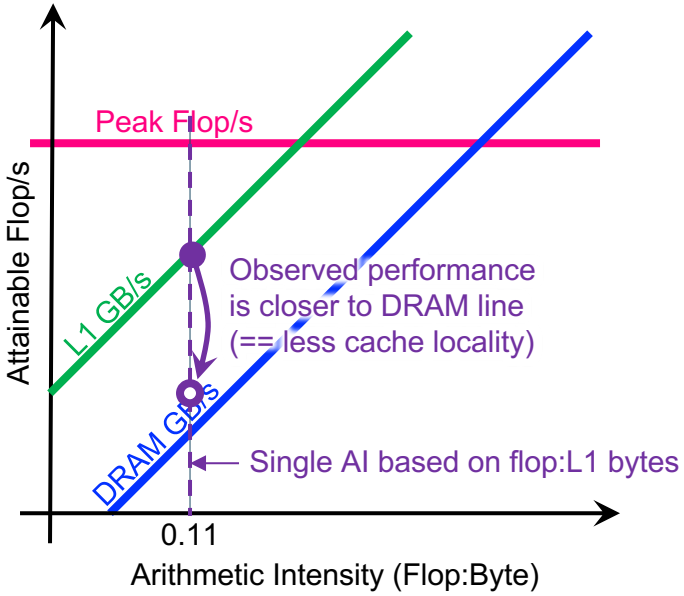
- Hierarchical Roofline tells you about **cache locality**

Cache-Aware Roofline Model (CARM)

Hierarchical Roofline



Cache-Aware Roofline



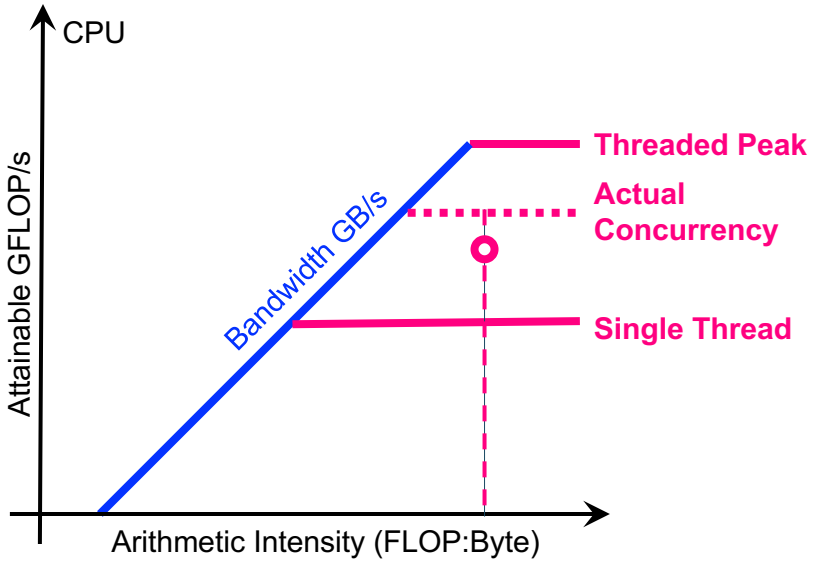
7-point Stencil

Hierarchical vs Cache-Aware

Hierarchical	Cache-Aware
Captures cache effects	Captures cache effects
AI is Flop:Bytes after being **filtered by lower cache levels**	AI is Flop:Bytes **as presented to the L1 cache (plus non-temporal stores)**
Multiple Arithmetic Intensities (one per level of memory)	Single Arithmetic Intensity
AI **dependent** on problem size (capacity misses reduce AI)	AI **independent** of problem size
Memory/Cache/Locality effects are **observed as decreased AI**	Memory/Cache/Locality effects are **observed as decreased performance**
Requires **performance counters or cache simulator** to correctly measure AI	Requires static analysis or **binary instrumentation** to measure AI

Multiple Compute Ceilings

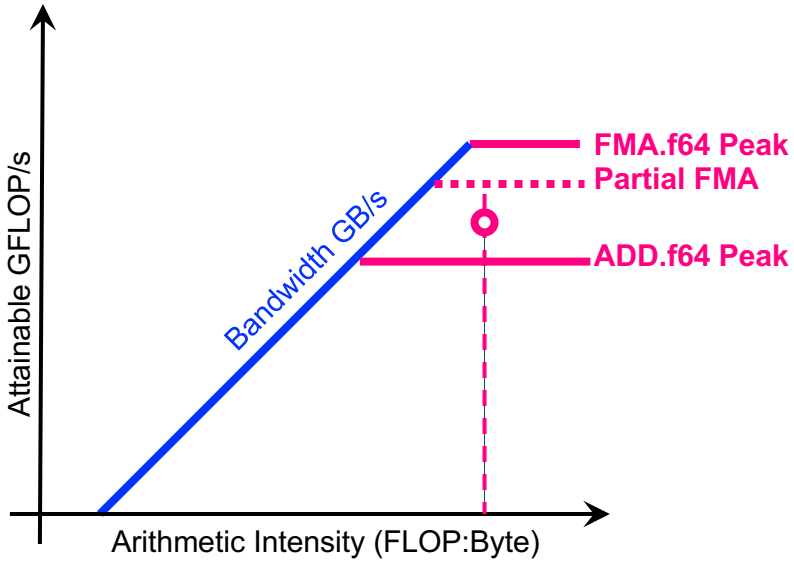
- Impact of **execution configuration**
- Concurrency affects your peak
  - OpenMP thread concurrency
  - SM occupancy
  - load balance
  - threadblock/thread configuration



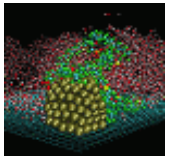
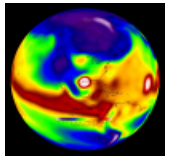
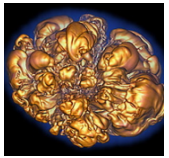
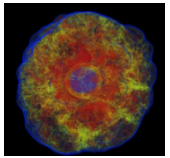
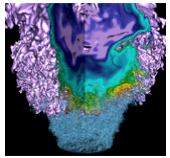
- Performance is bound by the **actual concurrency** ceiling

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

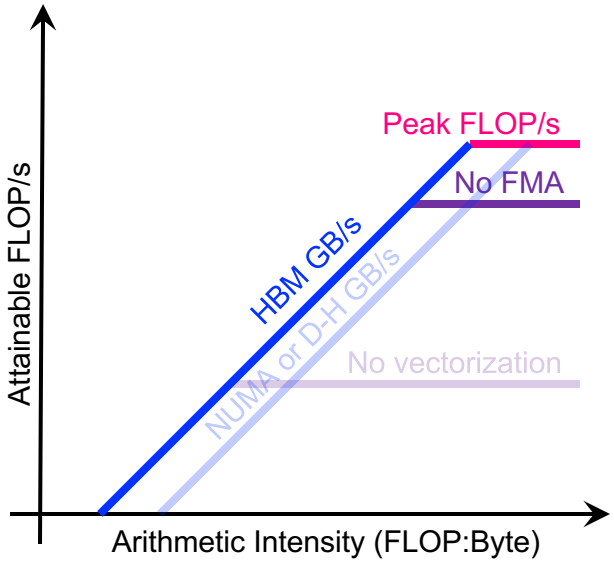


1.2 Roofline Drives Optimization

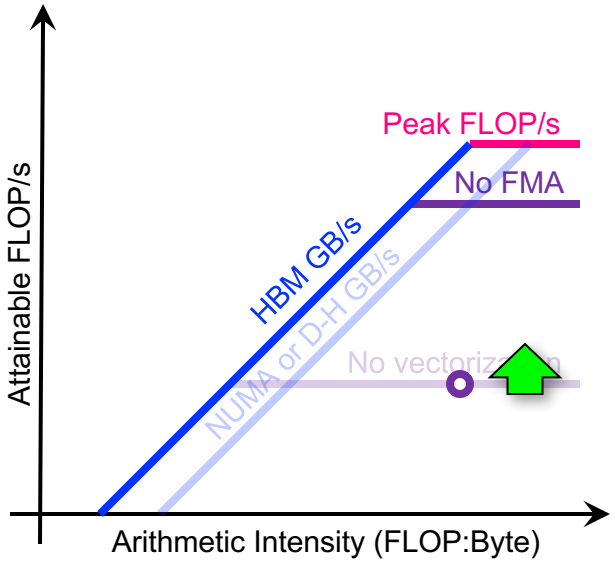
General Optimization Strategy

- **Broadly speaking, three approaches to improving performance:**



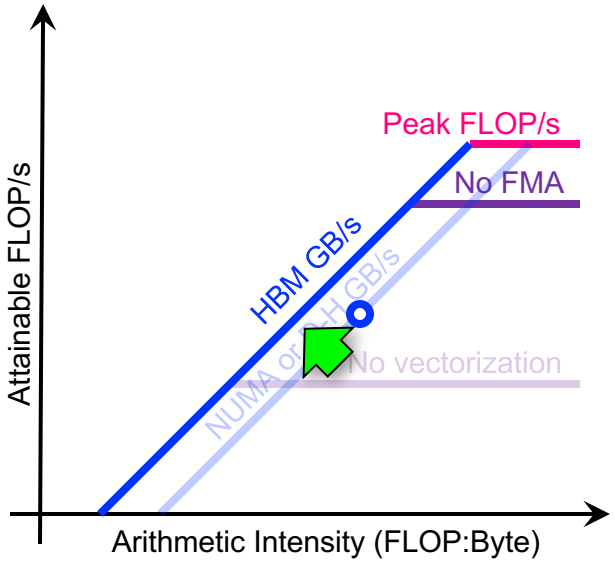
General Optimization Strategy

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



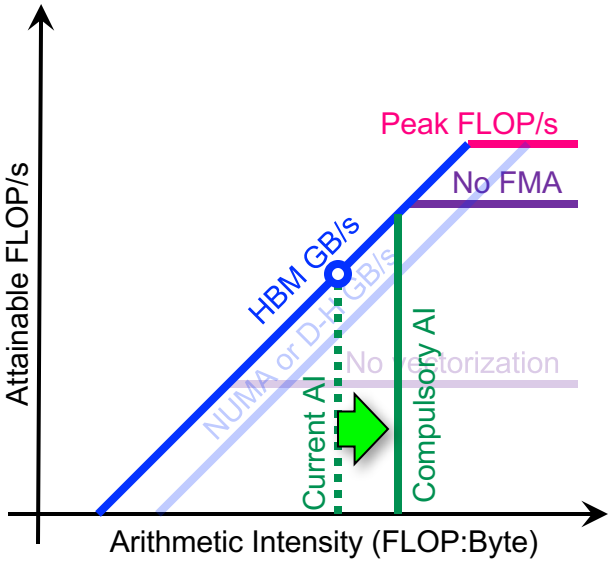
General Optimization Strategy

- 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

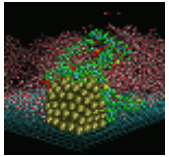
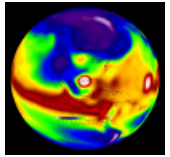
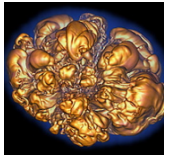
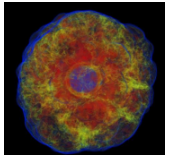
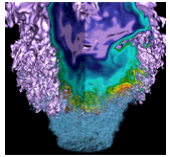


General Optimization Strategy

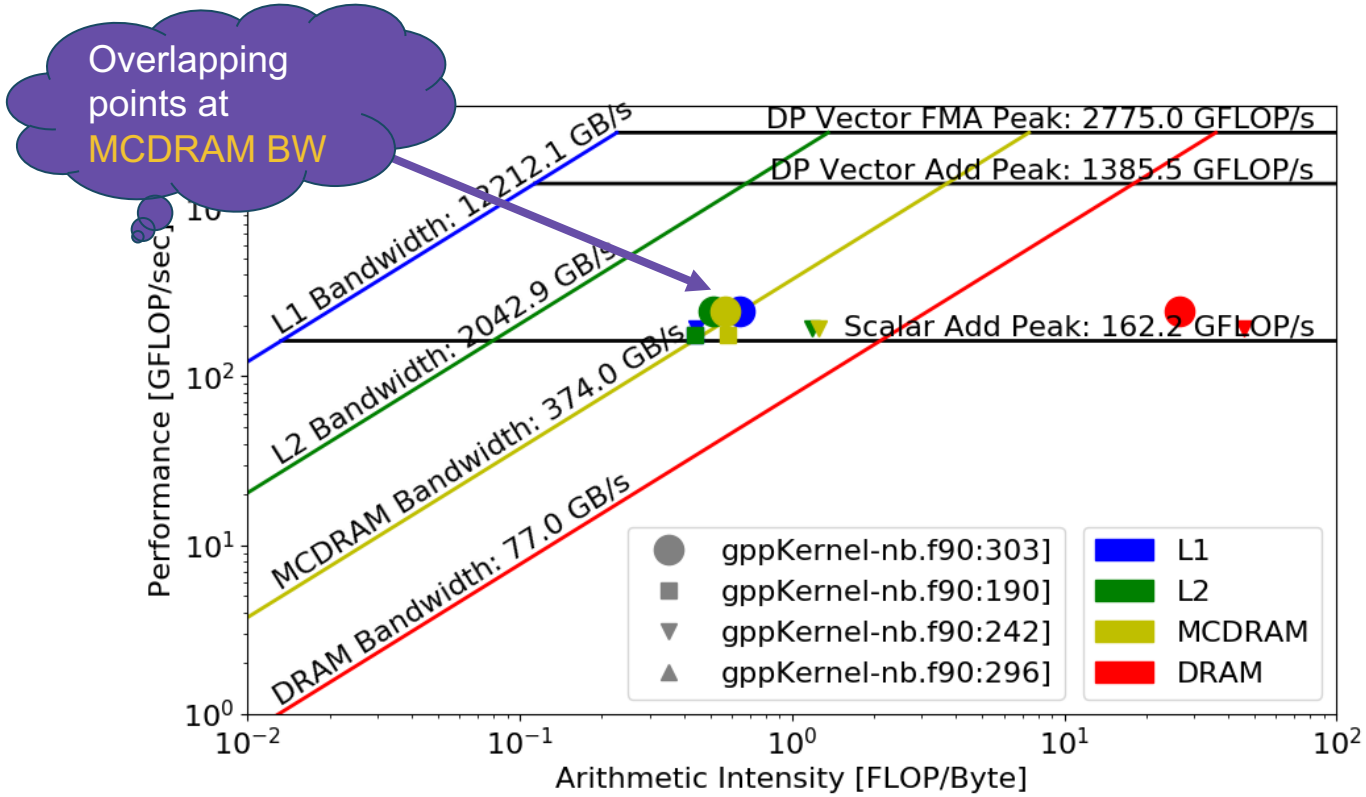
- Broadly speaking, three approaches to improving performance:
- Maximize compute performance
- Maximize memory bandwidth
- Improve AI
  - minimize data movement
  - exploit cache reuse



1.3 Example Applications

Example 1: GPP, KNL, Cache Blocking



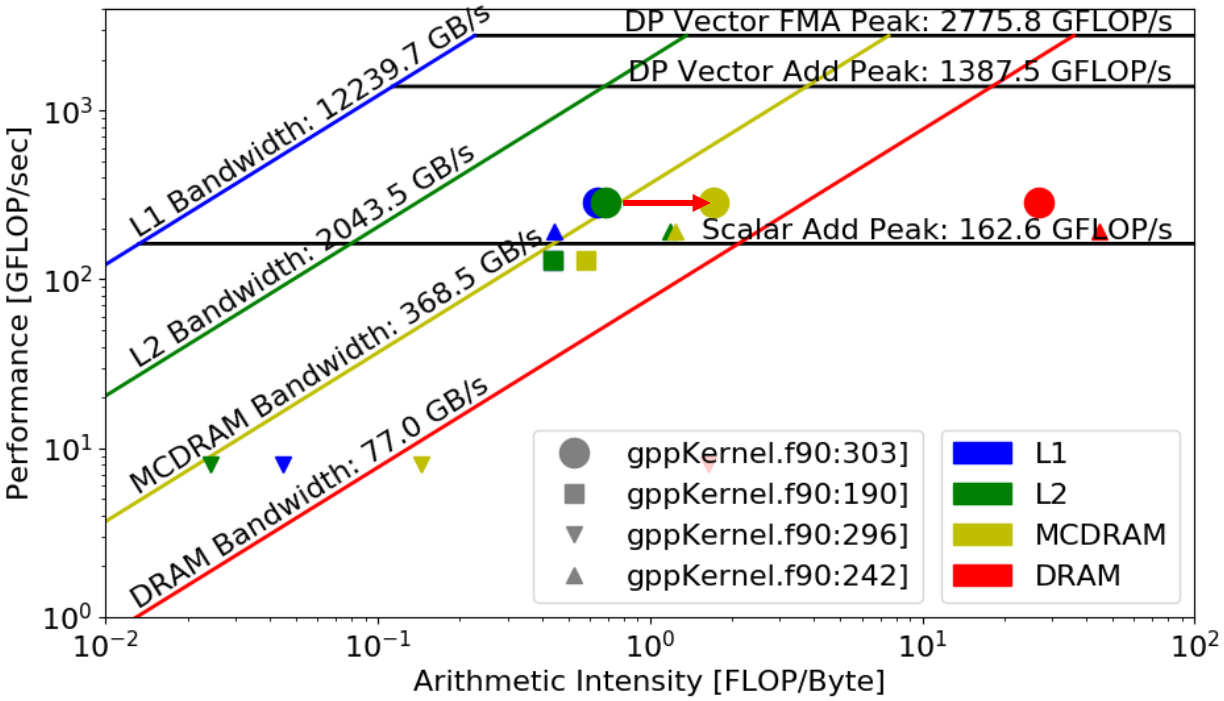
242 GFflop/s, **Bound by MCDRAM Bandwidth**

Most Flops in the main loop (O)

Read/Write 2MB of data per inner loop iteration
➤ No reuse of data in L1/L2, shown by overlapping points at MCDRAM bandwidth

BW Bound ➤ Increase MCDRAM AI by adding cache locality

Example 1: GPP, KNL, Cache Blocking



Cache blocking implemented to achieve L2 data reuse

3x Increase in MCDRAM AI

Performance increased from 242 to 287 GFlop/s (+18%)

Why not 3x Flops increase?
➤ Not BW bound any more, divide, shuffle and unpack instructions involved

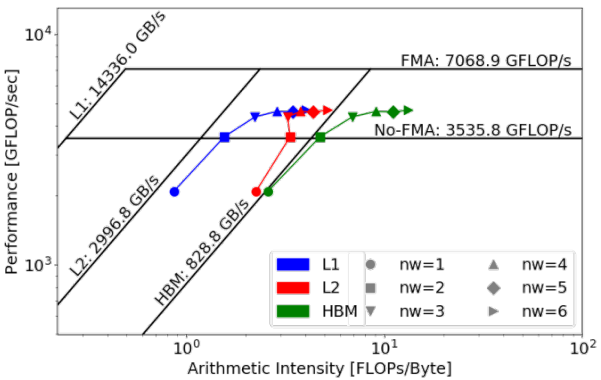
- T. Koskela, Z. Matveev, C. Yang, A. Adetokunbo, R. Belenov, P. Thierry, Z. Zhao, R. Gayatri, H. Shan, L. Oliker, J. Deslippe, R. Green, and S. Williams, A Novel Multi-Level Integrated Roofline Model Approach for Performance Characterization, ISC'2018 Research Paper, Jun 24-28 2018, Frankfurt

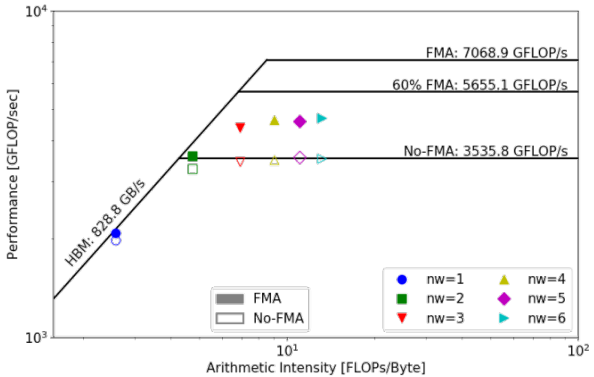
Example 1: GPP, V100, Hierarchical

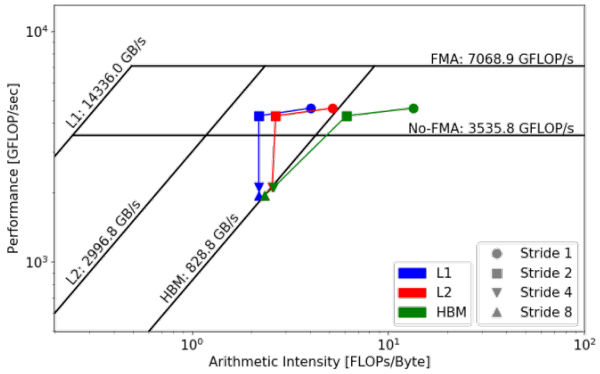
Three experiments to study the effects of

- cache reuse (varying `nw` from 1 to 6)
- instruction mix (FMA vs. Mul/Add)
- memory coalescing

```
do band = 1, nbands        #blockIdx.x
   do igp = 1, ngpown      #blockIdx.y
      do ig = 1, ncouls    #threadIdx.x
         do iw = 1, nw     #unrolled
            compute; reductions
```

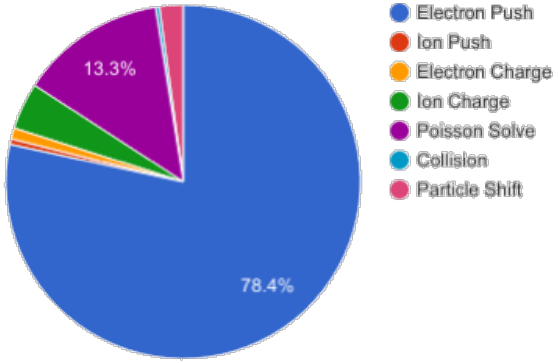


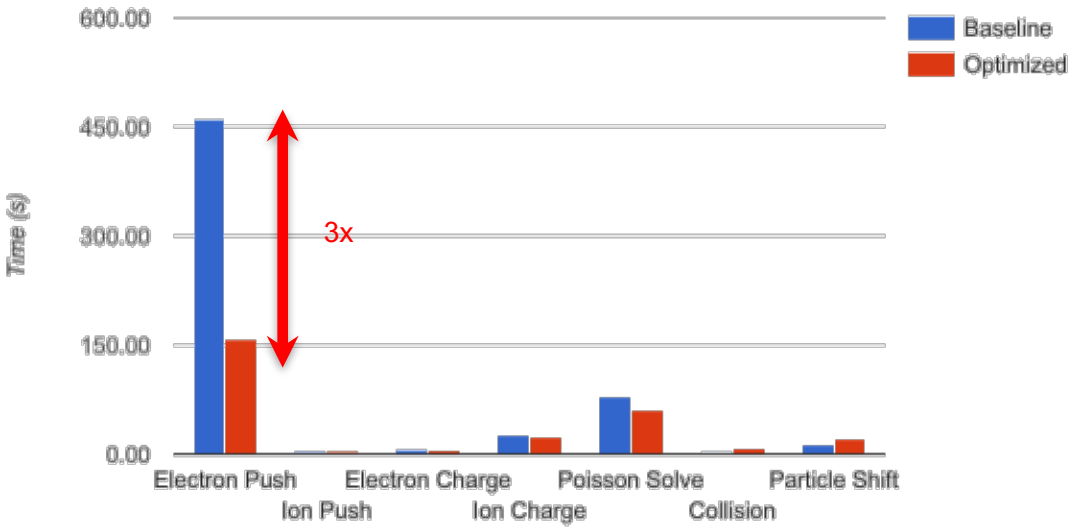




- Charlene Yang, Thorsten Kurth, Samuel Williams, "Hierarchical Roofline Analysis for GPUs: Accelerating Performance Optimization for the NERSC-9 Perlmutter System", Cray User Group (CUG), May 2019.

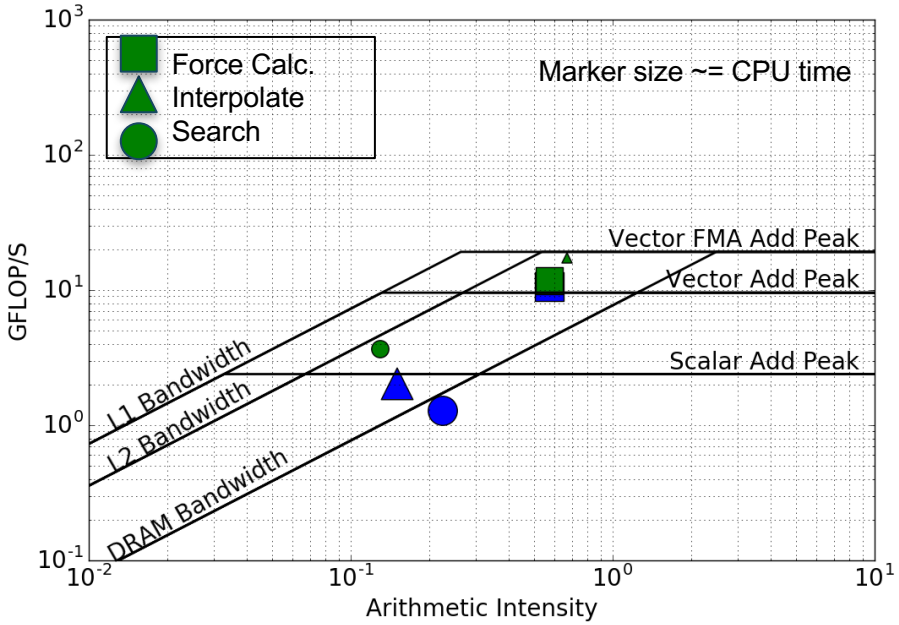
Example 2: XGC1, KNL





(Left) Hotspots for unoptimized XGC1 on 1024 Cori KNL nodes in Quad-Flat mode;
(Right) Speedup in XGC1 Electron Push routine after back porting the optimizations made in ToyPush kernel

Example 2: ToyPush from XGC1



- **Force Kernel:**
- close to vector add peak
- not much optimization done

- **Interpolate Kernel:**
- L1 blocking, indirect memory access
- memory alignment, more efficient vectorization
- **10x speedup**, closer to vector FMA peak

- **Search Kernel:**
- multiple exits, simd private, enable vectorization
- **3x speedup**, closer to L2 bandwidth roof

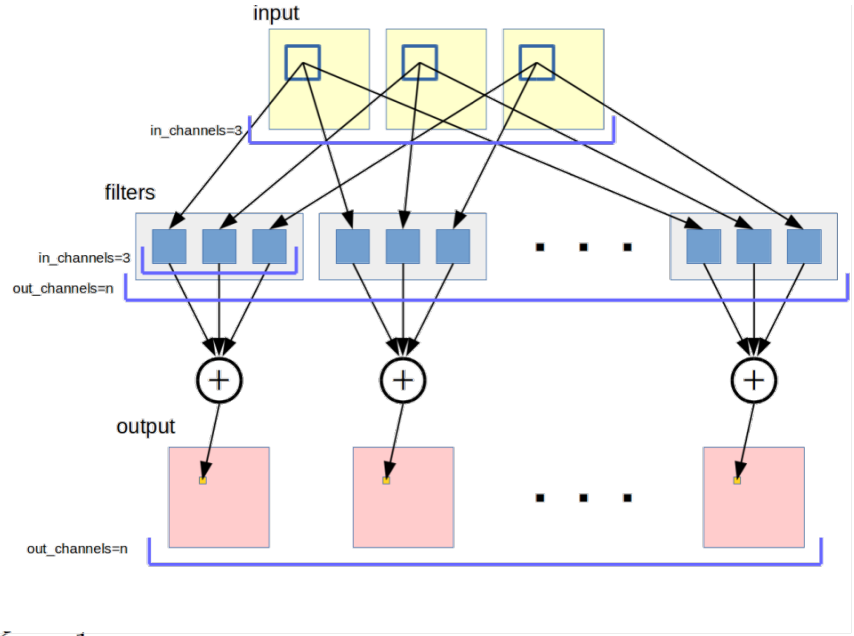
- Code is available at
- https://github.com/tkoskela/toypush

Example 3: conv2d from TensorFlow

- **Kernel tf.nn.conv2d**

TensorFlow

https://www.tensorflow.org



$$B_{nhwc} = \sum_{m=0}^{C-1} \sum_{k_h=0}^{K_H-1} \sum_{k_w=0}^{K_W-1} A_{n\,h+k_h\,w+w_h\,m} K_{k_h\,k_w\,m\,c}$$

Example 3: conv2d from TensorFlow

exec_op:

- forward pass -- conv in 2D
- backward pass -- conv + derivative
- calibrate -- tensor generation

```
#choose operation depending on pass
if pass=="forward":
    with tf.device(gpu_dev):
    exec_op = output_result
elif pass=="backward":
    with tf.device(gpu_dev):
    opt = tf.train.Gradient\
        DescentOptimizer(0.5)
    exec_op = opt.compute\
        _gradients(output_result)
elif pass=="calibrate":
    with tf.device(gpu_dev):
    exec_op = input_image
```

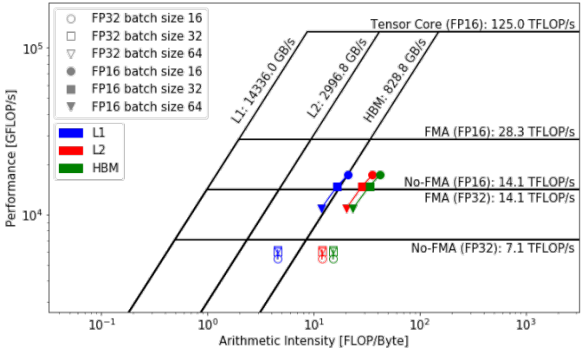
```
#generate random input tensor
input_image = tf.random_uniform(shape=input_size, minval=0., maxval=1., dtype=dtype)
#create network
output_result = conv2d(input_image, 'NHWC', kernel_size, stride_size, dtype)
```

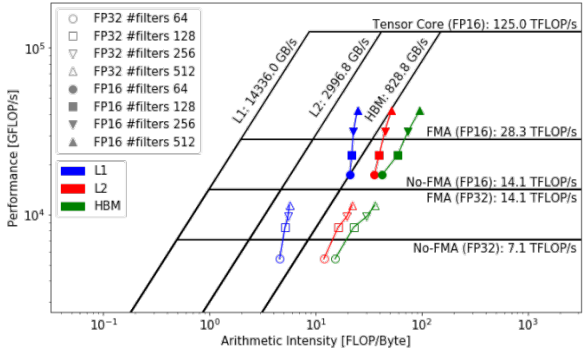
Example 3: conv2d from TensorFlow

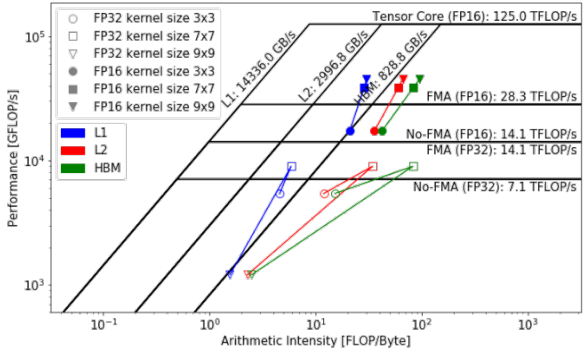
- Each TensorFlow kernel translates to a series of subkernels
  - padding, shuffling, data conversion, *etc*
- TensorFlow based on heuristics decides what subkernels to call
- cuDNN also has some algorithm selection mechanism
- We **INCLUDE** the housekeeping subkernels in our measurements, but **EXCLUDE** the autotuning subkernels

- C. Yang, S. Williams, Hierarchical Roofline Analysis for GPUs: Accelerating Performance Optimization for the NERSC-9 Perlmutter System, CUG'2019, May 5-9 2019, Montreal, Canada

Example 3: TF / Forward Pass







#Batch Size

- Constant performance(**no!**)
- **FP16 performance anti-correlated with batch size**
- Performance << TC peak
- Transformation kernels
- Low L2 locality

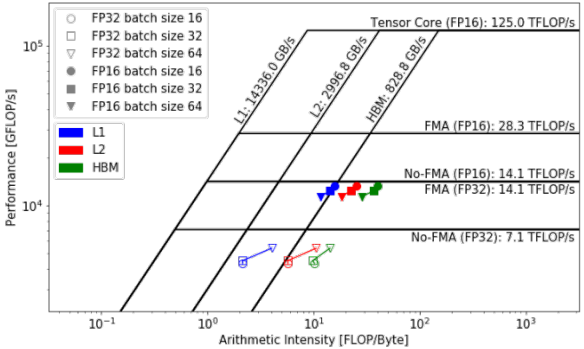
#Filters

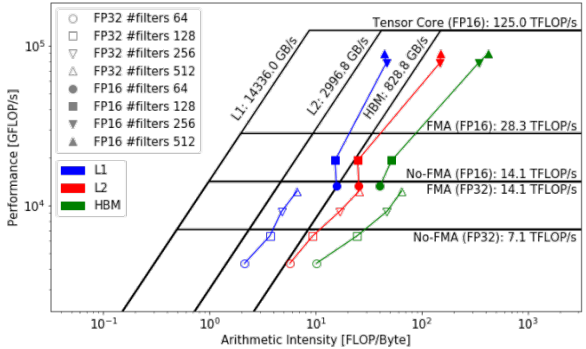
- Intensity ∝ #Filters
- Low L2 data locality
- Some use of TC's (>FP16 FMA)… ***partial TC ceiling***

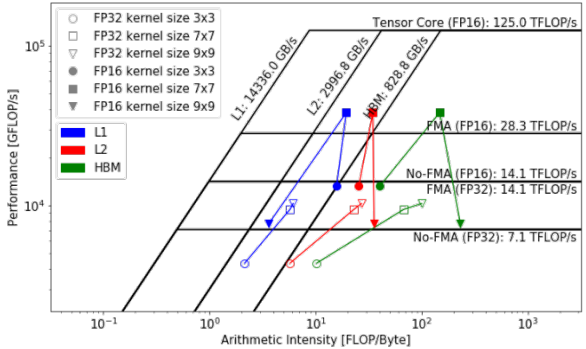
#Kernel Size

- Intensity ∝ kernel size
- Low L2 data locality
- **Autotuner switched FP32 algorithm to FFT at 9x9**
- Some use of TC's (>FP16 FMA)… ***partial TC ceiling***

Example 3: TF / Backward Pass







#Batch Size

- Autotuner chose different (**better**) algorithm for FP32 with batch size = 64 (boost)

#Filters

- Close to FP16 TC peak
- Close to FP32 FMA peak

#Kernel Size

- Good FP32 performance trend (almost peak)
- **Autotuner chose to run 9x9 FP16 in FP32 !!**

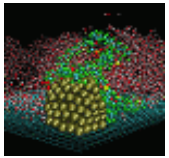
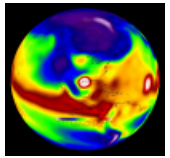
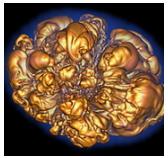
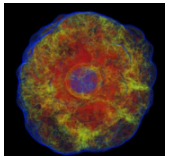
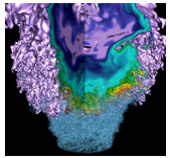
Summary

- Useful for characterization as well as optimization of HPC applications

- Roofline has a wide applicability
  - different architectures (KNL, V100, …)
  - different algorithms (Simulation, Machine Learning, …)

2. Vendor Integration

Intel VTune, LIKWID, Intel Advisor, NVIDIA nvprof

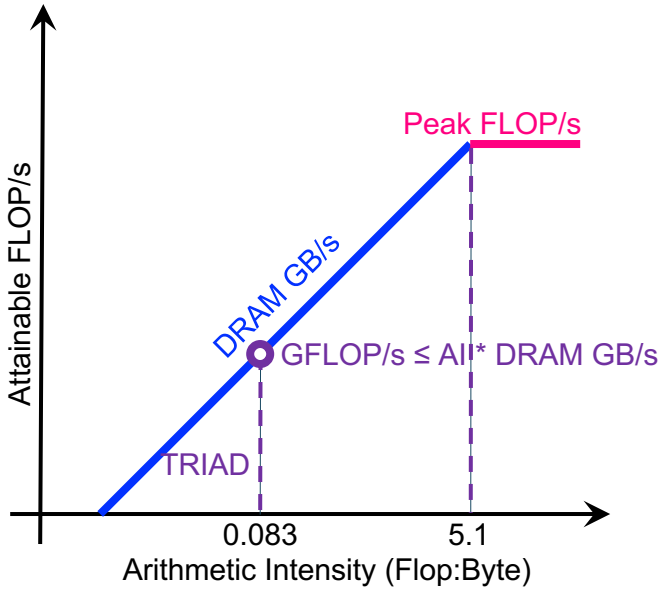
Pen and Paper

- **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]

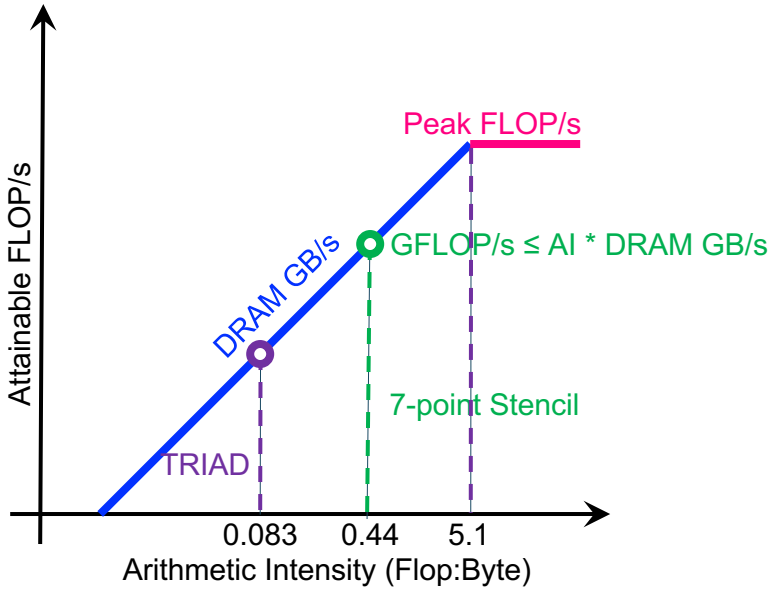
- AI = 0.083 FLOPs per byte
- Memory bound



Pen and Paper

- **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  ];
}}}
```

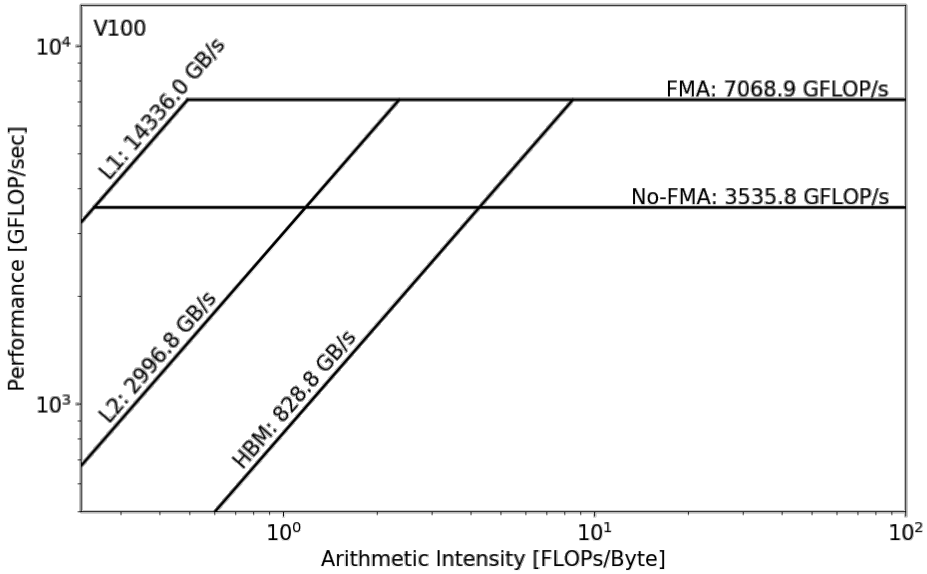


Pen and Paper

- 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

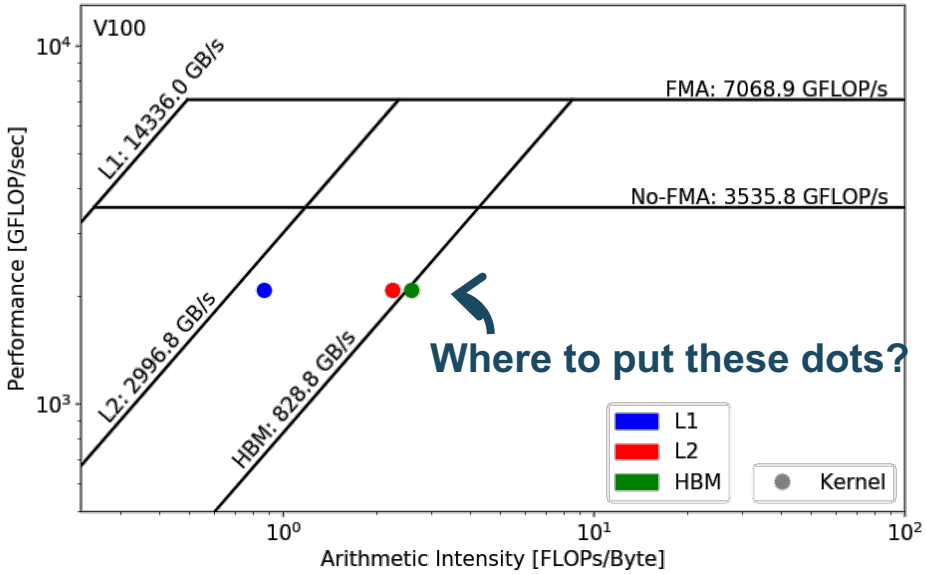
We need tools!

We Need Tools!

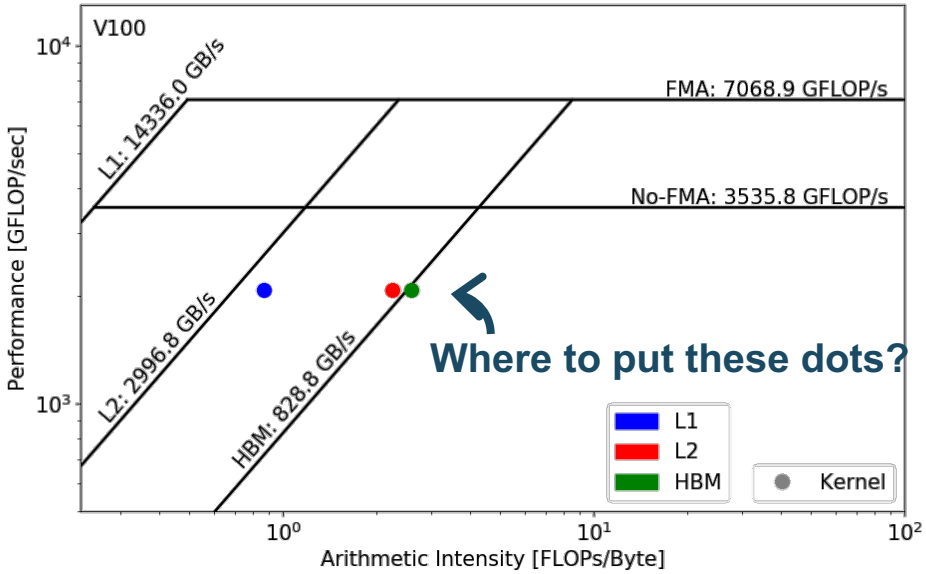


- **Roofline ceilings**
  - **vendor specifications**
  - **empirical measurements**
    - **ERT**
    - https://bitbucket.org/berkeleylab/cs-roofline-toolkit

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:

$$\text{Arithmetic Intensity} = \frac{\text{FLOPs}}{\text{Data Movement}}$$

(FLOPs/Byte)

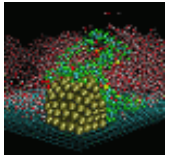
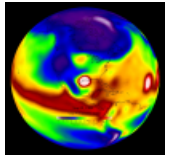
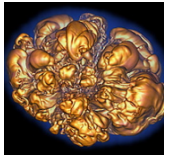
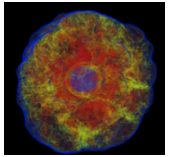
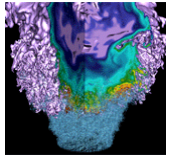
$$\text{Performance} = \frac{\text{FLOPs}}{\text{Runtime}}$$

(GFLOP/s)

Methodology to Construct Roofline

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

2.1 Intel CPUs and NVIDIA GPUs

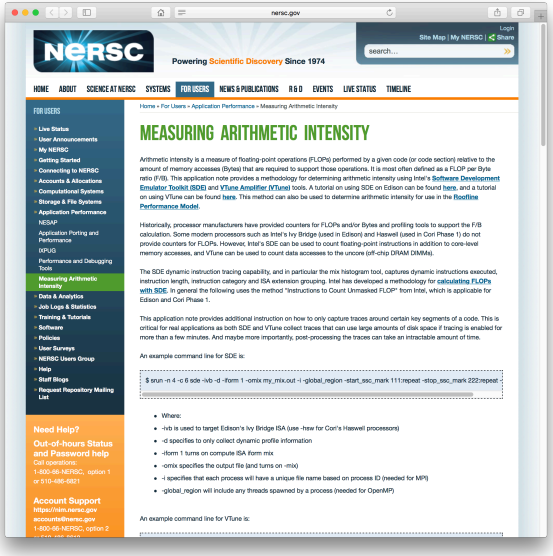
Data Collection on Intel CPUs

The not-so-automated way 1:

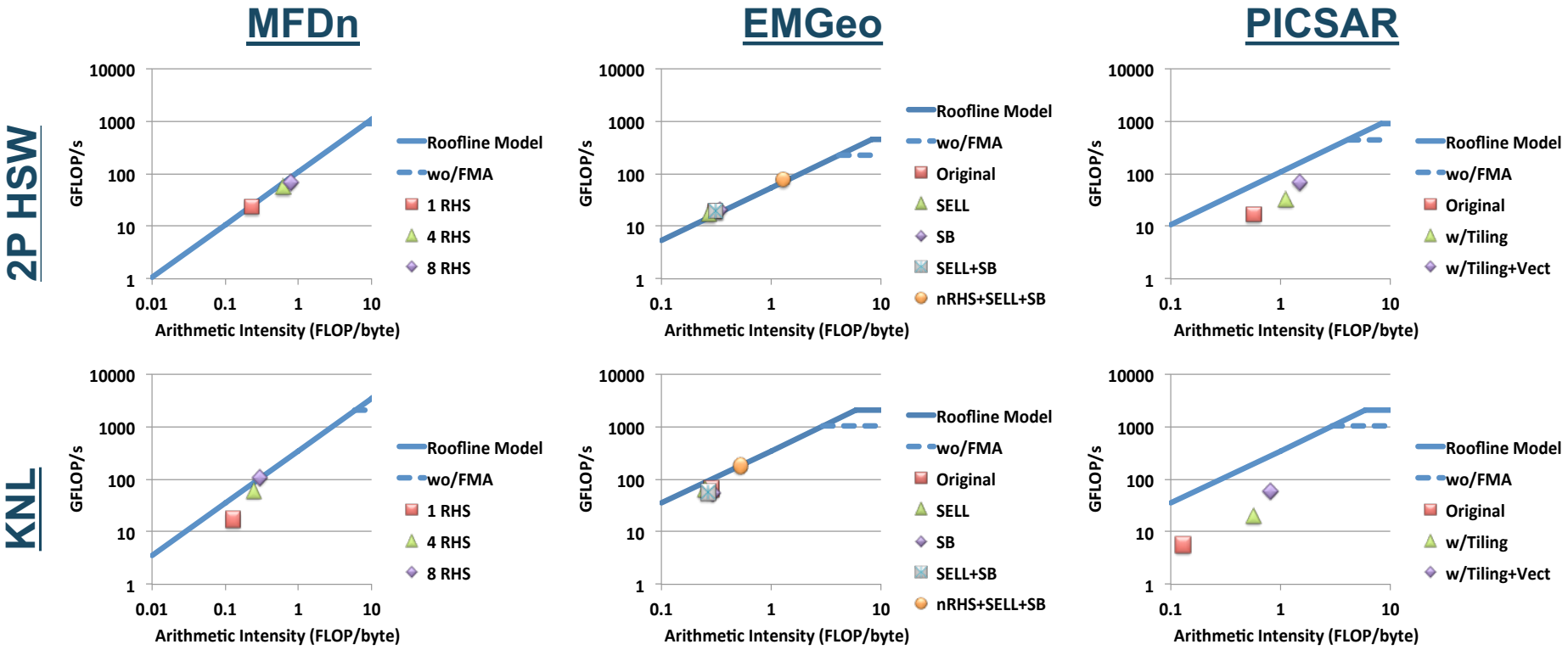
- **Intel SDE** for FLOPs (emulation)
- **Intel VTune** for DRAM bytes (HW counters)
- Runtime

- **DRAM Roofline** only

- Used by NESAP for Cori
  - NERSC Exascale Science Application Program
  - http://www.nersc.gov/users/application-performance/measuring-arithmetic-intensity/



Data Collection on Intel CPUs



DRAM Rooflines of NESAP Codes

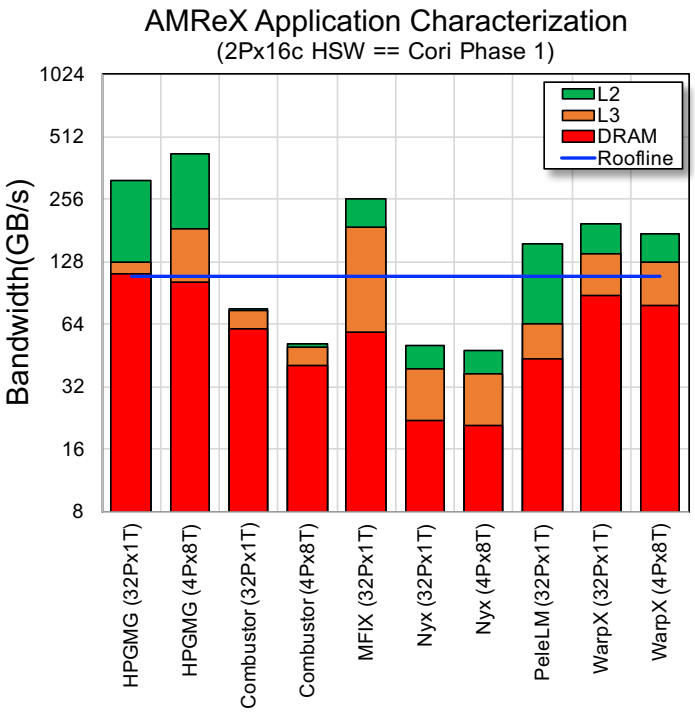
Data Collection on Intel CPUs

The not-so-automated way 2:

- **LIKWID for FLOPs and bytes**
  - **Both are based on HW counters**
- **Runtime**

- **Hierarchical Roofline**

- **Limited by quality of HW counters**
- **High-level characterization, no callstack**



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

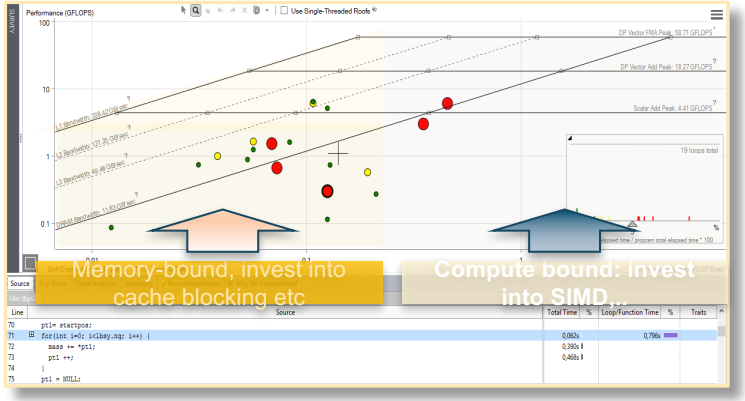
Data Collection on Intel CPUs

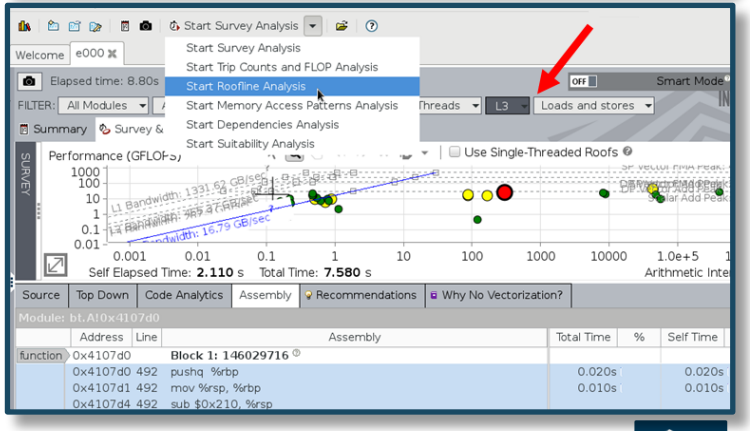
The fully automated way:

- 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



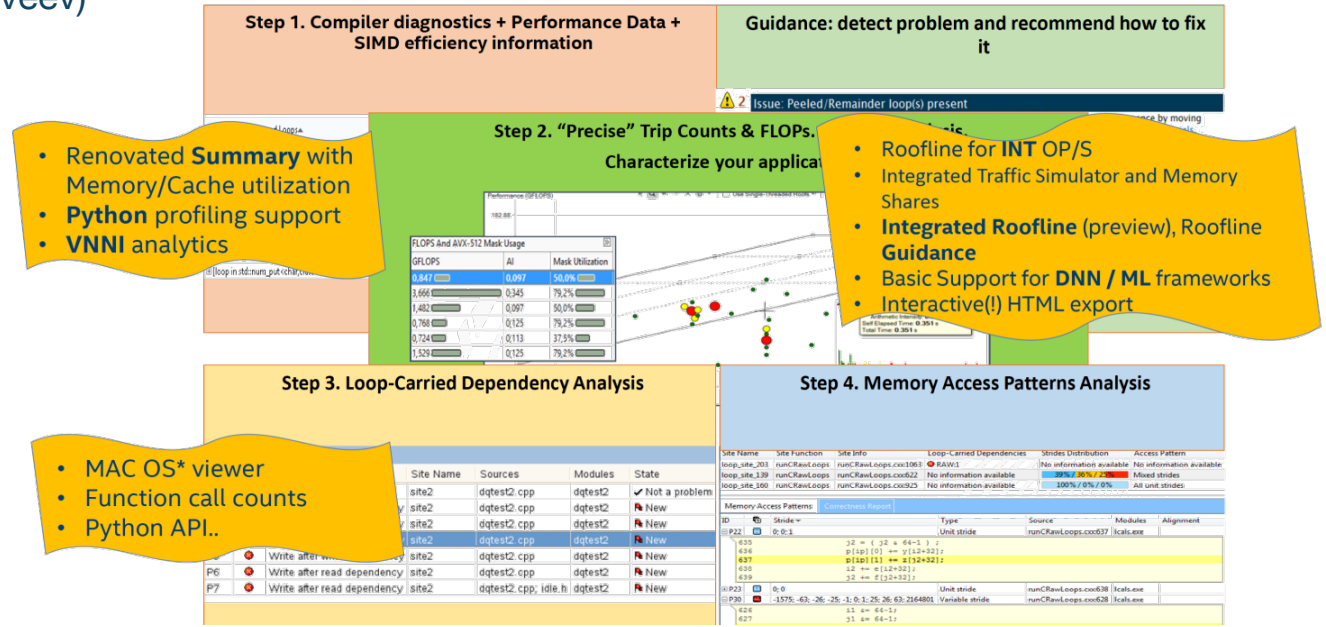


Data Collection on Intel CPUs

New features in Intel Advisor 2019

(picture courtesy of Z. Matveev)

https://software.intel.com/en-us/intel-advisor-2019-release-notes



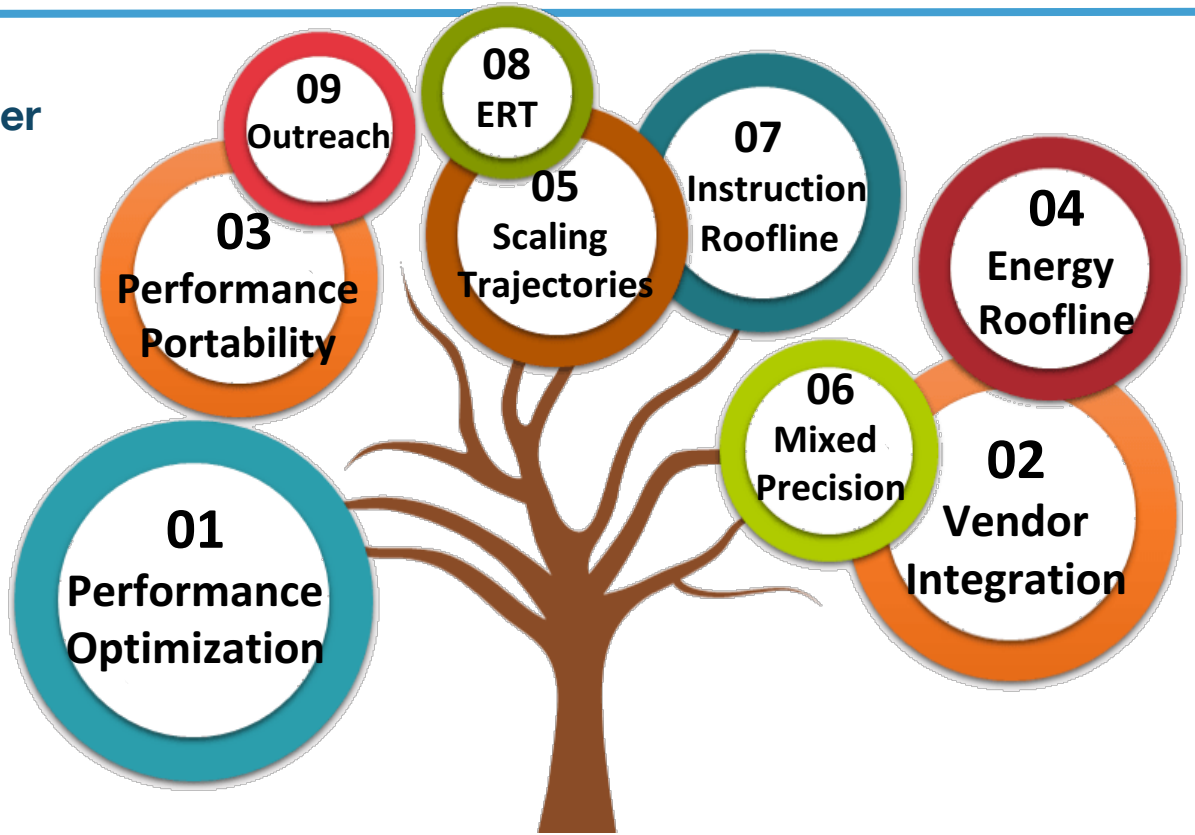
Data Collection on NVIDIA GPUs

- Still 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 Tree

Brings People Together

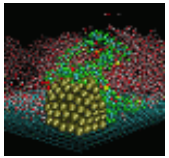
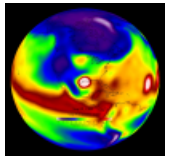
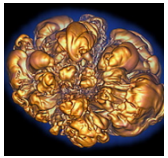
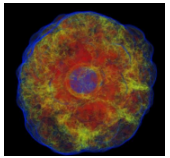
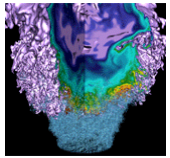
- NESAP
- CRD
- Intel
- NVIDIA
- all HPCers



Roofline Performance Model

3. Performance Portability

Definition, Metric, Roofline, KNL, V100

Introduction

- No consensus on the definition or metric for performance portability
- But Pennycook *et al*…

$$\boldsymbol{\Phi}(a,p,\boldsymbol{H}) = \begin{cases} \dfrac{|\boldsymbol{H}|}{\sum_{i\in\boldsymbol{H}} \dfrac{1}{\boldsymbol{e_i(a,p)}}} & \text{if } i \text{ is supported}, \forall i \in \boldsymbol{H} \\ 0 & \text{otherwise} \end{cases}$$

Application's Efficiency on platform $\boldsymbol{i}$

- Application's Efficiency ?
  - application performance on platform i / peak performance of platform i
  - application performance on platform i / application's best performance on all platforms of interest $\boldsymbol{H}$

Architectural Efficiency

Application Efficiency

S. J. Pennycook, J. D. Sewall, and V. Lee, "A metric for performance portability," arXiv:1611.07409, 2016.

Introduction

- No consensus on the definition or metric for performance portability
- But Pennycook *et al*…

$$\boldsymbol{\Phi}(a,p,\boldsymbol{H}) = \begin{cases} \dfrac{|\boldsymbol{H}|}{\sum_{i\in\boldsymbol{H}} \frac{1}{e_i(a,p)}} & \text{if } i \text{ is supported}, \forall i \in \boldsymbol{H} \\ 0 & \text{otherwise} \end{cases}$$

- Architectural Efficiency [Williams *et al*]

Actual Application Performance

$$e_i(a,p) = \frac{P_i(a,p)}{\min(F_i,\ B_i \times I_i(a,p))}$$

Max Attainable Performance defined by Roofline

- S. J. Pennycook, J. D. Sewall, and V. Lee, "A metric for performance portability," arXiv:1611.07409, 2016.
- 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.

Introduction

- No consensus on the definition or metric for performance portability
- But Pennycook *et al*…

$$\boldsymbol{\Phi}(a,p,\boldsymbol{H}) = \begin{cases} \dfrac{|\boldsymbol{H}|}{\sum_{i\in\boldsymbol{H}} \frac{1}{e_i(a,p)}} & \text{if } i \text{ is supported}, \forall i \in \boldsymbol{H} \\ 0 & \text{otherwise} \end{cases}$$

- Architectural Efficiency [Williams *et al*]

$$e_i(a,p) = \frac{P_i(a,p)}{\min(F_i,\ B_i \times I_i(a,p))}$$

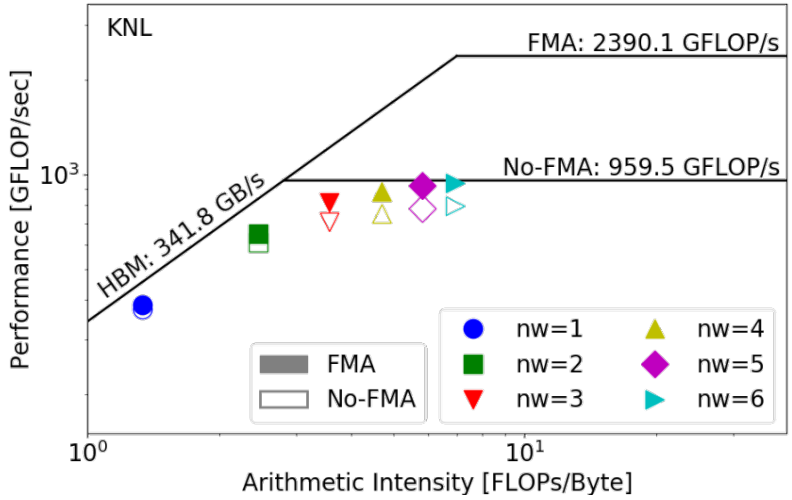
Peak FLOP/s

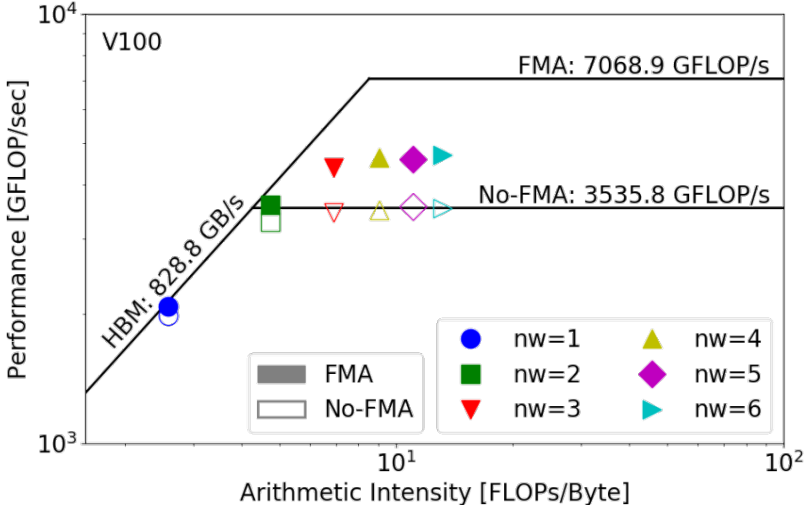
Peak Bandwidth

Arithmetic Intensity

Bottleneck Changes

- **Bottleneck shifts at $nw = 2$ on KNL vs. V100 (no-FMA performance)**
- **Easier to achieve no-FMA ceiling on V100 than KNL, due to higher ratio of instruction issue bandwidth vs. instruction execution bandwidth**





- C. Yang, R. Gayatri, T. Kurth, P. Basu, Z. Ronaghi, A. Adetokunbo, B. Friesen, B. Cook, D. Doerfler, L. Oliker, J. Deslippe, S. Williams, An Empirical Roofline Methodology for Quantitatively Assessing Performance Portability, SC'2018 P3HPC Workshop, Nov 11-16 2018, Dallas

Bottleneck Changes

- **No FMA: performance portability consistently > 80%**
- **FMA: benefit is far less than 2x at high nw; architectural efficiency suffers (so does performance portability)**
- **Could regain some architectural efficiency if non-floating-point vector operations were considered**

	Architectural Efficiency	$nw = 1$	$nw = 2$	$nw = 3$	$nw = 4$	$nw = 5$	$nw = 6$
FMA	KNL	84.98%	77.50%	66.77%	55.28%	46.56%	39.65%
	V100	97.36%	91.50%	76.70%	65.44%	65.07%	66.38%
	Performance Portability	**90.76%**	**83.92%**	**71.39%**	**59.93%**	**54.28%**	**49.65%**
No-FMA	KNL	82.06%	72.95%	73.74%	78.72%	81.28%	82.81%
	V100	92.88%	92.88%	97.43%	98.91%	1	99.73%
	Performance Portability	**87.14%**	**81.72%**	**83.95%**	**87.67%**	**89.93%**	**90.49%**

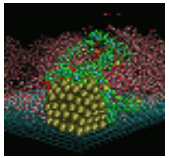
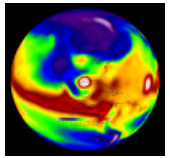
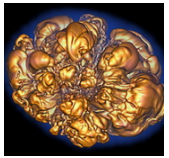
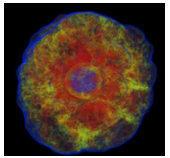
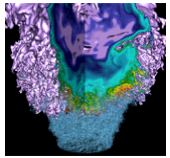
Summary

- Roofline is very powerful in capturing changes in machine and application characteristics such as
  - compute/bandwidth bound, problem size
  - instruction issue bandwidth, strided memory access

- It is important to
  - measure bandwidth/compute ceilings empirically
  - account for non-multiply/add instructions appropriately
  - select relevant ceilings in performance analysis and performance portability analysis

4. Energy Roofline

Performance, Power Consumption, Energy Efficiency

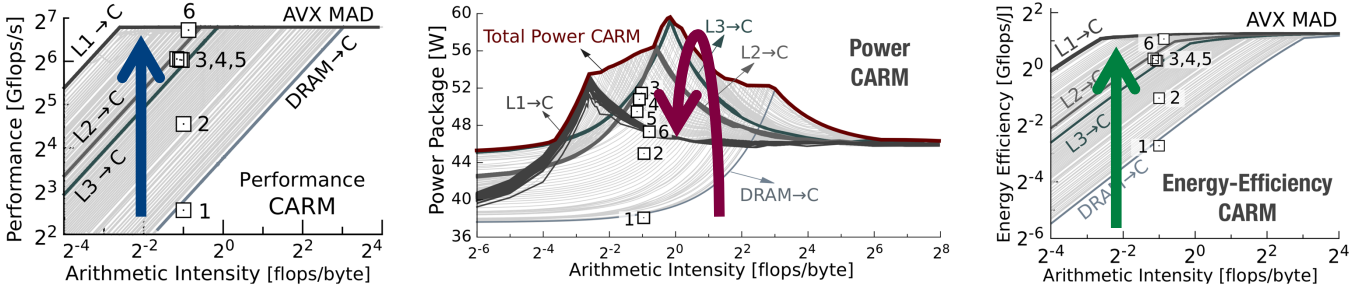






U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

Energy Roofline - GEMM

Cache-aware Roofline Models



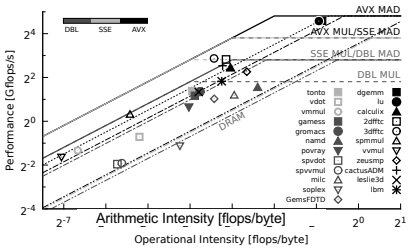
- **Power Consumption based on CARM**
  - **Relates Watts with FLOPs/bytes**
  - **Defines power envelope for different types of FP and memory operations**

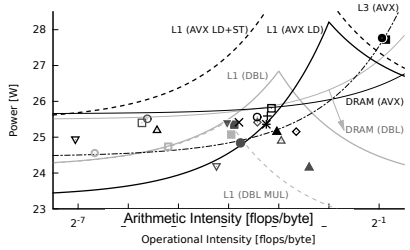
VERSION	OPTIMIZATION STRATEGY
1	Basic implementation: Row-major matrices
2	Improved memory access by transposing B matrix
3, 4, 5	Blocking for caches: L3 (pt. 3), L2 (pt. 4) and L1 (pt. 5)
6	Highly optimized Intel MKL implementation

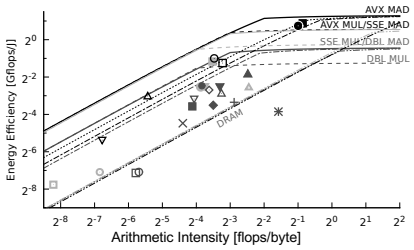
- A. Ilic, F. Pratas, and L. Sousa, “Cache-aware Roofline model: Upgrading the loft”, IEEE Computer Architecture Letters (2013)
- A. Ilic, F. Pratas and L. Sousa, “Beyond the Roofline: Cache-aware Power & Energy-Efficiency…”, IEEE Transactions on Computers (2017)

Use Cases

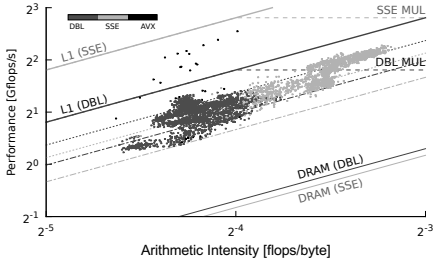
Application Characterization

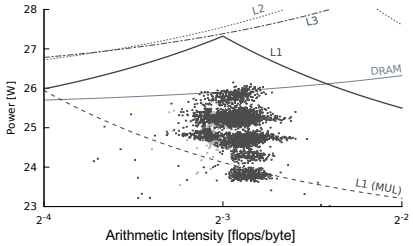


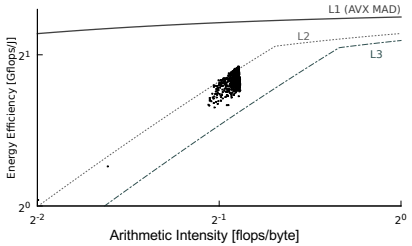




Online Monitoring



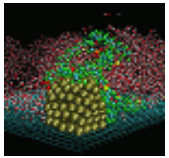
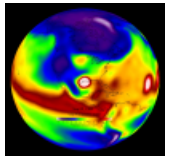
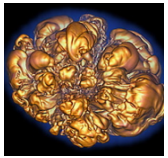
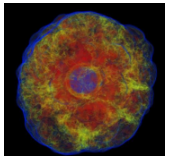
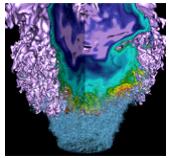




- Ilic, A., Pratas, F. and Sousa, L., “Beyond the Roofline: Cache-aware Power & Energy-Efficiency Modeling…”, IEEE Transactions on Computers (2017)
- Antão, D., et.al.,“Monitoring Performance and Power for Application Characterization with CARM”, PPAM’13

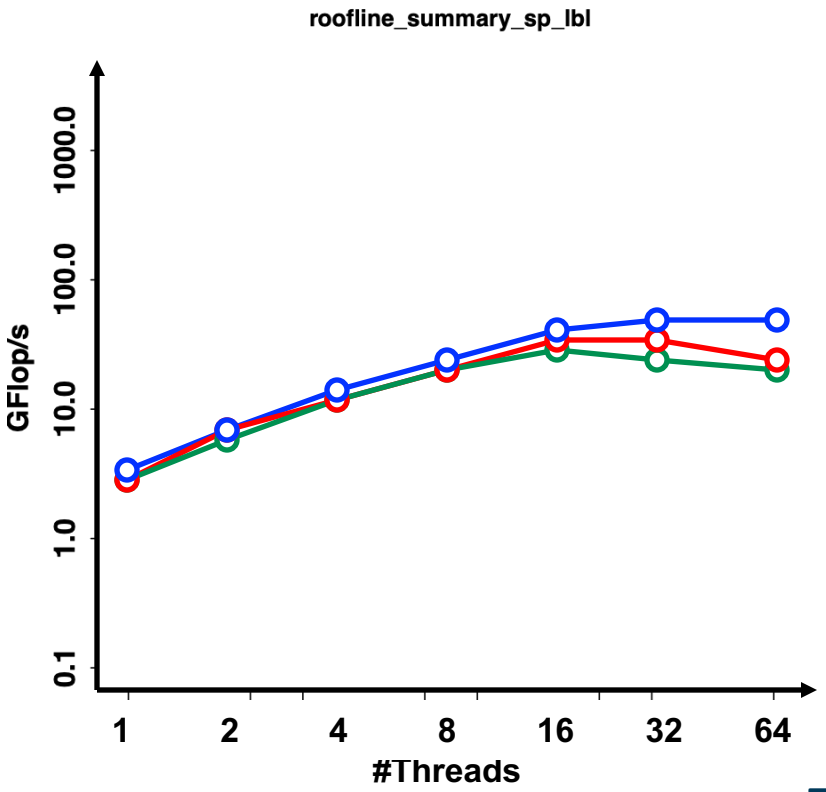
5. Scaling Trajectories

What’s causing bad scaling from Roofline point of view?

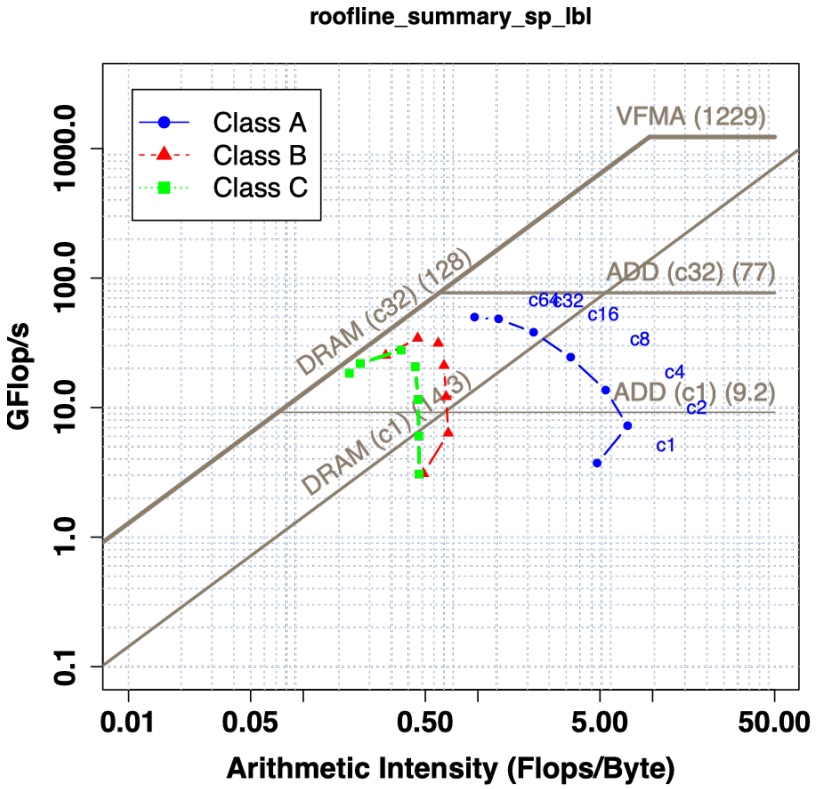
Roofline Scaling Trajectories

- We often plot performance as a function of thread concurrency
  - Carries no insight or analysis
  - Provides no actionable info



Roofline Scaling Trajectories

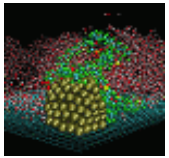
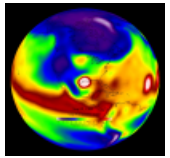
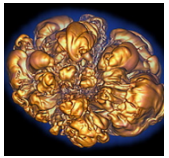
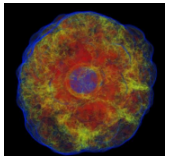
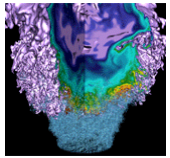
- We often plot performance as a function of thread concurrency
  - Carries no insight or analysis
  - Provides no actionable information.
- Use Roofline to analyze thread (or process) scalability
  - 2D scatter plot of performance as a function of intensity and concurrency
  - Identify loss in performance due to increased cache pressure (data movement)



- Khaled Ibrahim, Samuel Williams, Leonid Oliker, "Roofline Scaling Trajectories: A Method for Parallel Application and Architectural Performance Analysis", HPBench, July 2018

6. Mixed Precision

FP64, FP32, FP16, CPU, GPU

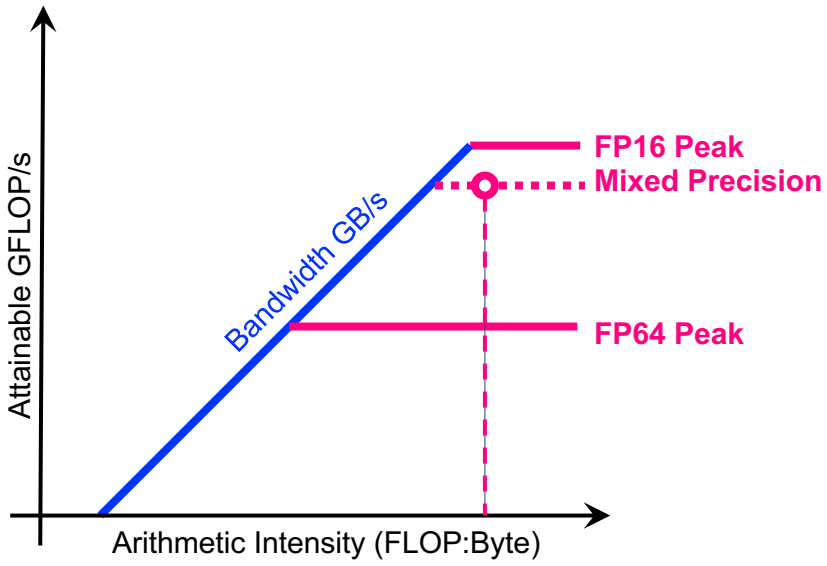






Mixed Precision

Benefits of reduced/mixed precision:

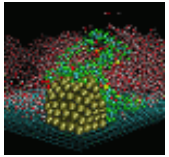
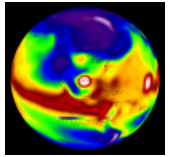
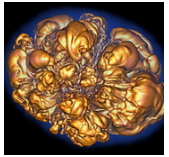
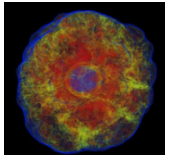
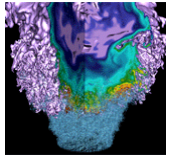
- From FP64 to FP32
  - 2x due to bandwidth savings or compute unit availability
  - similar for network communication
- More support on modern architectures
  - ~15x FP16 over FP64 for some ops

NESAP collaboration with **CRD (Costin Iancu)** and **NVIDIA (Chris Newburn)**



7. Instruction Roofline

FLOP, INTOP, IPC

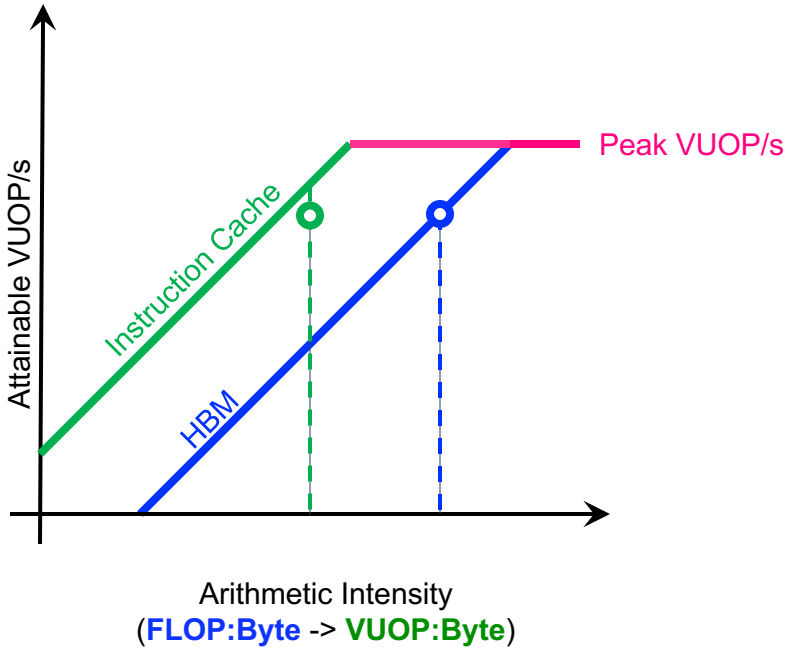






U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

Instruction Roofline

- FP instructions can be the minority in many HPC codes
- Emerging domains have ~no FP
  - Graphs
  - Hash tables
  - Bloom filters
  - Searches
- FLOPs is agnostic of precision, scalar/vectors/tensors, …



- **Instruction Roofline**

Instruction Roofline

FLOPs-based Roofline

- FMA doesn't change Arithmetic Intensity (FMA == FMUL+FADD)
- Vectors/tensors don't change Arithmetic Intensity
- Vector integer operations don't change Arithmetic Intensity
- Reducing precision (64b, 32b, 16b) **increases** Arithmetic Intensity

➢ **Tells us about performance**

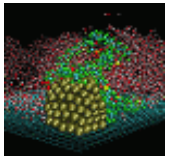
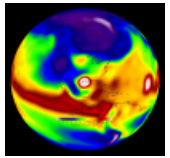
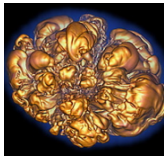
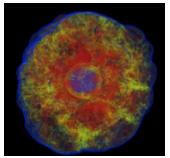
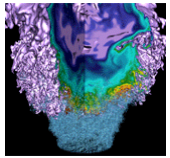
VUOPs-based Roofline

- FMA cuts Arithmetic Intensity in **half** (half the number of VUOPS)
- vectors/tensors reduce Arithmetic Intensity (**SIMD cuts VUOPS by 8x**)
- Vector integer operations **increases** Arithmetic Intensity
- Changing precision doesn't change Arithmetic Intensity

➢ **Tells us about VPU/pipeline utilization and bottlenecks**

8. Empirical Roofline Toolkit (ERT)

Machine Characterization, Peak FLOP/s, Bandwidths

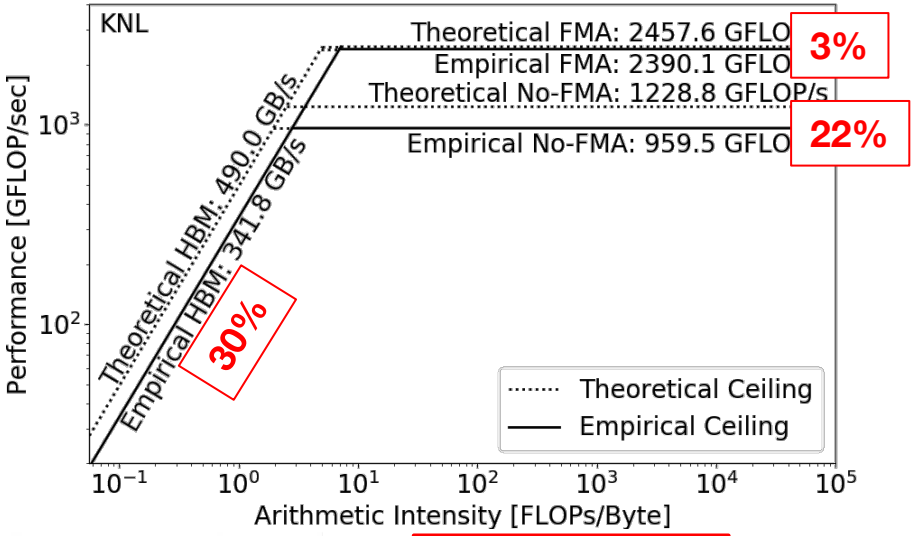
Empirical vs. Theoretical Ceilings

Theoretical compute ceiling on KNL:

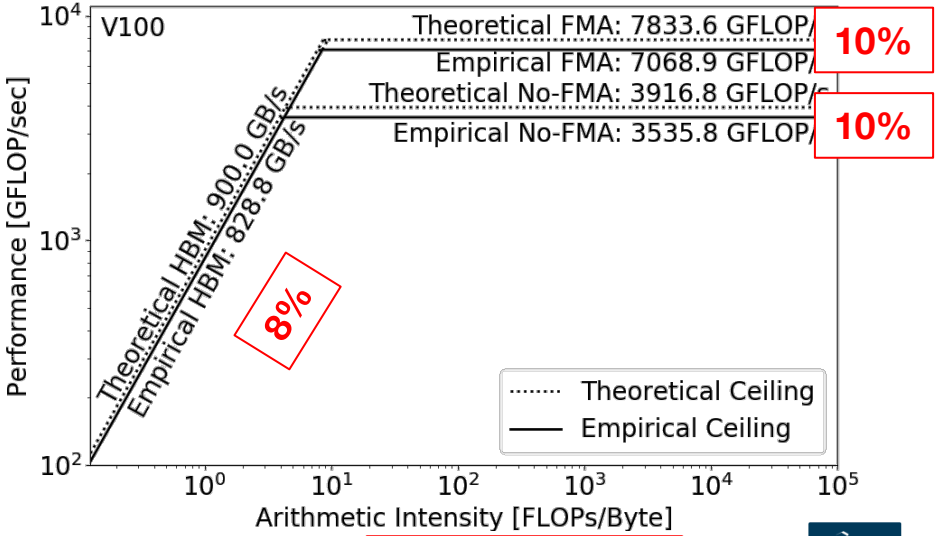
$$64 \text{ cores} \times 8 \text{ DP/vector} \times 2 \text{ FLOPs/FMA} \times 2 \text{ vectors} \times 1.2 \text{ GHz} = 2.46 \text{ TFLOP/s}$$

Theoretical compute ceiling on V100:

$$80 \text{ SMs} \times 32 \text{ FP64 cores/SM} \times 2 \text{ FLOPs/FMA} \times 1.53\text{GHz} = 7.83 \text{ TFLOP/s}$$



Cori KNL partition



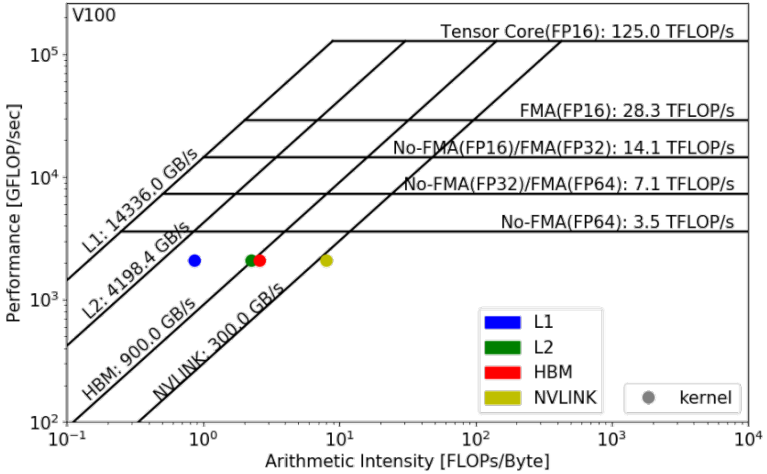
Voltar at UOregon

Machine Characterization

- ERT can't detect all the ceilings yet - IN DEVELOPMENT!
  - Haswell/KNL: L1, L2, L3/HBM, DDR
  - V100: L2, HBM, DDR

- Our goal is to incorporate
  - the full memory hierarchy
  - instruction mix (e.g. FMA/no-FMA)
  - data type (e.g. FP64, FP32, FP16)
  - compute units
    (e.g. CPU/CUDA core/Tensor core)

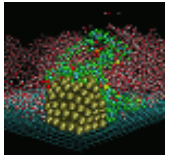
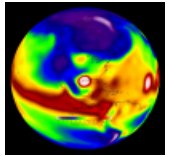
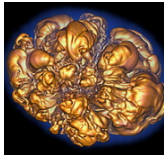
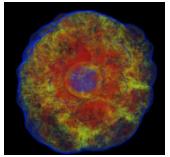
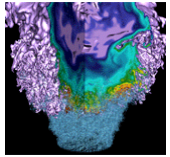
- Ceilings can be omitted if irrelevant



Empirical Roofline Toolkit (ERT). https://bitbucket.org/berkeleylab/cs-roofline-toolkit/

9. Outreach

Tutorials, Talks, Publications, Collaboration

Tutorials

- T. Koskela, A. Ilic, Z. Matveev, S. Williams, P. Thierry, C. Yang, Performance Tuning of Scientific Codes with the Roofline Model, SC'2017 Half-Day Tutorial, Nov 12-17 2017, Denver
- T. Koskela, A. Ilic, Z. Matveev, R. Belenov, C. Yang, L. Sousa, A Practical Approach to Application Performance tuning with the Roofline Model, ISC'2018 Half-Day Tutorial, Jun 24-28 2018, Frankfurt
- S. Williams, A. Ilic, Z. Matveev, C. Yang, Performance Tuning of Scientific Codes with the Roofline Model, SC'2018 Half-Day Tutorial, Nov 11-16 2018, Dallas
- S. Williams, J. Deslippe, C. Yang, P. Basu, Performance Tuning of Scientific Codes with the Roofline Model, Exascale Computing Project 2nd Annual Meeting, Feb 5-9, 2018, Knoxville
- C. Yang, Z. Matveev, A. Ilic, D. Marques, Performance Optimization of Scientific Codes with the Roofline Model, ISC'2019 Half-Day Tutorial, Jun 16-20 2019, Frankfurt
- S. Williams, J. Deslippe, C. Yang, Performance Tuning of Scientific Codes with the Roofline Model, Exascale Computing Project (ECP) Annual Meeting, Jan 14-18, 2019, Houston

Talks

- C. Yang, S. Williams, Performance Analysis of GPU-Accelerated Applications using the Roofline Model, GTC'2019, Mar 17-21 2019, San Jose
- C. Yang, Roofline Performance Analysis with nvprof, NERSC/NVIDIA F2F, Feb 7 2019, Berkeley
- C. Yang, S. Williams, Performance tools and performance counters, Intel PathForward Hack-a-Thon, Apr 10-12 2018, Santa Clara
- C. Yang, Performance Analysis of GPU-Accelerated Applications using the Roofline Model, Cray COE Webinar, Apr 11 2019, Berkeley
- J. Pennycook, C. Yang, J. Deslippe, Quantitatively Assessing Performance Portability with Roofline, IDEAS webinar, Jan 23 2019, https://ideas-productivity.org/
- C. Yang, T. Kurth, Roofline Performance Model and Intel Advisor, Performance Analysis and Modeling (PAM) Workshop, Feb 14-15 2018, Brookhaven National Laboratory
- C. Yang, Introduction to Performance & Scalability Tools, DOE CSGF Annual Review, Jul 15-19 2018, Arlington
- C. Yang, Using Intel Tools at NERSC, Intel KNL Training, May 22-23 2019, Berkeley

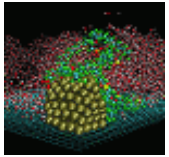
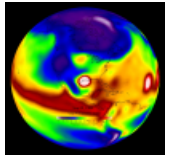
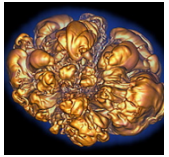
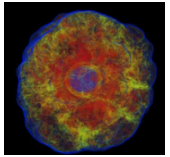
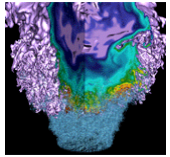
All Publications…

LBNL CRD Roofline Research:

https://crd.lbl.gov/departments/computer-science/PAR/research/roofline/publications/

Collaborate with us!

Closing

Roofline Helps

- Drive performance optimization
  - Application characterization, application readiness
- Create a dialogue between Applied Maths and CS
  - Communication-avoiding algorithms, high-order methods, new algorithms
- Facilitate cross-architecture comparisons for procurements
  - CPUs, GPUs, other accelerators

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