Roofline on Manycore and Accelerated Systems

Samuel Williams
Computational Research Division
Lawrence Berkeley National Lab
SWWilliams@lbl.gov

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 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.
- This research used resources of the Oak Ridge Leadership Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

Introduction

Why Use 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.

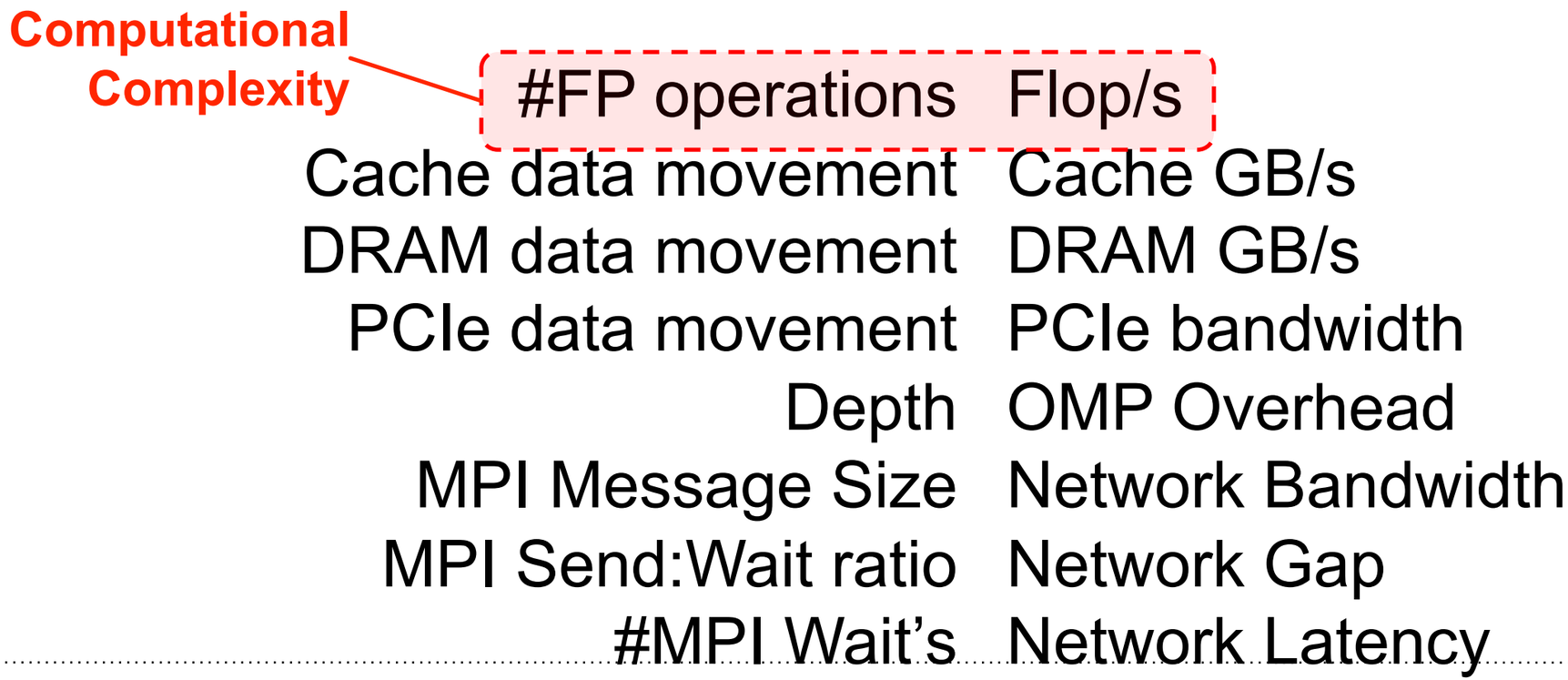
Performance Models

- Many different components can contribute to kernel run time.
- Some are characteristics of the application, some are characteristics of the machine, and some are both (memory access pattern + caches).

#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

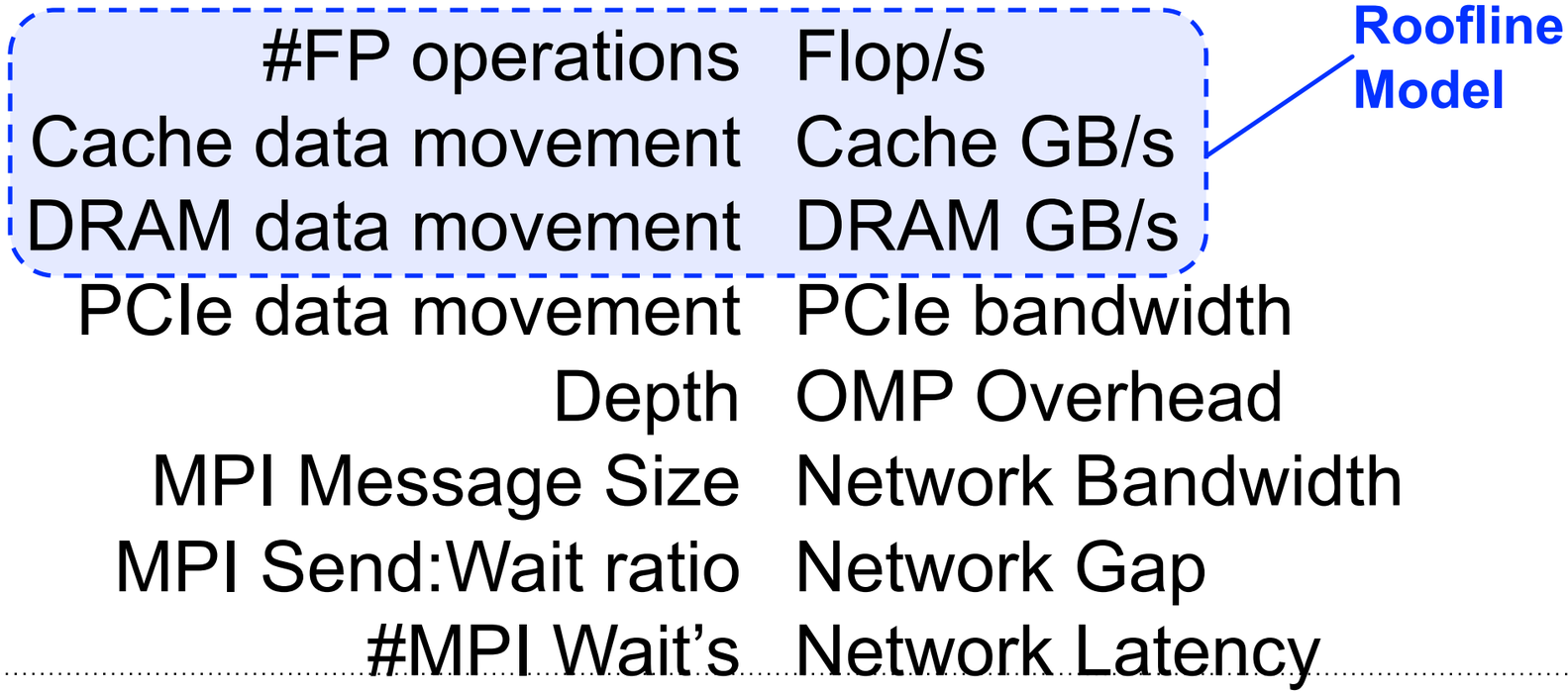
Performance Models

- Can’t think about all these terms all the time for every application…



Performance Models

- Because there are so many components, performance models often conceptualize the system as being dominated by one or more of these components.



Williams et al, "Roofline: An Insightful Visual Performance Model For Multicore Architectures", CACM, 2009.

Performance Models

- Because there are so many components, performance models often conceptualize the system as being dominated by one or more of these 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

LogP

Culler, et al, "LogP: a practical model of parallel computation", CACM, 1996.

Performance Models

- Because there are so many components, performance models often conceptualize the system as being dominated by one or more of these components.

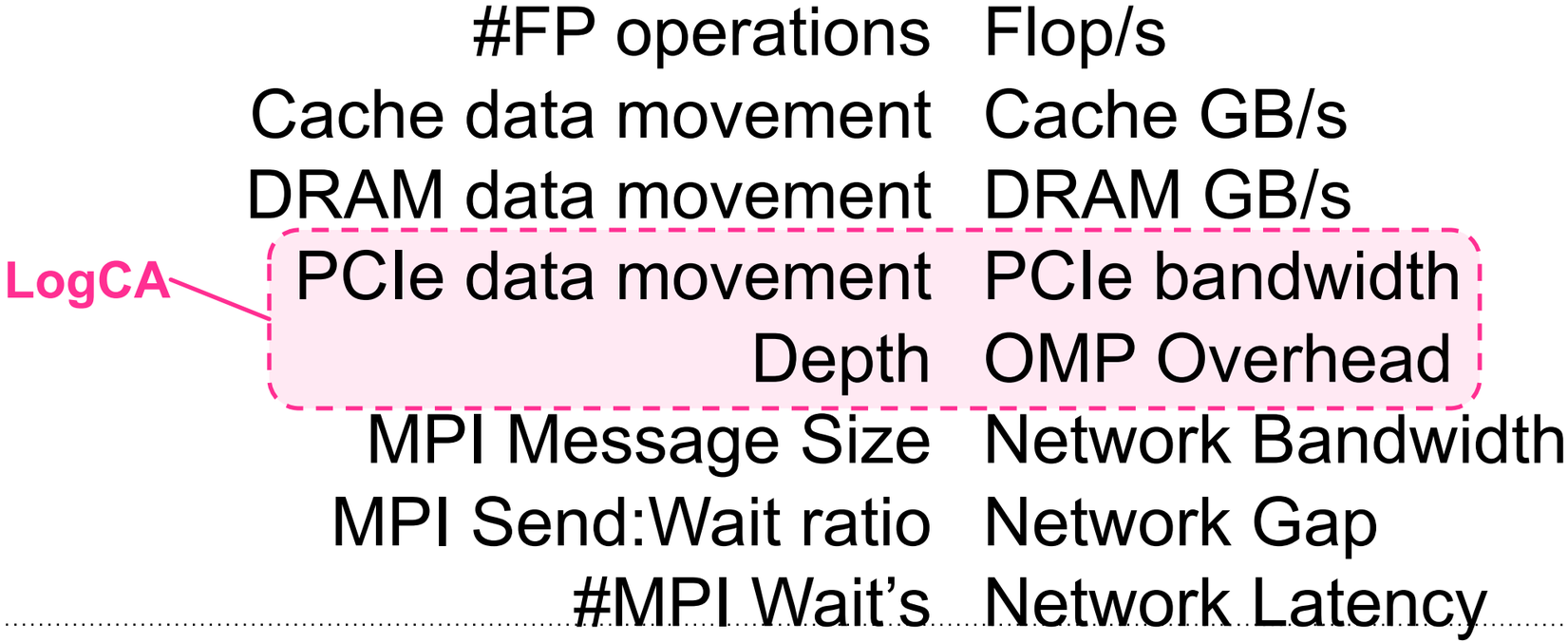
#FP operations	Flop/s
Cache data movement	Cache GB/s
DRAM data movement	DRAM GB/s
PCIe data movement	PCIe bandwidth
Depth	OMP Overhead
MPI Message Size	Network Bandwidth
MPI Send:Wait ratio	Network Gap
#MPI Wait's	Network Latency

LogGP

Alexandrov, et al, "LogGP: incorporating long messages into the LogP model - one step closer towards a realistic model for parallel computation", SPAA, 1995.

Performance Models

- Because there are so many components, performance models often conceptualize the system as being dominated by one or more of these components.



Bin Altaf et al, "LogCA: A High-Level Performance Model for Hardware Accelerators", ISCA, 2017.

Introduction to the Roofline Model

Performance Models / Simulators

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

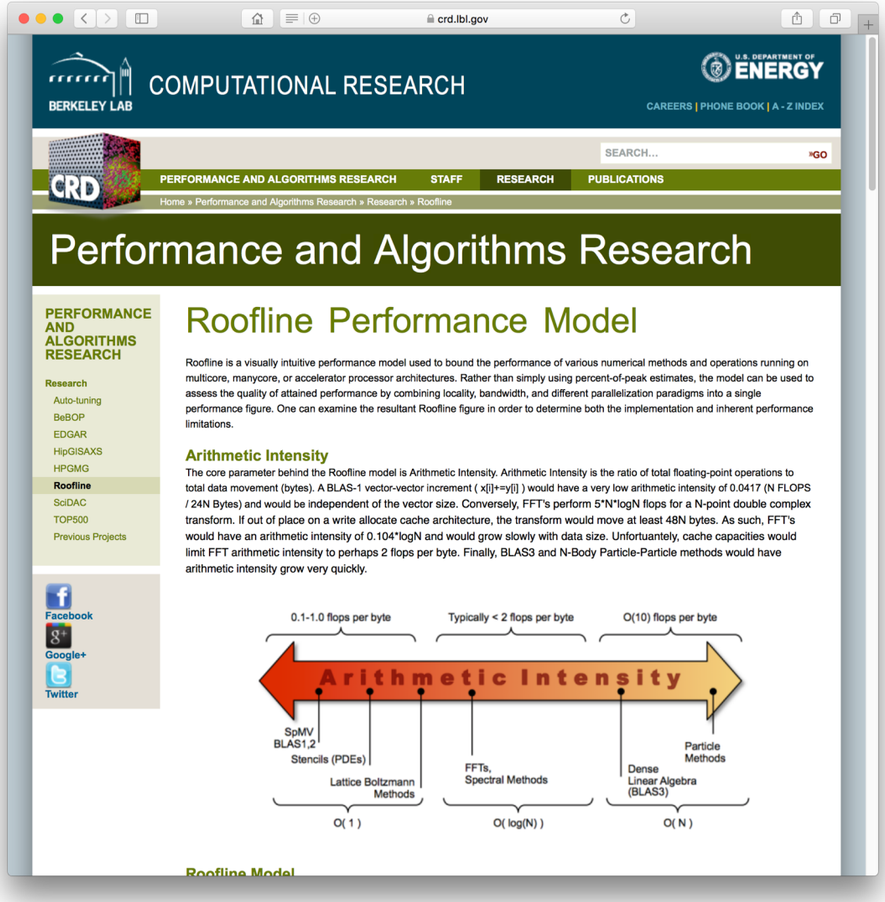
- The last two decades saw a number of latency-hiding techniques…
  - Out-of-order execution (hardware discovers parallelism to hide latency)
  - HW stream prefetching (hardware speculatively loads data)
  - Massive thread parallelism (independent threads satisfy the latency-bandwidth product)

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

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…)

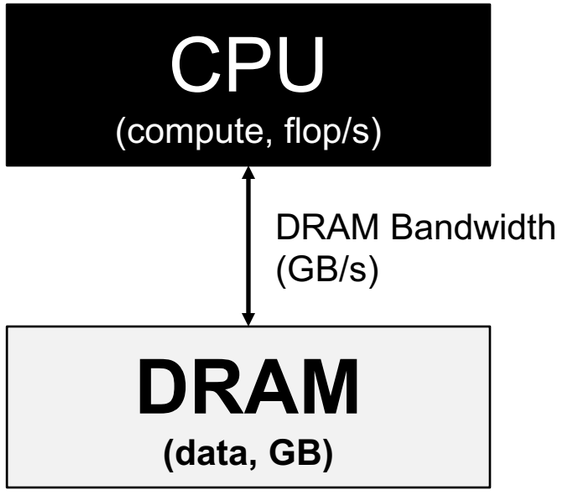


https://crd.lbl.gov/departments/computer-science/PAR/research/roofline

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

(DRAM) Roofline

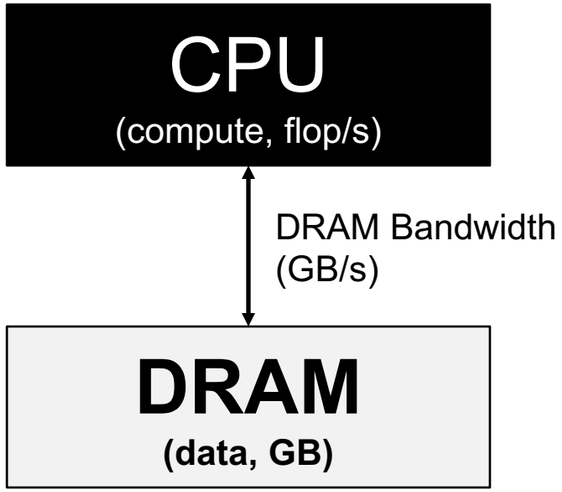
- One could hope to always attain peak performance (Flop/s)
- However, finite locality (reuse) and bandwidth limit performance.
- Assume:
  - Idealized processor/caches
  - Cold start (data in DRAM)



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

(DRAM) Roofline

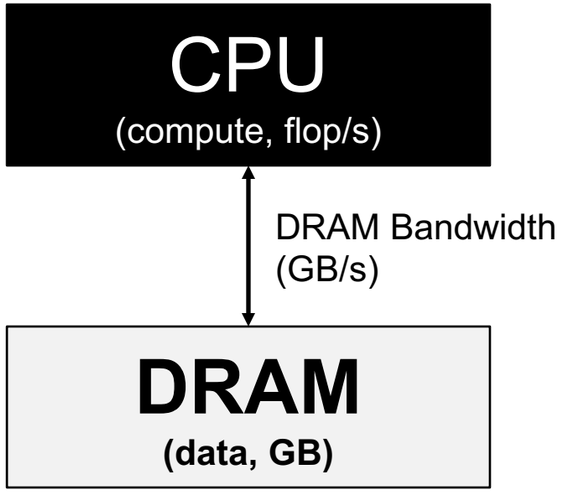
- One could hope to always attain peak performance (Flop/s)
- However, finite locality (reuse) and bandwidth limit performance.
- Assume:
  - Idealized processor/caches
  - Cold start (data in DRAM)



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

(DRAM) Roofline

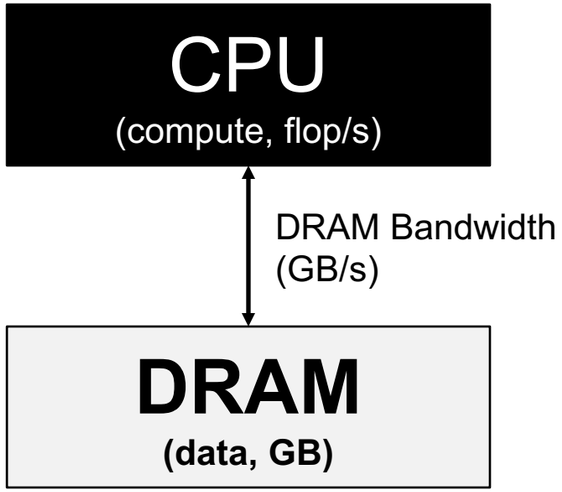
- One could hope to always attain peak performance (Flop/s)
- However, finite locality (reuse) and bandwidth limit performance.
- Assume:
  - Idealized processor/caches
  - Cold start (data in DRAM)



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

(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)

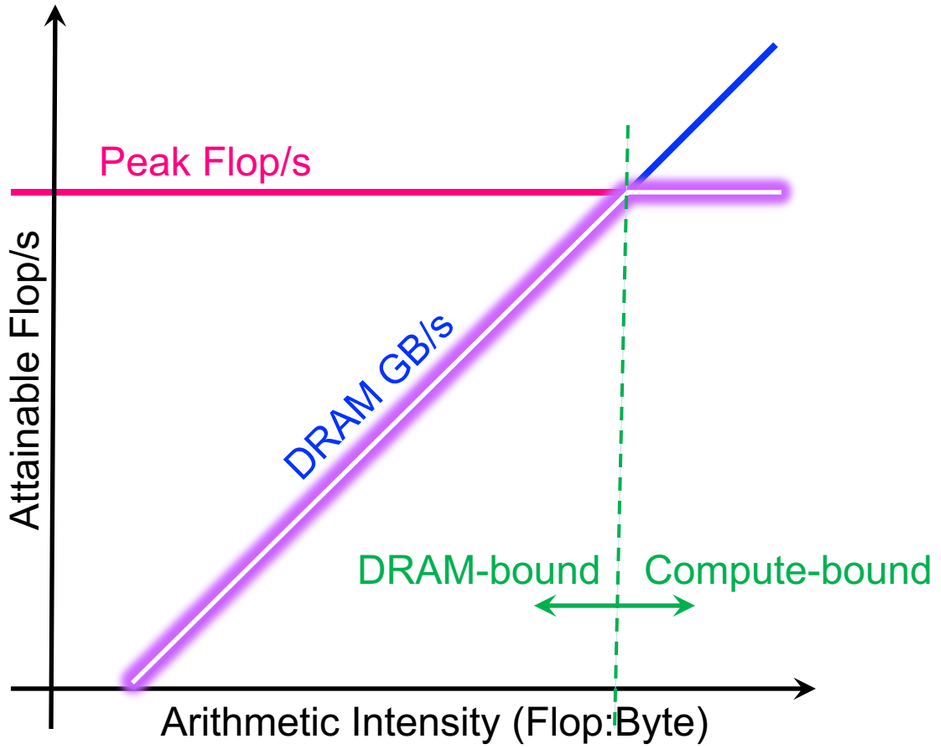


GFlop/s = min { Peak GFlop/s ; AI * Peak GB/s }

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

(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 (we'll refine this later…)



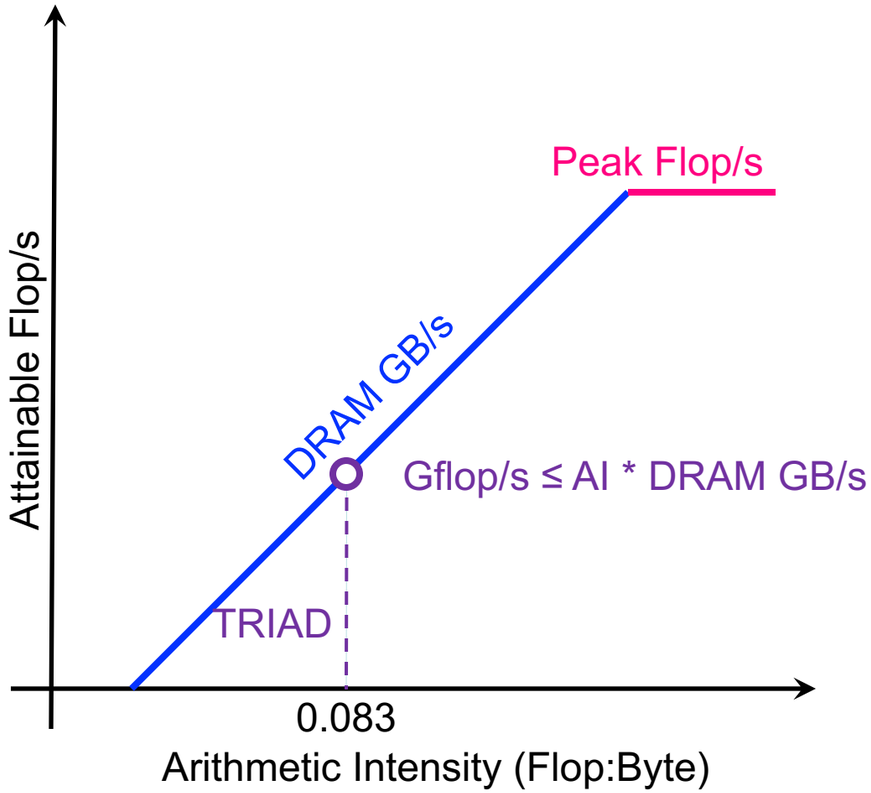
Roofline Example #1

- 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**

- 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**

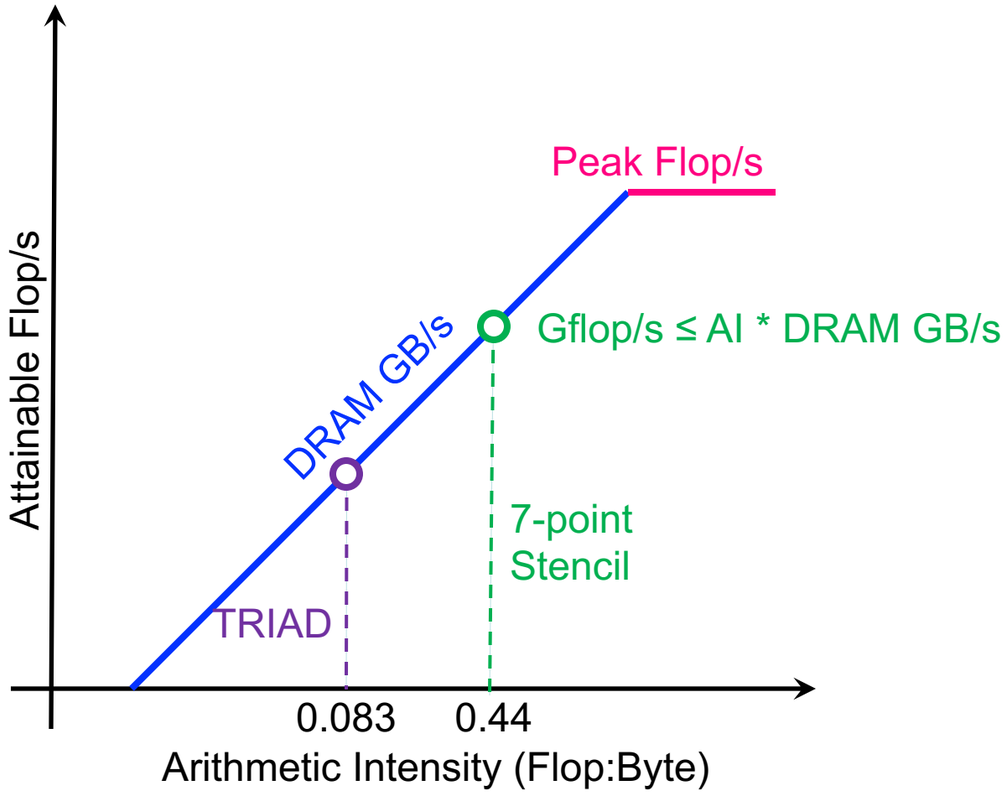


Roofline Example #2

- 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,**

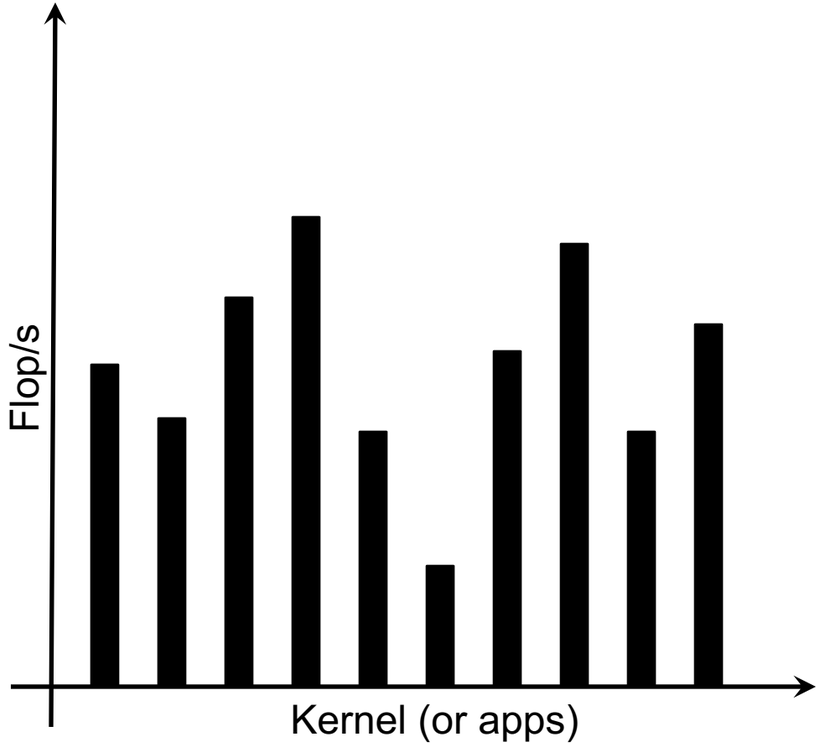
    but 5x the flop rate

```
#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++){
  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];
}}}
```



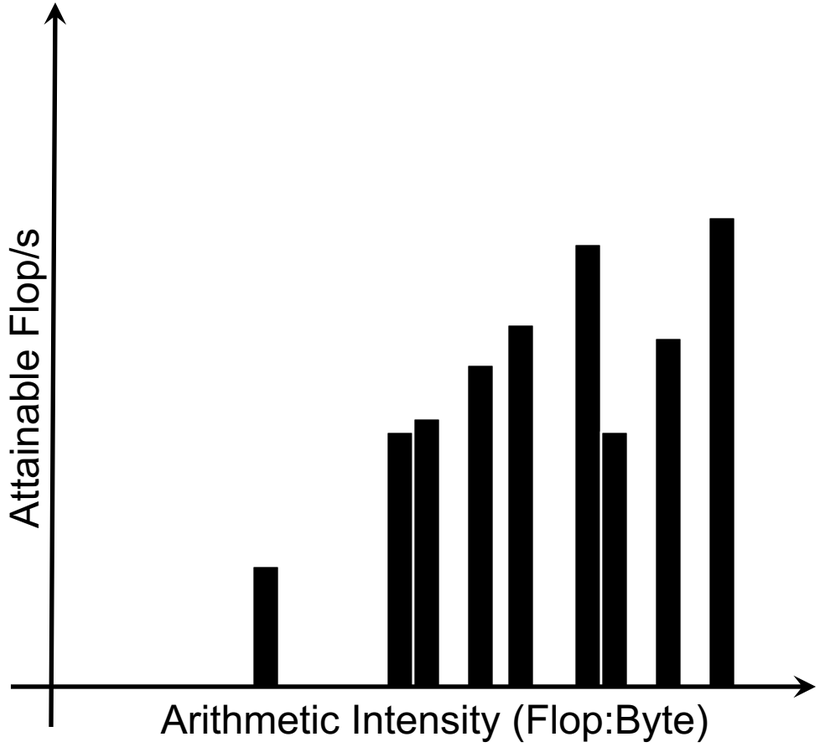
General Guidelines

- Imagine a mix of loop nests (or applications)
- Flop/s alone may not be useful for understanding performance



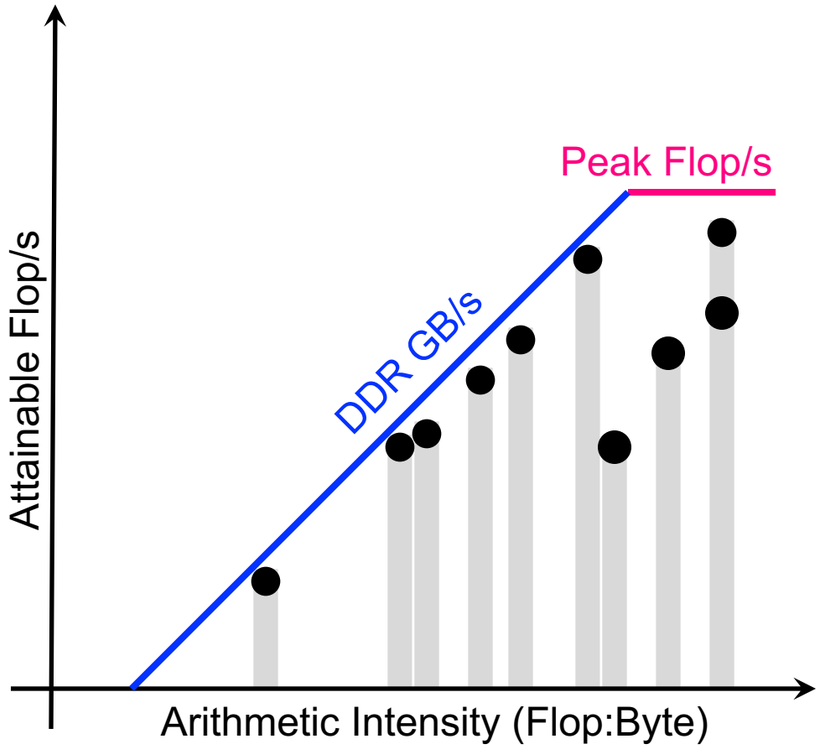
General Guidelines

- We can sort kernels (or apps) by AI …



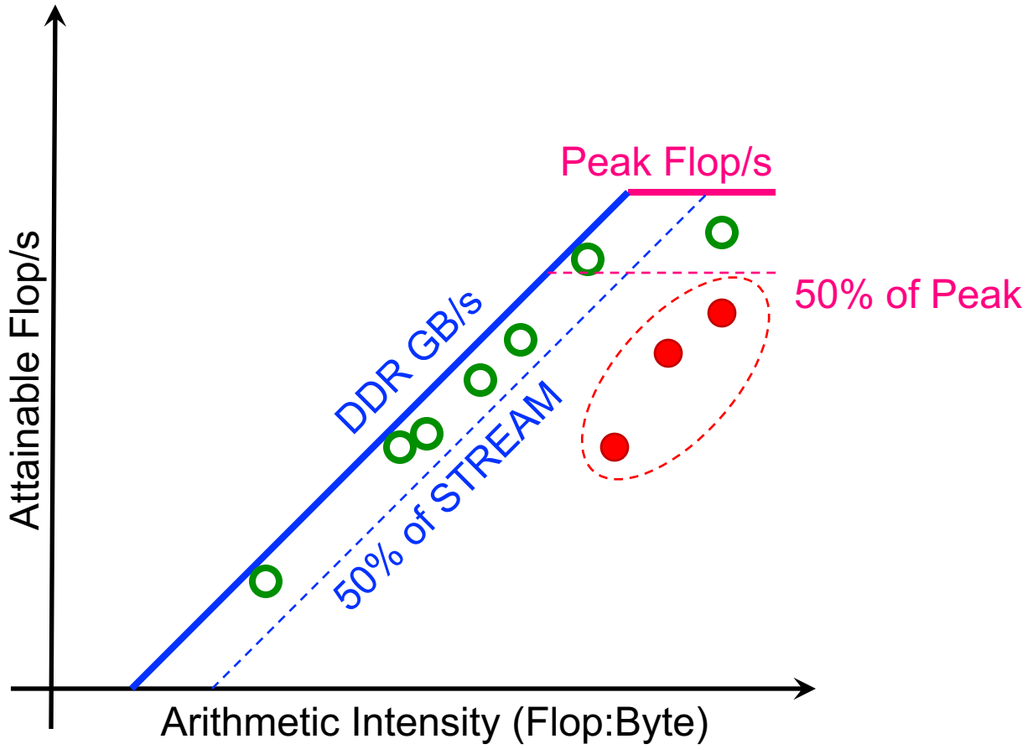
General Guidelines

- We can sort kernels (or apps) by AI …
- … and compare performance relative to machine capabilities



General Guidelines

- Applications near the roofline are making good use of computational resources
  - Kernels can have low performance (Gflop/s), but make good use of a machine
  - Kernels can have high performance (Gflop/s), but make poor use of a machine



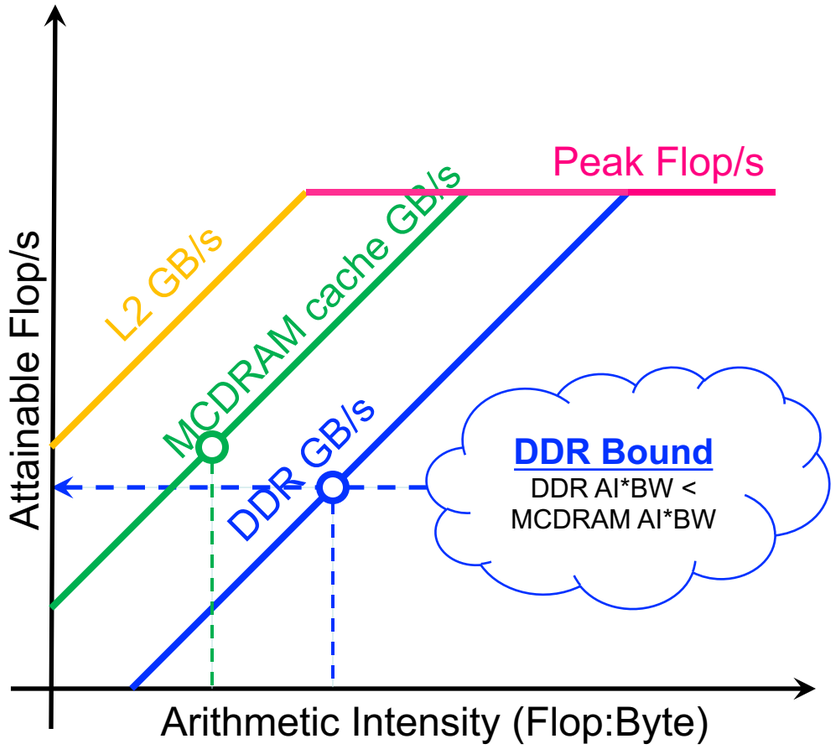
Roofline Model: Cache Effects

Hierarchical Roofline

- Real processors have multiple levels of memory
  - Registers
  - L1, L2, L3 cache
  - MCDRAM/HBM (KNL/GPU device memory)
  - DDR (main memory)
  - NVRAM (non-volatile memory)
- Applications can have locality in each level
  - Unique data movements imply unique AI's
  - Moreover, each level will have a unique bandwidth

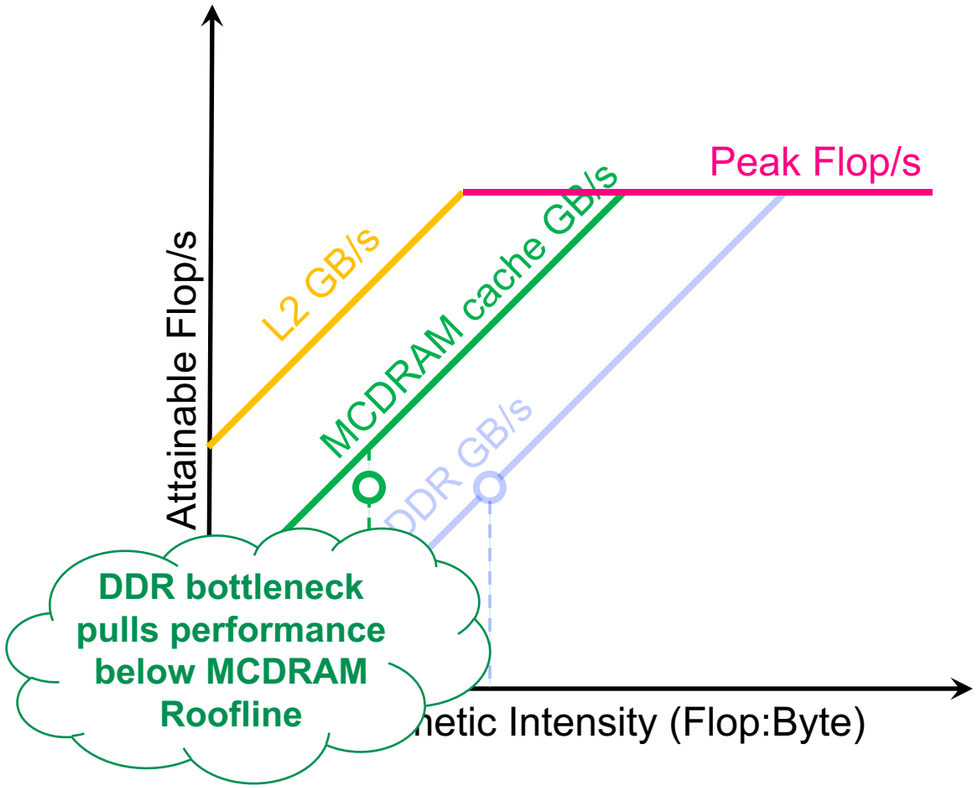
Hierarchical Roofline

- Construct superposition of Rooflines…
  - Measure a bandwidth
  - Measure AI for each level of memory
  - Although an loop nest may have multiple AI's and multiple bounds (flops, L1, L2, … DRAM)…
  - **… performance is bound by the minimum**



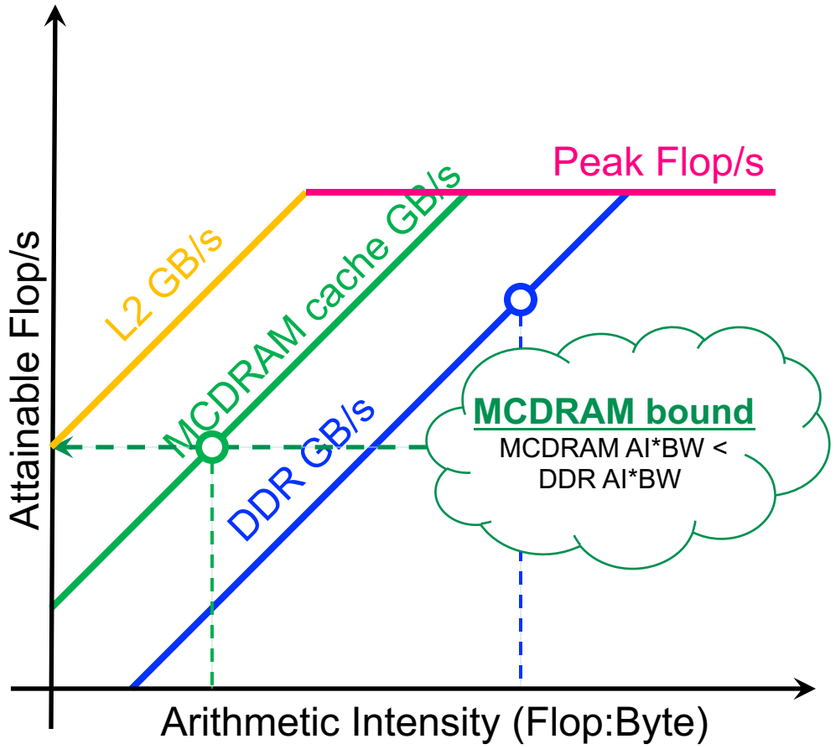
Hierarchical Roofline

- Construct superposition of Rooflines…
  - Measure a bandwidth
  - Measure AI for each level of memory
  - Although an loop nest may have multiple AI's and multiple bounds (flops, L1, L2, … DRAM)…
  - **… performance is bound by the minimum**



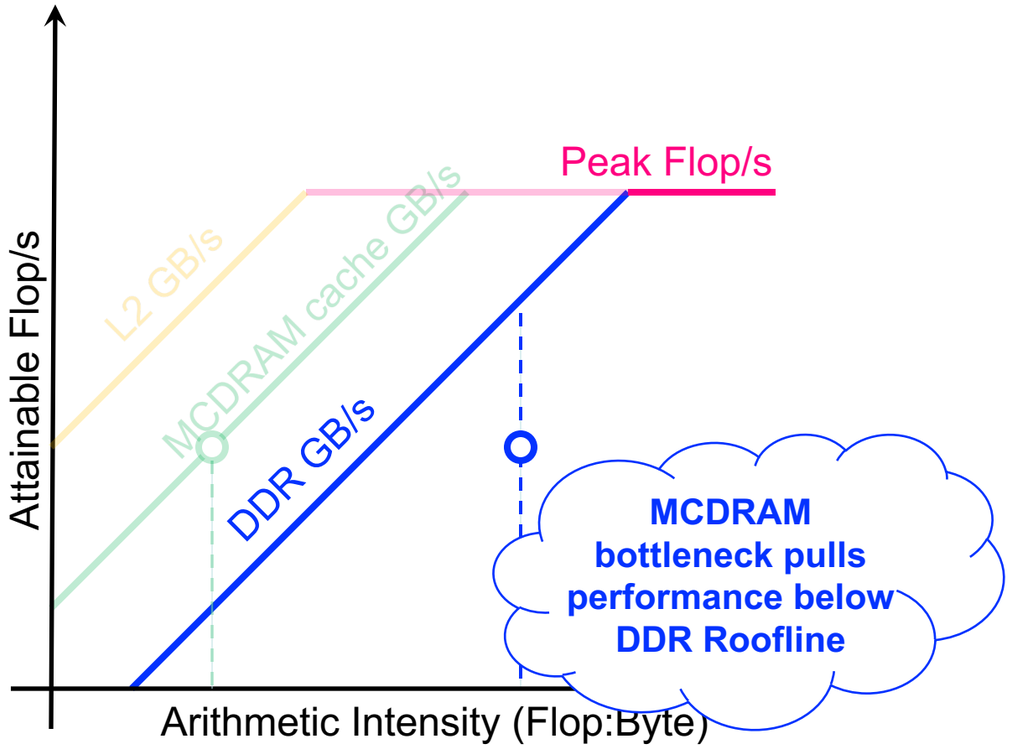
Hierarchical Roofline

- Construct superposition of Rooflines…
  - Measure a bandwidth
  - Measure AI for each level of memory
  - Although an loop nest may have multiple AI's and multiple bounds (flops, L1, L2, … DRAM)…
  - **… performance is bound by the minimum**



Hierarchical Roofline

- Construct superposition of Rooflines…
  - Measure a bandwidth
  - Measure AI for each level of memory
  - Although an loop nest may have multiple AI's and multiple bounds (flops, L1, L2, … DRAM)…
  - **… performance is bound by the minimum**



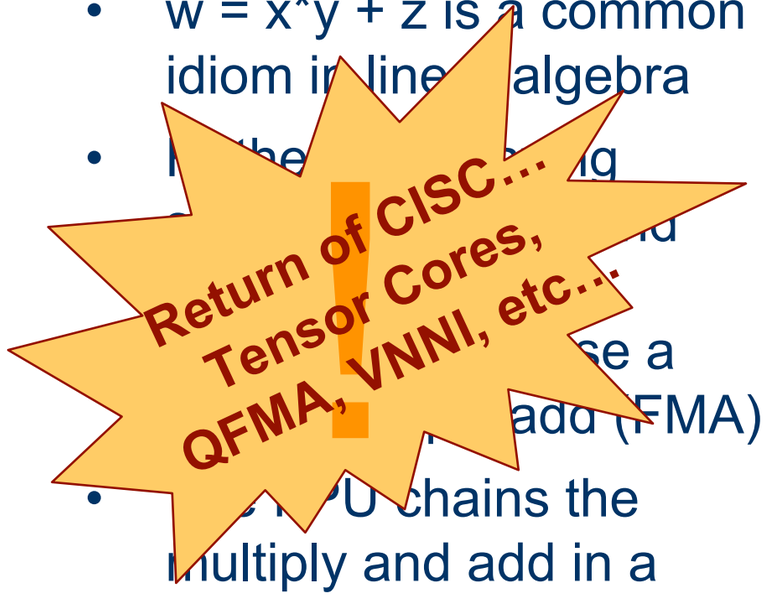
Roofline Model: In-Core Effects

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
- [illegible]
- [illegible] a fused multiply add (FMA)
- The FPU chains the multiply and add in a single pipeline so that it can complete FMA/cycle



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…

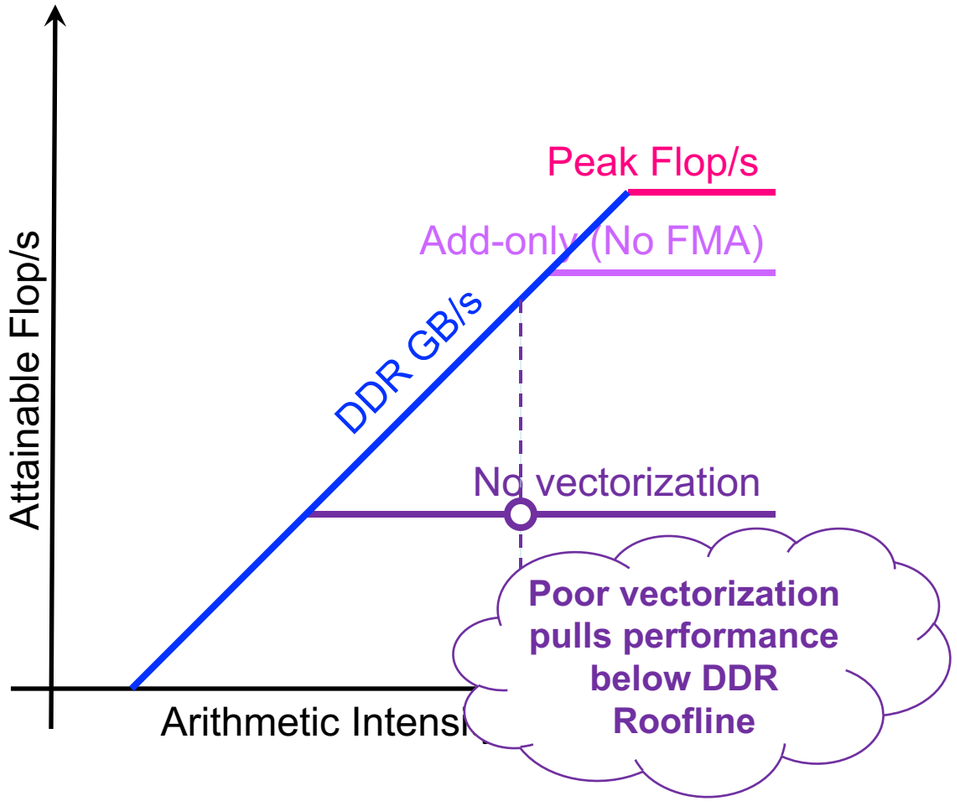
  x [0:7] *y [0:7] + z [0:7]
- Vector FPUs complete 8 vector operations/cycle

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

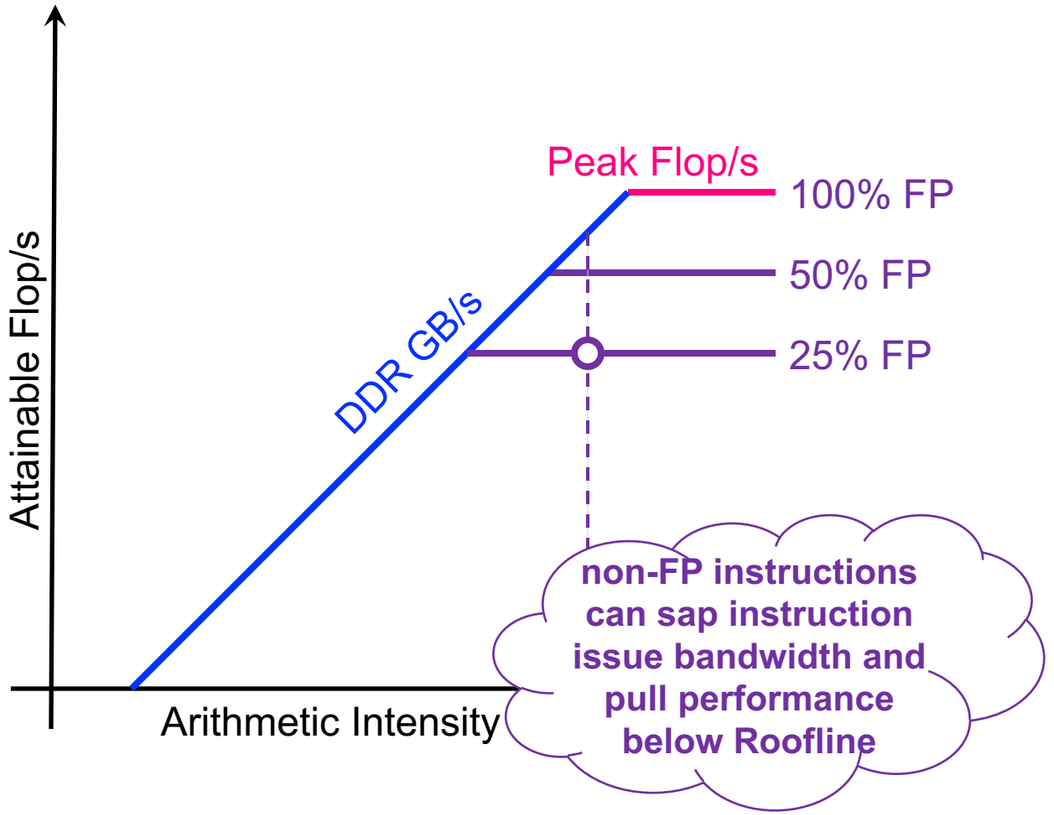
Data, Instruction, Thread-Level Parallelism…

- If every instruction were an ADD (instead of FMA), **performance would drop by 2x on KNL or 4x on Haswell**
- Similarly, if one had no vector instructions, performance would drop by **another 8x on KNL and 4x on Haswell**
- FP Divides can be even worse.
- Lack of threading will reduce performance by 64x on KNL.



Superscalar vs. instruction mix

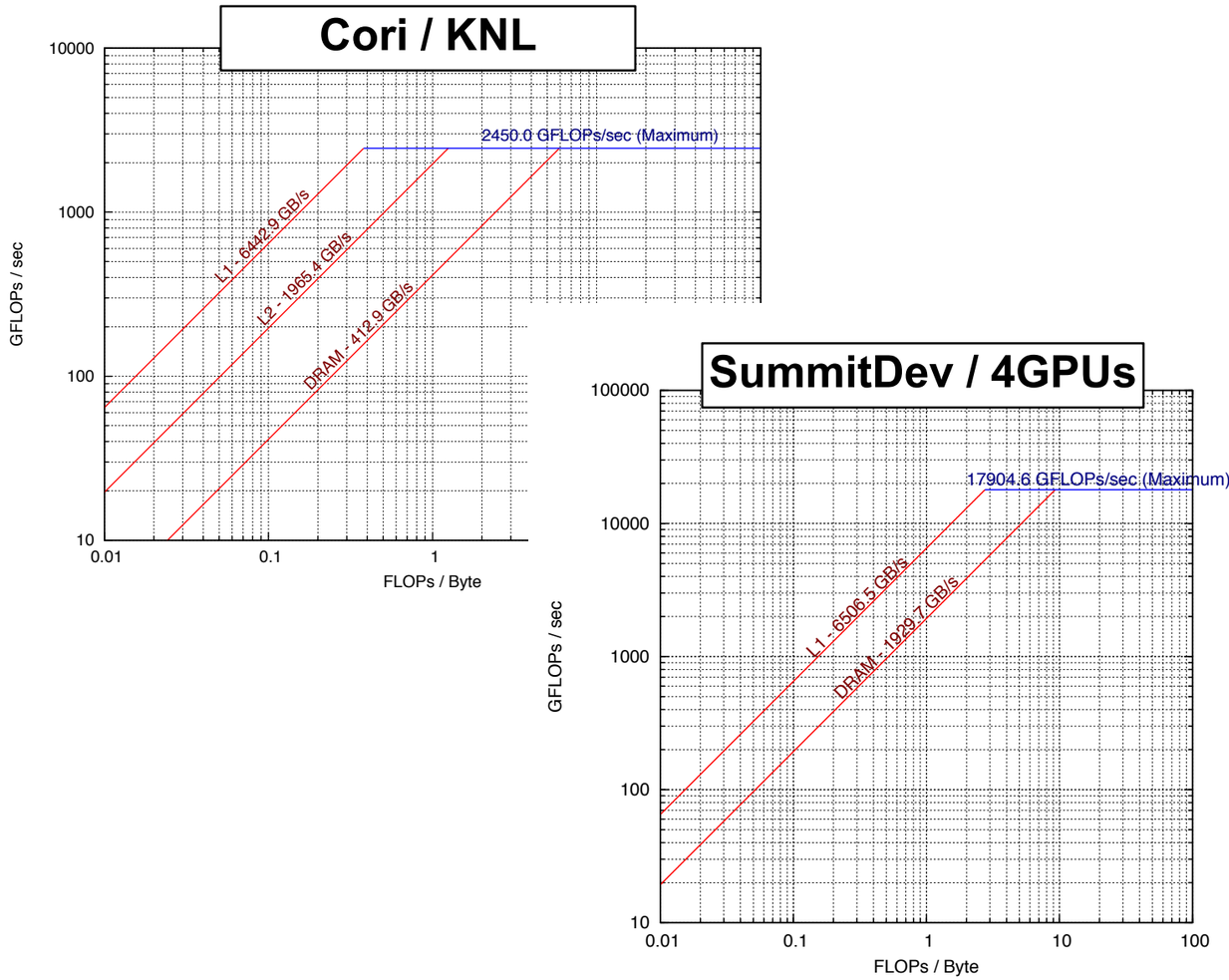
- Define in-core ceilings based on instruction mix…
- e.g. Haswell
  - 4-issue superscalar
  - Only 2 FP data paths
  - Requires 50% of the instructions to be FP to get peak performance
- e.g. KNL
  - 2-issue superscalar
  - 2 FP data paths
  - Requires 100% of the instructions to be FP to get peak performance



Node Characterization

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**



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

BERKELEY LAB

Application Characterization

Measuring AI

- To characterize application execution with Roofline we need…
  - Time
  - Flops (=> flop's / time)
  - Data movement between each level of memory (=> Flop's / GB's)
- We can look at the full application…
  - Coarse grained, 30-min average
  - Misses many details and bottlenecks
- … or we can look at individual loop nests
  - Requires auto-instrumentation on a loop by loop basis
  - Moreover, we should probably differentiate data movement or flops on a core-by-core basis.

Measuring Data Movement

Manual Counting

- Go thru each loop nest and estimate how many bytes will be moved
- Use a mental model of caches

✓ Works best for simple loops that stream from DRAM (stencils, FFTs, spare, …)

✗ N/A for complex caches

✗ Not scalable

Perf. Counters

- Read counter before/after

✓ Applies to full hierarchy (L2, DRAM,

✓ Accurate

✓ Low overhead (<%) == can run full MPI applications

✓ Can detect load imbalance

✗ Requires privileged access

✗ Requires manual instrumentation (+overhead) or full-app characterization

Cache Simulation

- Build a full cache simulator driven by memory addresses

✓ Applies to full hierarchy and multicore

✓ Can detect load imbalance

✓ Automated application to multiple loop nests

✗ Ignores prefetchers

✗ >10x overhead (limited to short runs / single node)

Measuring Flop's

Manual Counting

- Go thru each loop nest and count the number of FP operations
- ✓ Works best for deterministic loop bounds
- ✓ or parameterize by the number of iterations (recorded at run time)
- ✗ Not scalable

Perf. Counters

- Read counter before/after
- ✓ Accurate
- ✓ Low overhead (<%) == can run full MPI applications
- ✓ Can detect load imbalance
- ✗ Requires privileged access
- ✗ Requires manual instrumentation (+overhead) or full-app characterization
- ✗ Broken counters = garbage
- ✗ May not differentiate FMA from add
- ✗ No insight into special pipelines

Binary Instrumentation

- Automated inspection of assembly at run time
- ✓ Can count instructions by class/type
- ✓ FMA-, VL-, and mask-aware
- ✓ Can detect load imbalance
- ✓ Can include effects from non-FP instructions
- ✓ Automated application to multiple loop nests
- ✗ >10x overhead (limited to short runs / single node)

LIKWID

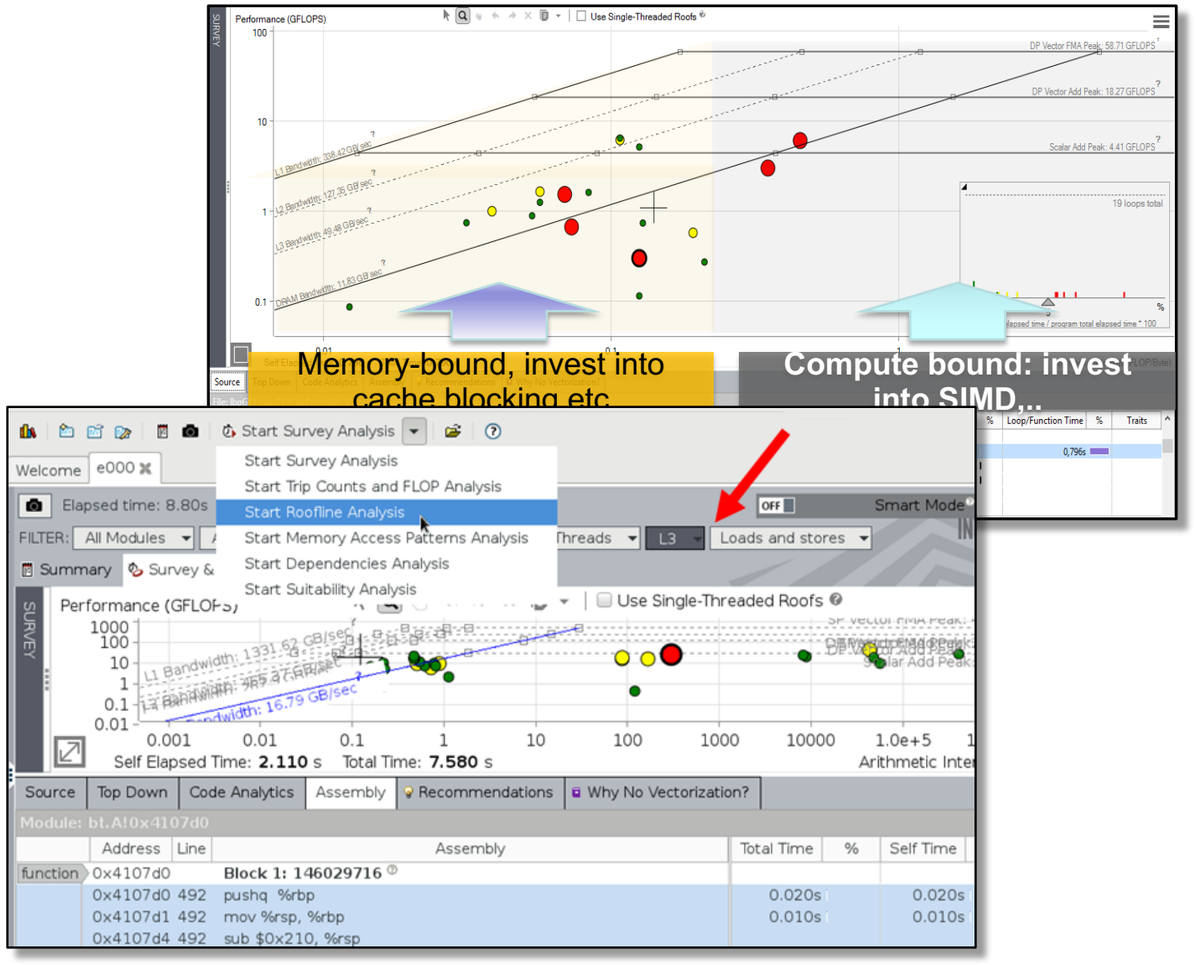
- LIKWID provides easy to use wrappers for measuring performance counters...
  - ✓ **Works on NERSC production systems**
  - ✓ Distills counters into user-friendly metrics (e.g. MCDRAM Bandwidth)
  - ✓ Minimal overhead (<1%)
  - ✓ Scalable in distributed memory (MPI-friendly)
  - ✓ Fast, high-level characterization
  - ✗ No timing breakdowns
  - ✗ **Suffers from Garbage-in/Garbage Out (e.g. broken Haswell FP counters)**

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

http://www.nersc.gov/users/software/performance-and-debugging-tools/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



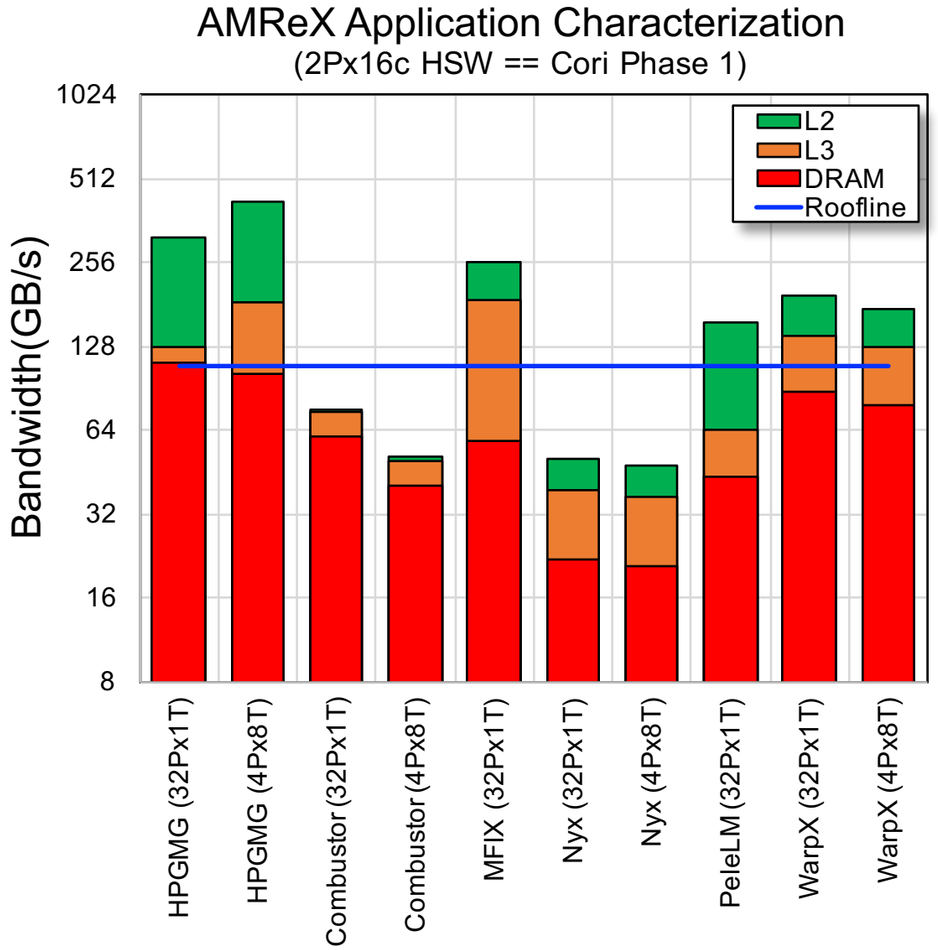
http://www.nersc.gov/users/training/events/roofline-training-1182017-1192017

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

Example Use Cases

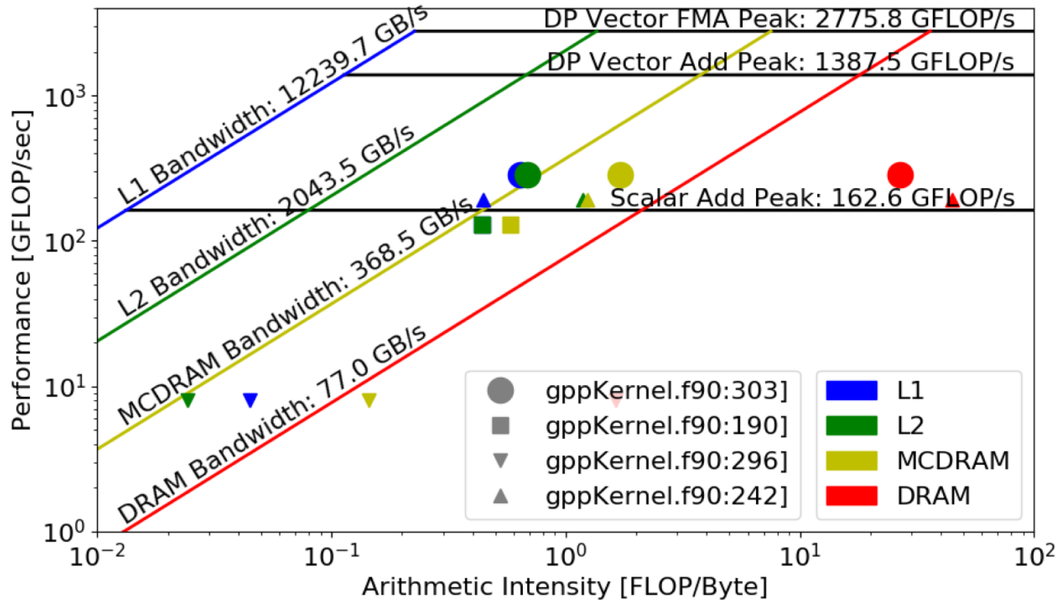
LIKWID on AMReX apps

- Used LIKWID to characterize AMReX applications
- Measured cache and DRAM bytes.
  - Averaged over 30min executions and 32 processes
  - Only 2 applications (not counting HPGMG proxy) used >50% of memory bandwidth on average
  - Used this data to estimate average AI for each level of the memory hierarchy
  - Used this data to infer requisite cache tapering



Using Intel Advisor at NERSC

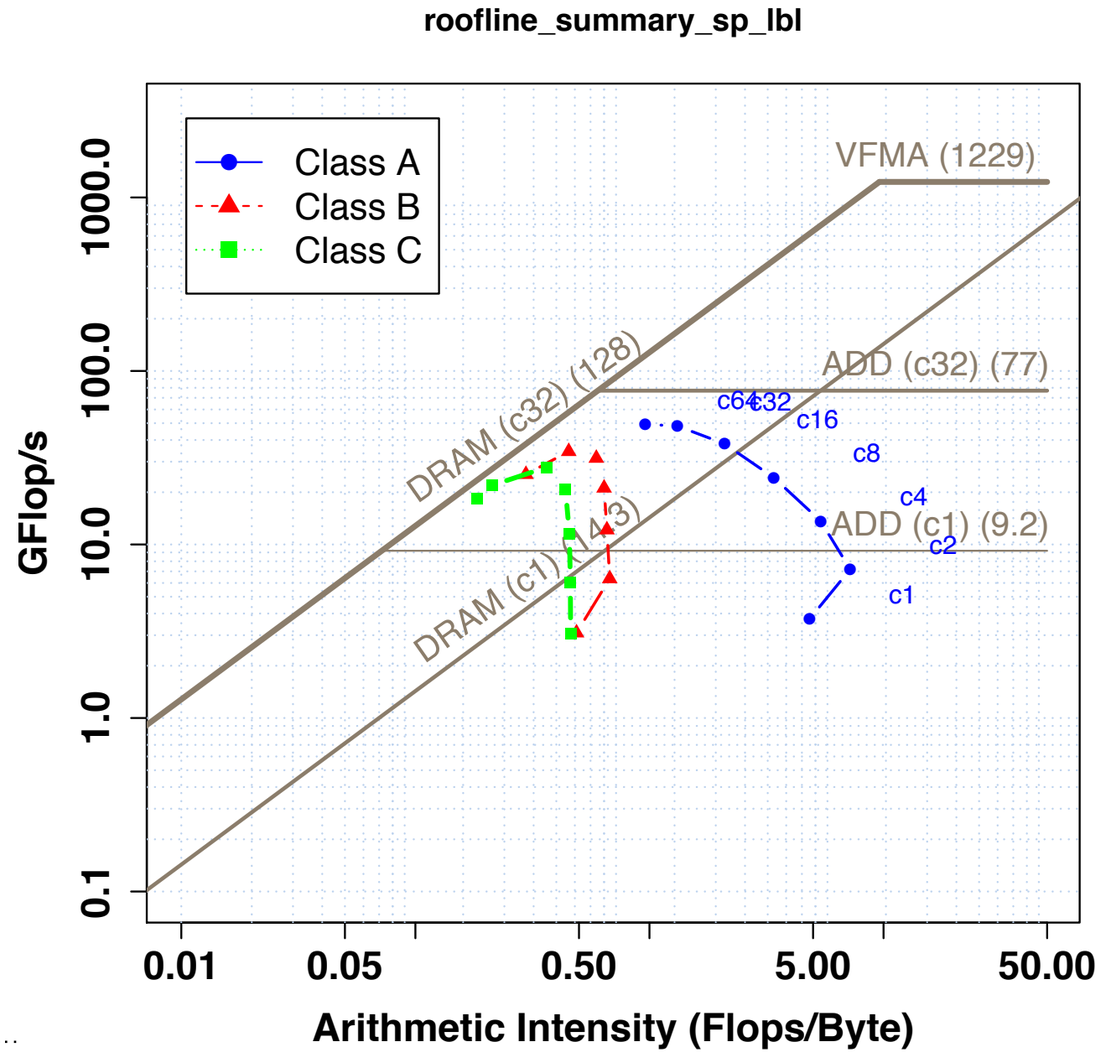
- Used Advisor to analyze cache/MCDRAM/DDR behavior of multiple apps on KNL
  - Some loops bound by L2
  - Some by MCDRAM
  - Some loops had no clear memory or flop bound
  - **%FMA?**
  - **%Vectorized?**
  - **Non-FP vector instructions?**
  - **Non-vector instructions?**
  - **Unpipelined instructions (e.g. divide)?**



Tuomas Koskela, Zakhar Matveev, Charlene Yang, Adetokunbo Adedoyin, Roman Belenov, Philippe Thierry, Zhengji Zhao, Rahulkumar Gayatri, Hongzhang Shan, Leonid Oliker, Jack Deslippe, Ron Green, and Samuel Williams, "A Novel Multi-Level Integrated Roofline Model Approach for Performance Characterization", ISC, June 2018.

Roofline Scaling Trajectories

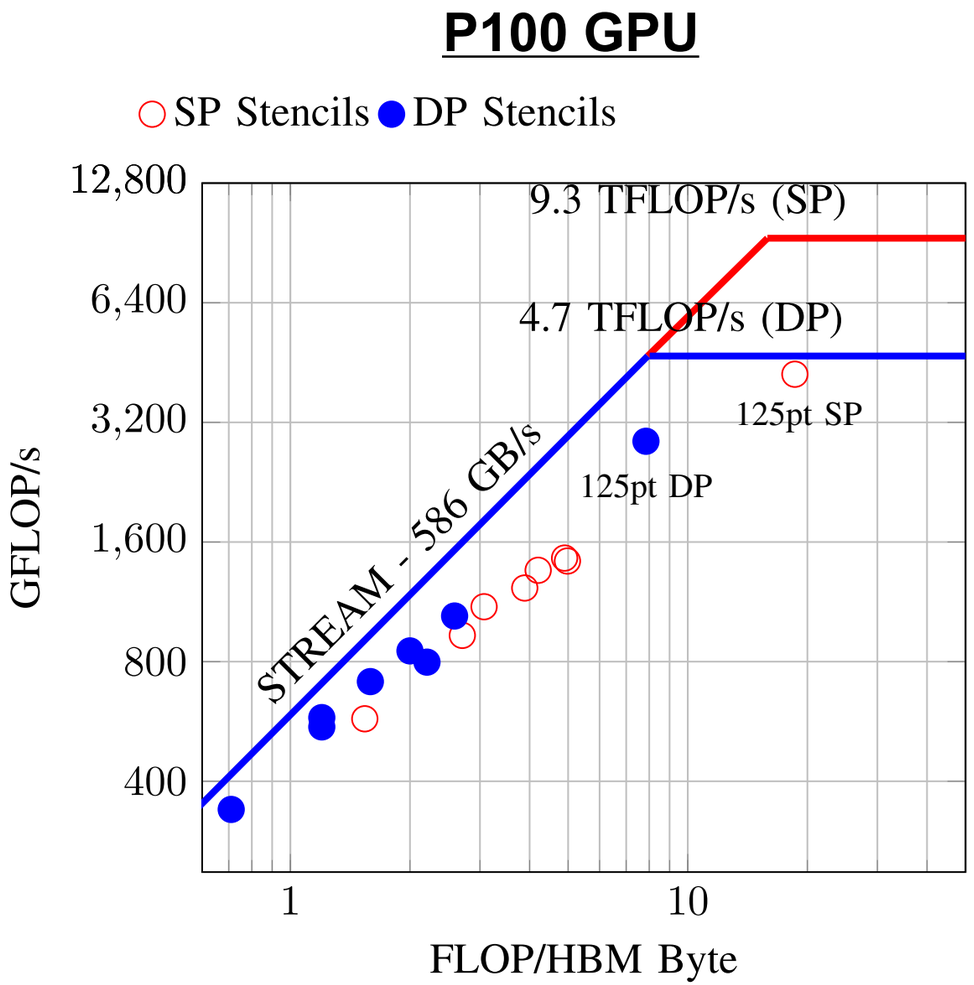
- Often, one plots performance as a function of thread concurrency
  - Carries no insight or analysis
  - Provides no actionable information.
- Khaled Ibrahim developed a new way of using Roofline to analyze thread (or process) scalability
  - Create a 2D scatter plot of performance as a function of AI and thread concurrency
  - Can identify loss in performance due to increased cache pressure



Khaled Ibrahim, Samuel Williams, Leonid Oliker, "Roofline Scaling Trajectories: A Method for Parallel Application and Architectural Performance Analysis", HPCS Special Session on High Performance Computing Benchmarking and Optimization (HPBench), July 2018.

Roofline on GPUs

- We can similarly use NVProf to record HBM data movement on GPUs.
- We used this technique to evaluate performance of autotuning stencil library developed under an ECP ST project.



Tuowen Zhao, Mary Hall, Protonu Basu, Samuel Williams, Hans Johansen, ” Performance Portability for Stencils across CPUs and GPUs Using Bricks", (submitted to) Supercomputing, 2018.

Future Directions

Understanding/Visualizing Load Imbalance

- Generally, Roofline presumes computation and data movement is balanced
  - Obviously, a single core cannot hit full-socket flop/s
  - **More subtly, a single core cannot come anywhere close to socket GB/s**
  - We are investigating how to visualize this in Roofline **(e.g. load imbalance ceiling)**

- On Heterogeneous systems, the issue is more subtle
  - Nominally, Roofline treats heterogeneous systems as two homoegeneous (sub)systems
  - Examining work partitioning in accelerated applications and visualization

New Architectures

- End of Dennard Scaling + End of Moore's Law
  - Exponential growth in transistors will slow (end)
  - Can't simply increase system size due to power & energy
  - Extract exponential performance from a ~fixed number transistors
  - **The emergence of specialization**

- Emergence of Specialization and Multimodal Heterogeneity
  - Return to CISC (tensor cores, QFMA, VNNI, …)
  - (multiple) specialized cores/accelerators (big/little, but we can think more profound)
  - (multiple) specialized discrete accelerators / node types
  - Multiple memory types (can't get capacity, bandwidth, and energy in a single type)

Extending Roofline to New Architectures

- Requirements…
  - Bandwidth (vertical, horizontal, and asymmetry)
  - Measuring Data movement (vertical and horizontal)
  - Special in-core ceilings (presence and exploitation)
  - Overheads

- Targets…
  - AI ISA extensions
  - NNPs and TPUs
  - FPGAs (interesting tradeoffs)
  - NVM / HBM

Consumers?

- Those wanting to understand performance on new architectures
- Those building runtimes/mappers/compilers that need accurate cost models
- Those wanting to develop new algorithms/discretizations appropriate for exascale systems.

Bottlenecks:
End of the Road or New Opportunities?

Memory (2004-?)

- Conventional Wisdom:
  - Computation is limited by how fast we can move data (flops are free)
  - We must minimize data movement
  - We must have more bandwidth
- New Conventional Wisdom:
  - Flop-heavy methods that were cost prohibitive are now attractive if they reduce (net) data movement
  - ➢ **If Flop/s are free, use them (there's no penalty).**
  - Recompute terms on the fly (rather than storing in memory)
  - Use high-order discretizations (equal error for reduced total data movement)
  - Communication-Avoiding Algorithms

Latency/Overhead (2021-?)

- Conventional Wisdom:
  - Limits on performance (can't get 90% of peak on any CUDA kernel that does less than 1.4B FP ops
  - Numerical methods and applications limited by Computational Depth (20us * #synchronization points)
- New Conventional Wisdom:
  - You can do anything you want (locally) every 20us
  - Flop- and Bandwidth-heavy methods that were cost prohibitive are now attractive if they reduce (net) synchronization…
  - ➢ **If Flop/s and GB/s are free, use them**
  - No penalty for redundant computation (or redundant data movement) if it reduces synchronization
  - Synchronization-Avoiding Algorithms
  - Speculative Execution

We're Hiring...

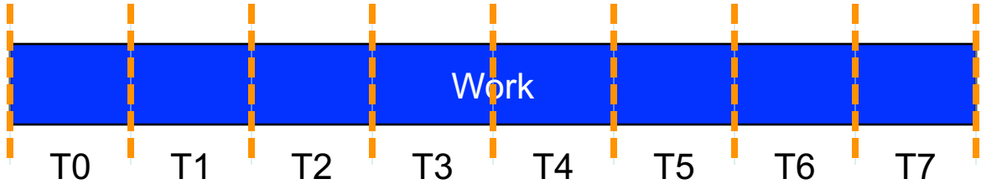
jobs.lbl.gov
search for #85373

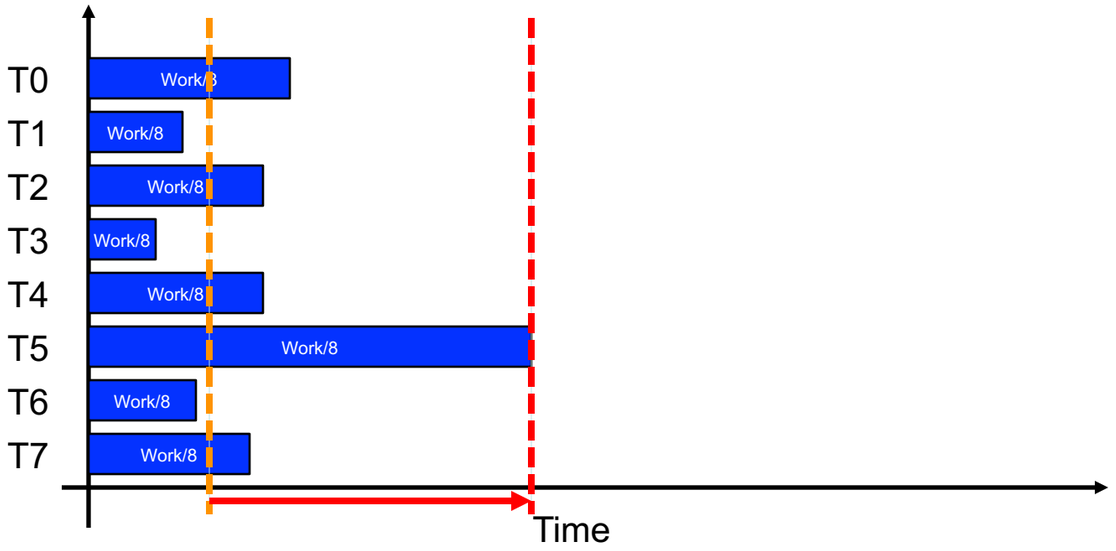
Questions

Backup

Load Balancing

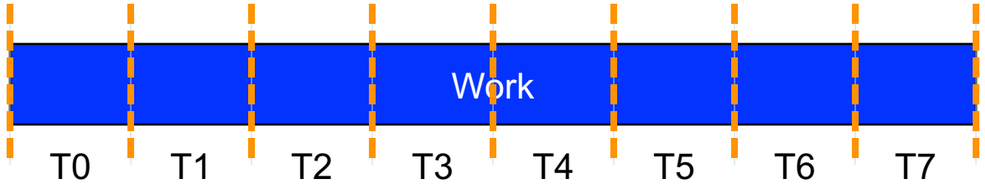
- Unfortunately, some loop iterations may be more expensive, or some threads may run slower (e.g. cache effects)
  - As a result, we can observe load imbalance where run time is limited by the slowest thread
  - We can assess the degree of load imbalance by measuring **max/average**
  - **A slow outlier may substantially hurt performance**, but…

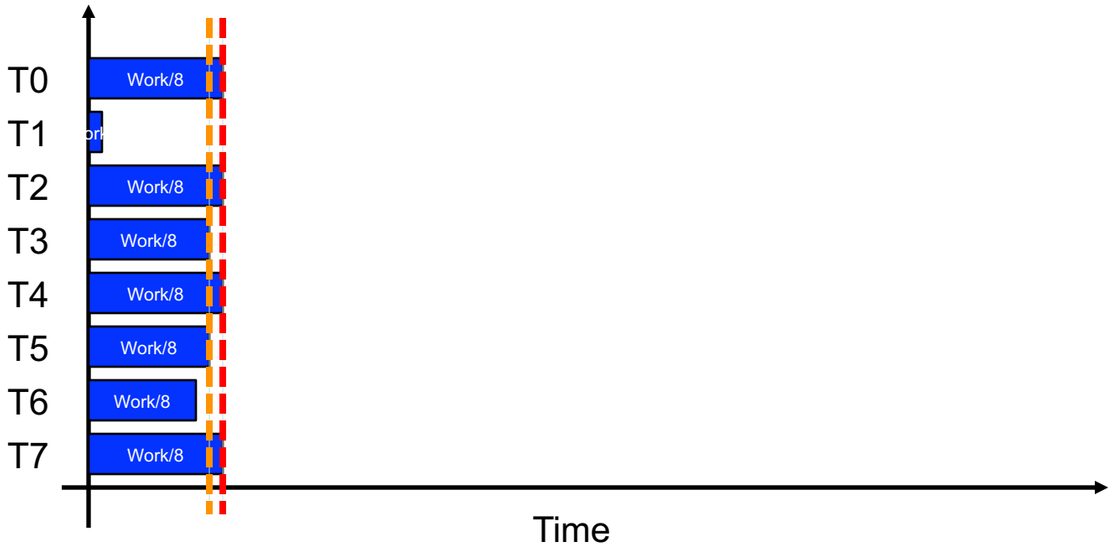




Load Balancing

- Unfortunately, some loop iterations may be more expensive, or some threads may run slower (e.g. cache effects)
  - As a result, we can observe load imbalance where run time is limited by the slowest thread
  - We can assess the degree of load imbalance by measuring **max/average**…
  - **A slow outlier may substantially hurt performance**, but…
  - …a fast thread may not help or hurt much





Lack of Parallelism

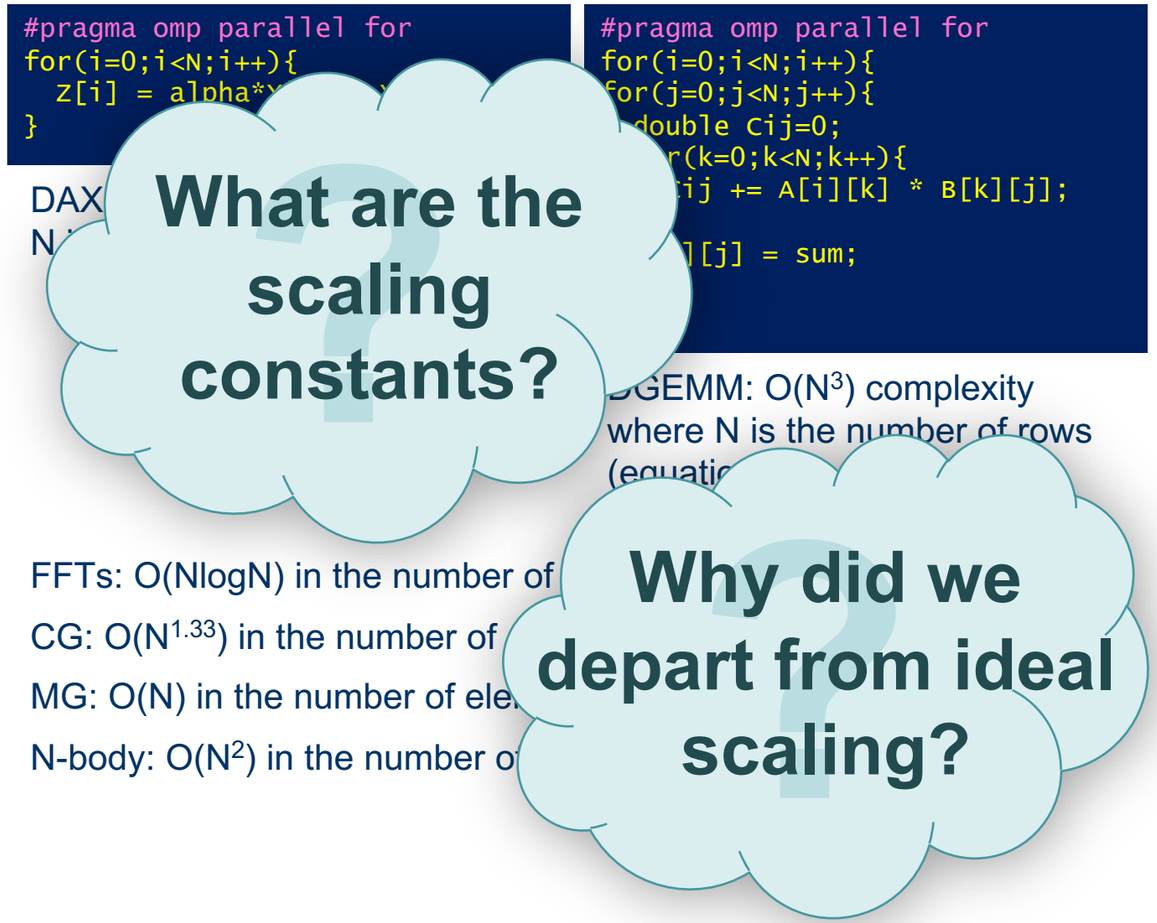
- Trends in architecture have enabled >>1000-way parallelism on a chip (#FPUs * FPU latency)
- Not all loop nests support 1000-way parallelization
- Loop nests with <1000-way parallelism underutilize HW resources

- Often codes must be restructured to enable more parallelism
  - Loops are reordered/fused (OMP collapse(3))
  - Variables(arrays) are privatized and reduced
  - Nominally sequential functions/solvers on independent variables are performed concurrently (MPI sub communicators or OMP Tasks)
  - Workflows/multiphysics are parallelized at launch (SLURM MPMD)

Performance Models

Computational Complexity

- Assume run time is correlated with the number of operations (e.g. FP ops)
- Users define parameterize their algorithms, solvers, kernels
- Count the number of operations as a function of those parameters
- Demonstrate run time is correlated with those parameters



Data Movement Complexity

- Assume run time is correlated with the amount of data accessed (or moved)
- Easy to calculate amount of data accessed… count array accesses
- Data moved is more complex as it requires understanding cache behavior…
  - Compulsory[1] data movement (array sizes) is a good initial guess…
  - … but needs refinement for the effects of finite cache capacities

Operation	Flop's	Data
DAXPY	O(N)	O(N)
DGEMV	$O(N^2)$	$O(N^2)$
DGEMM	$O(N^3)$	$O(N^2)$
FFTs	O(NlogN)	O(N)
CG	$O(N^{1.33})$	
MG		
N-body		

Which is more expensive…

Performing Flop's, or

Moving words from memory

[1]Hill et al, "Evaluating Associativity in CPU Caches", IEEE Trans. Comput., 1989.

Machine Balance and Arithmetic Intensity

- Data movement and computation can operate at different rates
- We define machine balance as the ratio of…

$$\textbf{Balance} = \frac{\textbf{Peak DP Flop/s}}{\textbf{Peak Bandwidth}}$$

- …and arithmetic intensity as the ratio of…

$$\textbf{AI} = \frac{\textbf{Flop's Performed}}{\textbf{Data Moved}}$$

Operation	Flop's	Data	AI (ideal)
DAXPY	O(N)	O(N)	O(1)
DGEMV	$O(N^2)$	$O(N^2)$	O(1)
DGEMM	$O(N^3)$		O(N)
FFTs	O(N		O(logN)
CG	O(N		
MG			O(1)
N-body			O(N)

Kernels with AI less than machine balance are ultimately bandwidth-limited

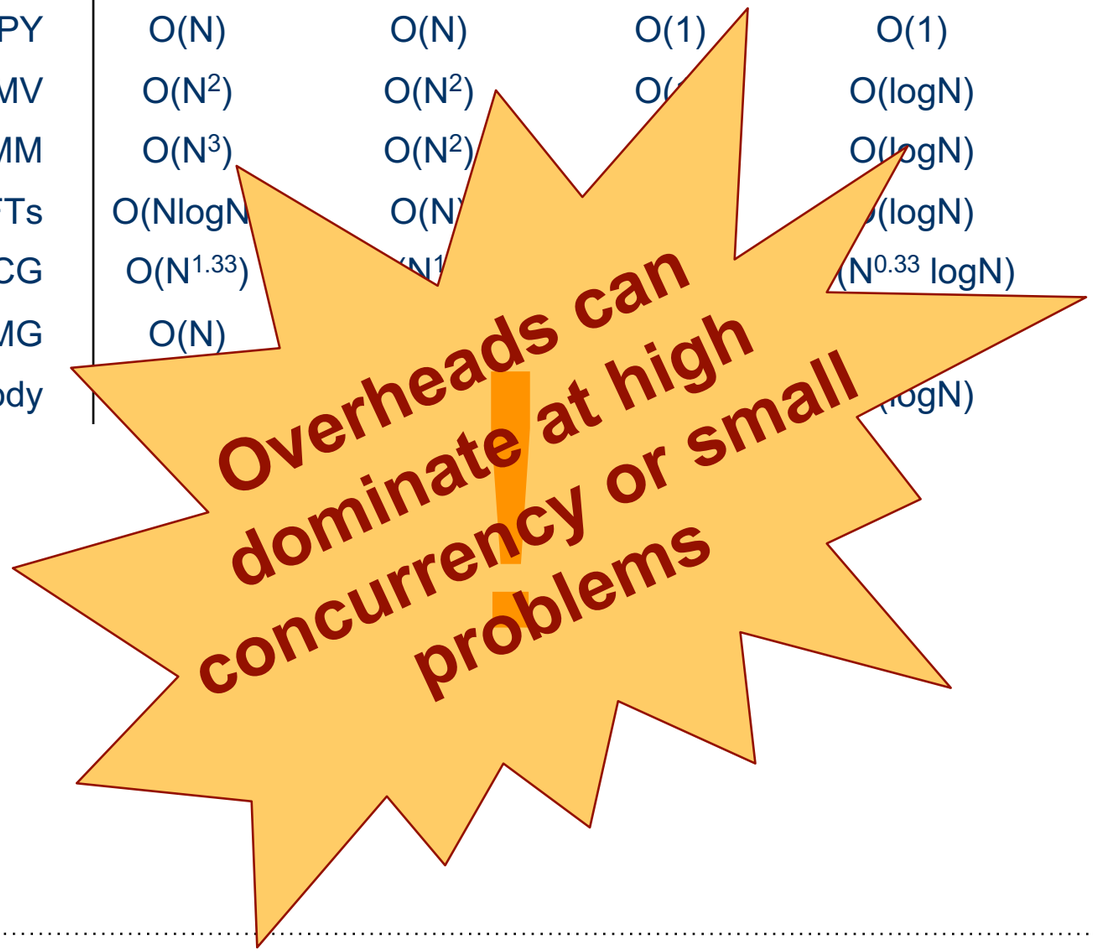
Distributed Memory Performance Modeling

- In distributed memory, one communicates by sending messages between processors.
- Messaging time can be constrained by several components…
  - Overhead (CPU time to send/receive a message)
  - Latency (time message is in the network; can be hidden)
  - Message throughput (rate at which one can send small messages… messages/second)
  - Bandwidth (rate one can send large messages… GBytes/s)
- Bandwidths and latencies are further constrained by the interplay of network architecture and contention
- Distributed memory versions of our algorithms can be differently stressed by these components depending on N and P (#processors)

Computational Depth

- Imagine a world of infinite parallelism & bandwidth, but finite latencies
- We can classify algorithms by **depth** (max depth of the algorithm's dependency chain)
- For iterative algorithms, this is product of iterations and depth per iteration

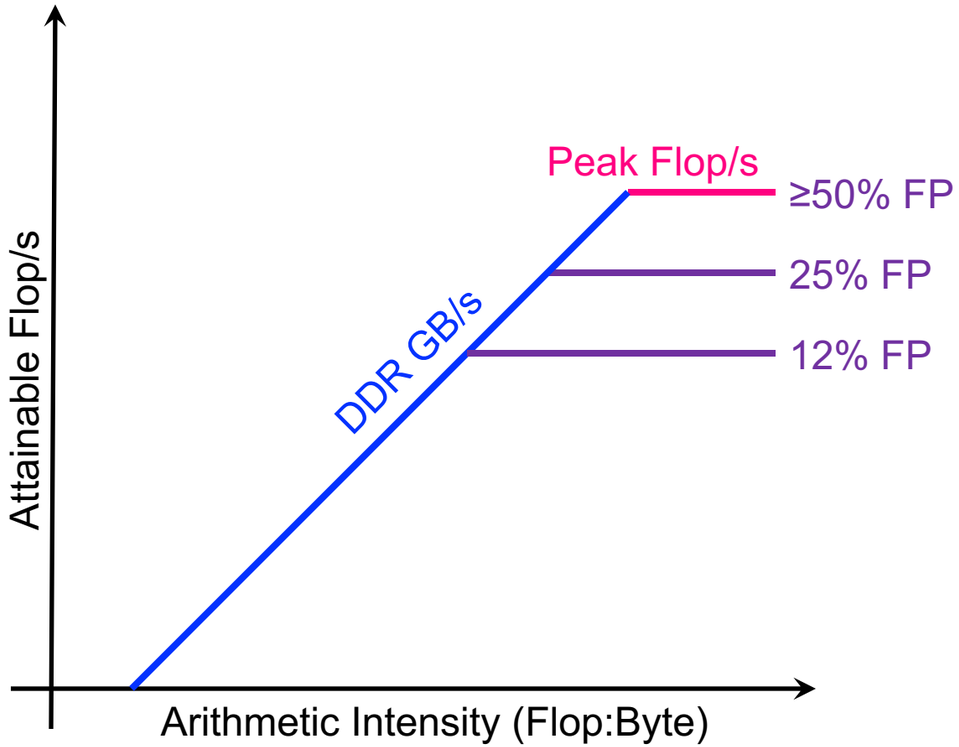
Operation	Flop's	Data	AI (ideal)	Depth
DAXPY	O(N)	O(N)	O(1)	O(1)
DGEMV	$O(N^2)$	$O(N^2)$	O(	O(logN)
DGEMM	$O(N^3)$	$O(N^2)$		O(logN)
FFTs	O(NlogN	O(N		(logN)
CG	$O(N^{1.33})$	(N1		$(N^{0.33}$ logN)
MG	O(N)			
N-body				logN)



Roofline Model: In-Core Effects

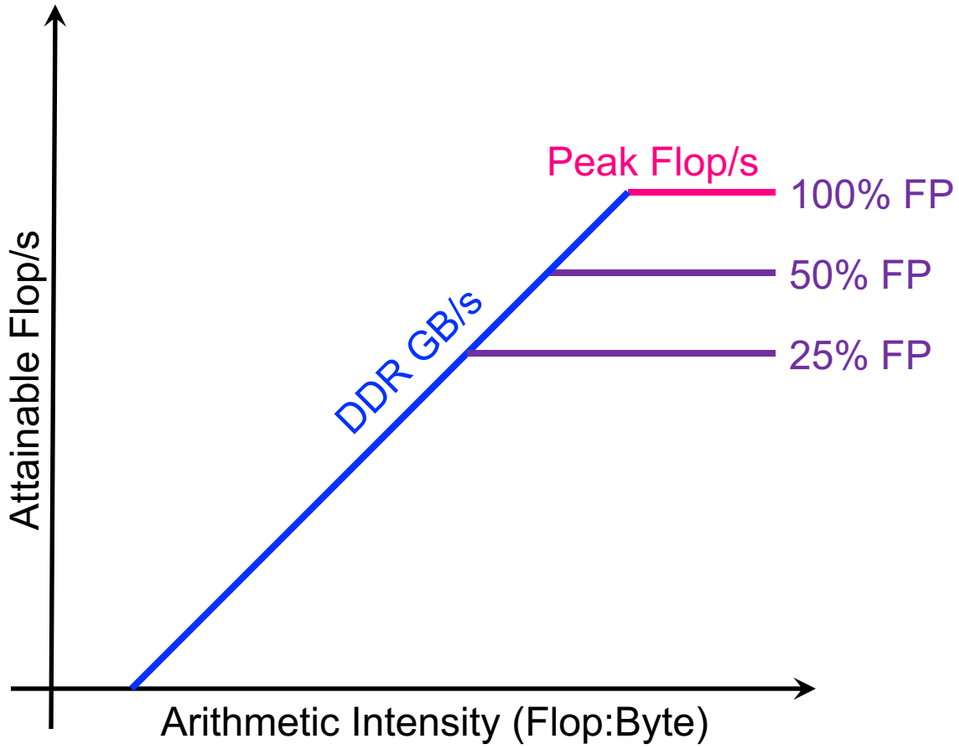
Superscalar vs. instruction mix

- 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



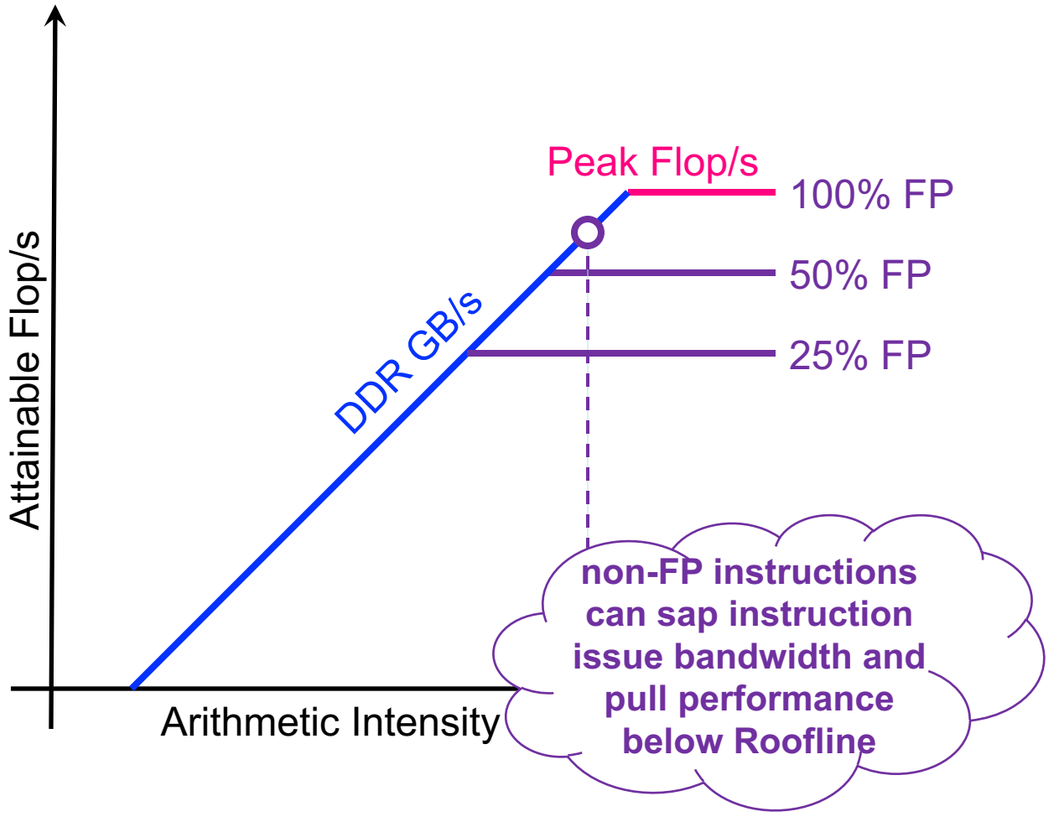
Superscalar vs. instruction mix

- 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



Superscalar vs. instruction mix

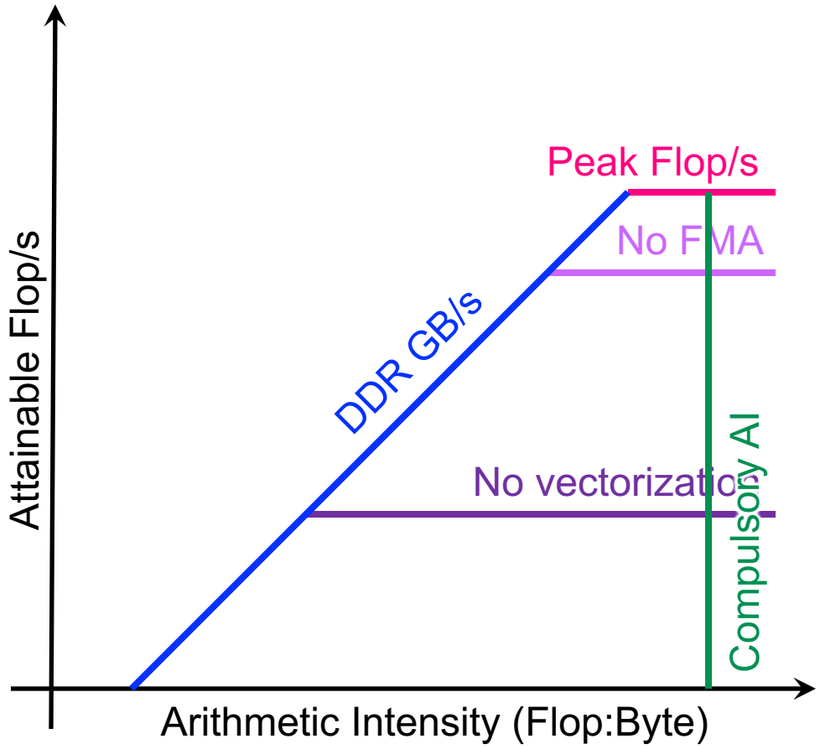
- Define in-core ceilings based on instruction mix…
- e.g. Haswell
  - 4-issue superscalar
  - Only 2 FP data paths
  - Requires 50% of the instructions to be FP to get peak performance
- e.g. KNL
  - 2-issue superscalar
  - 2 FP data paths
  - Requires 100% of the instructions to be FP to get peak performance



Roofline Model: Cache Effects

Locality Walls

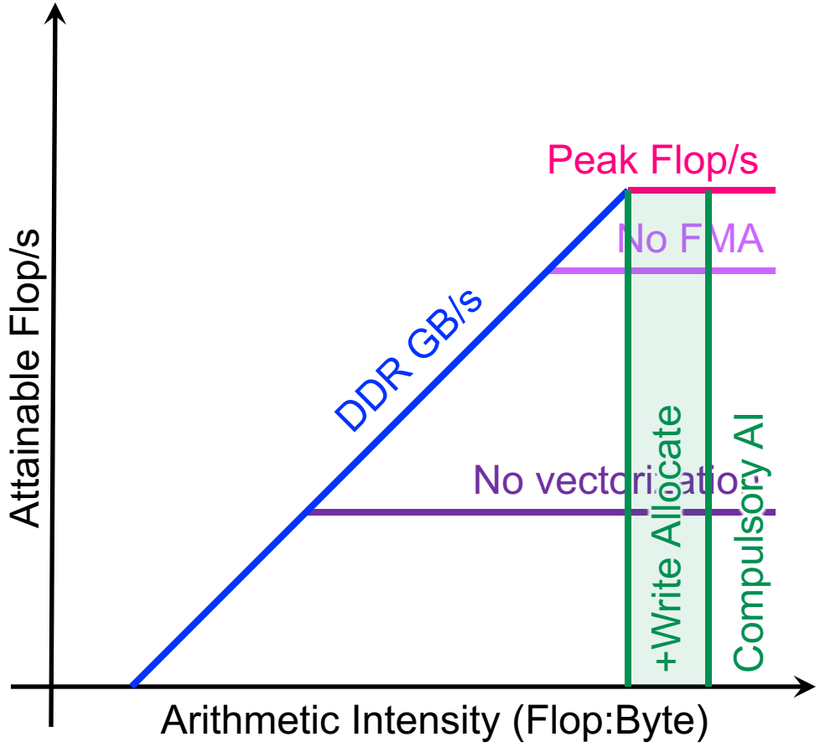
- Naively, we can bound AI using only compulsory cache misses



$$AI = \frac{\#Flop's}{Compulsory\ Misses}$$

Locality Walls

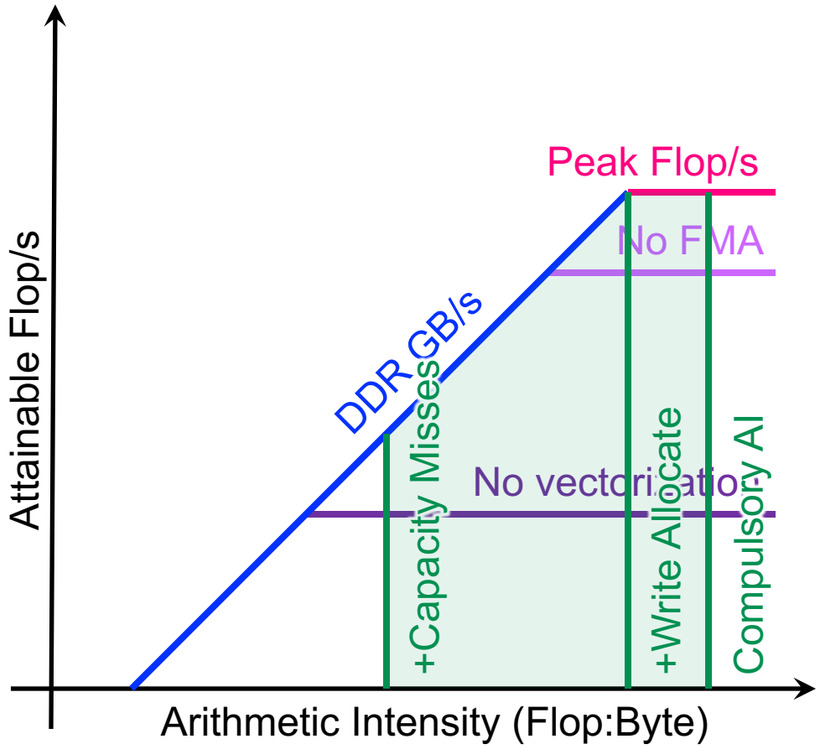
- Naively, we can bound AI using only compulsory cache misses
- However, write allocate caches can lower AI



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

Locality Walls

- 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

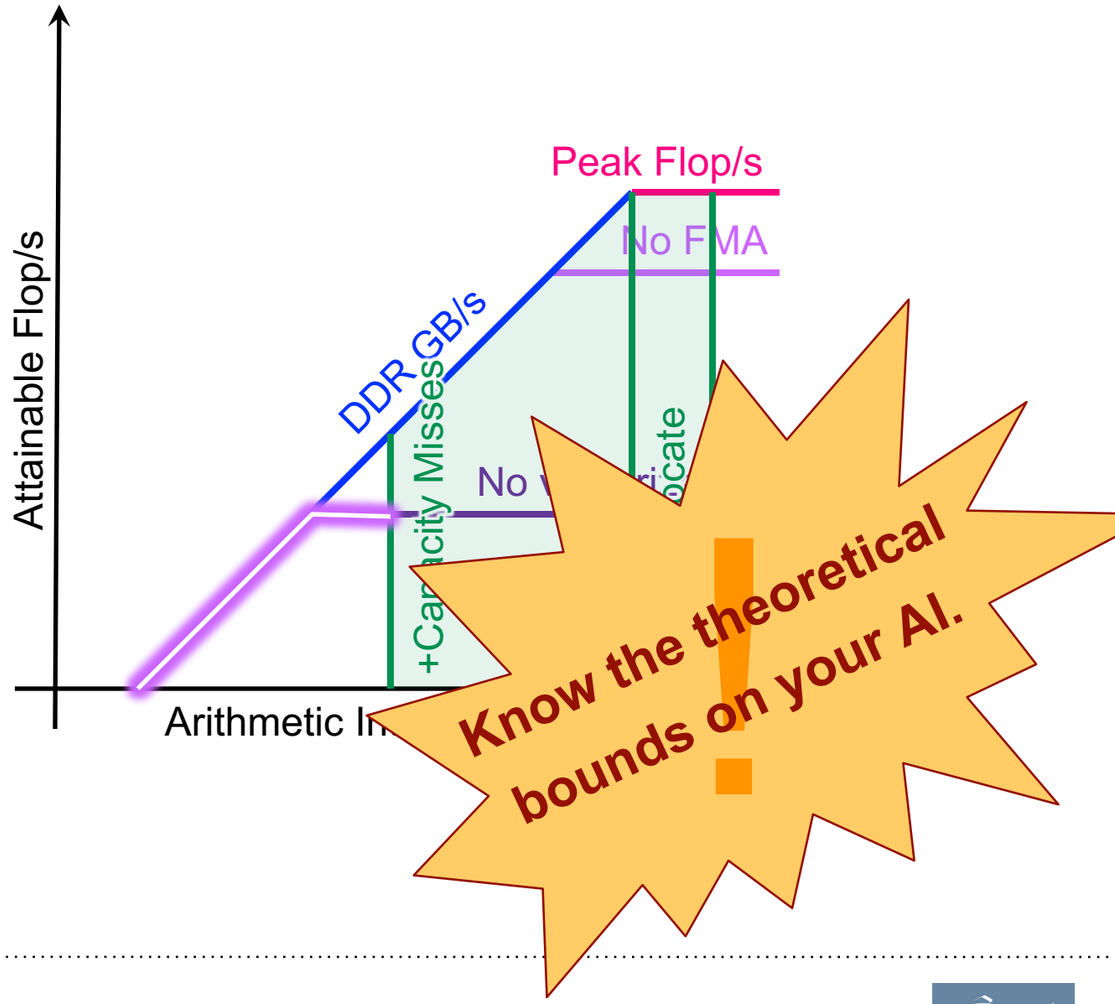


$$AI = \frac{\#Flop's}{Compulsory\ Misses + Write\ Allocates + Capacity\ Misses}$$

Locality Walls

- 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**

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



Hierarchical Roofline vs. Cache-Aware Roofline

…understanding different Roofline formulations in Advisor

There are two Major Roofline Formulations:

- Hierarchical Roofline (original Roofline w/ DRAM, L3, L2, …)…
  - **Williams, et al, "Roofline: An Insightful Visual Performance Model for Multicore Architectures", CACM, 2009**
  - **Chapter 4 of "Auto-tuning Performance on Multicore Computers", 2008**
  - Defines multiple bandwidth ceilings and multiple AI's per kernel
  - Performance bound is the minimum of flops and the memory intercepts (superposition of original, single-metric Rooflines)
- Cache-Aware Roofline
  - **Ilic et al, "Cache-aware Roofline model: Upgrading the loft", IEEE Computer Architecture Letters, 2014**
  - Defines multiple bandwidth ceilings, but uses a single AI (flop:L1 bytes)
  - As one looses cache locality (capacity, conflict, …) performance falls from one BW ceiling to a lower one at constant AI
- Why Does this matter?
  - Some tools use the Hierarchical Roofline, some use cache-aware **== Users need to understand the differences**
  - Cache-Aware Roofline model was integrated into production Intel Advisor
  - Evaluation version of Hierarchical Roofline[1] (cache simulator) has also been integrated into Intel Advisor

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

Hierarchical Roofline

- Captures cache effects
- AI is Flop:Bytes after being ***filtered by lower cache levels***
- Multiple Arithmetic Intensities (one per level of memory)
- AI ***dependent*** on problem size (capacity misses reduce AI)
- Memory/Cache/Locality effects are ***observed as decreased AI***
- Requires ***performance counters or cache simulator*** to correctly measure AI

Cache-Aware Roofline

- Captures cache effects
- AI is Flop:Bytes ***as presented to the L1 cache (plus non-temporal stores)***
- Single Arithmetic Intensity
- AI ***independent*** of problem size
- Memory/Cache/Locality effects are ***observed as decreased performance***
- Requires static analysis or ***binary instrumentation*** to measure AI

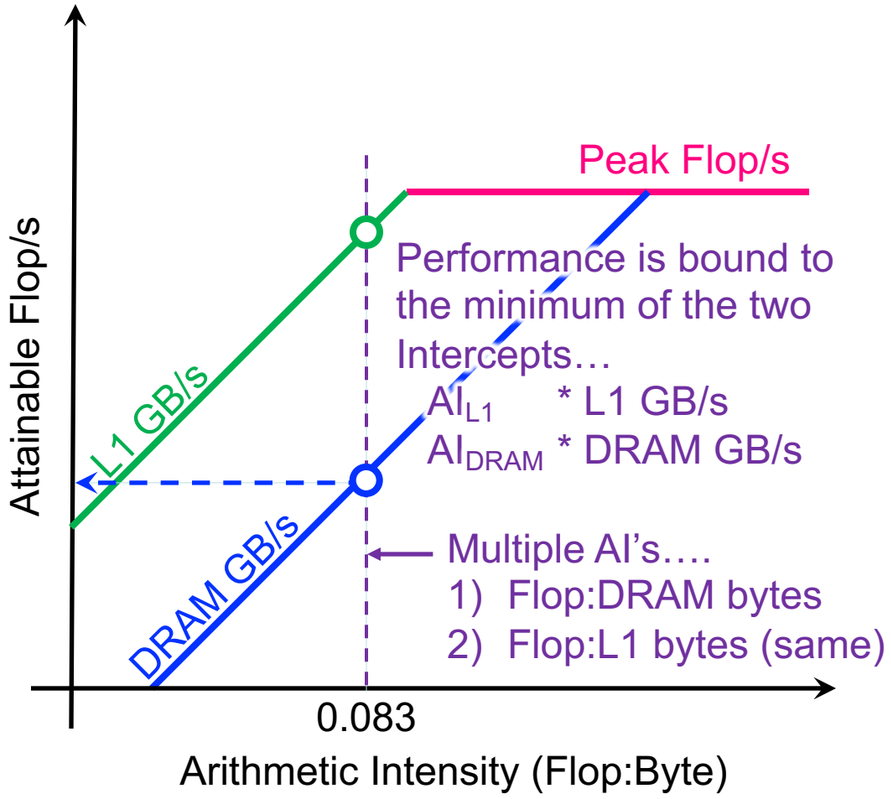
Example: STREAM

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

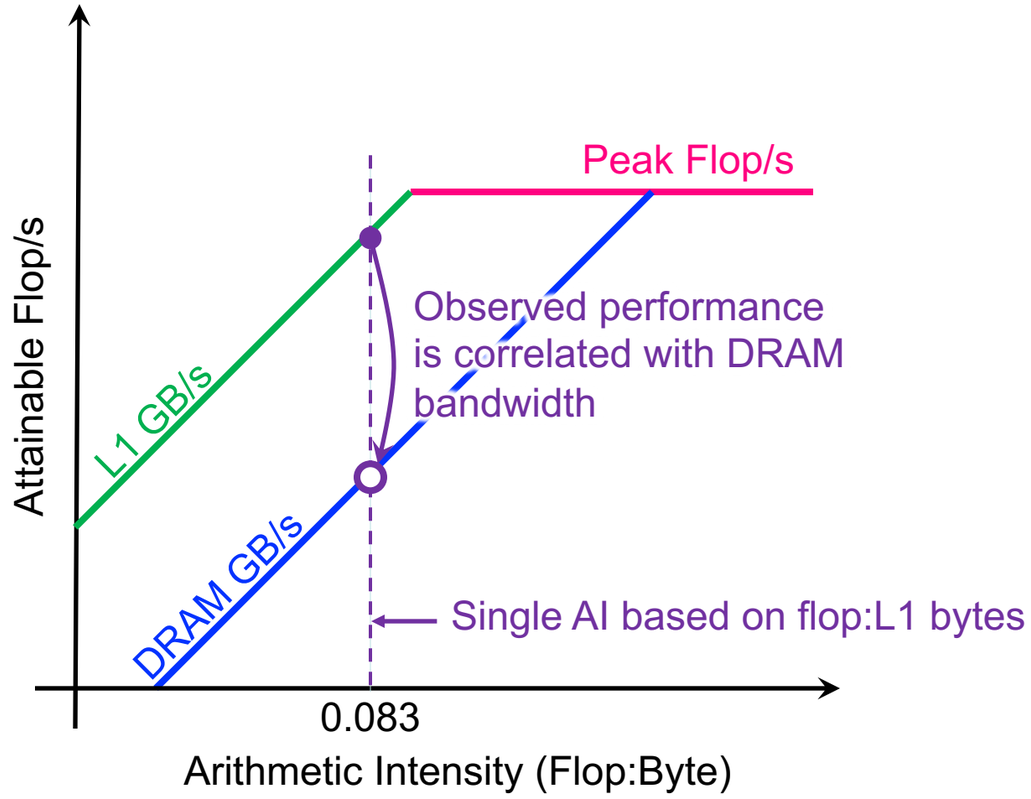
```
#pragma omp parallel for
for(i=0;i<N;i++){
  Z[i] = X[i] + alpha*Y[i];
}
```

Example: STREAM

Hierarchical Roofline



Cache-Aware Roofline



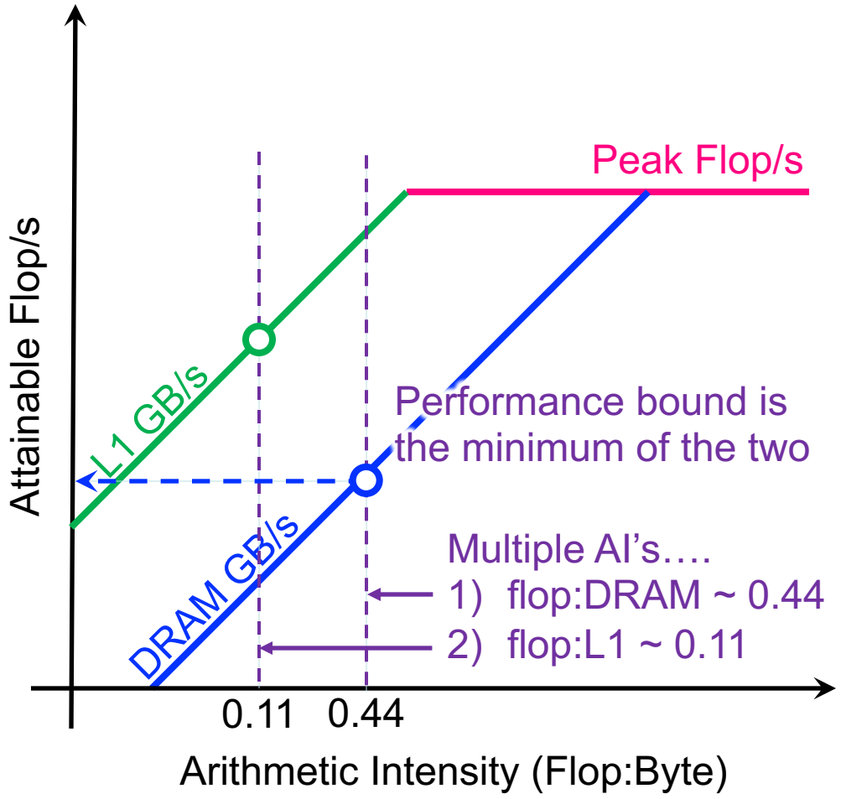
Example: 7-point Stencil (Small Problem)

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

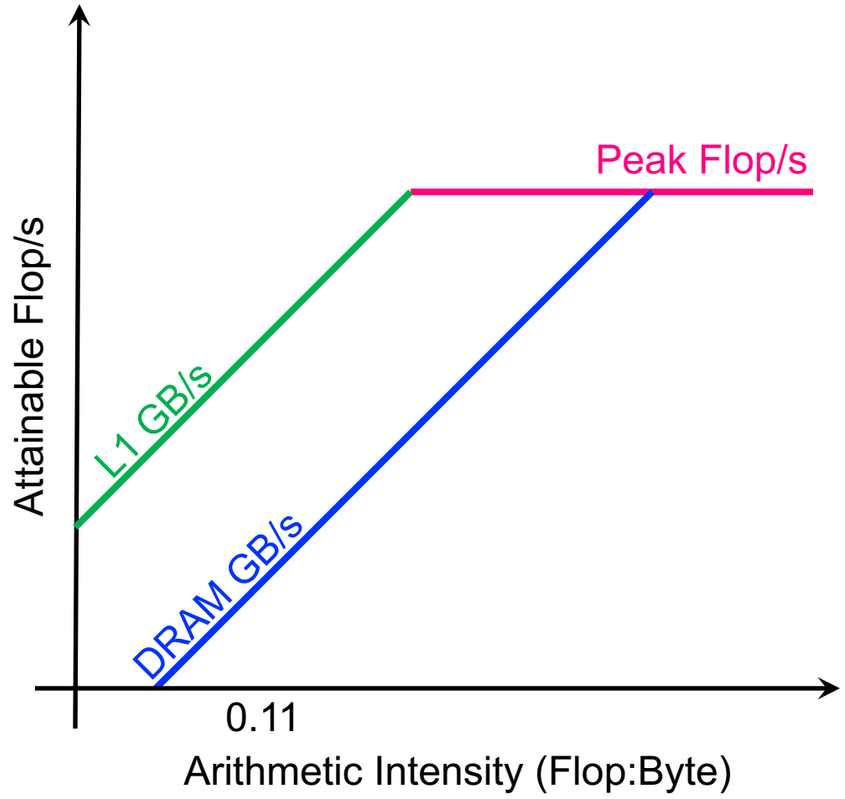
```
#pragma omp parallel for
for(k=1;k<dim+1;k++){
for(j=1;j<dim+1;j++){
for(i=1;i<dim+1;i++){
  int ijk = i + j*jStride + k*kStride;
  new[ijk] = -6.0*old[ijk          ]
                 + old[ijk-1       ]
                 + old[ijk+1       ]
                 + old[ijk-jStride]
                 + old[ijk+jStride]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

Example: 7-point Stencil (Small Problem)

Hierarchical Roofline

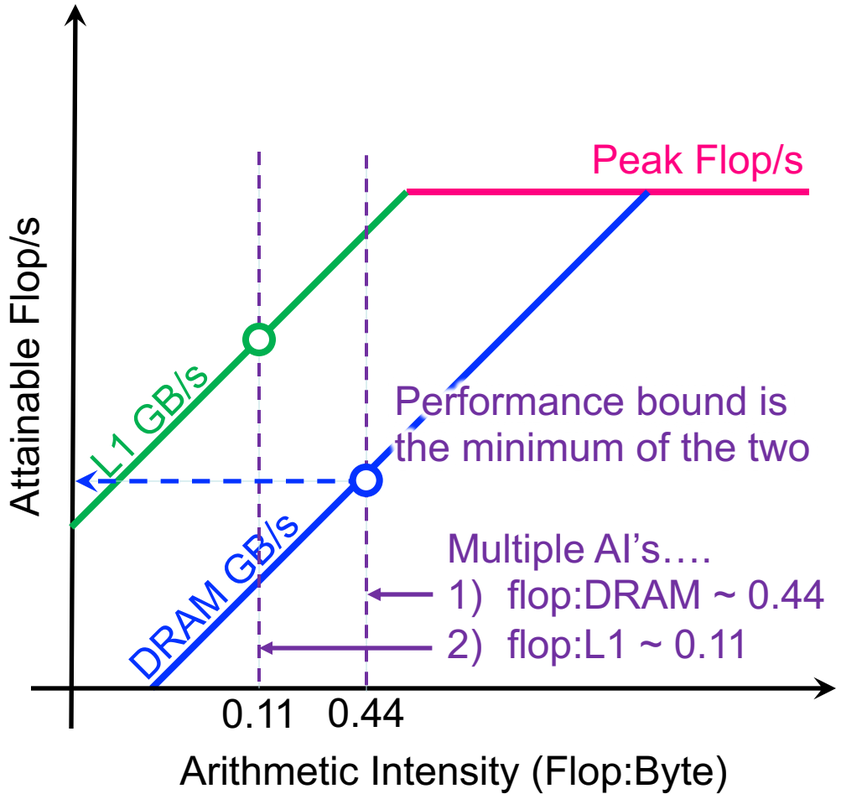


Cache-Aware Roofline

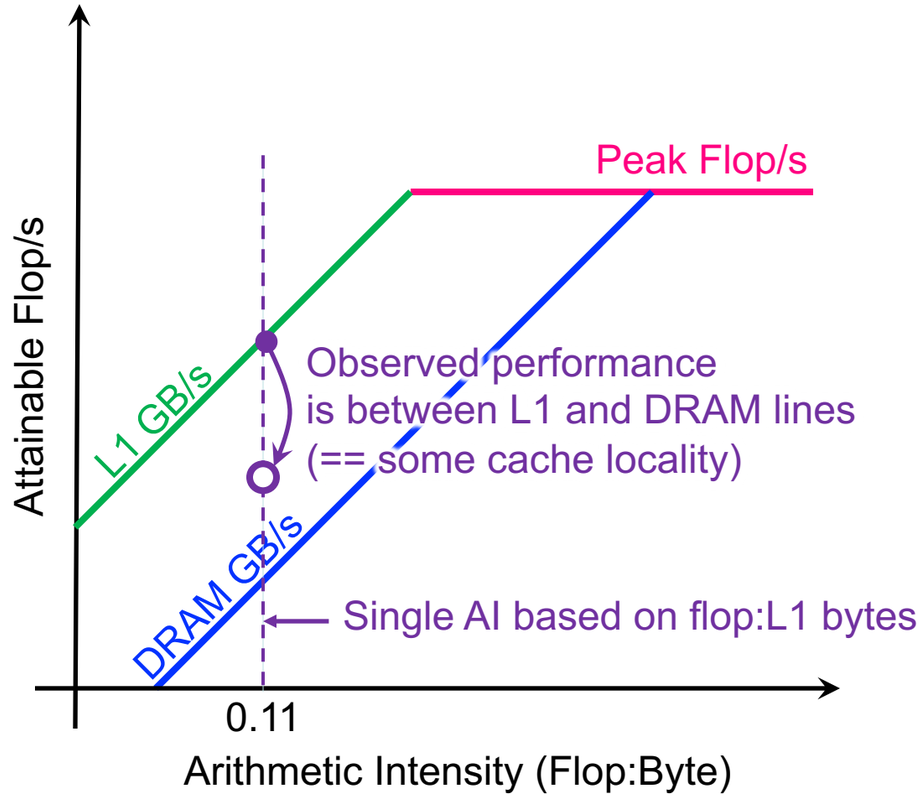


Example: 7-point Stencil (Small Problem)

Hierarchical Roofline

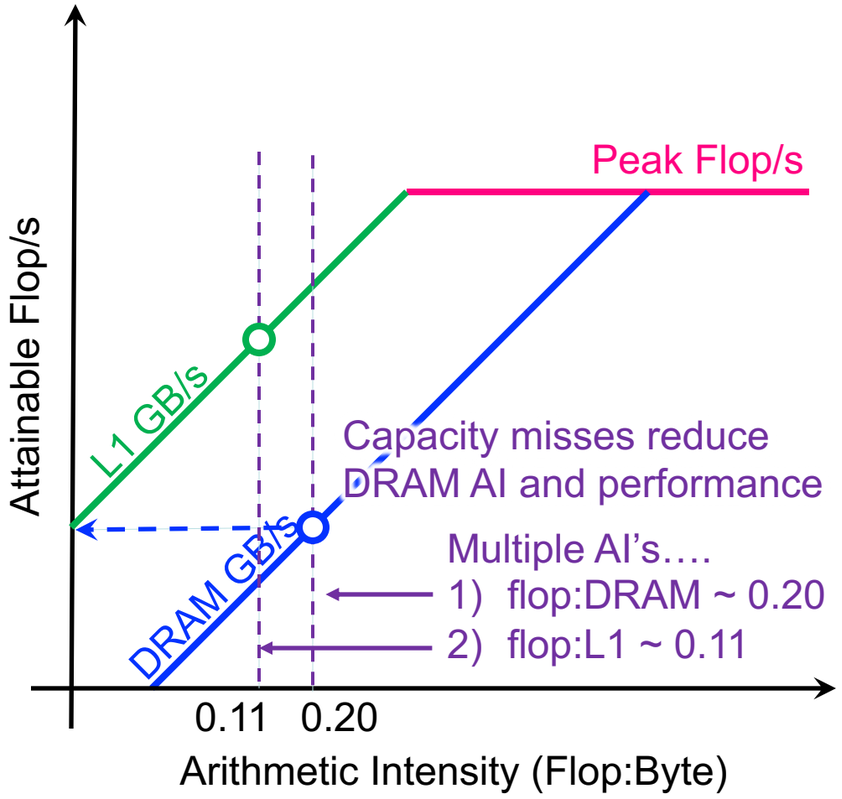


Cache-Aware Roofline

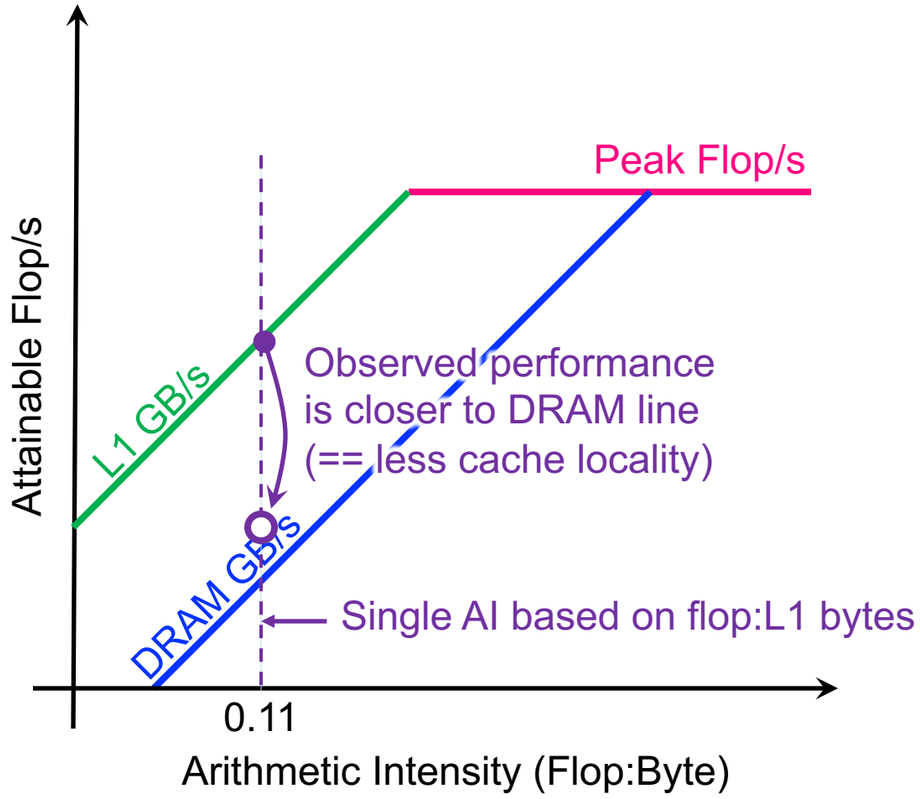


Example: 7-point Stencil (Large Problem)

Hierarchical Roofline

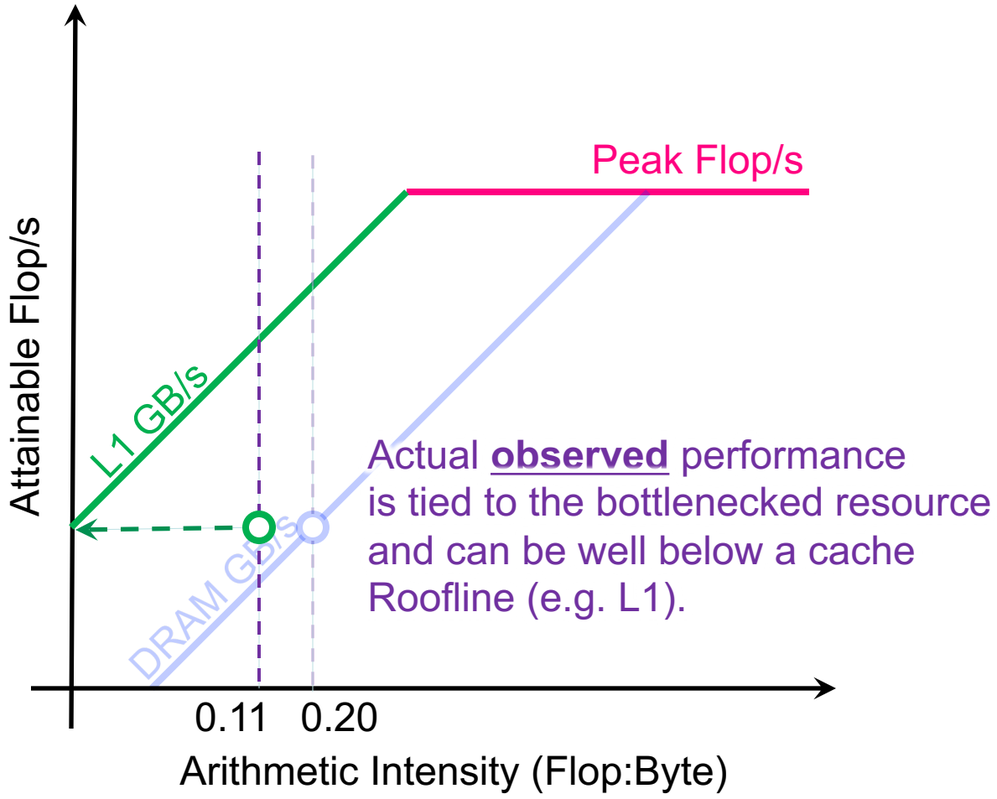


Cache-Aware Roofline

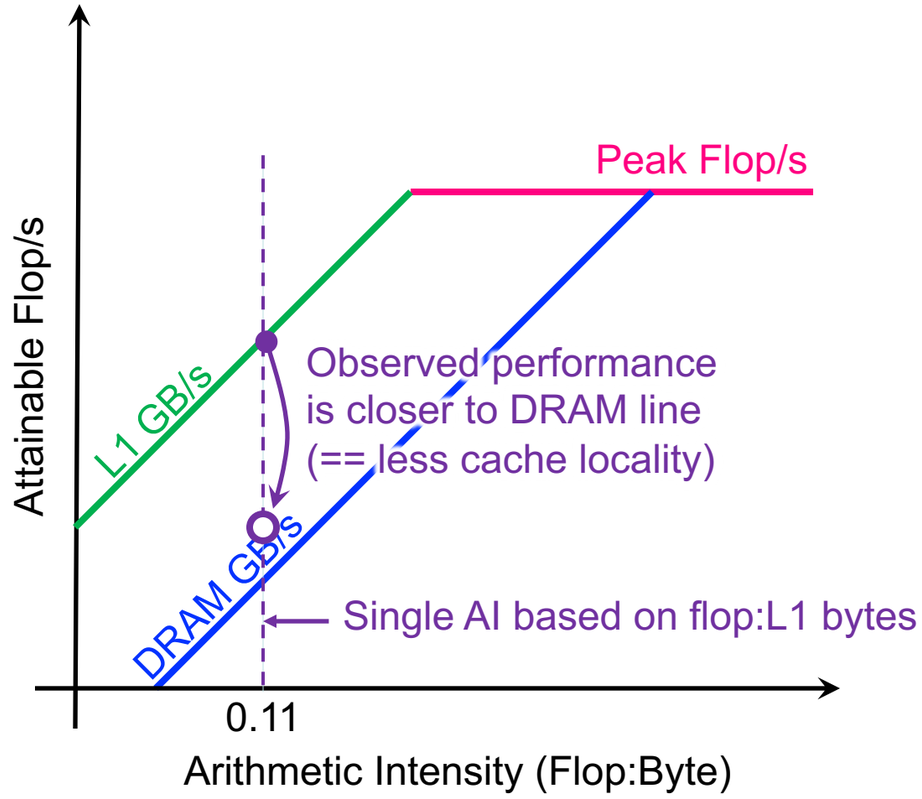


Example: 7-point Stencil (Observed Perf.)

Hierarchical Roofline

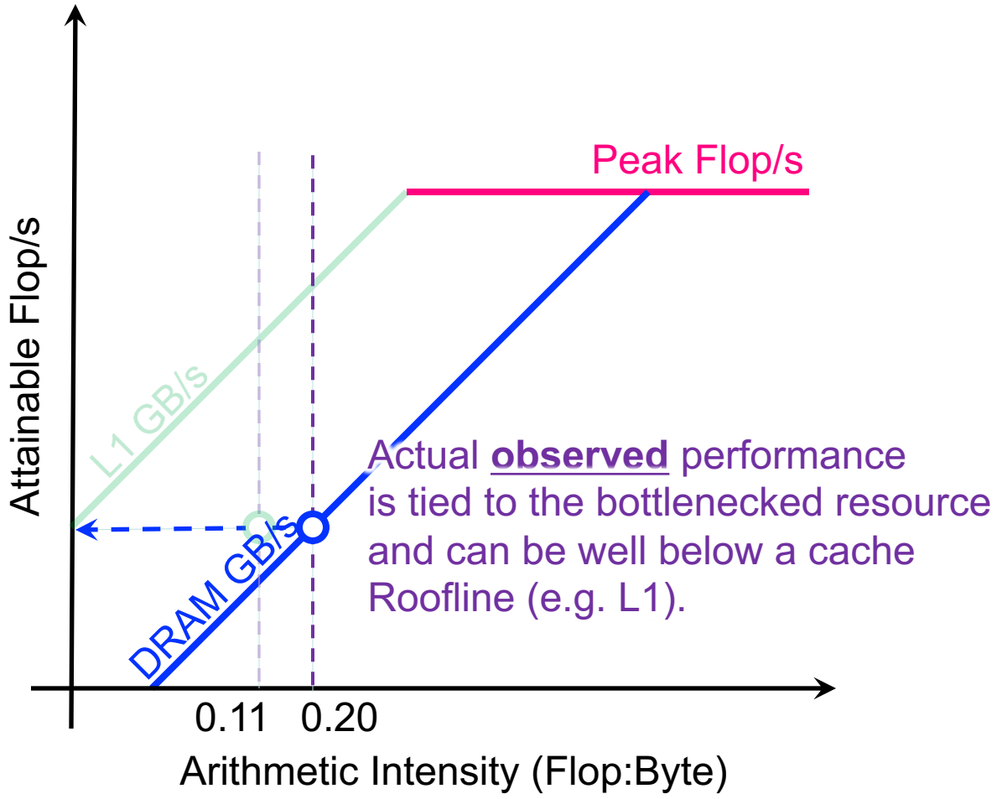


Cache-Aware Roofline

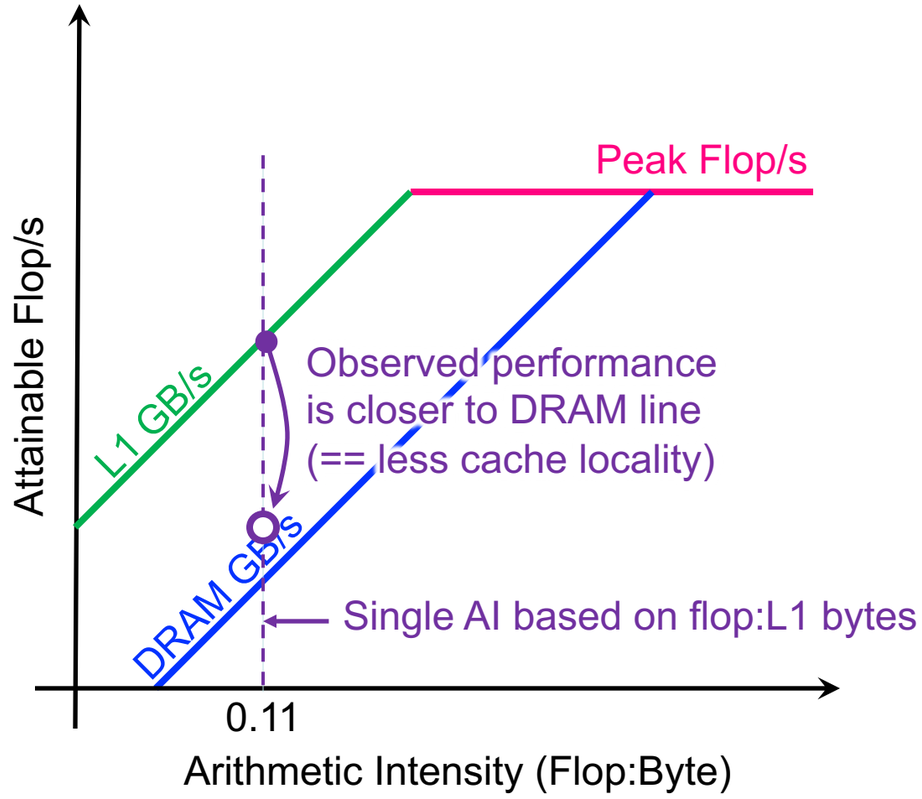


Example: 7-point Stencil (Observed Perf.)

Hierarchical Roofline



Cache-Aware Roofline



Little's Law Redux

Little's Law Redux…

- Recast latency-bandwidth product for OMP/CUDA overheads & flop/s...
- Haswell (Xeon CPU):
  - 100 GB/s, 1.3 Tflop/s, ~1us OMP overhead
  - **Can't hit peak bandwidth on any kernel that moves less than 100KB**
  - **Can't hit peak flops on any kernel that does less than 1M FP operations**
- KNL (Xeon Phi Manycore):
  - 400 GB/s, 2.5 Tflop/s, ~5us OMP overhead
  - **Can't hit peak bandwidth on any kernel that moves less than 2MB**
  - **Can't hit peak flops on any kernel that does less than 13M FP operations**
- Volta GPU:
  - 800 GB/s, 7 Tflop/s, ~20us CUDA launch overhead
  - **Can't hit peak bandwidth on any kernel that moves less than 16MB**
  - **Can't hit peak flops on any kernel that does less than 140M FP operations**