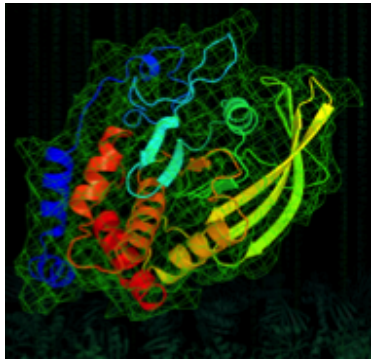
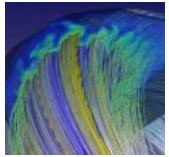
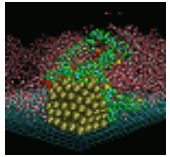
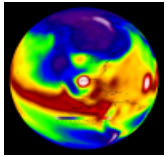
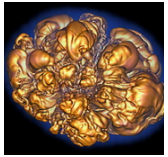
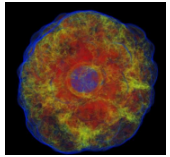
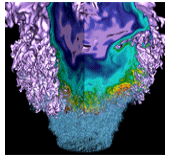
Introduction to the Roofline Model



Charlene Yang

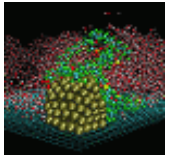
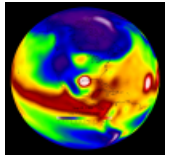
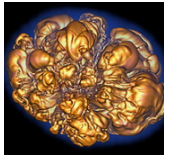
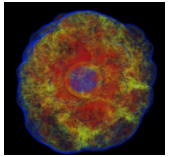
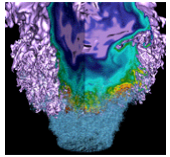
Lawrence Berkeley National Laboratory

June 24, 2018, Frankfurt

Outline

- Performance modeling: Why use performance models or tools?
- Roofline Model:
  - Arithmetic intensity (AI) and bandwidth
    - DRAM Roofline, stream/7pt stencil example
    - Hierarchical Roofline is superposition of rooflines
  - Modeling in-core performance effects
    - Data/instruction/thread level parallelism gives different roofs!
    - Divides/sqrts affect roofs as well!
  - Modeling cache effects - Locality matters!
  - General optimization strategy (In-core/memory bandwidth/data locality)
- Constructing a Roofline Model requires knowledge of machine/application/problem/*etc*
- Performance tools: tools available, NESAP, Advisor's Roofline feature
- Comparison of Hierarchical Roofline and CARM: stream/7pt stencil example

Performance Modeling

Why Performance Models or Tools?

- **Identify performance bottlenecks**
- **Motivate software optimizations**
- **Determine when we're done optimizing**
  - Assess performance relative to machine capabilities
  - Motivate need for algorithmic changes
- **Predict performance on future machines / architectures**
  - Sets realistic expectations on performance for future procurements
  - Used for HW/SW Co-Design to ensure future architectures are well-suited for the computational needs of today's applications.

Contributing Factors

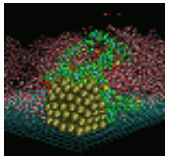
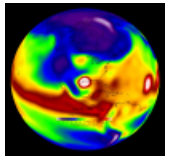
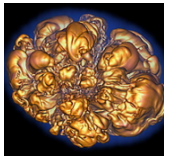
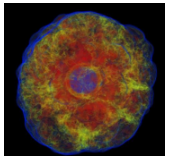
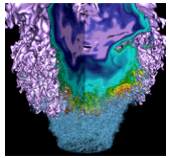
- **Many different components contribute to kernel run time.**
- **Characteristics of the application, machine, or both.**
- **Focus on one or two dominant components.**

#FP operations	Flop/s
Cache data movement	Cache GB/s
DRAM data movement	DRAM GB/s
PCIe data movement	PCIe bandwidth
Depth	OMP Overhead
MPI Message Size	Network Bandwidth
MPI Send:Wait ratio	Network Gap
#MPI Wait's	Network Latency

Roofline Model

Right model depends on application size and problem size!

Roofline Model:
Arithmetic Intensity and Bandwidth

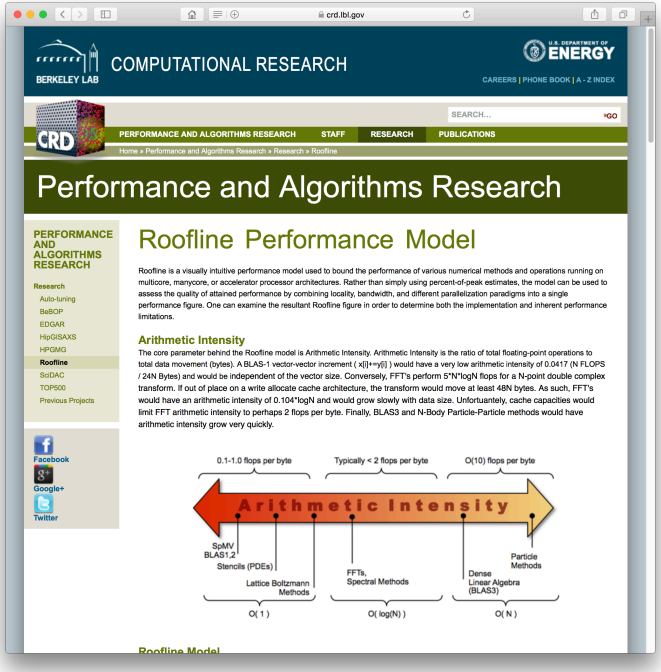






U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

Roofline Model

- Roofline Model is a **throughput-oriented** performance model…
  - Tracks rates not times
  - Augmented with Little's Law
    (concurrency = latency*bandwidth)
  - Independent of ISA and architecture
    (applies to CPUs, GPUs, Google TPUs[1], etc…)

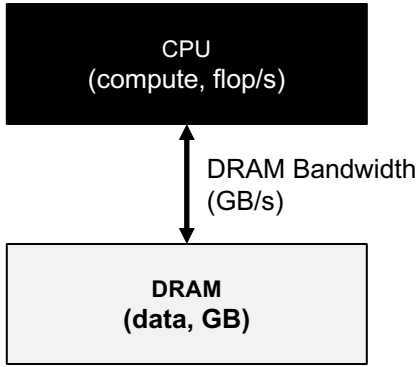


https://crd.lbl.gov/departments/computer-science/PAR/research/roofline

[1]Jouppi et al, "In-Datacenter Performance Analysis of a Tensor Processing Unit", ISCA, 2017.

(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)



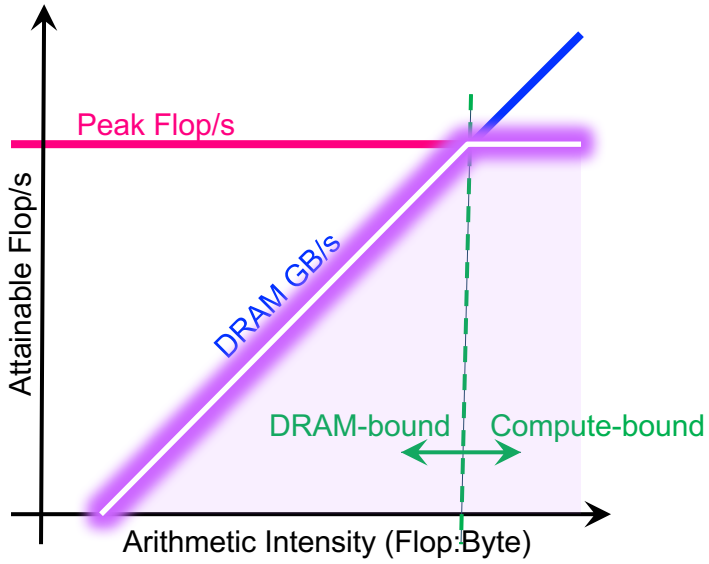
$$\text{GFlop/s} = \min \begin{cases} \text{Peak GFlop/s} \\ \text{AI} * \text{Peak GB/s} \end{cases}$$

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

Machine Balance (MB) = Peak Gflop/s / Peak GB/s

(DRAM) Roofline

- Plot Roofline bound using Arithmetic Intensity as the x-axis
- **Log-log scale** makes it easy to doodle, extrapolate performance along Moore's Law, etc…
- Kernels with AI less than **machine balance** are ultimately DRAM bound
- Typical machine balance is 5-10 flops per byte…
  - 40-80 flops per double to exploit compute capability
  - Artifact of technology and money
  - **Unlikely to improve**

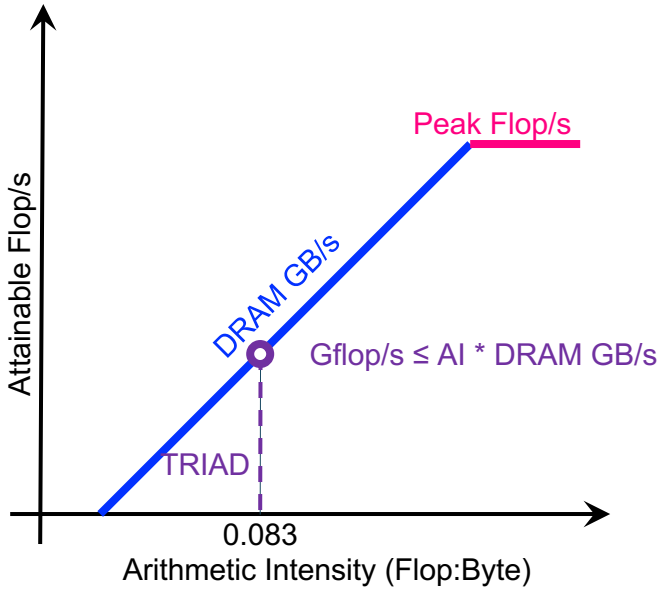


Roofline Example #1

- Consider STREAM Triad…

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

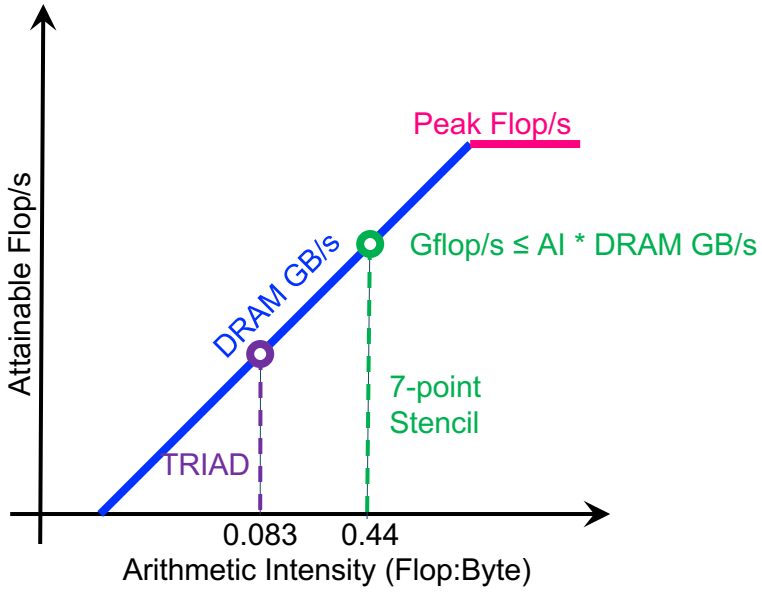
  - 2 flops per iteration
  - Transfer 24 bytes per iteration (read X[i], Y[i], write Z[i])
  - **AI = 0.083 flops per byte == Memory bound**



Roofline Example #2

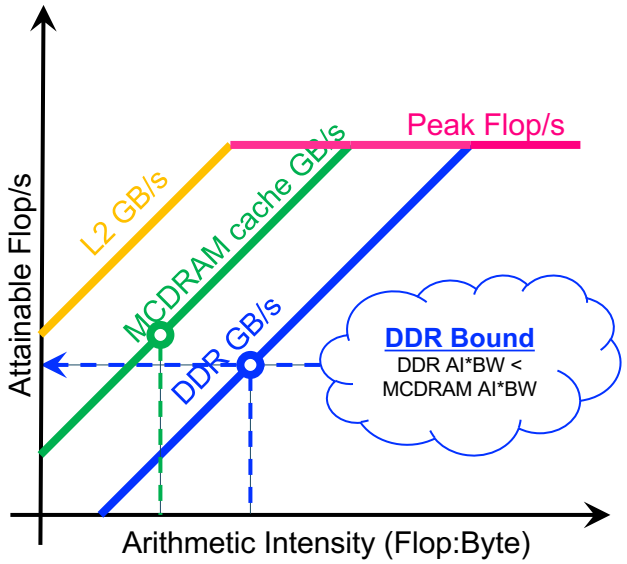
- Conversely, 7-point constant coefficient stencil…
  - 7 flops
  - 8 memory references (7 reads, 1 store) per point
  - Cache can filter all but 1 read and 1 write per point
  - **AI = 0.44 flops per byte == memory bound, but 5x the flop rate**

```
#pragma omp parallel for
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
];
}}}
```



Hierarchical Roofline

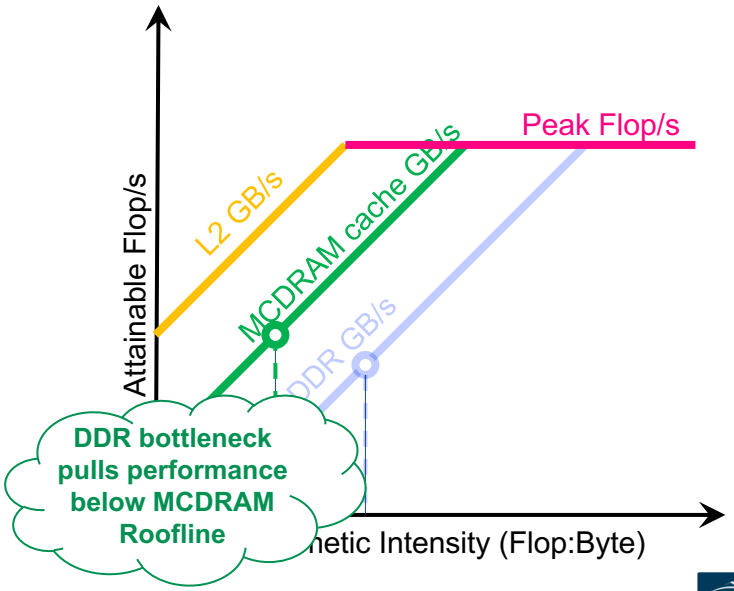
- **Multiple levels** of memory on real processors
  - Registers, L1, L2, L3 cache, MCDRAM/HBM (KNL/GPU device memory), DDR (main memory), NVRAM (non-volatile memory)
- A different bandwidth/data movement/AI for each memory level
- Construct **superposition** of Rooflines…
  - Measure a bandwidth
  - Measure an AI for each memory level
- Although a loop nest may have multiple AI's and multiple bounds (flops, L1, L2, … DRAM), performance is **bound by the minimum**



Hierarchical Roofline

- Although a loop nest may have multiple AI's and multiple bounds (flops, L1, L2, … DRAM), performance is **bound by the minimum**

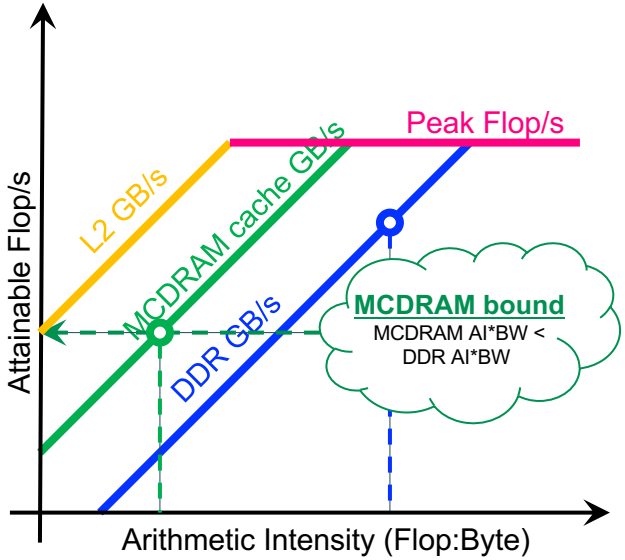
DDR AI*BW < MCDRAM AI*BW



Hierarchical Roofline

- Although a loop nest may have multiple AI's and multiple bounds (flops, L1, L2, … DRAM), performance is **bound by the minimum**

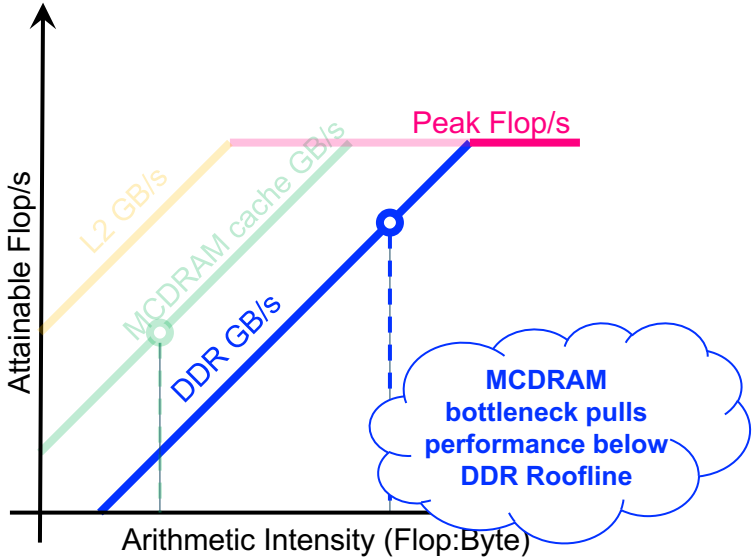
MCDRAM AI*BW < DDR AI*BW



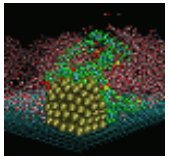
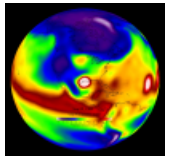
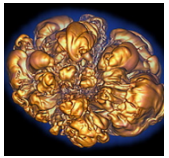
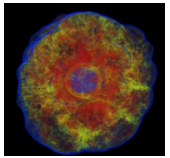
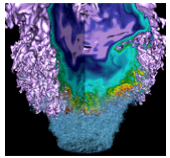
Hierarchical Roofline

- Although a loop nest may have multiple AI's and multiple bounds (flops, L1, L2, … DRAM), performance is **bound by the minimum**

MCDRAM AI*BW < DDR AI*BW



Roofline Model:
Modeling In-core Performance Effects

U.S. DEPARTMENT OF ENERGY | Office of Science

Data, Instruction, Thread-Level Parallelism

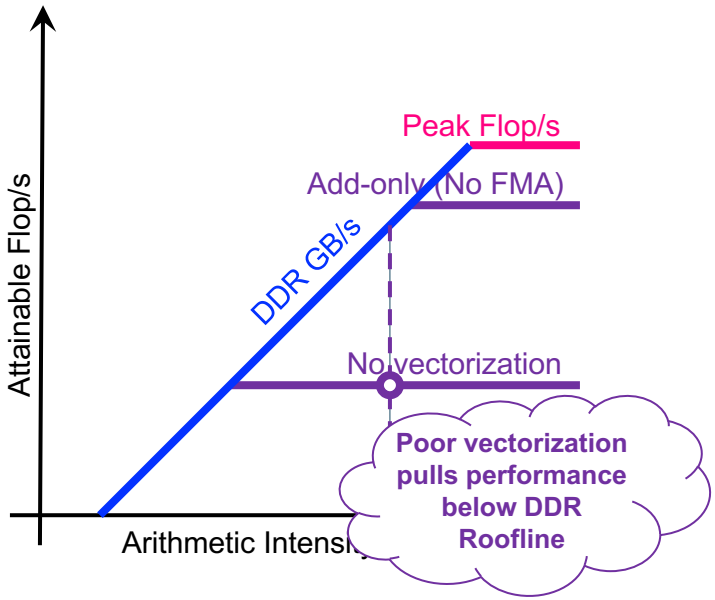
- If every instruction were an ADD (instead of FMA), performance would

  drop by 2x on KNL or 4x on Haswell !!

- Similarly, if one failed to vectorize, performance would drop by

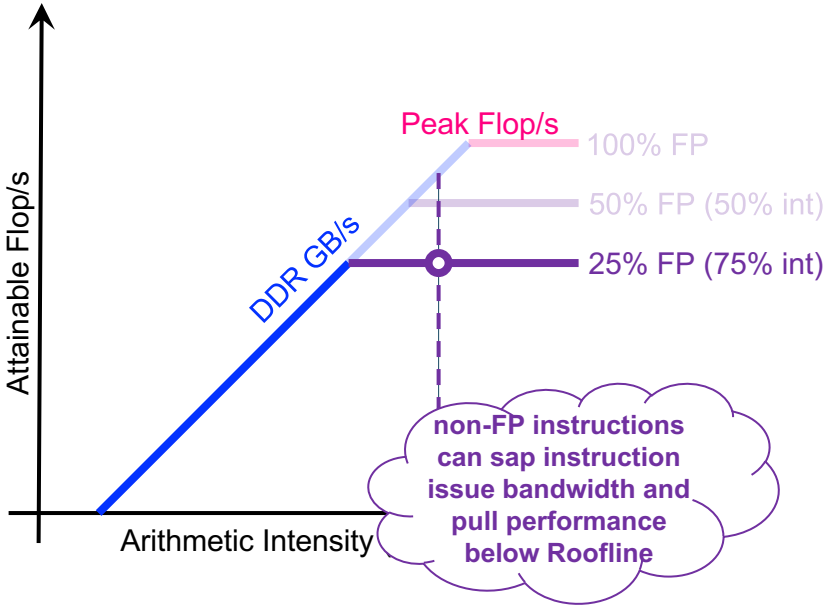
  another 8x on KNL and 4x on Haswell !!!

- Lack of threading (or load imbalance) will reduce performance by another 64x on KNL.



Superscalar vs. Instruction Mix

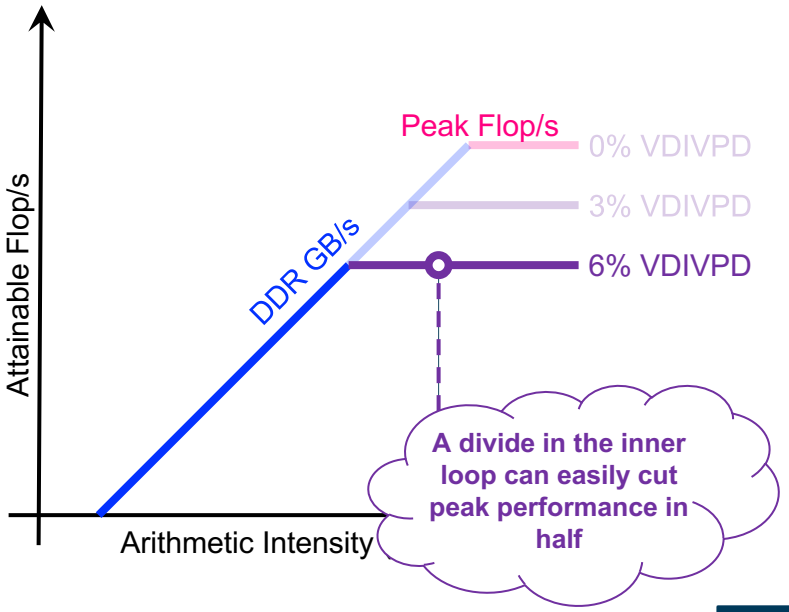
- Define in-core ceilings based on instruction mix…
- e.g. Haswell
  - 4-issue superscalar
  - Only 2 FP data paths
  - Requires 50% of the instructions to be FP to get peak performance
- e.g. KNL
  - 2-issue superscalar
  - 2 FP data paths
  - Requires 100% of the instructions to be FP to get peak performance



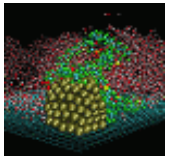
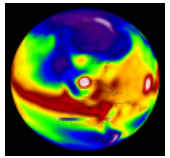
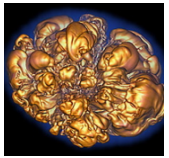
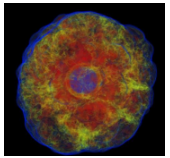
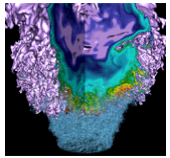
Divides and other Slow FP instructions

- FP Divides (sqrt, rsqrt, …) might support only limited pipelining
- As such, their throughput is substantially lower than FMA's
- If divides constitute even if 3% of the flop's come from divides, performance can be **cut in half !!**

Penalty varies substantially between architectures and generations (e.g. IVB, HSW, KNL, …)



Roofline Model: Modeling Cache Effects

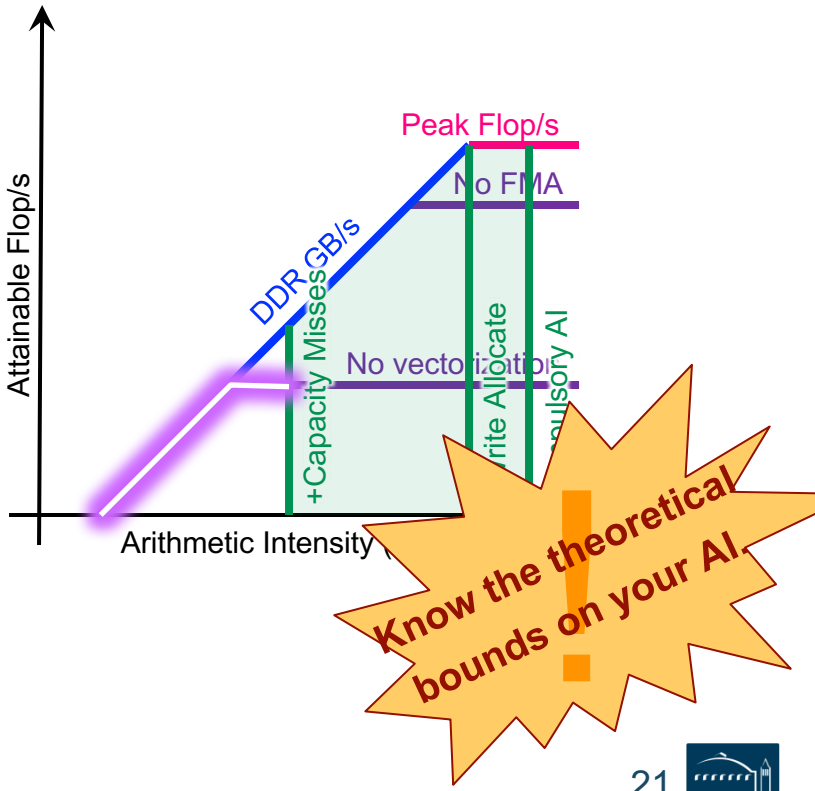
U.S. DEPARTMENT OF ENERGY | Office of Science

Locality Matters!

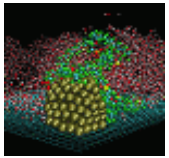
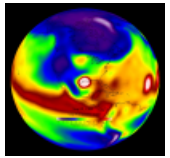
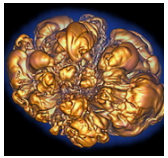
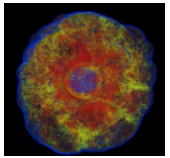
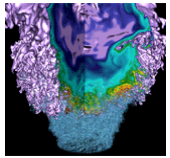
- Naively, we can bound AI using only compulsory cache misses
- However, write allocate caches can lower AI
- Cache capacity misses can have a huge penalty

Compute bound became memory bound!

$$AI = \frac{\#\text{Flop's}}{\text{Compulsory Misses} + \text{Write Allocates} + \text{Capacity Misses}}$$



Roofline Model:
General Strategy Guide

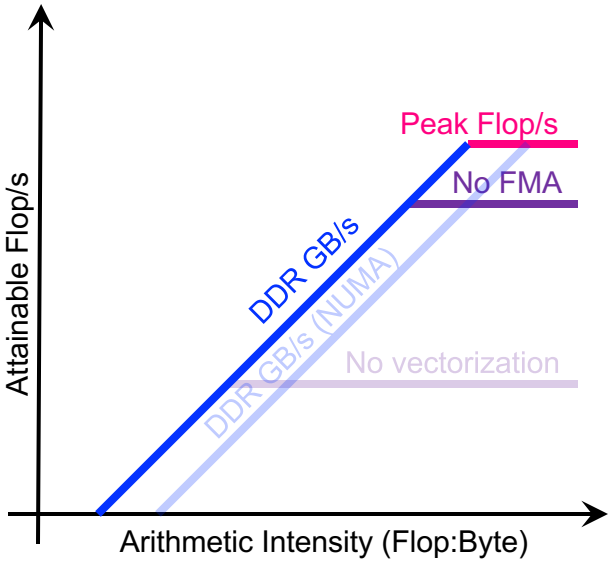






U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

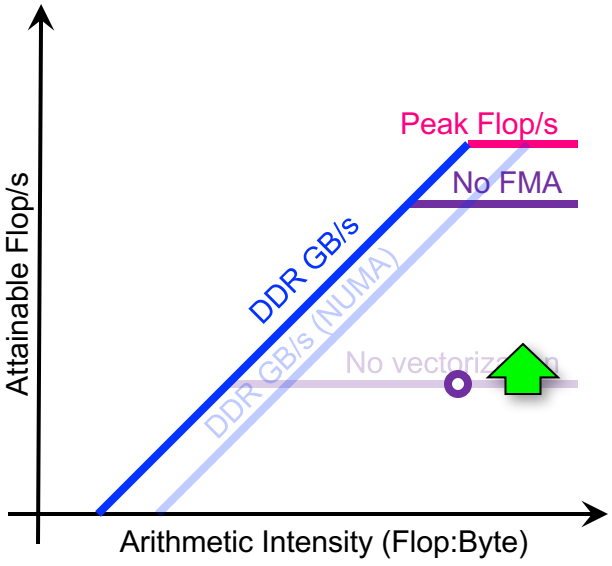
General Strategy Guide

- Broadly speaking, three approaches to improving performance:



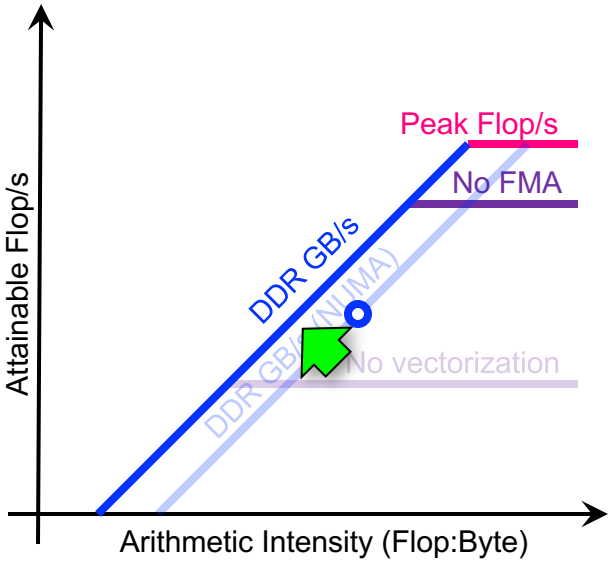
General Strategy Guide

- Broadly speaking, three approaches to improving performance:
- **Maximize in-core performance (e.g. get compiler to vectorize)**



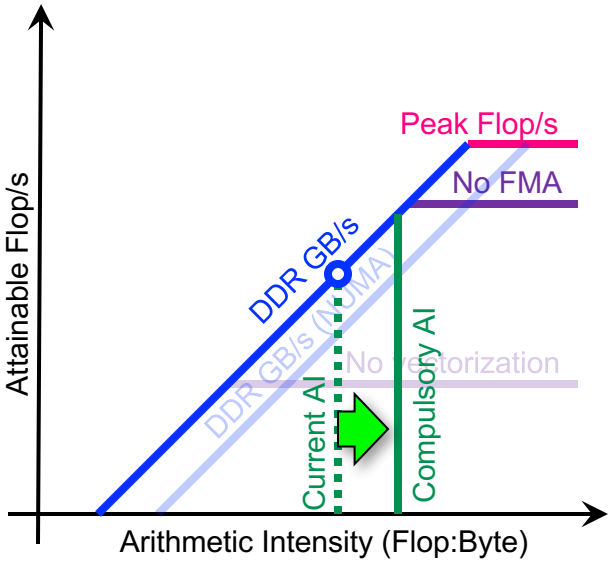
General Strategy Guide

- Broadly speaking, three approaches to improving performance:
- Maximize in-core performance (e.g. get compiler to vectorize)
- **Maximize memory bandwidth (e.g. NUMA-aware allocation)**

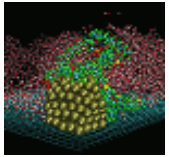
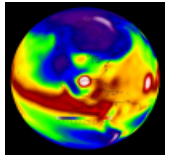
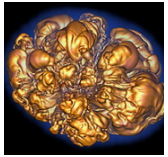
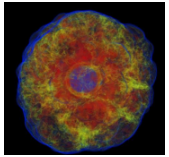
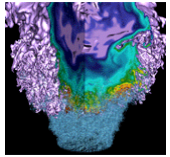


General Strategy Guide

- Broadly speaking, three approaches to improving performance:
- Maximize in-core performance (e.g. get compiler to vectorize)
- Maximize memory bandwidth (e.g. NUMA-aware allocation)
- **Minimize data movement (increase AI)**



Constructing a Roofline Model requires answering some questions…

Questions can overwhelm users…

What is my machine's peak flop/s?

How many vectorization or FMA on my machine?

What is my machine's DDR GB/s? L2 GB/s?

Properties of the target machine

(Benchmarking)

How much data did my kernel actually move?

Did my kernel vectorize?

How many flop's did my kernel actually do?

How much did that divide hurt?

Properties of an application's execution

(Instrumentation)

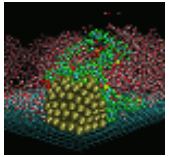
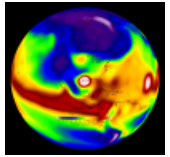
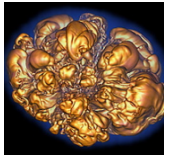
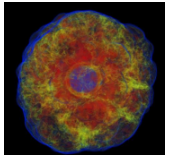
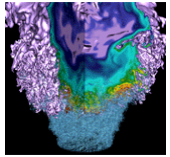
What is my kernel's computational intensity (communication lower bounds)

Can my kernel ever be vectorized?

Fundamental properties of the kernel constrained by hardware

(Theory)

We need tools…

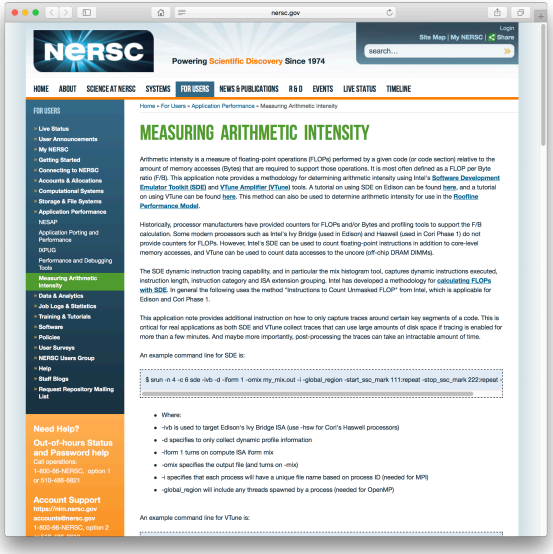
Forced to Cobble Together Tools…

DRAM Roofline:

- Use tools known/observed to work on NERSC's Cori (KNL, HSW)…
  - Used **Intel SDE** (Pin binary instrumentation + emulation) to create software Flop counters
  - Used **Intel VTune** performance tool (NERSC/Cray approved) to access uncore counters

➢ Accurate measurement of Flop's (HSW) and DRAM data movement (HSW and KNL)

➢ Used by **NESAP** (NERSC KNL application readiness project) to characterize apps on Cori…



http://www.nersc.gov/users/application-performance/measuring-arithmetic-intensity/

Initial Roofline Analysis of NESAP Codes

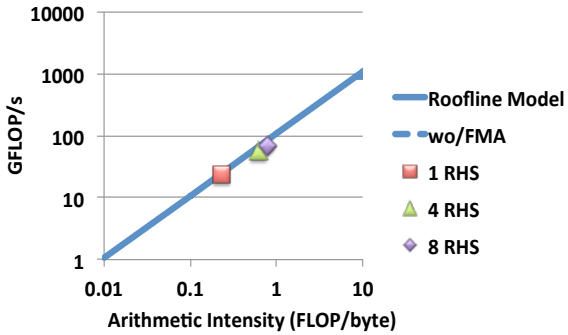
MFDn

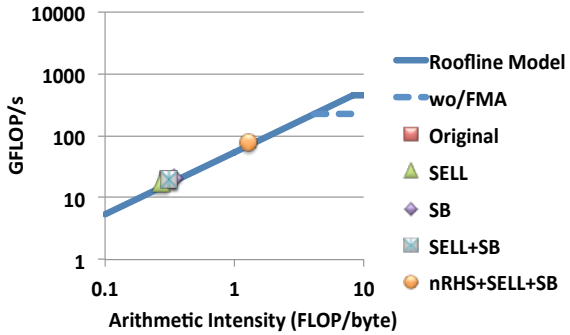
EMGeo

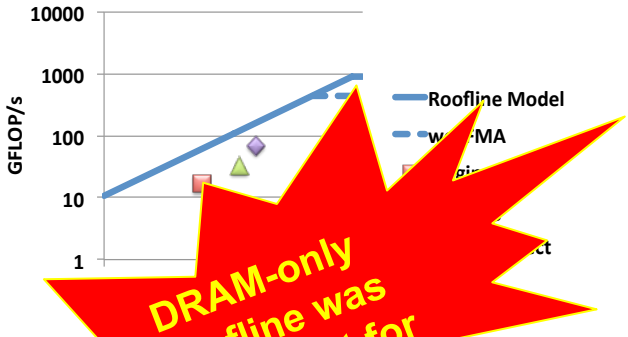
PICSAR

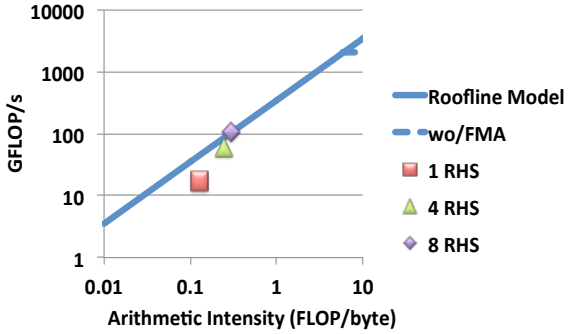
2P HSW

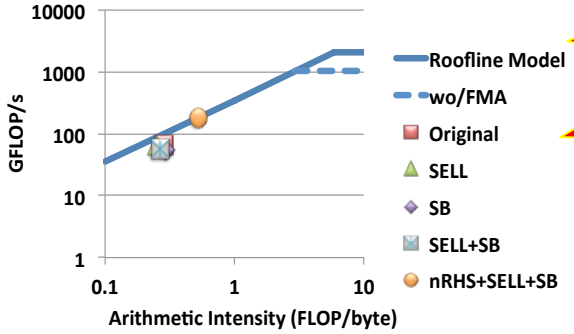
KNL

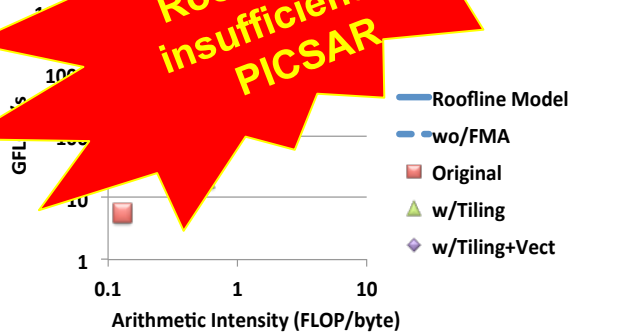








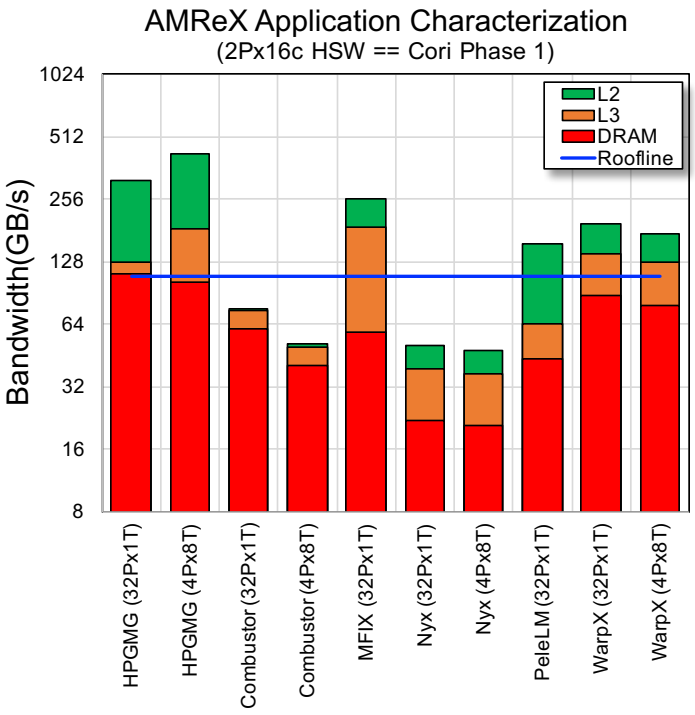




DRAM-only Roofline was insufficient for PICSAR

Evaluation of LIKWID

- LIKWID provides easy to use wrappers for measuring performance counters…
  - ✓ **Works on NERSC production systems**
  - ✓ Minimal overhead (<1%)
  - ✓ Scalable in distributed memory (MPI-friendly)
  - ✓ Fast, high-level characterization
  - x **No detailed timing breakdown or optimization advice**
  - x **Limited by quality of hardware performance counter implementation (garbage in/garbage out)**

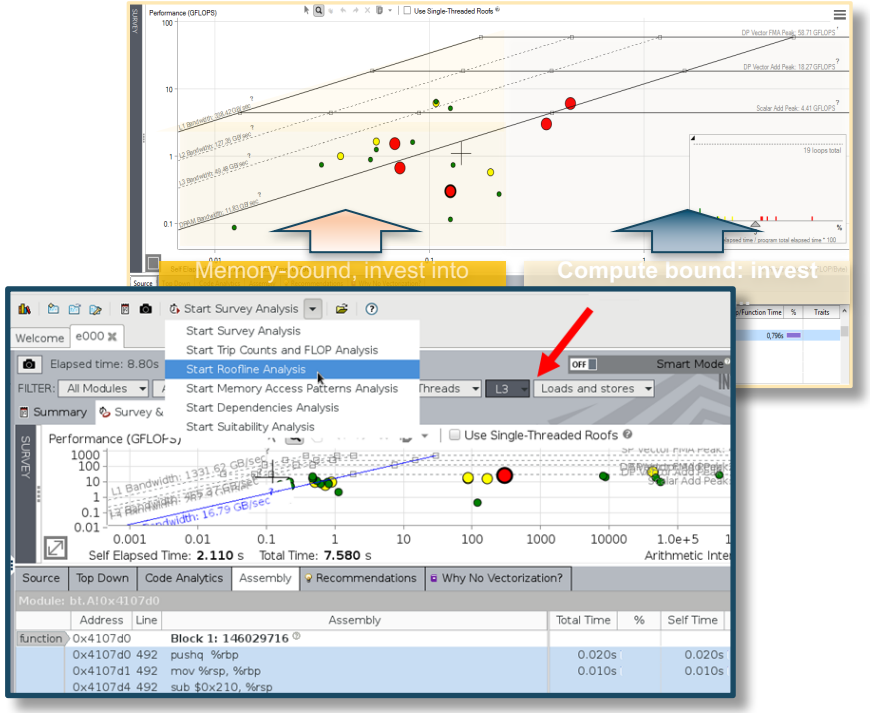


Useful tool that complements other tools!

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

Intel Advisor

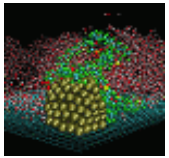
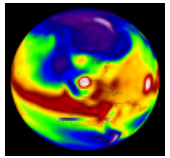
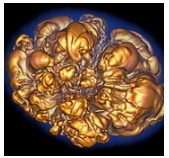
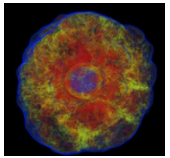
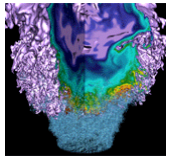
- Includes Roofline Automation…
  - ✓ Automatically instruments applications (one dot per loop nest/function)
  - ✓ Computes FLOPS and AI for each function (**CARM**)
  - ✓ AVX-512 support that incorporates masks
  - ✓ **Integrated Cache Simulator[1] (hierarchical roofline / multiple AI's)**
  - ✓ Automatically benchmarks target system (calculates ceilings)
  - ✓ Full integration with existing Advisor capabilities



http://www.nersc.gov/users/training/events/roofline-training-1182017-1192017

[1]Technology Preview, not in official product roadmap so far.

Hierarchical Roofline vs. Cache-Aware Roofline

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

Two Major Roofline Formulations:

- Hierarchical Roofline (original Roofline w/ DRAM, L3, L2, …)…
  - Defines multiple bandwidth ceilings and multiple AI's per kernel
  - Performance bound is the minimum of flops and the memory intercepts (superposition of single-metric Rooflines)
- Cache-Aware Roofline Model (CARM)
  - Defines multiple bandwidth ceilings, but uses a single AI (flop:L1 bytes)
  - As one looses cache locality, performance falls from one BW ceiling to a lower one at constant AI

- CARM has been integrated into production Intel Advisor; evaluation version of Hierarchical Roofline (cache simulator) has also been integrated into Intel Advisor (Technology Preview version)

Hands-on Session shows you both !

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

Example: STREAM

- L1 AI…
  - 2 flops
  - 2 x 8B load (old)
  - 1 x 8B store (new)
  - = 0.08 flops per byte

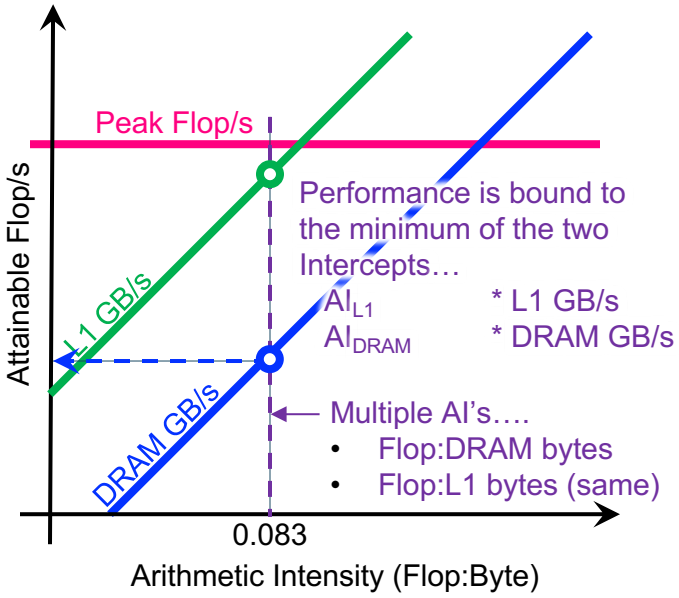
```
#pragma omp parallel for
for(i=0;i<N;i++){
  Z[i] = X[i] + alpha*Y[i];
}
```

- No cache reuse…
  - Iteration *i* doesn't touch any data associated with iteration *i+delta* for any *delta*.

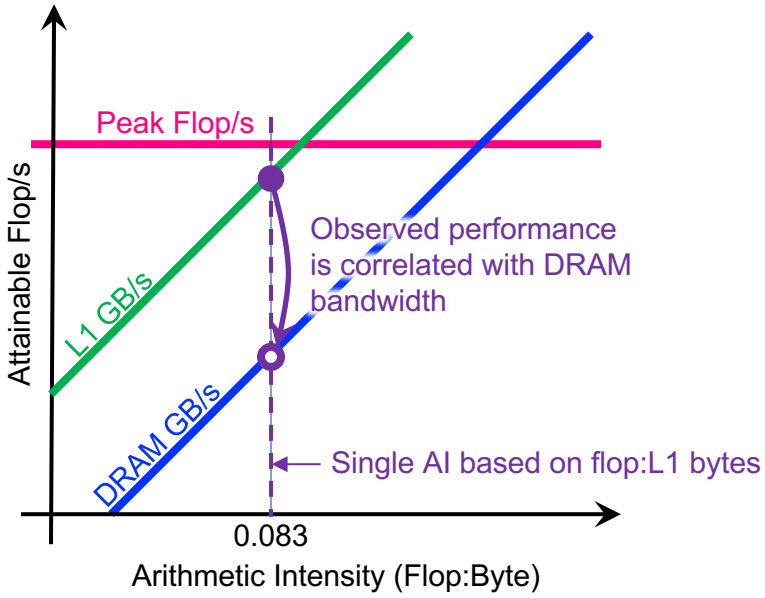
- … leads to a **DRAM AI equal to the L1 AI**

Example: STREAM

Hierarchical Roofline



Cache-Aware Roofline



Example: 7-point Stencil (Small)

- L1 AI…
  - 7 flops
  - 7 x 8B load (old)
  - 1 x 8B store (new)
  - = 0.11 flops per byte
  - compilers may do register shuffles to reduce number of loads

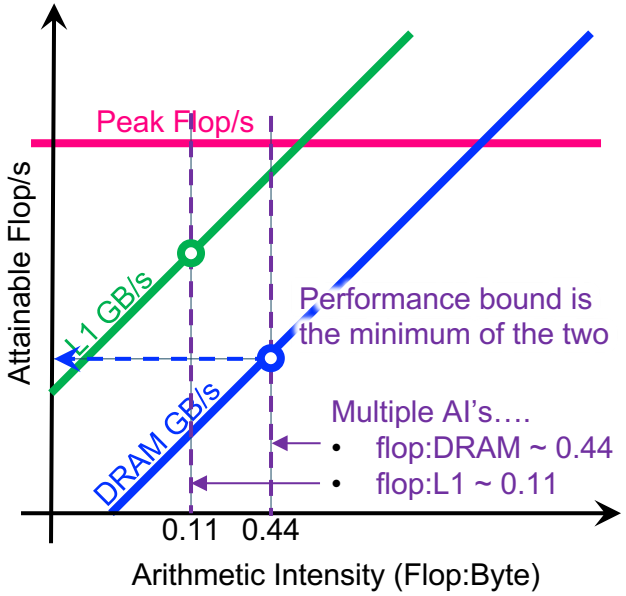
```
#pragma omp parallel for
for(k=1;k<dim+1;k++){
for(j=1;j<dim+1;j++){
for(i=1;i<dim+1;i++){
  int ijk = i + j*jStride + k*kStride;
  new[ijk] = -6.0*old[ijk        ]
                + old[ijk-1      ]
                + old[ijk+1      ]
                + old[ijk-jStride]
                + old[ijk+jStride]
                + old[ijk-kStride]
                + old[ijk+kStride];
}}}
```

- Moderate cache reuse…
  - old[ijk] is reused on subsequent iterations of i,j,k
  - old[ijk-1] is reused on subsequent iterations of i.
  - old[ijk-jStride] is reused on subsequent iterations of j.
  - old[ijk-kStride] is reused on subsequent iterations of k.

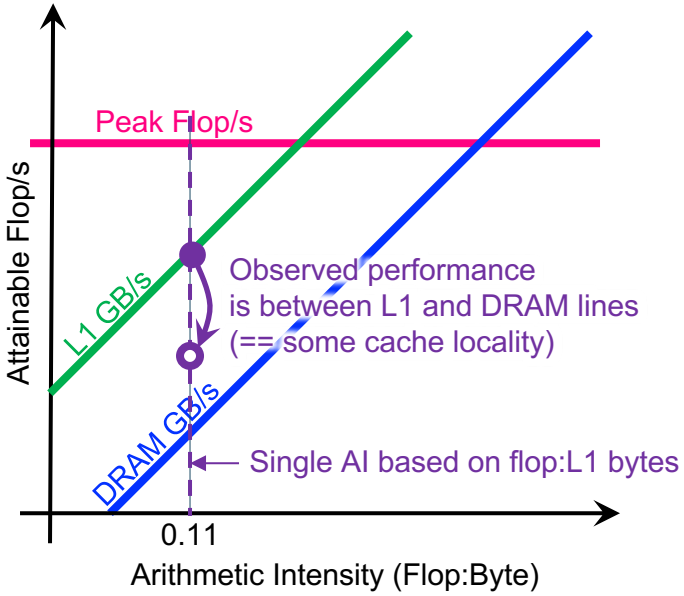
- … leads to **DRAM AI larger than the L1 AI**

Example: 7-point Stencil (Small)

Hierarchical Roofline



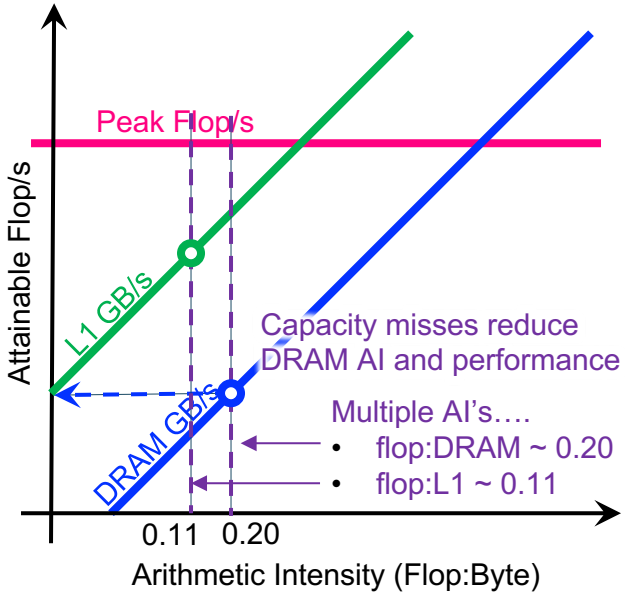
Cache-Aware Roofline



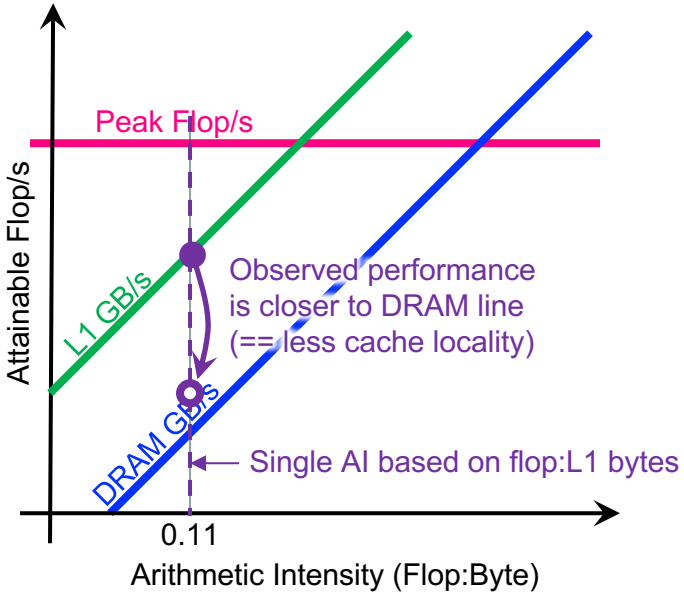
(Small Problem)

Example: 7-point Stencil (Large)

Hierarchical Roofline



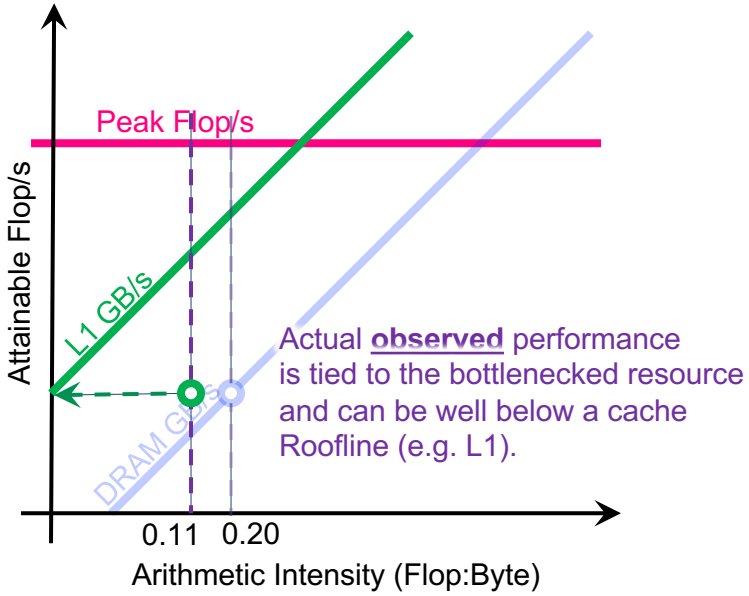
Cache-Aware Roofline



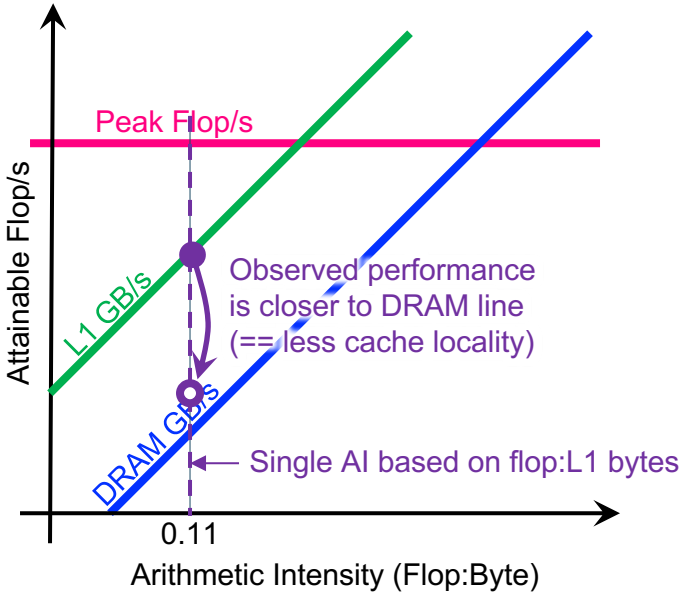
(Large Problem)

Example: 7-point Stencil (Large)

Hierarchical Roofline



Cache-Aware Roofline



(Large Problem)

Summary

- Answered the questions: Why use performance models or tools?
- Introduced Roofline model:
  - Arithmetic intensity and bandwidth
  - Two formulations: DRAM Roofline, Hierarchical Roofline
  - General optimization strategy (In-core/memory bandwidth/data locality)
- Performance tools: tools available in the market, NESAP, Intel Advisor
- Two examples: stream and 7-pointt stencil
  - Differences between Hierarchical Roofline and CARM

Aleks' talk: CARM, Hierarchical (ORM) Roofline, and Integrated Roofline in Advisor

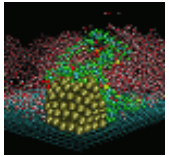
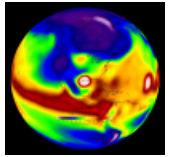
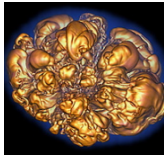
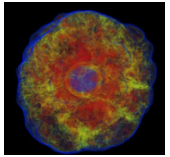
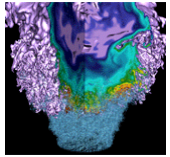
Acknowledgements

- 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 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 No. DE-AC02-05CH11231.

Thank You

Backup Slides

Performance Models / Simulators

- Historically, many performance models and simulators tracked latencies to predict performance (i.e. counting cycles)

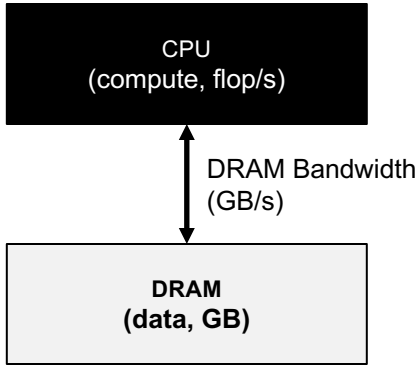
- The last two decades saw a number of **latency-hiding** techniques…
  - Out-of-order execution (hardware discovers parallelism to hide latency)
  - HW stream prefetching (hardware speculatively loads data)
  - Massive thread parallelism (independent threads satisfy the latency-bandwidth product)

- Effective latency hiding has resulted in a shift from a latency-limited computing regime to a **throughput-limited** computing regime

(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)

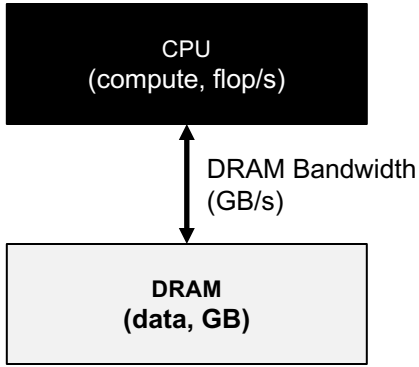
Time = max { **#FP ops / Peak GFlop/s**, **#Bytes / Peak GB/s** }



(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)

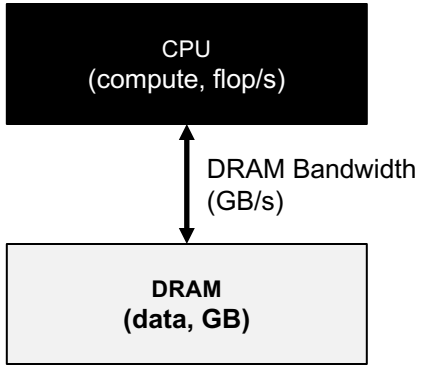
$$\frac{\text{Time}}{\text{\#FP ops}} = \max \begin{cases} \text{1 / Peak GFlop/s} \\ \text{\#Bytes / \#FP ops / Peak GB/s} \end{cases}$$



(DRAM) Roofline

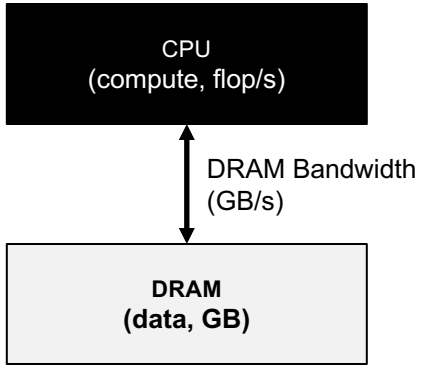
- One could hope to always attain peak performance (Flop/s)
- However, finite locality (reuse) and bandwidth limit performance.
- Assume:
  - Idealized processor/caches
  - Cold start (data in DRAM)

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



(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)



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

Note, Arithmetic Intensity (AI) = Flops / Bytes (as presented to DRAM)

Data, Instruction, Thread-Level Parallelism

- Modern CPUs use several techniques to increase per core Flop/s

Fused Multiply Add

- w = x*y + z is a common idiom in linear algebra
- Rather than [illegible] ing separate [illegible] and [illegible] use a [illegible] add (FMA)
- The FPU chains the multiply and add in a single pipeline so that it can complete FMA/cycle

Resurgence… Tensor Cores, QFMA, etc…

Vector Instructions

- Many HPC codes apply the same operation to a vector of elements
- Vendors provide vector instructions that apply the same operation to 2, 4, 8, 16 elements…

  x [0:7] *y [0:7] + z [0:7]
- Vector FPUs complete 8 vector operations/cycle

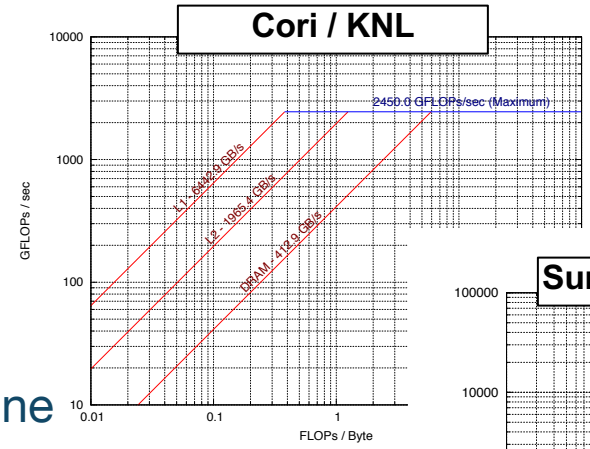
Deep pipelines

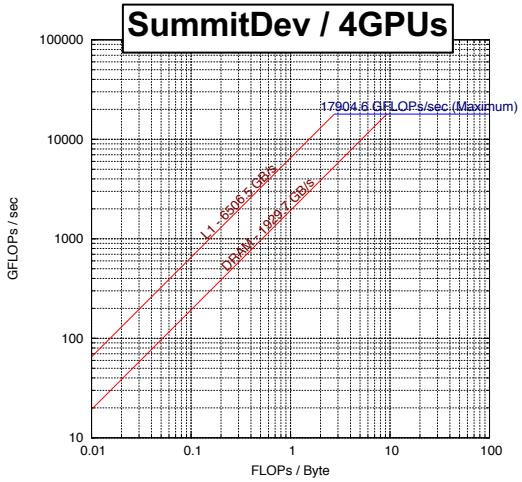
- The hardware for a FMA is substantial.
- Breaking a single FMA up into several smaller operations and pipelining them allows vendors to increase GHz
- Little's Law applies… need FP_Latency * FP_bandwidth independent instructions

Node Characterization?

- "Marketing Numbers" can be deceptive…
  - Pin BW vs. real bandwidth
  - TurboMode / Underclock for AVX
  - compiler failings on high-AI loops.

- LBL developed the Empirical Roofline Toolkit (ERT)…
  - Characterize CPU/GPU systems
  - Peak Flop rates
  - Bandwidths for each level of memory
  - **MPI+OpenMP/CUDA == multiple GPUs**





Instrumentation with Performance Counters?

Characterizing applications with performance counters can be problematic…

- x Flop Counters can be **broken/missing** in production processors
- x Vectorization/**Masking** can complicate counting Flop's
- x Counting Loads and Stores doesn't capture cache reuse while counting cache misses doesn't account for **prefetchers**
- x DRAM counters (Uncore PMU) might be accurate, but…
  - x are **privileged** and thus nominally inaccessible in user mode
  - x may need vendor (e.g. Cray) and center (e.g. NERSC) approved **OS/kernel changes**

Tools/Platforms for Roofline Modeling

Use LIKWID for fast, scalable app-level instrumentation

Use ERT to benchmark systems

Use Advisor for loop-level instrumentation and analysis on Intel targets

	Metric	STREAM	Intel SDE	Intel Advisor	NVIDIA NVProf
Benchmark	Peak MFlops	✗	✗	✓	✗
	Pe[illegible]	✗	✗	✗	
	[illegible]	✓	✗	✓	
	[illegible]	✗	✗	[illegible]	
Execution	[illegible]	✗	✓	[illegible]	
	%S[illegible]	✗	✓	✓	
	MIPS	✗	✓	✗	
	DRAM BW	✗	✗	✓	✓
	Cache BW	✗	✗	✓	✓
	Auto-Roofline	✗	✗	✓	✗
Platforms	Intel CPUs	✓	✓	✓	✗
	IBM Power8	✓	✗	✗	✗
	NVIDIA GPUs	✓	✗	✗	✓
	AMD CPUs	✓	?	?	✗
	AMD GPUs	✓	✗	✗	✗
	ARM	✓	✗	✗	✗