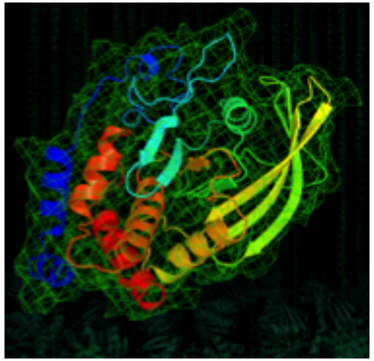
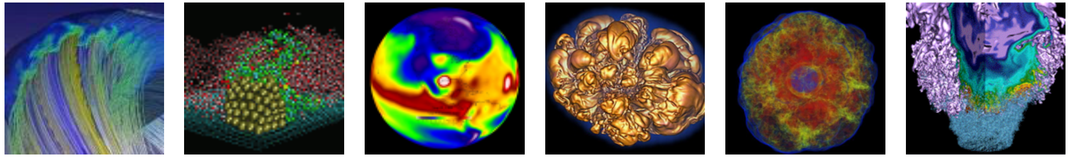
Performance-Related Activities at NERSC/CRD





Charlene Yang

Application Performance Specialist
NERSC, LBNL

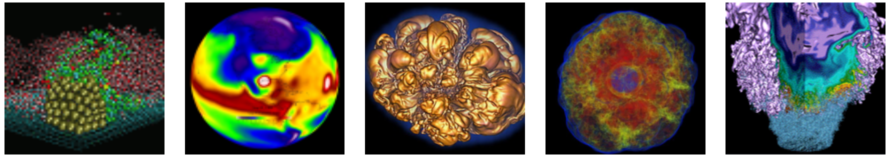
Activities at NERSC/CRD

- **Roofline performance model**
  - NESAP, vendor integration
  - Performance portability, scaling trajectories
  - Instruction Roofline, integer Roofline, mixed Precision
  - ERT, energy Roofline, Roofline for FPGA

- **LDMS for mass performance data collection**
  - *#SBATCH --profile=<tool>:<group>*
  - <tool> = vtune, likwid, ldms; <group> = flops, mem, bandwidth, …

- **PAPI for Roofline**
  - *#SBATCH --profile=timemory:roofline*

Roofline Performance Model



Roofline Performance Model

- Sustainable performance is bound by

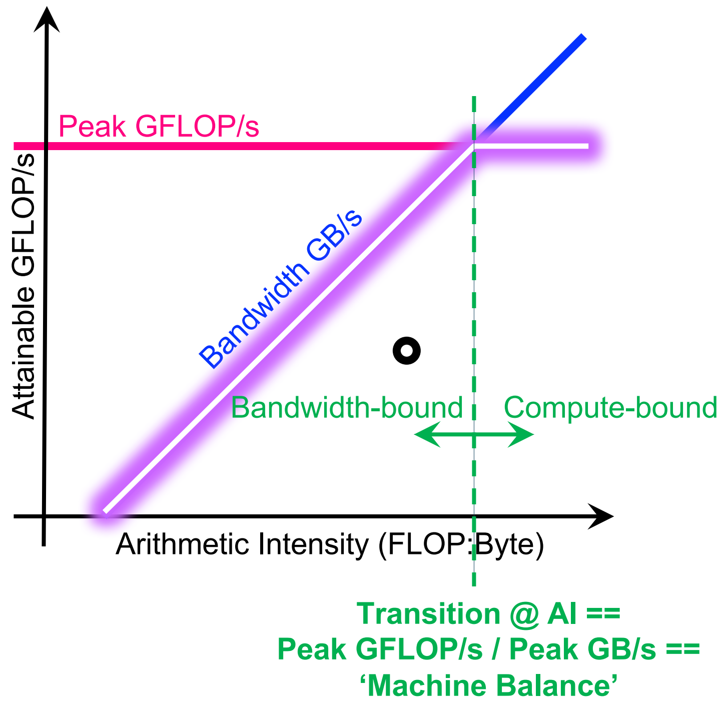
$$\text{GFLOP/s} = \min \begin{cases} \text{Peak GFLOP/s} \\ \text{AI * Peak GB/s} \end{cases}$$

- Arithmetic Intensity (AI) =

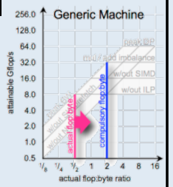
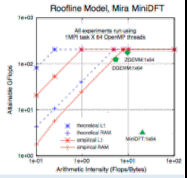
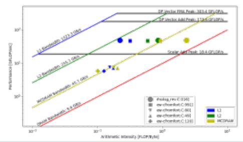
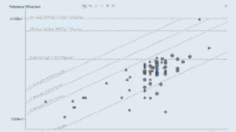
  FLOPs / Bytes

- How did this come about?

  → A CPU DRAM example



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…

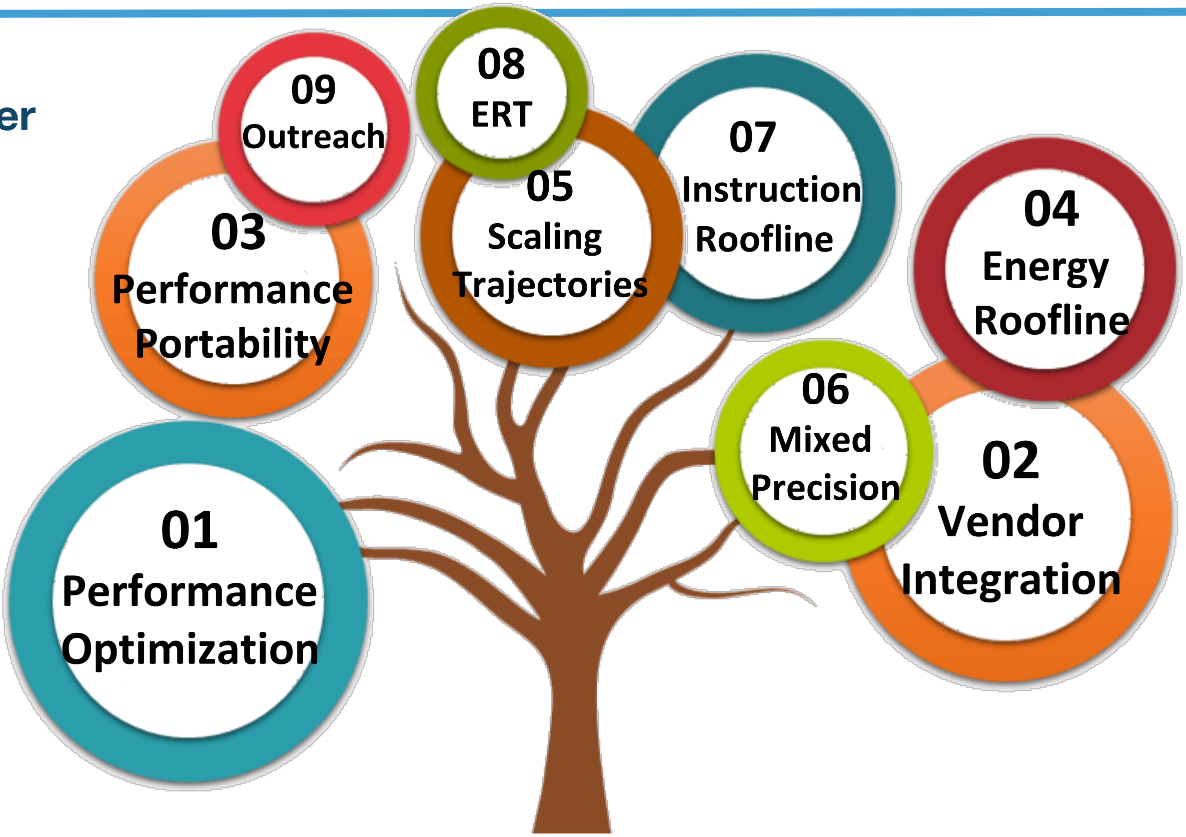
- Zakhar Matveev (Intel)
- Max Katz, Magnus Strengert (NVIDIA)
- Constantios Evangelinos (IBM)
- Protonu Basu (Facebook; formerly LBL/CRD)
- Linda Lo (Facebook; formerly U. Utah)
- David Patterson (Google, formerly UC Berkeley)

Thank You!

The Roofline Tree

Brings People Together

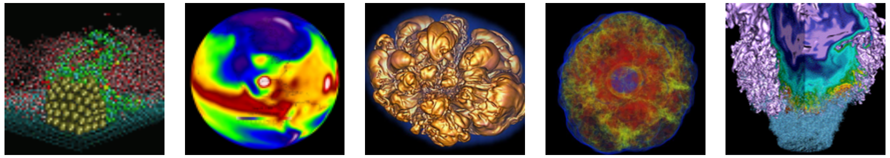
- NESAP
- CRD
- Intel
- NVIDIA
- all HPCers



Roofline Performance Model

1. Roofline drives optimization

NESAP



Roofline Drives Optimization

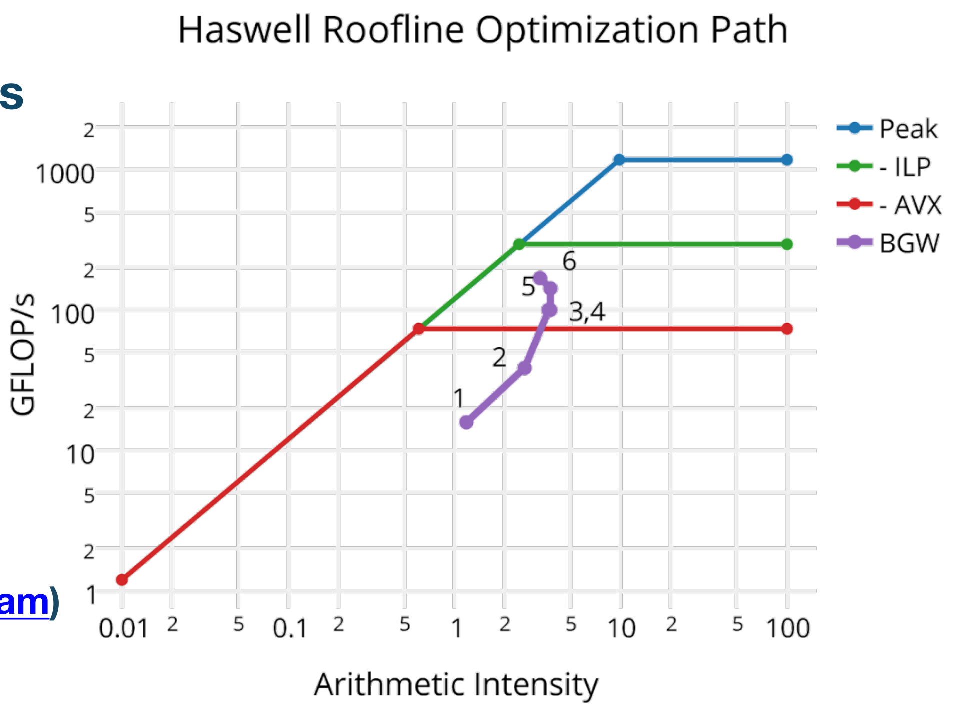
The Roofline Model

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

An example:

- **NESAP for Cori - BerkeleyGW**

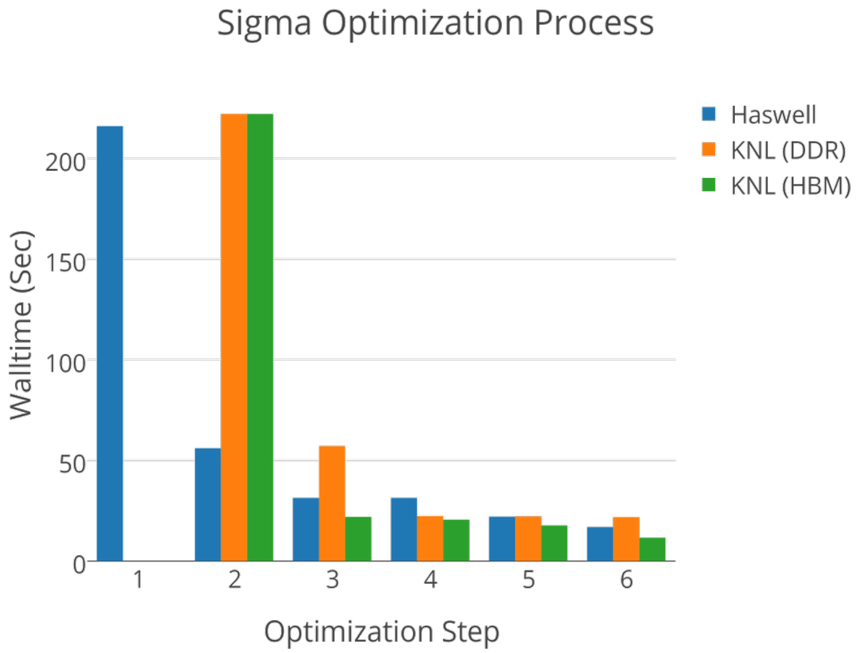
(NERSC Exascale Scientific Application Program)



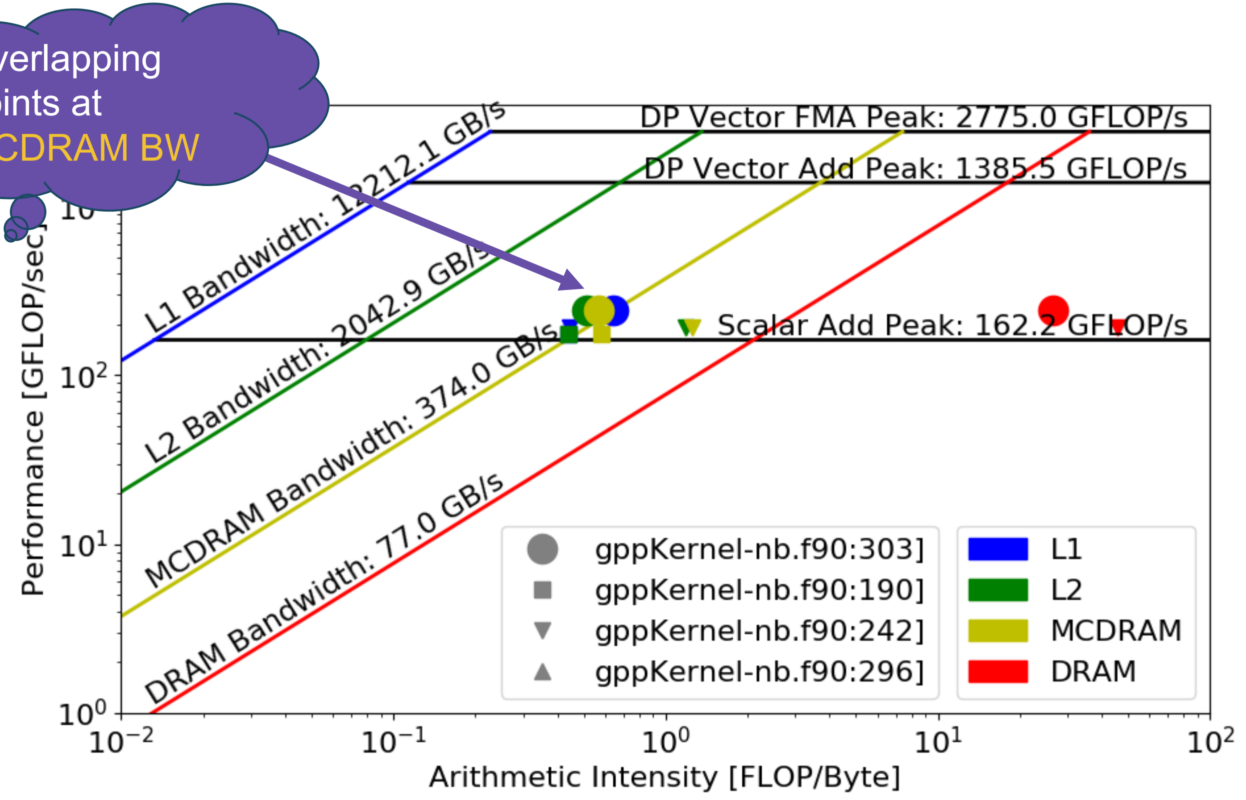
Roofline Drives Optimization

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**



Example 1: GPP, KNL, Cache Blocking



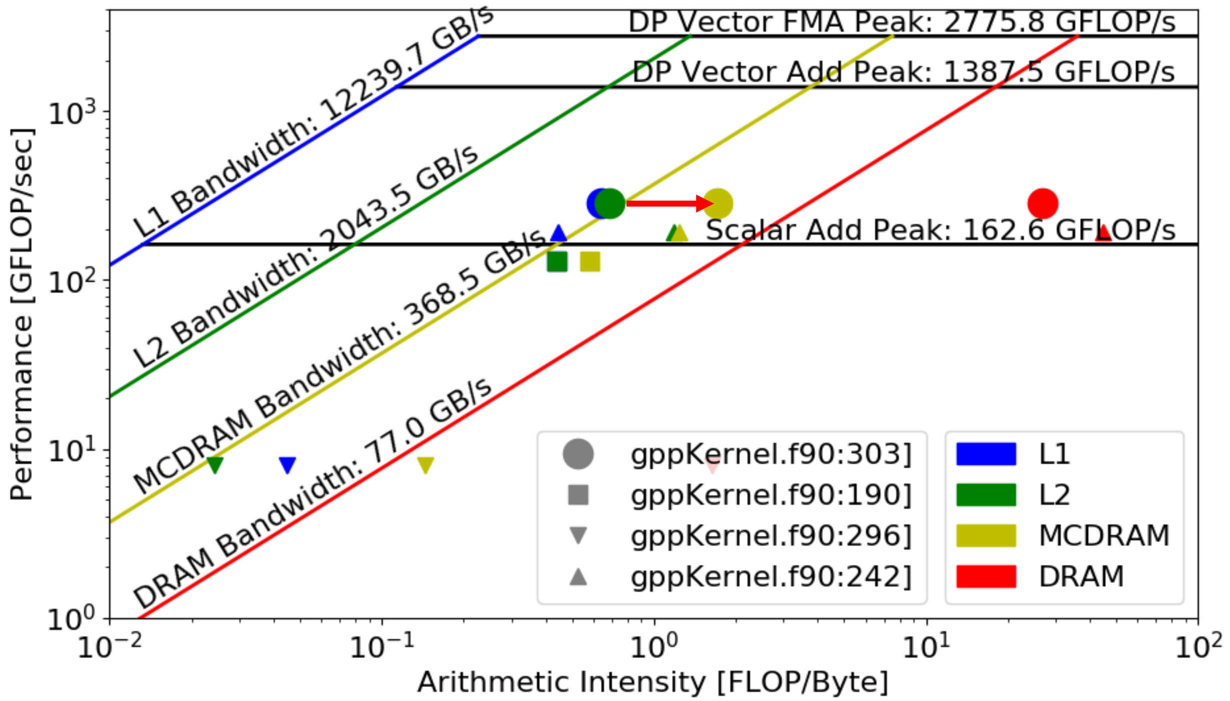
242 GFflop/s, **Bound by MCDRAM Bandwidth**

Most Flops in the main loop (⭕)

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

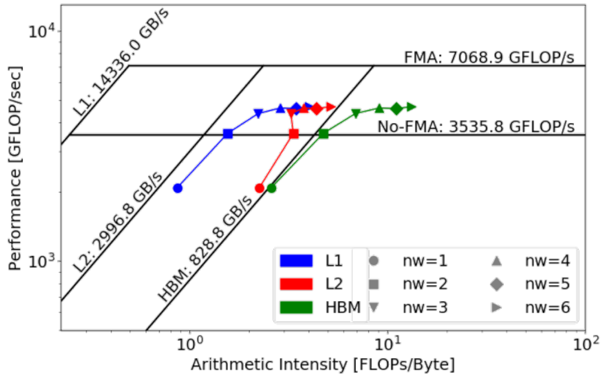
- T. Koskela, Z. Matveev, C. Yang, A. Adetokunbo, R. Belenov, P. Thierry, Z. Zhao, R. Gayatri, H. Shan, L. Oliker, J. Deslippe, R. Green, and S. Williams, A Novel Multi-Level Integrated Roofline Model Approach for Performance Characterization, ISC'2018 Research Paper, Jun 24-28 2018, Frankfurt

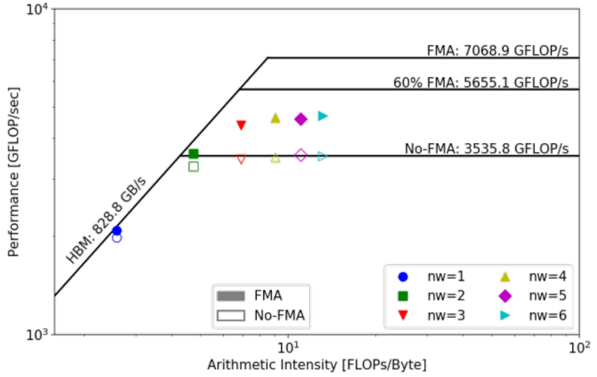
Example 1: GPP, V100, Hierarchical

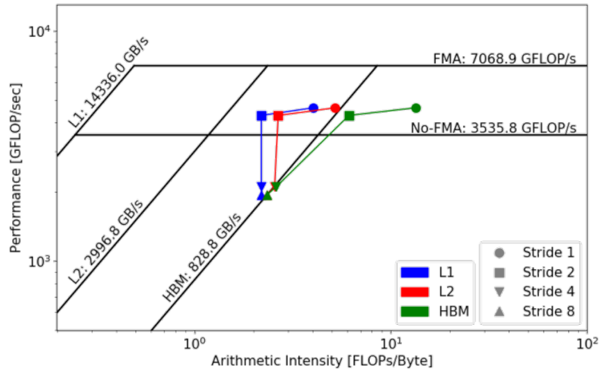
Three experiments to study the effects of

- 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
```

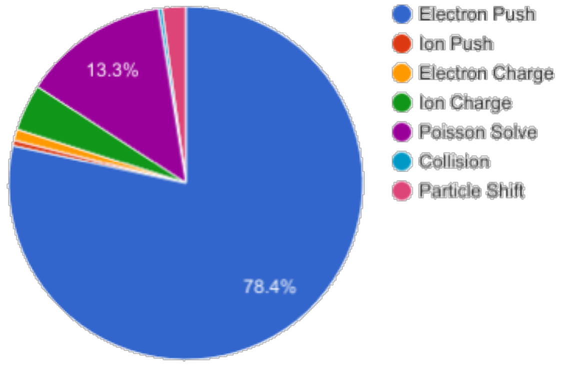


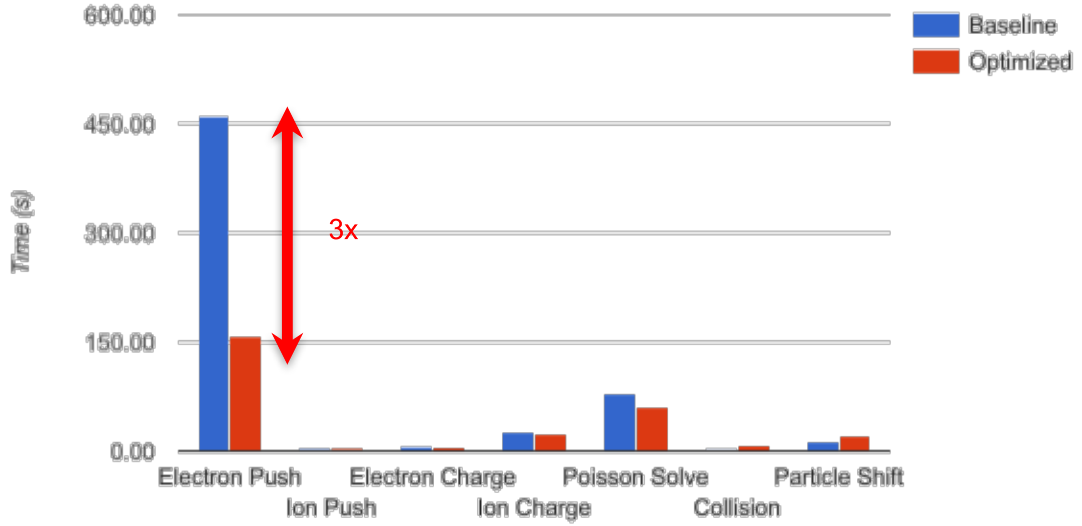




- Charlene Yang, Thorsten Kurth, Samuel Williams, "Hierarchical Roofline Analysis for GPUs: Accelerating Performance Optimization for the NERSC-9 Perlmutter System", Cray User Group (CUG), May 2019.

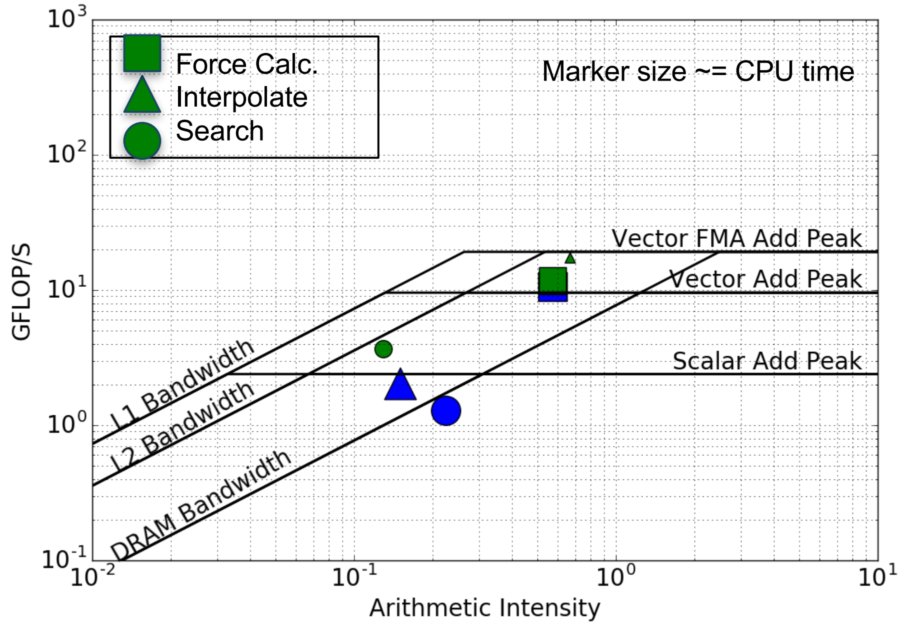
Example 2: XGC1, KNL





(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

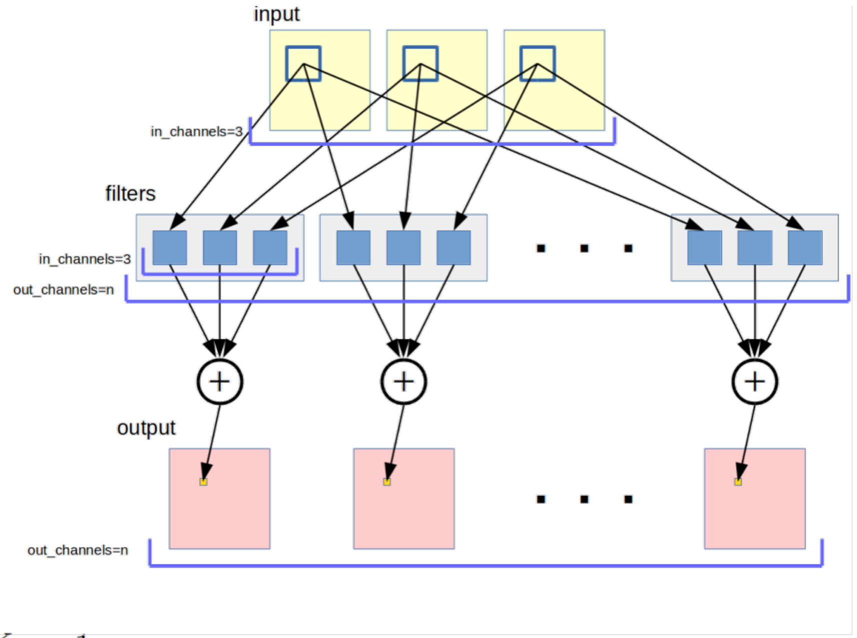
- Code is available at
- https://github.com/tkoskela/toypush

Example 3: conv2d from TensorFlow

- **Kernel tf.nn.conv2d**

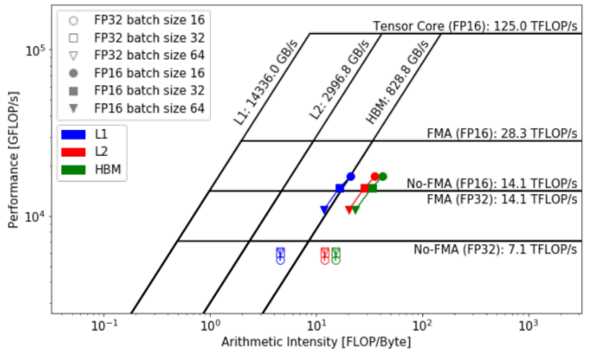
TensorFlow

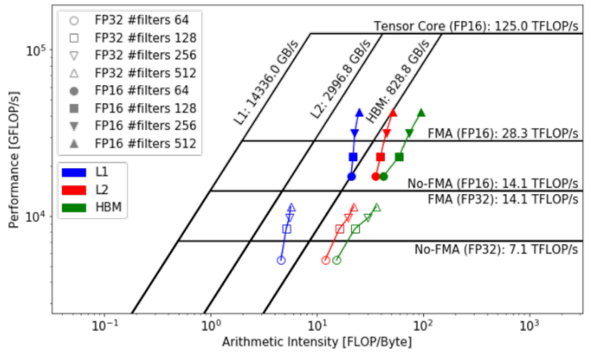
https://www.tensorflow.org

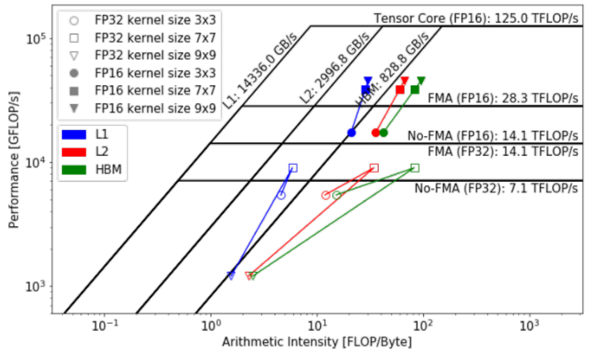


$$B_{nhwc} = \sum_{m=0}^{C-1} \sum_{k_h=0}^{K_H-1} \sum_{k_w=0}^{K_W-1} A_{n\, h+k_h\, w+w_h\, m} K_{k_h\, k_w\, m\, c}$$

Example 3: TF / Forward Pass







#Batch Size

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

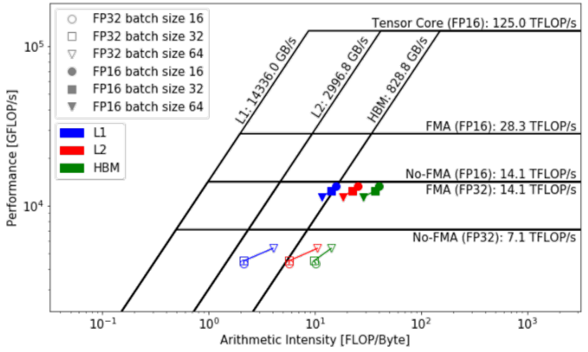
#Filters

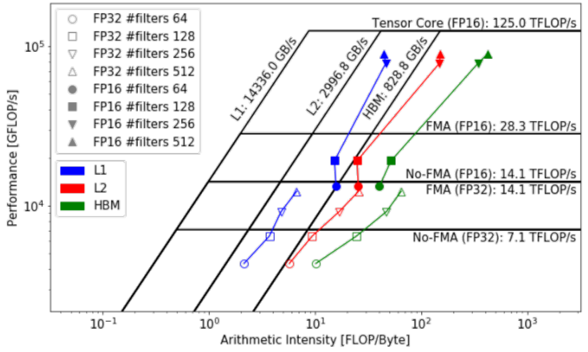
- Intensity ∝ #Filters
- Low L2 data locality
- Some use of TC's (>FP16 FMA)… ***partial TC ceiling***

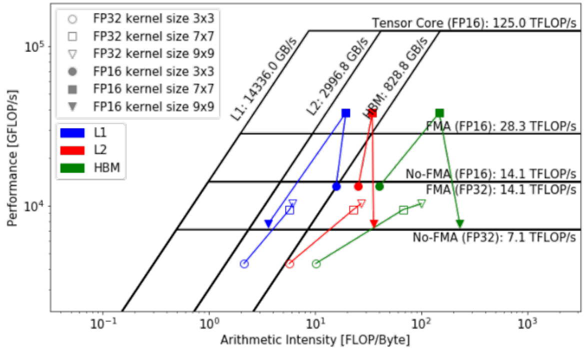
#Kernel Size

- 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

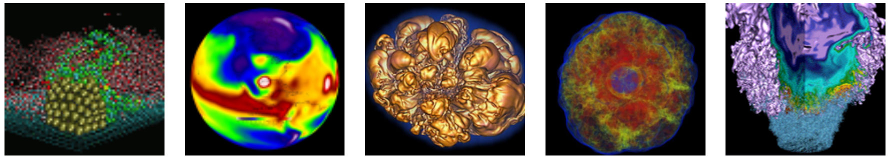
- Close to FP16 TC peak
- Close to FP32 FMA peak

#Kernel Size

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

2. Vendor Integration

Intel VTune, LIKWID, Intel Advisor, NVIDIA nvprof



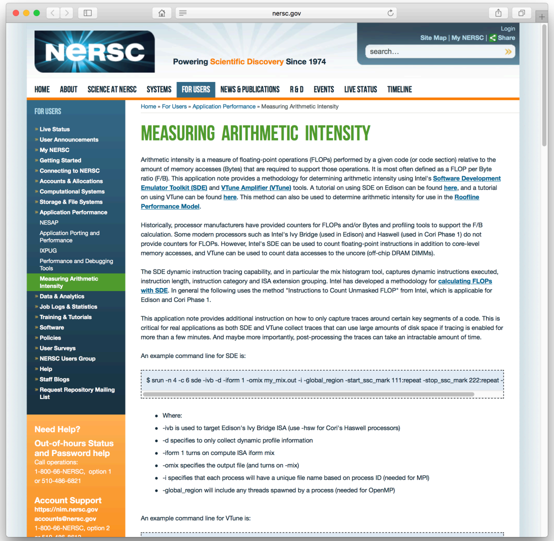
Data Collection on Intel CPUs

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/



Home » For Users » Application Performance » Measuring Arithmetic Intensity

MEASURING ARITHMETIC INTENSITY

Arithmetic intensity is a measure of floating-point operations (FLOPs) performed by a given code (or code section) relative to the amount of memory accesses (Bytes) that are required to support those operations. It is most often defined as a FLOP per Byte ratio (F/B). This application note provides a methodology for determining arithmetic intensity using Intel's Software Development Emulator Toolkit (SDE) and VTune Amplifier (VTune) tools. A tutorial on using SDE on Edison can be found here, and a tutorial on using VTune can be found here. This method can also be used to determine arithmetic intensity for use in the Roofline Performance Model.

Historically, processor manufacturers have provided counters for FLOPs and/or Bytes and profiling tools to support the F/B calculation. Some modern processors such as Intel's Ivy Bridge (used in Edison) and Haswell (used in Cori Phase 1) do not provide counters for FLOPs. However, Intel's SDE can be used to count floating-point instructions in addition to core-level memory accesses, and VTune can be used to count data accesses to the uncore (off-chip DRAM DIMMs).

The SDE dynamic instruction tracing capability, and in particular the mix histogram tool, captures dynamic instructions executed, instruction length, instruction category and ISA extension grouping. Intel has developed a methodology for calculating FLOPs with SDE. In general the following uses the method "Instructions to Count Unmasked FLOP" from Intel, which is applicable for Edison and Cori Phase 1.

This application note provides additional instruction on how to only capture traces around certain key segments of a code. This is critical for real applications as both SDE and VTune collect traces that can use large amounts of disk space if tracing is enabled for more than a few minutes. And maybe more importantly, post-processing the traces can take an intractable amount of time.

An example command line for SDE is:

```
$ srun -n 4 -c 6 sde -ivb -d -iform 1 -omix my_mix.out -i -global_region -start_ssc_mark 111:repeat -stop_ssc_mark 222:repeat -
```

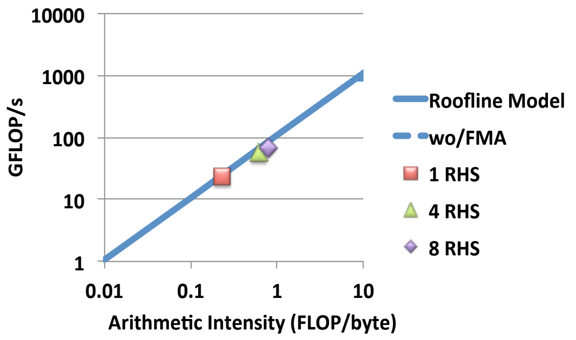
- Where:
- -ivb is used to target Edison's Ivy Bridge ISA (use -hsw for Cori's Haswell processors)
- -d specifies to only collect dynamic profile information
- -iform 1 turns on compute ISA iform mix
- -omix specifies the output file (and turns on -mix)
- -i specifies that each process will have a unique file name based on process ID (needed for MPI)
- -global_region will include any threads spawned by a process (needed for OpenMP)

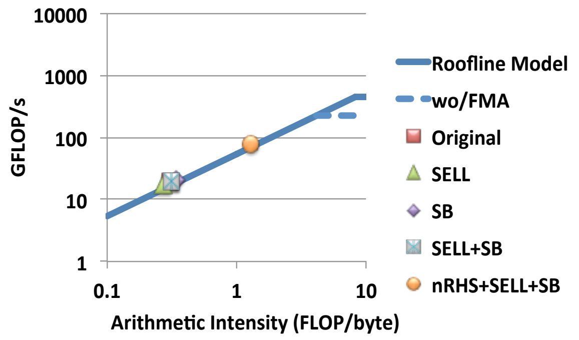
An example command line for VTune is:

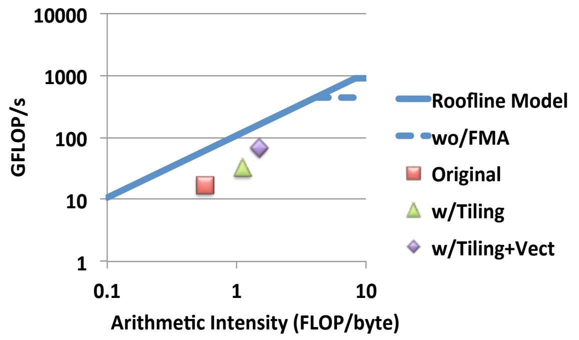
Data Collection on Intel CPUs

MFDn | EMGeo | PICSAR

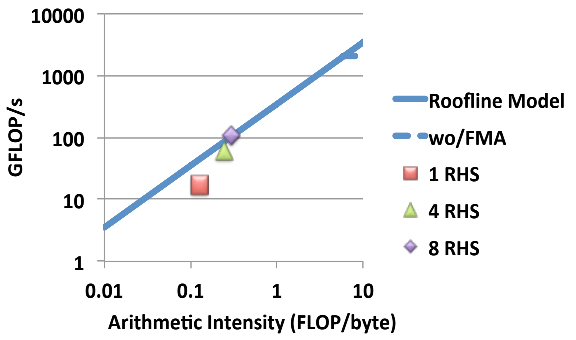
2P HSW

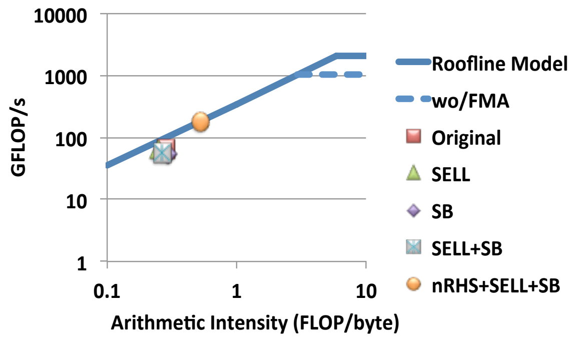


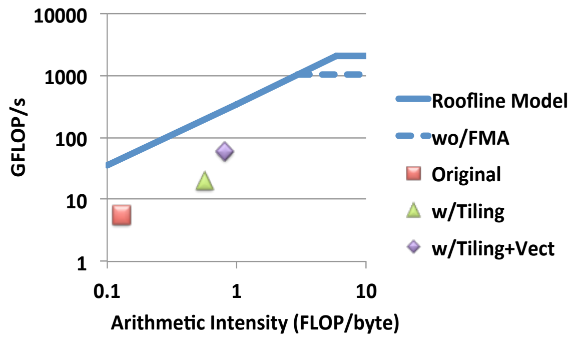




KNL





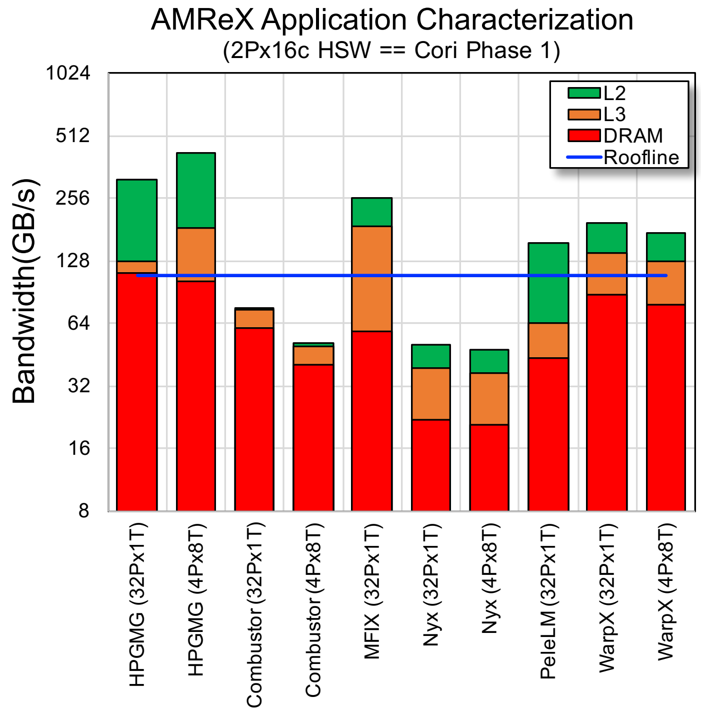


DRAM Rooflines of NESAP Codes

Data Collection on Intel CPUs

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 (need instrumentation)

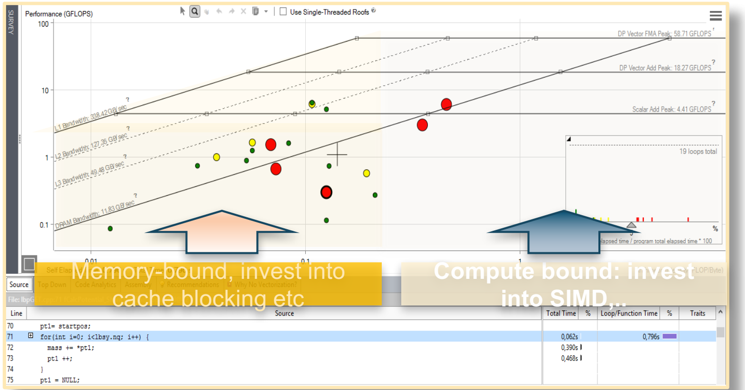


https://github.com/RRZE-HPC/likwid

Data Collection on Intel CPUs

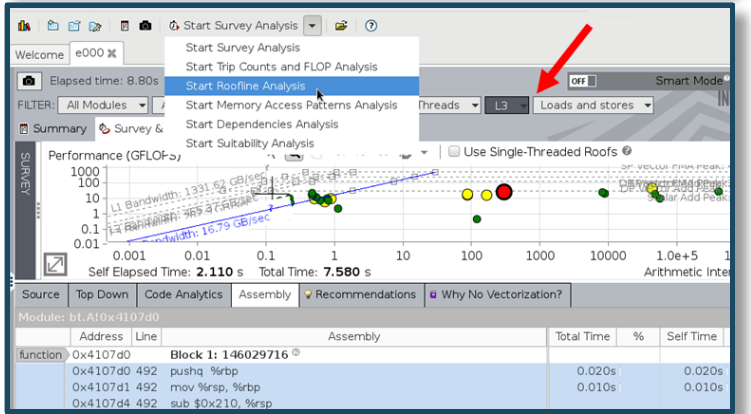
Way 3:

- 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



Performance (GFLOPS)
Use Single-Threaded Roofs
DP Vector FMA Peak: 58.71 GFLOPS
DP Vector Add Peak: 18.27 GFLOPS
Scalar Add Peak: 4.41 GFLOPS
L1 Bandwidth: 318.42 GB/sec
L2 Bandwidth: 127.35 GB/sec
L3 Bandwidth: 49.48 GB/sec
DRAM Bandwidth: 11.83 GB/sec
19 loops total
Memory-bound, invest into cache blocking etc
Compute bound: invest into SIMD,..
Source

Line	Source	Total Time	%	Loop/Function Time	%	Traits
70	`pt1= startpos;`					
71	`for(int i=0; i<lbsy.nq; i++) {`	0,062s		0,796s		
72	`mass += *pt1;`	0,390s				
73	`pt1 ++;`	0,468s				
74	`}`					
75	`pt1 = NULL;`					



Self Elapsed Time: **2.110** s Total Time: **7.580** s

Module: bt.A!0x4107d0

	Address	Line	Assembly	Total Time	%	Self Time
function	0x4107d0		**Block 1: 146029716**			
	0x4107d0	492	pushq %rbp	0.020s		0.020s
	0x4107d1	492	mov %rsp, %rbp	0.010s		0.010s
	0x4107d4	492	sub $0x210, %rsp			

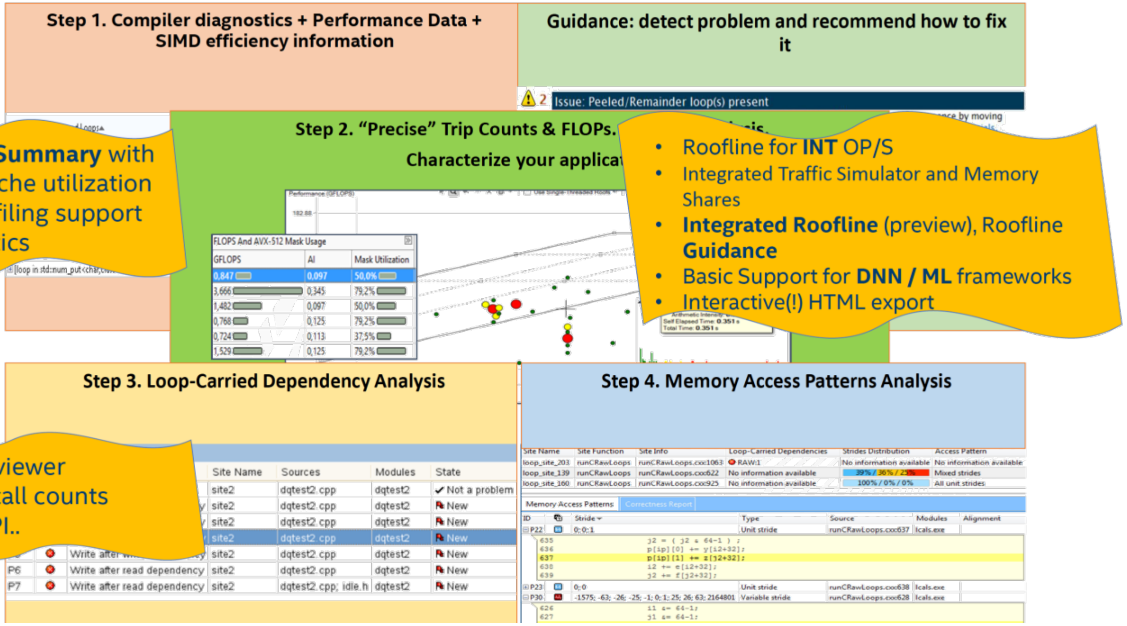
U.S. DEPARTMENT OF ENERGY | Office of Science

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



Step 1. Compiler diagnostics + Performance Data + SIMD efficiency information

Guidance: detect problem and recommend how to fix it

⚠ 2 Issue: Peeled/Remainder loop(s) present

Step 2. "Precise" Trip Counts & FLOPs.

Characterize your applica

FLOPS And AVX-512 Mask Usage

GFLOPS	AI	Mask Utilization
0,847	0,097	50,0%
3,666	0,345	79,2%
1,482	0,097	50,0%
0,768	0,125	79,2%
0,724	0,113	37,5%
1,529	0,125	79,2%

Step 3. Loop-Carried Dependency Analysis

			Site Name	Sources	Modules	State
			site2	dqtest2.cpp	dqtest2	✓ Not a problem
			site2	dqtest2.cpp	dqtest2	New
			site2	dqtest2.cpp	dqtest2	New
			site2	dqtest2.cpp	dqtest2	New
	●	Write after w	site2	dqtest2.cpp	dqtest2	New
P6	●	Write after read dependency	site2	dqtest2.cpp	dqtest2	New
P7	●	Write after read dependency	site2	dqtest2.cpp; idle.h	dqtest2	New

Step 4. Memory Access Patterns Analysis

Site Name	Site Function	Site Info	Loop-Carried Dependencies	Strides Distribution	Access Pattern
loop_site_203	runCRawLoops	runCRawLoops.cxx:1063	● RAW:1	No information available	No information available
loop_site_139	runCRawLoops	runCRawLoops.cxx:622	No information available	39% / 36% / 25%	Mixed strides
loop_site_160	runCRawLoops	runCRawLoops.cxx:925	No information available	100% / 0% / 0%	All unit strides

Memory Access Patterns | Correctness Report

ID		Stride	Type	Source	Modules	Alignment
P22		0; 0; 1	Unit stride	runCRawLoops.cxx:637	lcals.exe	

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

ID		Stride	Type	Source	Modules	Alignment
P23		0; 0	Unit stride	runCRawLoops.cxx:638	lcals.exe	
P30		-1575; -63; -26; -25; -1; 0; 1; 25; 26; 63; 2164801	Variable stride	runCRawLoops.cxx:628	lcals.exe	

```
i1 &= 64-1;
j1 &= 64-1;
```

- Renovated **Summary** with Memory/Cache utilization
- **Python** profiling support
- **VNNI** analytics

- Roofline for **INT** OP/S
- Integrated Traffic Simulator and Memory Shares
- **Integrated Roofline** (preview), Roofline **Guidance**
- Basic Support for **DNN / ML** frameworks
- Interactive(!) HTML export

- MAC OS* viewer
- Function call counts
- Python API..

Data Collection on NVIDIA GPUs

- Still manual at this stage, but we have a recipe using nvprof.

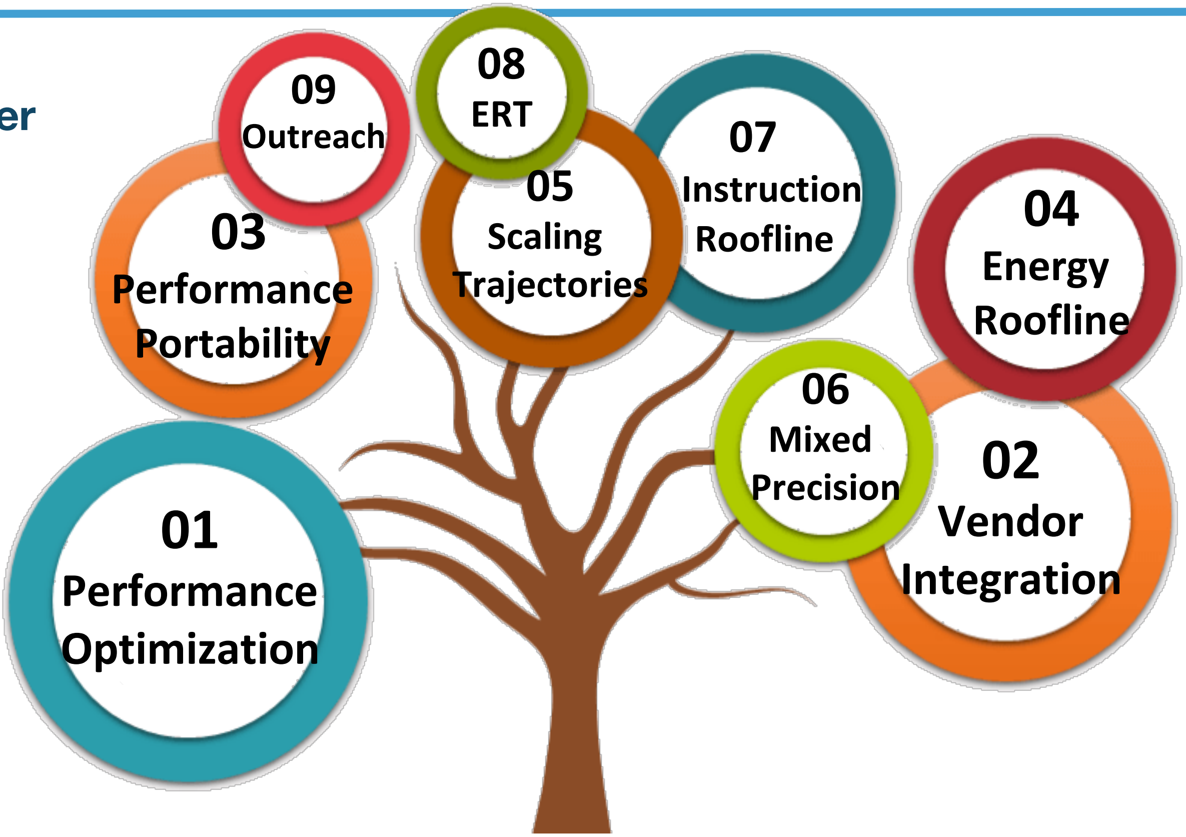
- Runtime:
  - Internal timers or `nvprof --print-gpu-trace`
- FLOPs:
  - DP/SP/HP counters and metrics, `nvprof --metrics 'flop_count_dp/sp/hp'` or ``` `tensor_precision_fu_utilization' ```
- Bytes for different cache levels:
  - Bytes = (read transactions + write transactions) x transaction size
 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

- Hierarchical Roofline

The Rest of the Tree

Brings People Together

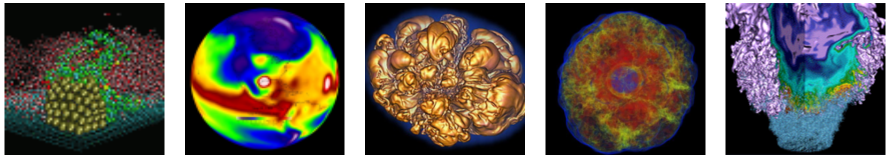
- NESAP
- CRD
- Intel
- NVIDIA
- all HPCers



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*…

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

- Architectual Efficiency [Williams *et al*]

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

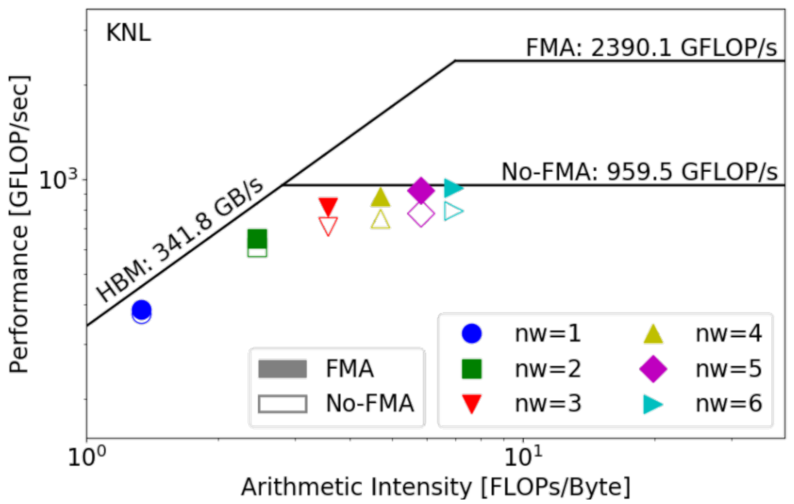
Peak FLOP/s

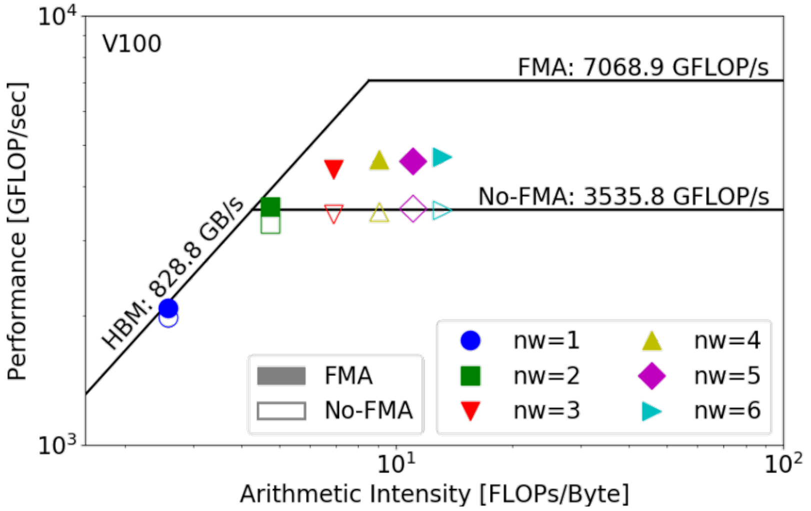
Arithmetic Intensity

Peak Bandwidth

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

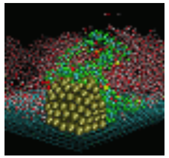
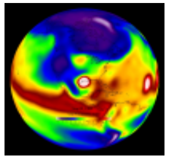
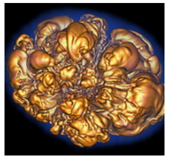
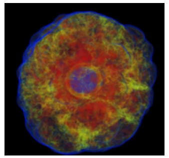
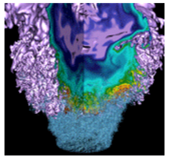
Bottleneck Changes

- **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%**

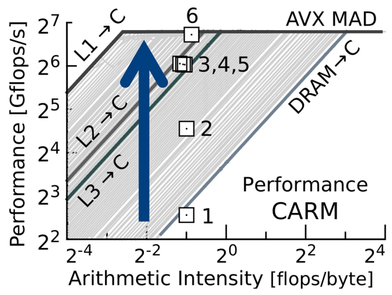
4. Energy Roofline

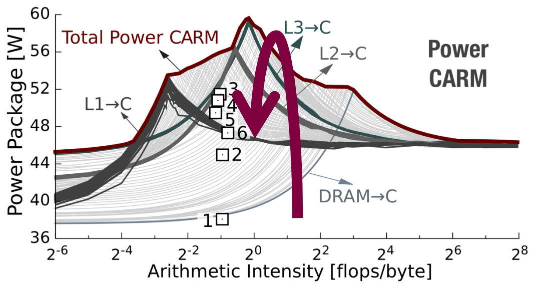
Performance, Power Consumption, Energy Efficiency

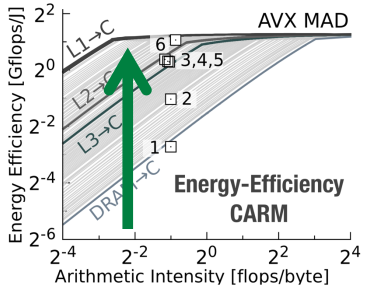






Energy Roofline - GEMM

Cache-aware Roofline Models







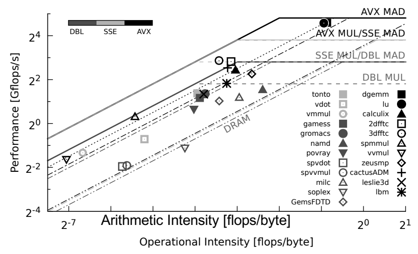
- **Power Consumption based on CARM**
  - **Relates Watts with FLOPs/bytes**
  - **Defines power envelope for different types of FP and memory operations**

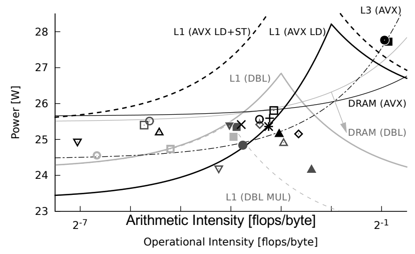
VERSION	OPTIMIZATION STRATEGY
1	Basic implementation: Row-major matrices
2	Improved memory access by transposing B matrix
3, 4, 5	Blocking for caches: L3 (pt. 3), L2 (pt. 4) and L1 (pt. 5)
6	Highly optimized Intel MKL implementation

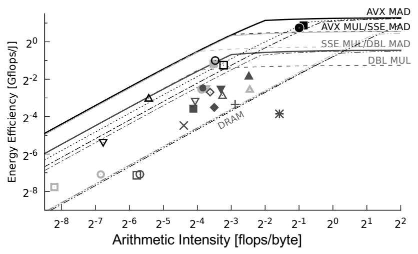
- A. Ilic, F. Pratas, and L. Sousa, “Cache-aware Roofline model: Upgrading the loft”, IEEE Computer Architecture Letters (2013)
- A. Ilic, F. Pratas and L. Sousa, “Beyond the Roofline: Cache-aware Power & Energy-Efficiency…”, IEEE Transactions on Computers (2017)

Energy Roofline - Use Cases

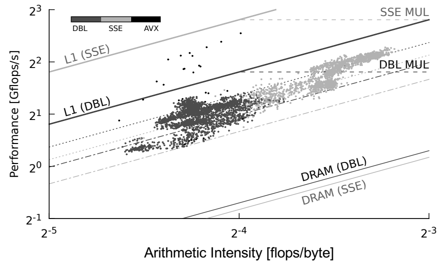
Application Characterization

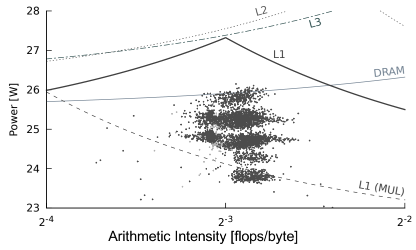


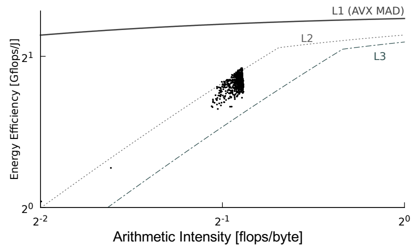




Online Monitoring



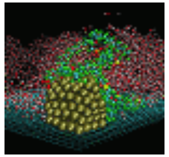
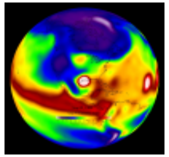
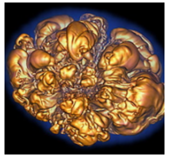
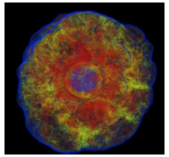
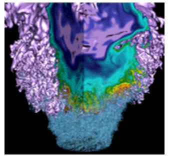




- Ilic, A., Pratas, F. and Sousa, L., “Beyond the Roofline: Cache-aware Power & Energy-Efficiency Modeling…”, IEEE Transactions on Computers (2017)
- Antão, D., et.al.,“Monitoring Performance and Power for Application Characterization with CARM”, PPAM’13

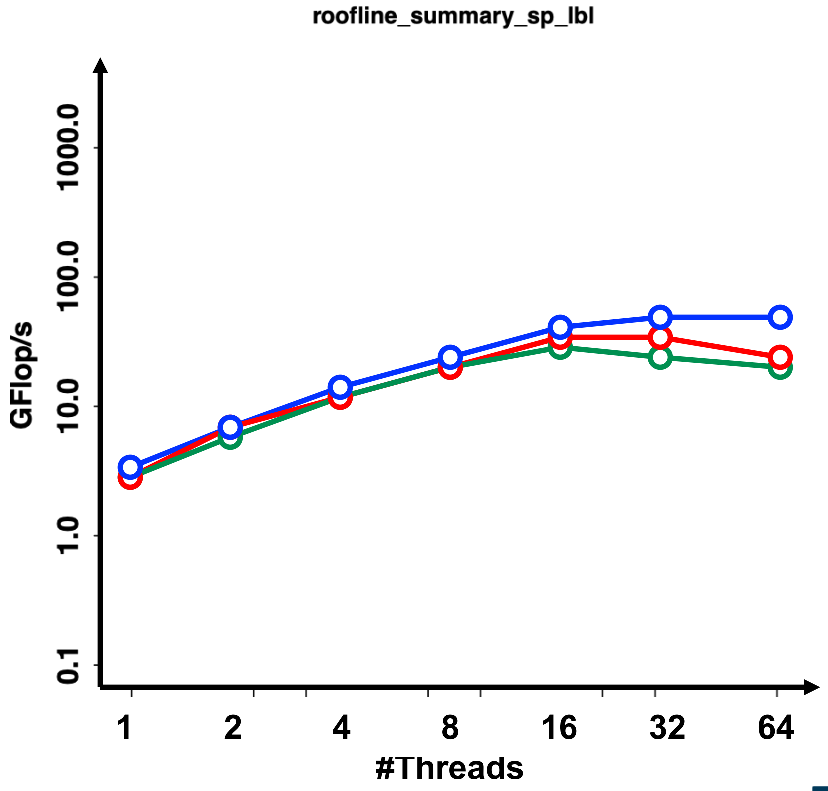
5. Scaling Trajectories

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

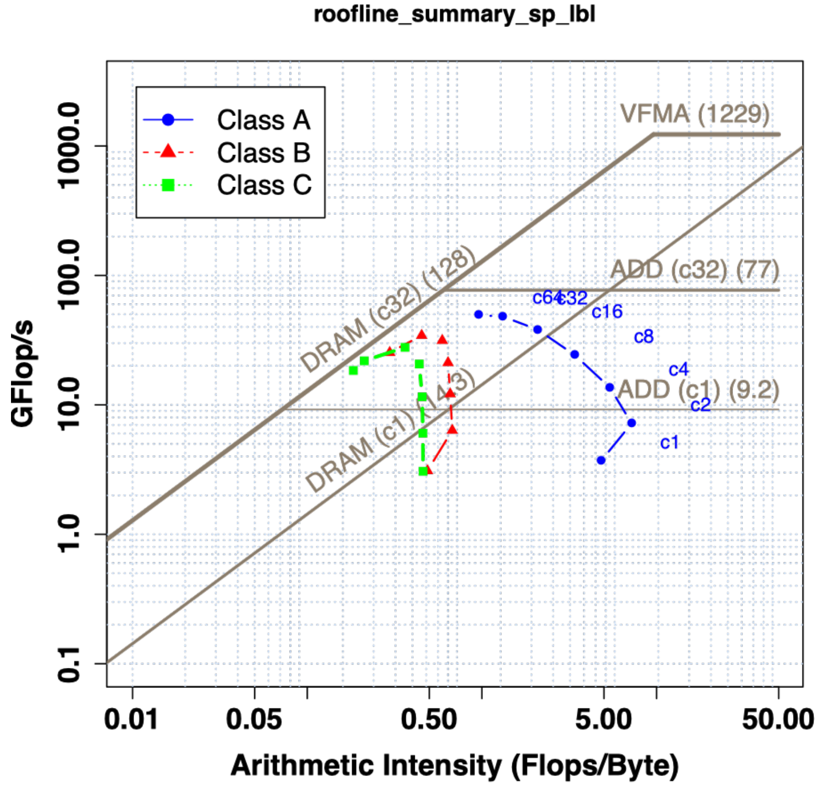
Roofline Scaling Trajectories

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



Roofline Scaling Trajectories

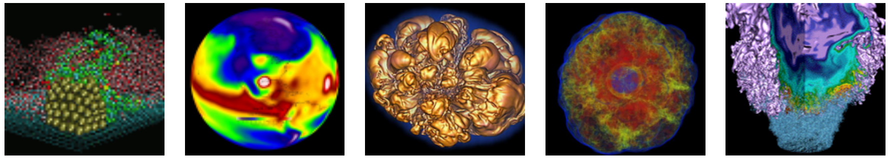
- We often plot performance as a function of thread concurrency
  - Carries no insight or analysis
  - Provides no actionable information.
- Use Roofline to analyze thread (or process) scalability
  - 2D scatter plot of performance as a function of intensity and concurrency
  - Identify loss in performance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

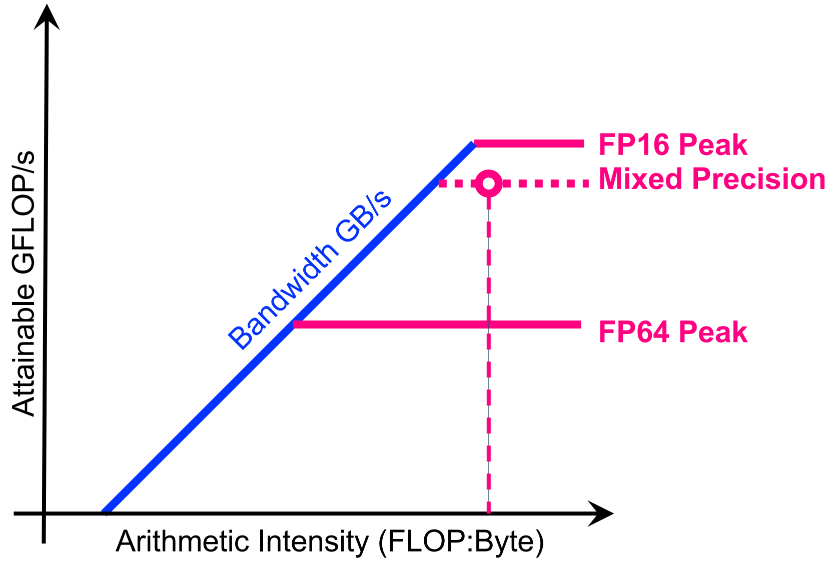


Mixed Precision

Benefits of reduced/mixed precision:

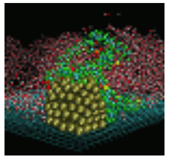
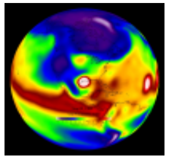
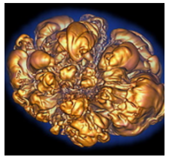
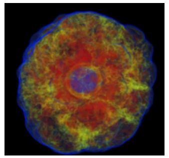
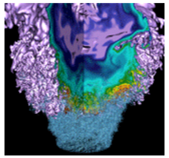
- From FP64 to FP32
  - 2x due to bandwidth savings or compute unit availability
  - similar for network communication
- More support on modern architectures
  - ~15x FP16 over FP64 for some ops

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



7. Instruction Roofline

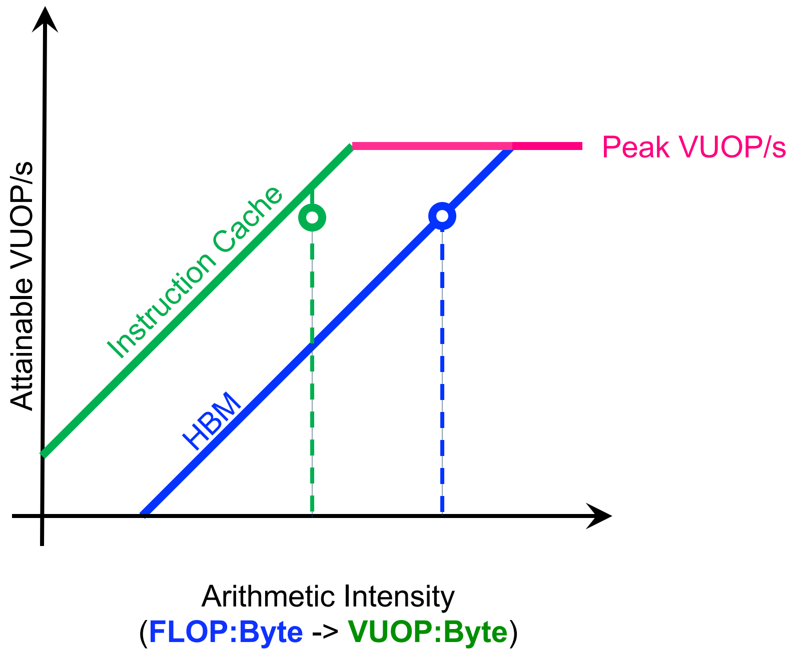
FLOP, INTOP, IPC

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**

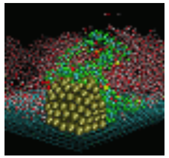
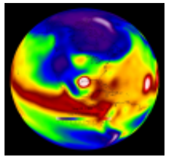
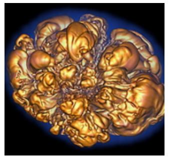
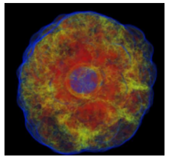
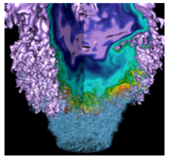
VUOPs-based Roofline

- 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

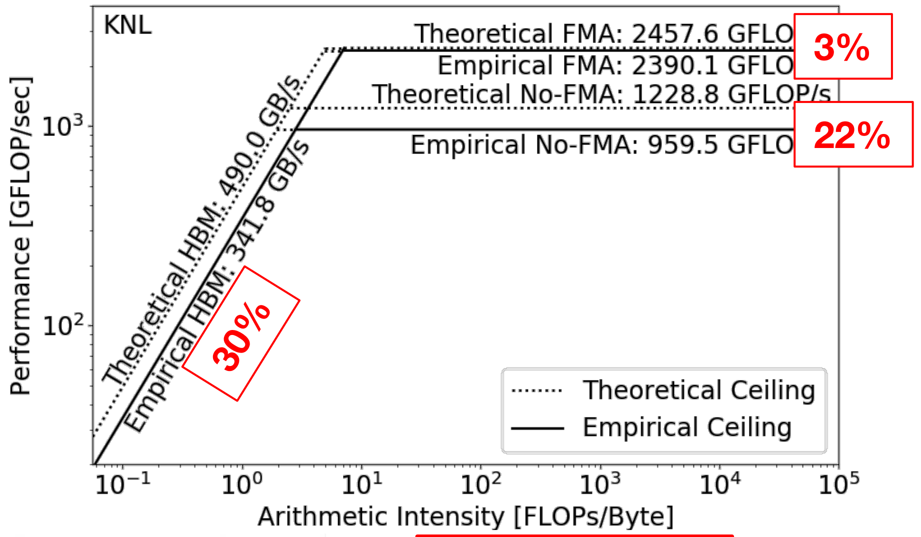
Empirical Roofline Toolkit (ERT)

Theoretical compute ceiling on KNL:

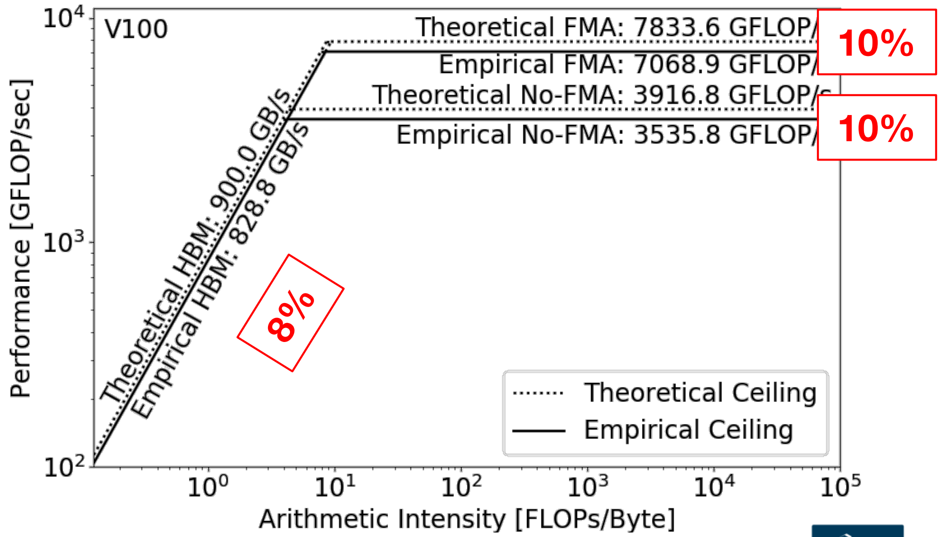
$$64\ \text{cores} \times 8\ \text{DP/vector} \times 2\ \text{FLOPs/FMA} \times 2\ \text{vectors} \times 1.2\ \text{GHz} = 2.46\ \text{TFLOP/s}$$

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



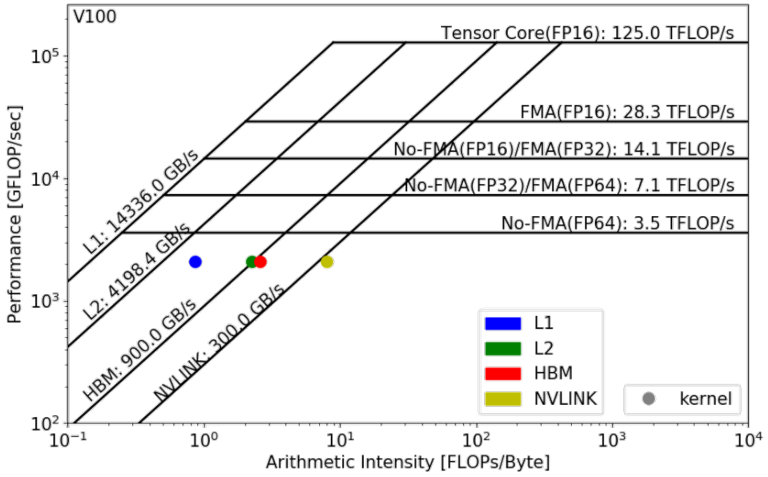
Voltar at UOregon

Empirical Roofline Toolkit (ERT)

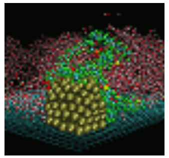
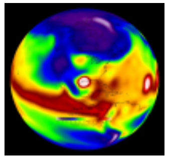
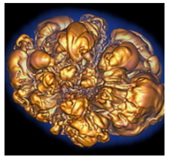
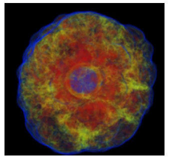
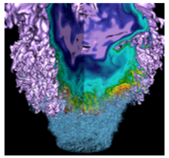
- ERT can't detect all the ceilings yet - IN DEVELOPMENT!
  - Haswell/KNL: L1, L2, L3/HBM, DDR
  - V100: L2, HBM, DDR

- 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



Closing

The Roofline Tree is Flourishing

LBNL CRD Roofline Research:

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

Collaborate with us!

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