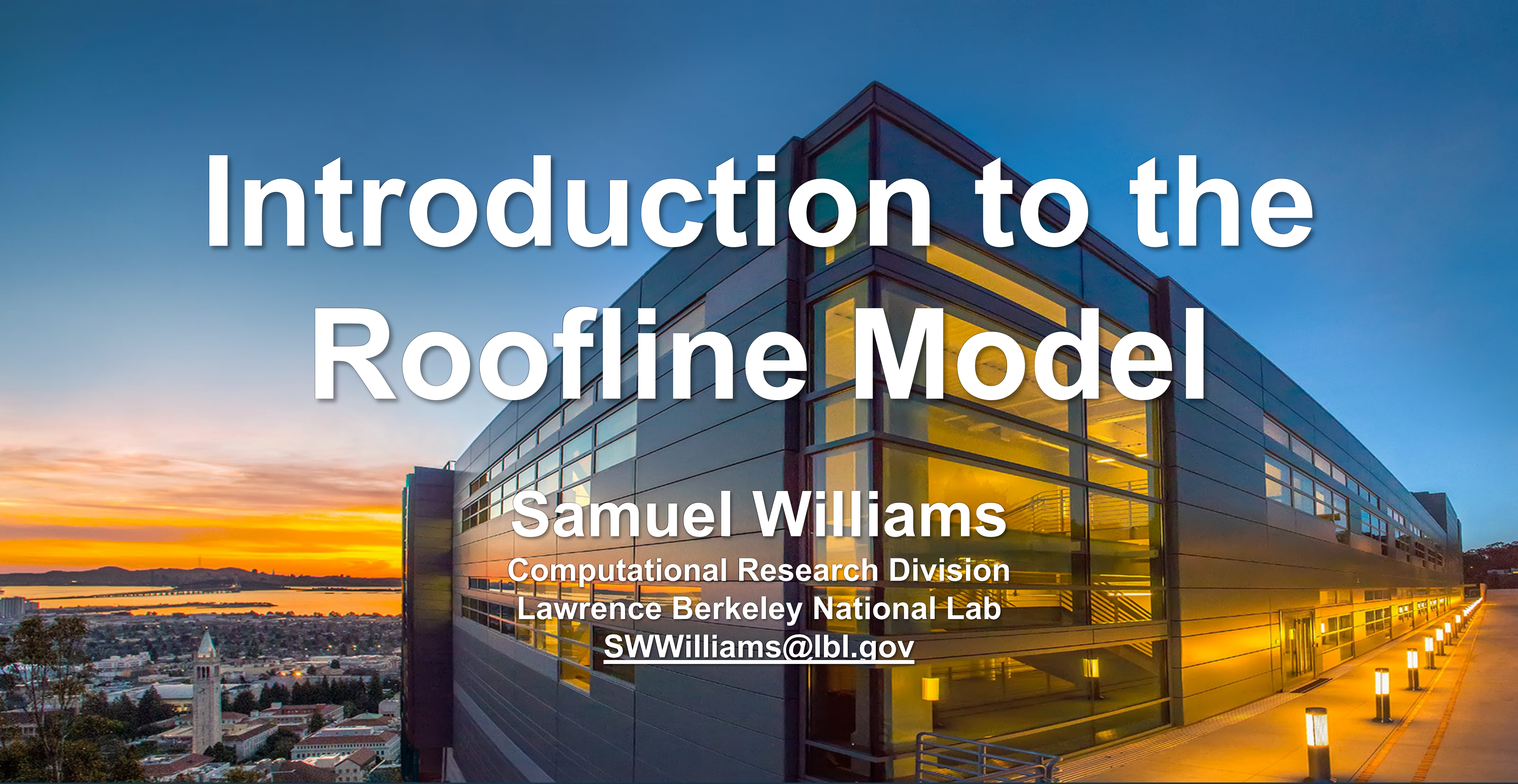


Introduction to the Roofline Model

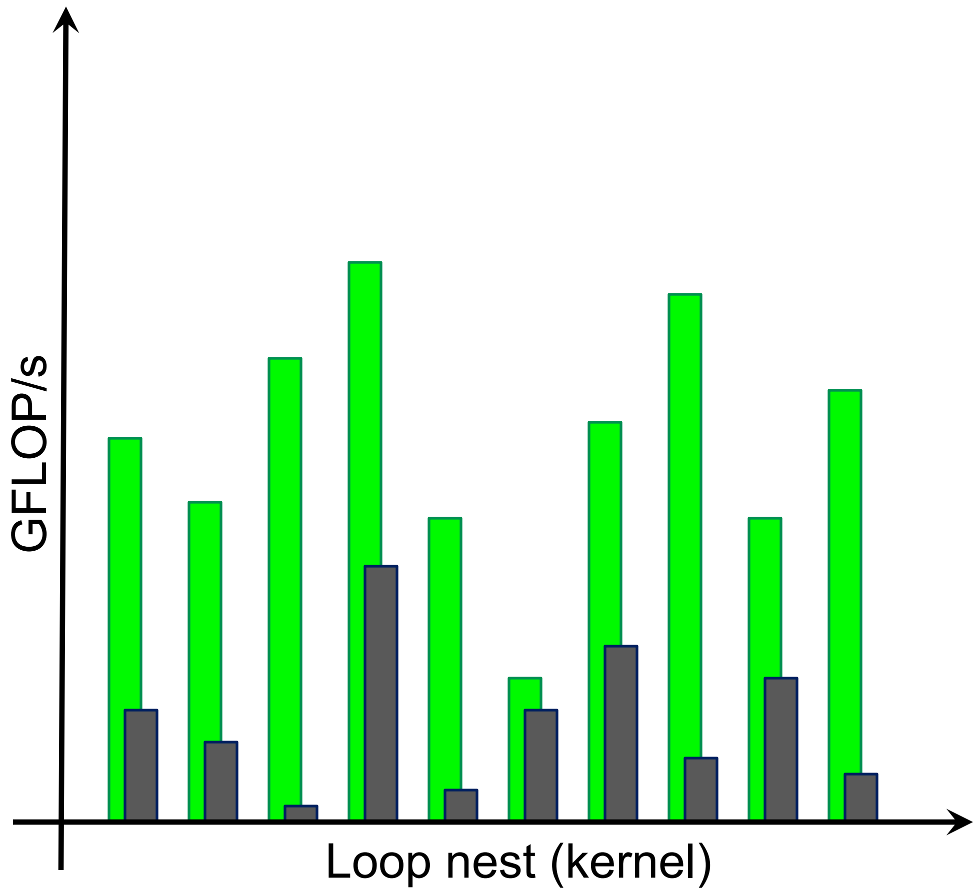
Samuel Williams
Computational Research Division
Lawrence Berkeley National Lab
SWWilliams@lbl.gov

You just spent 6 months porting your application to GPUs

Are you done?

What is “Good” Performance?

- Imagine profiling the mix of loop nests in an application when running on the GPU
  - GFLOP/s alone may not be particularly insightful
  - speedup relative to a Xeon may seem random



What is “Good” Performance?

- Two fundamental aspects to “Good” performance…

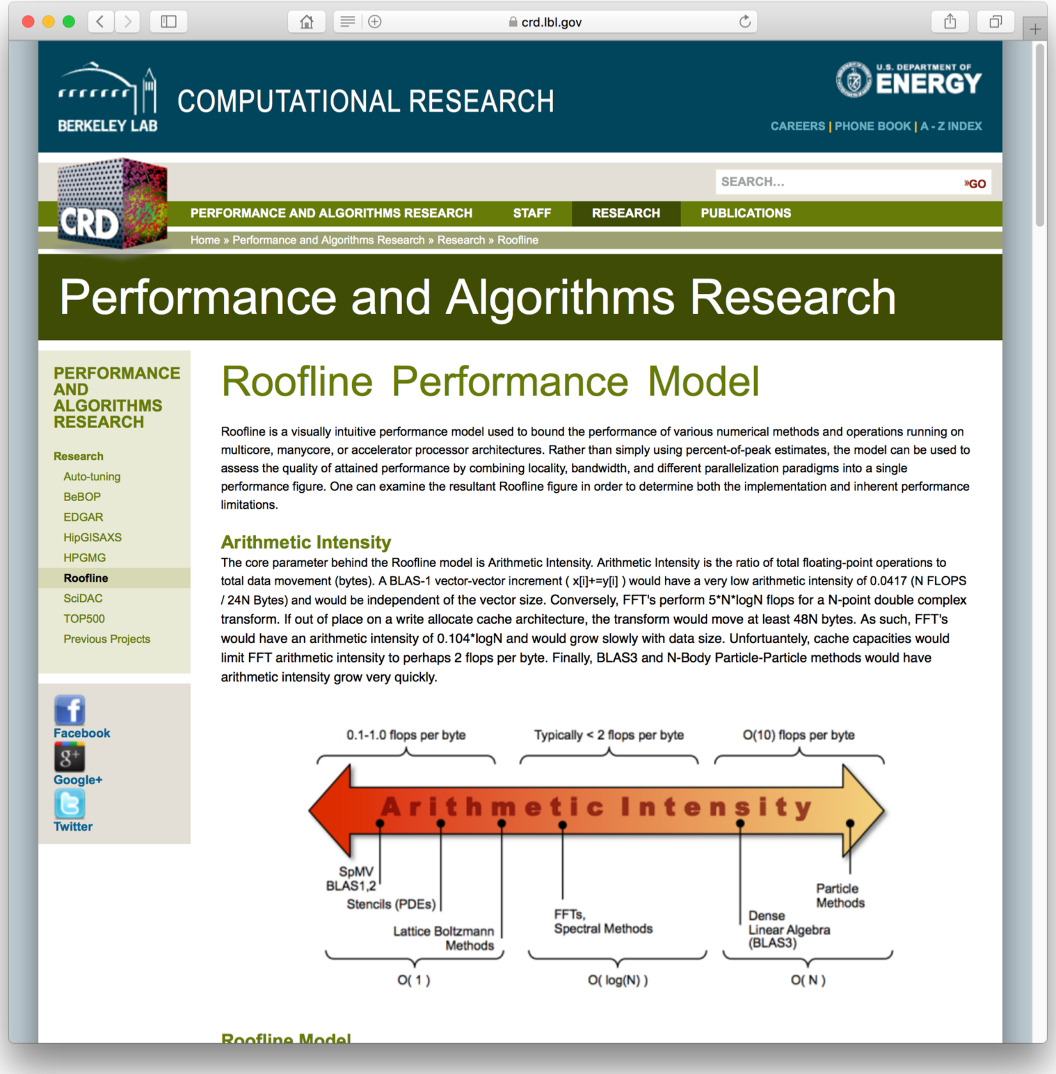
1. Operating in the throughput-limited regime
   not sensitive to Amdahl effects, D2H/H2D transfers, launch overheads, etc…

2. making good use of the GPU’s **compute** and/or **bandwidth** capabilities

➢ **Ultimately, we need a quantitative model rather than qualitative statements like “good”**

Roofline Model

- **Roofline Model** is a throughput-oriented performance model
- Tracks rates not times
- Independent of ISA and architecture
- applies to CPUs, GPUs, Google TPUs[1], FPGAs, etc…
- Helps quantify **Good Performance**



Performance and Algorithms Research

Roofline Performance Model

Roofline is a visually intuitive performance model used to bound the performance of various numerical methods and operations running on multicore, manycore, or accelerator processor architectures. Rather than simply using percent-of-peak estimates, the model can be used to assess the quality of attained performance by combining locality, bandwidth, and different parallelization paradigms into a single performance figure. One can examine the resultant Roofline figure in order to determine both the implementation and inherent performance limitations.

Arithmetic Intensity

The core parameter behind the Roofline model is Arithmetic Intensity. Arithmetic Intensity is the ratio of total floating-point operations to total data movement (bytes). A BLAS-1 vector-vector increment (x[i]+=y[i]) would have a very low arithmetic intensity of 0.0417 (N FLOPS / 24N Bytes) and would be independent of the vector size. Conversely, FFT's perform 5*N*logN flops for a N-point double complex transform. If out of place on a write allocate cache architecture, the transform would move at least 48N bytes. As such, FFT's would have an arithmetic intensity of 0.104*logN and would grow slowly with data size. Unfortuantely, cache capacities would limit FFT arithmetic intensity to perhaps 2 flops per byte. Finally, BLAS3 and N-Body Particle-Particle methods would have arithmetic intensity grow very quickly.

Roofline Model

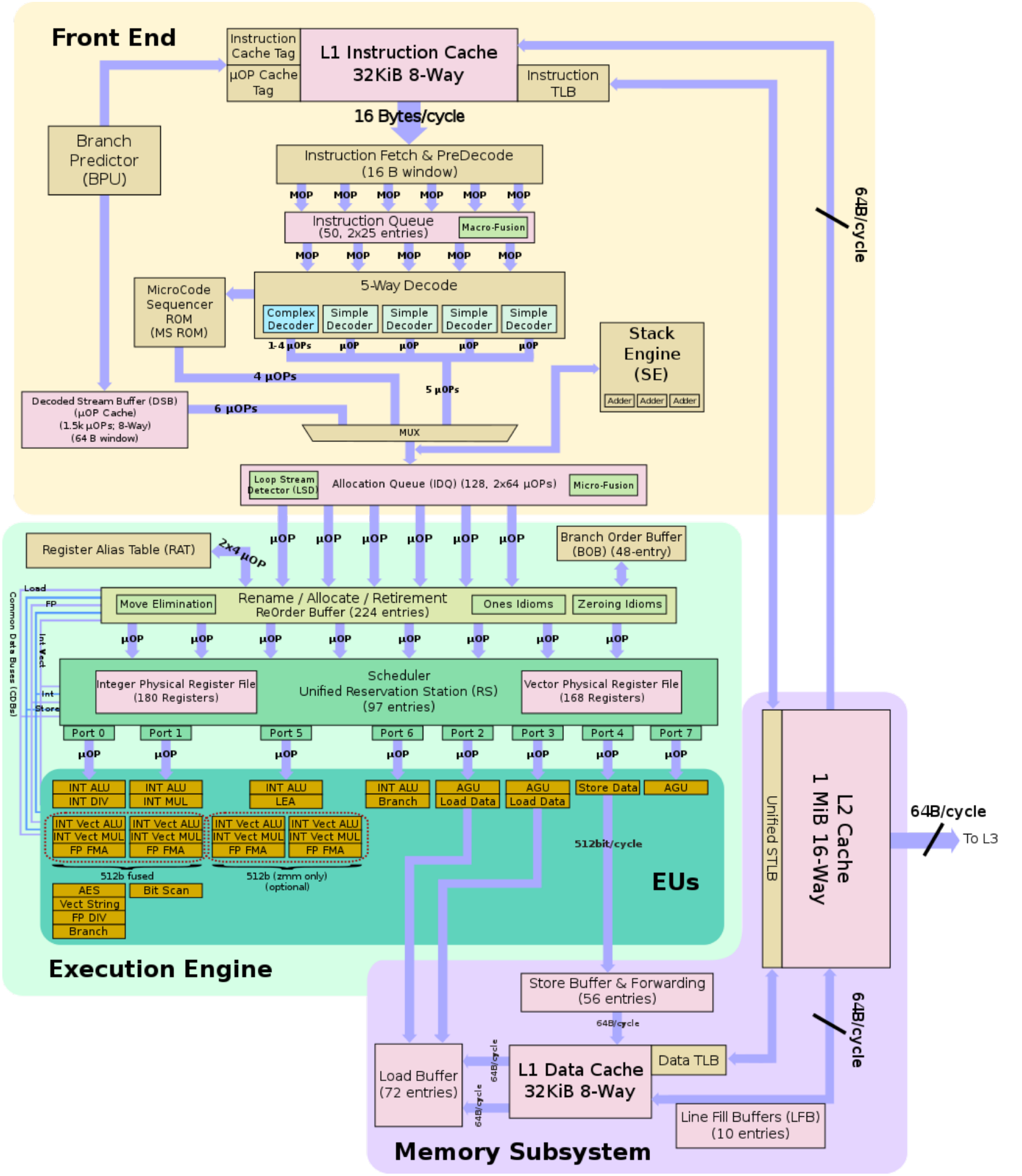
https://crd.lbl.gov/departments/computer-science/PAR/research/roofline

---

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

Reduced Model

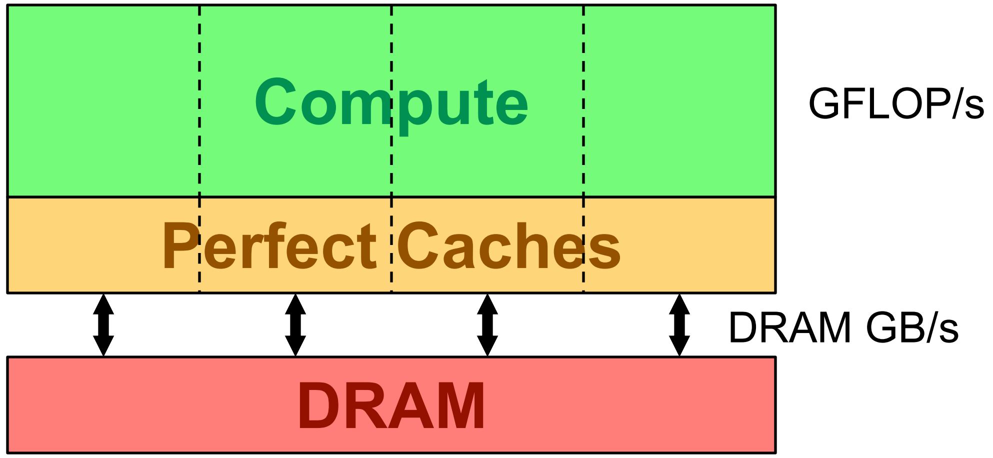
- Superscalar architectures can be complex
- Don’t model / simulate full architecture
- Created simplified processor architecture



https://en.wikichip.org/wiki/intel/microarchitectures/skylake_(server)

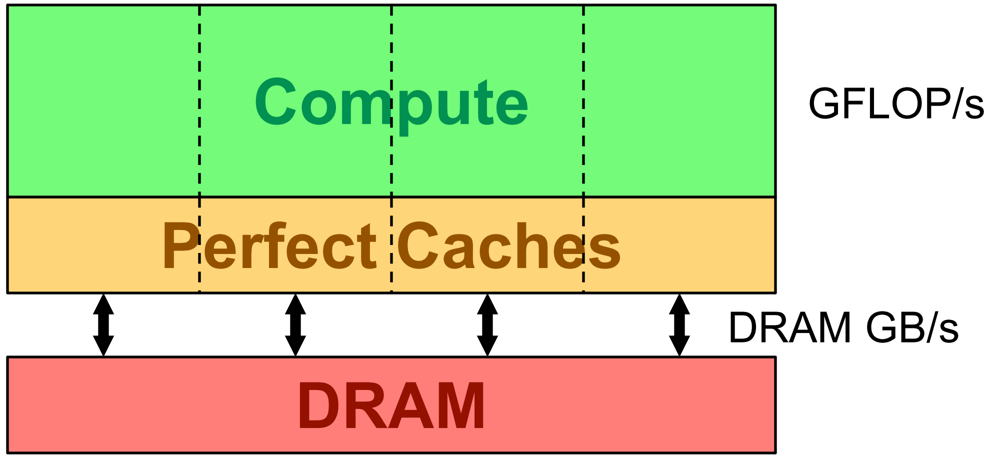
Reduced Model

- Superscalar architectures can be complex
- Don't model / simulate full architecture
- Created simplified processor architecture
- Make assumptions on performance and usage…
  - Cores can attain peak GFLOP/s on local data
  - Cores execute load-balanced SPMD code
  - NoC bisection bandwidth is sufficient
  - There is sufficient cache bandwidth and capacity such that they do not affect performance
  - ➢ **Basis for DRAM Roofline Model**



Data Movement or Compute?

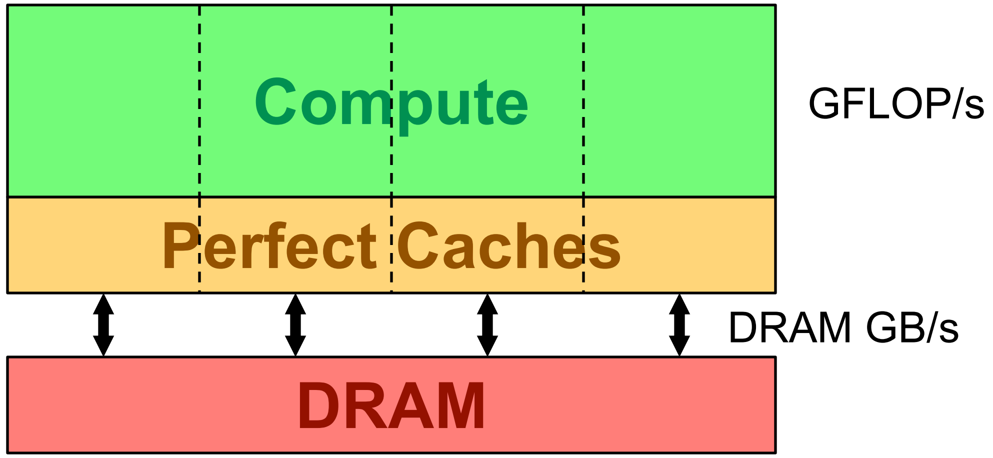
- Which takes longer?
  - Data Movement
  - Compute?



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

Data Movement or Compute?

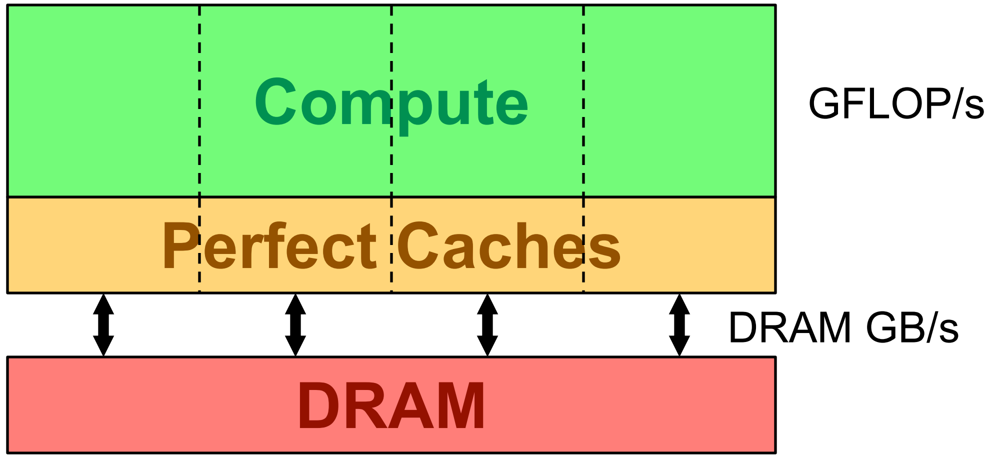
- Which takes longer?
  - Data Movement
  - Compute?
- Is performance limited by compute or data movement?



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

Data Movement or Compute?

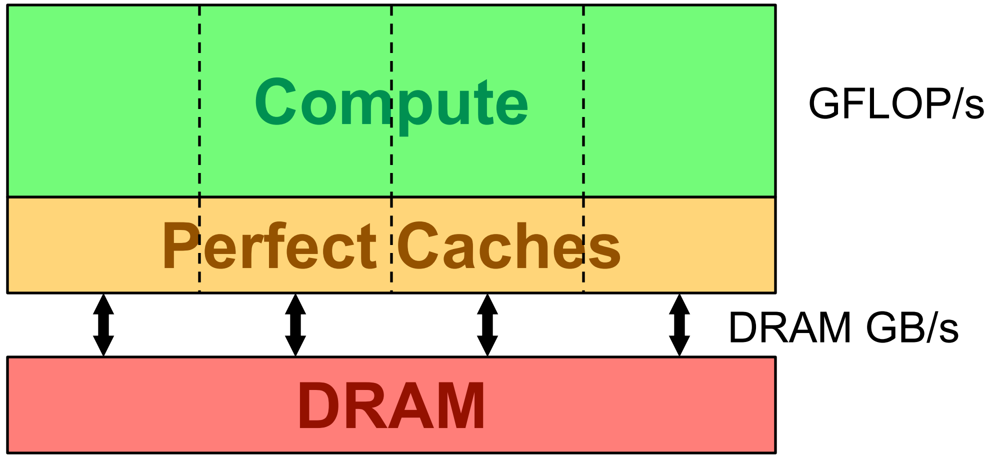
- Which takes longer?
  - Data Movement
  - Compute?
- Is performance limited by compute or data movement?



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

Data Movement or Compute?

- Which takes longer?
    - Data Movement
    - Compute?
- Is performance limited by compute or data movement?



Compute

GFLOP/s

Perfect Caches

DRAM GB/s

DRAM

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

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

Arithmetic Intensity

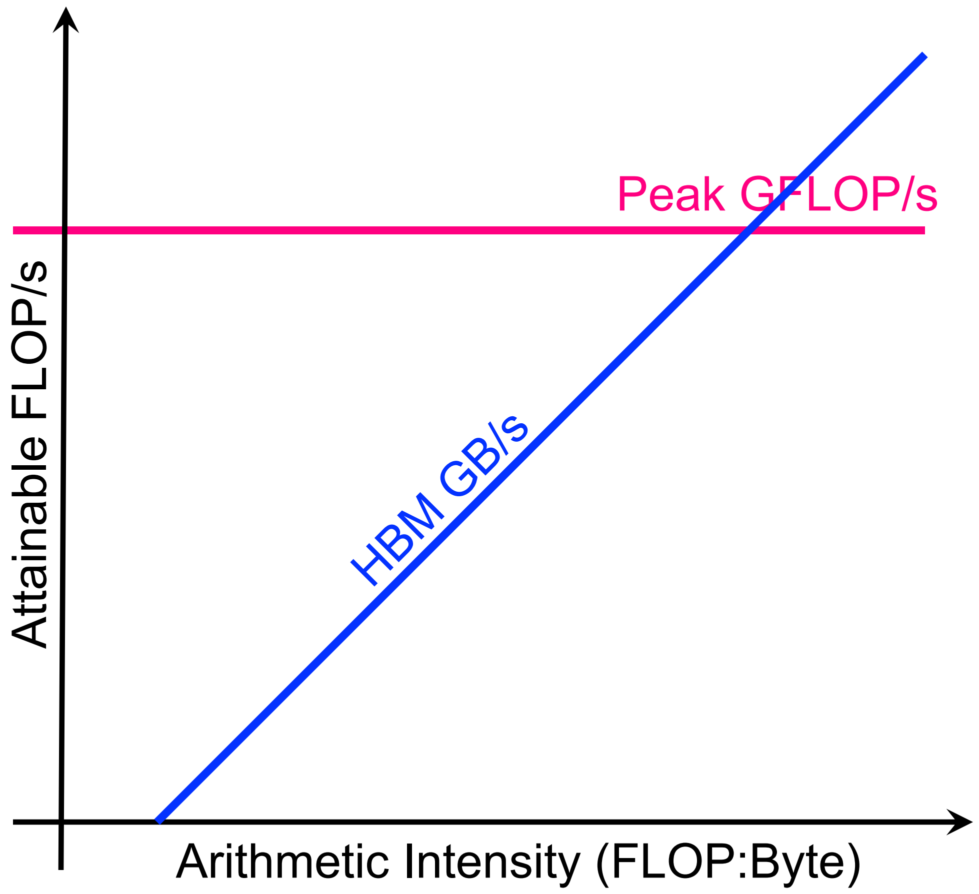
- Measure of data locality (data reuse)
- Ratio of **Total Flops** performed to **Total Bytes** moved
- For the DRAM Roofline…
  - Total Bytes to/from DRAM
  - Includes all cache and prefetcher effects
  - Can be very different from total loads/stores (bytes requested)
  - Equal to ratio of sustained GFLOP/s to sustained GB/s (time cancels)

(DRAM) Roofline Model

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

AI (Arithmetic Intensity) = FLOPs / Bytes (moved to/from DRAM)

- 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…

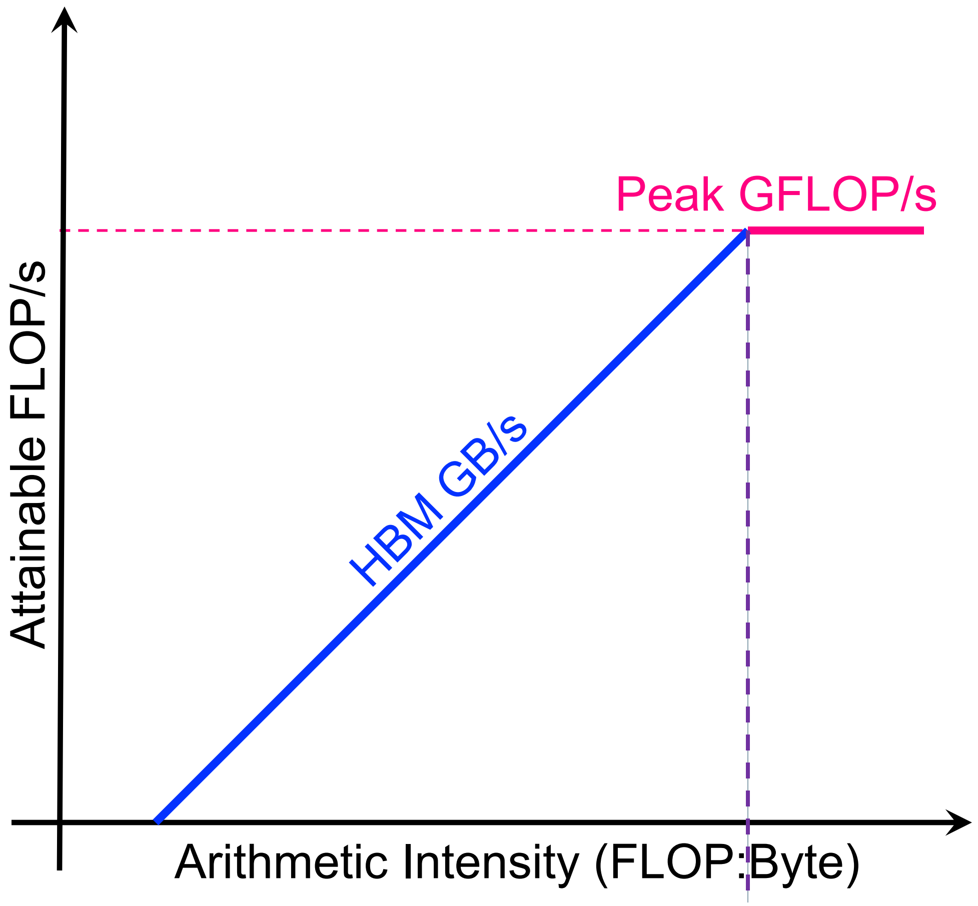


(DRAM) Roofline Model

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

AI (Arithmetic Intensity) = FLOPs / Bytes (moved to/from DRAM)

- 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…



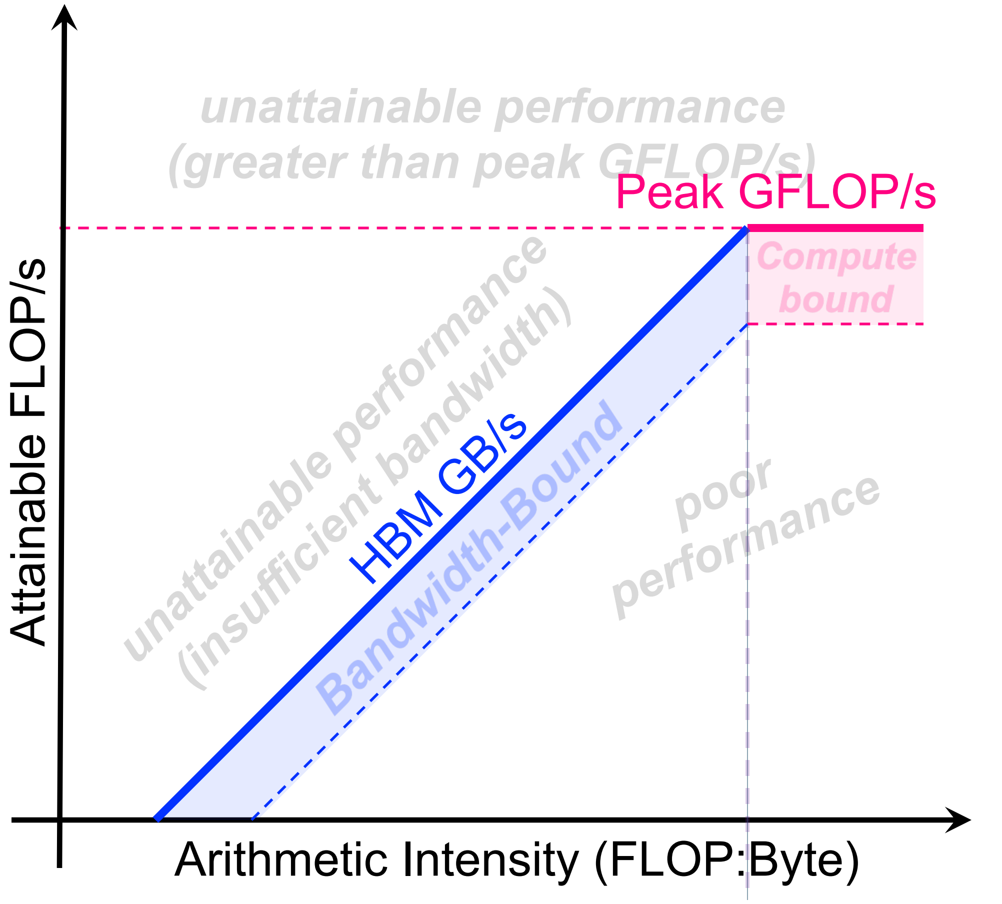
Transition @ AI ==
Peak GFLOP/s / Peak GB/s ==
'Machine Balance'

(DRAM) Roofline Model

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

AI (Arithmetic Intensity) = FLOPs / Bytes (moved to/from DRAM)

- Roofline tessellates this 2D view of performance into 5 regions…



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

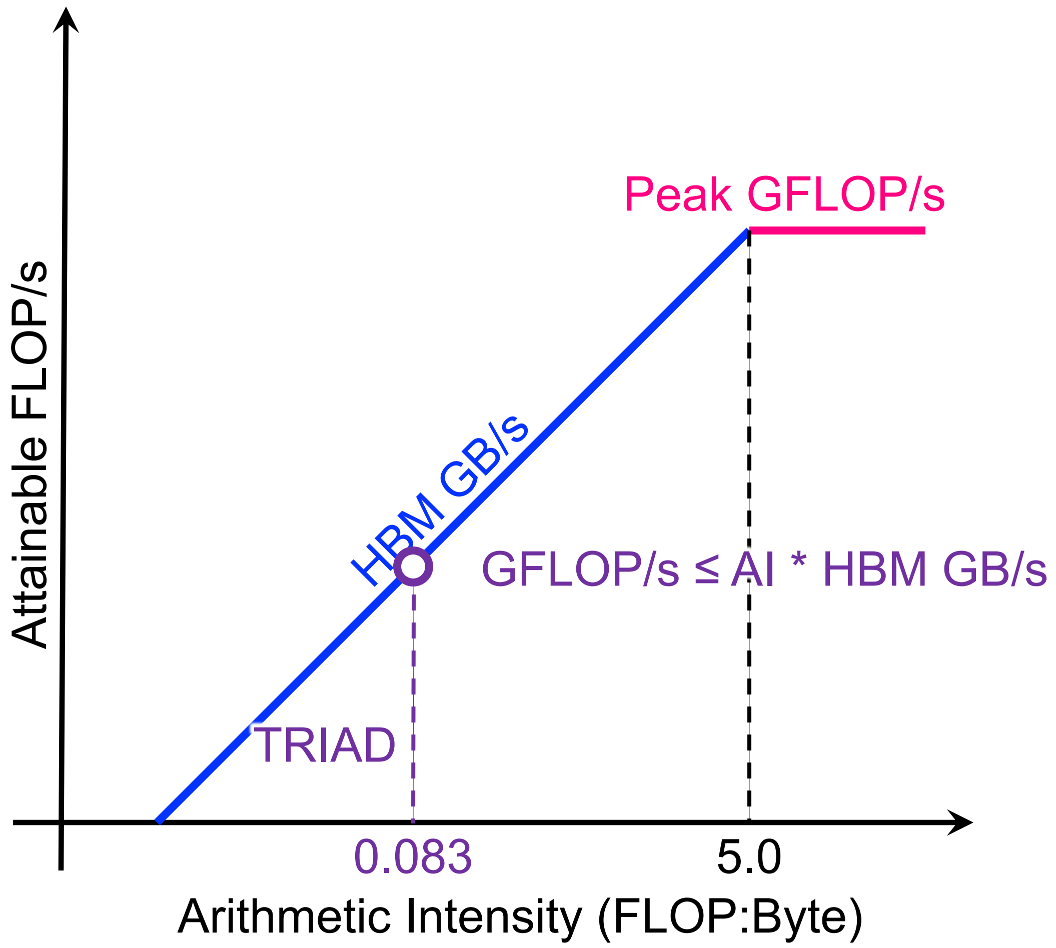
Roofline Example #1

- 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…

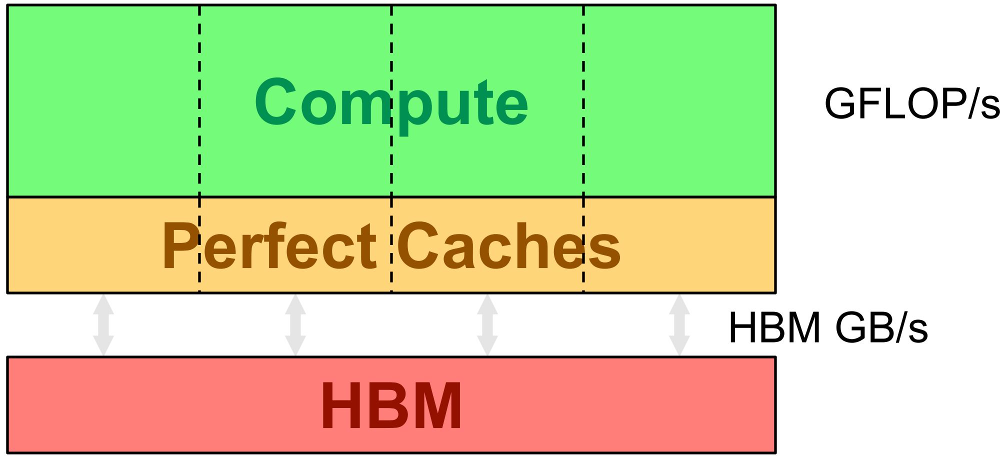
```
#pragma omp parallel for
for(i=0;i<N;i++){
  Z[i] = X[i] + alpha*Y[i];
}
```

  - 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…

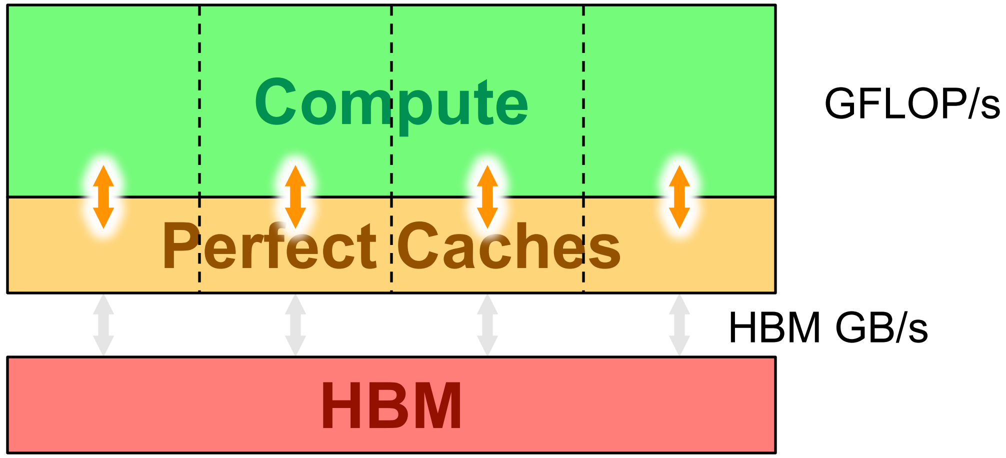


```
#pragma omp parallel for
for(k=1;k<dim+1;k++){
for(j=1;j<dim+1;j++){
for(i=1;i<dim+1;i++){
  new[k][j][i] = -6.0*old[k  ][j  ][i  ]
                    + old[k  ][j  ][i-1]
                    + old[k  ][j  ][i+1]
                    + old[k  ][j-1][i  ]
                    + old[k  ][j+1][i  ]
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

Roofline Example #2

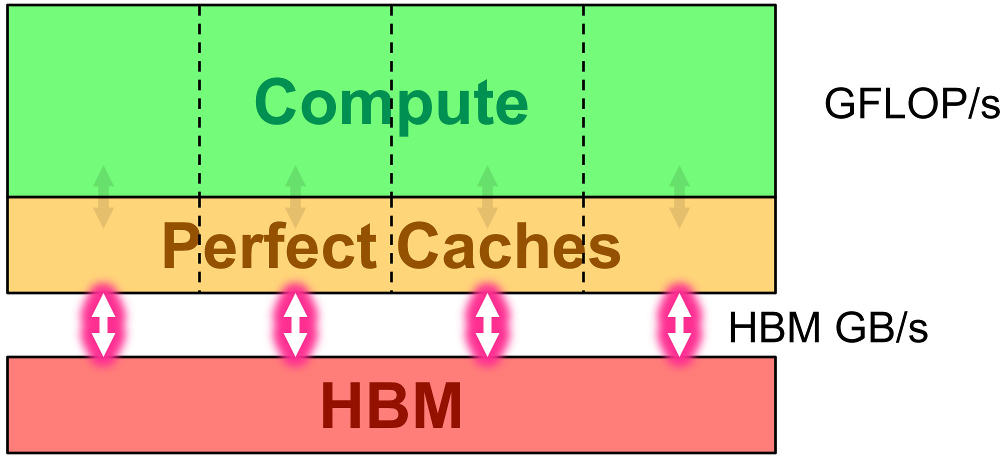
- Conversely, 7-point constant coefficient stencil…
  - 7 FLOPs
  - 8 memory references (7 reads, 1 store) per point
  - **AI = 7 / (8*8) = 0.11 FLOPs per byte (measured at the L1)**

```
#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  ]
}}}
```



Roofline Example #2

- Conversely, 7-point constant coefficient stencil…
    - 7 FLOPs
    - 8 memory references (7 reads, 1 store) per point
    - **Ideally, cache will filter all but 1 read and 1 write per point**

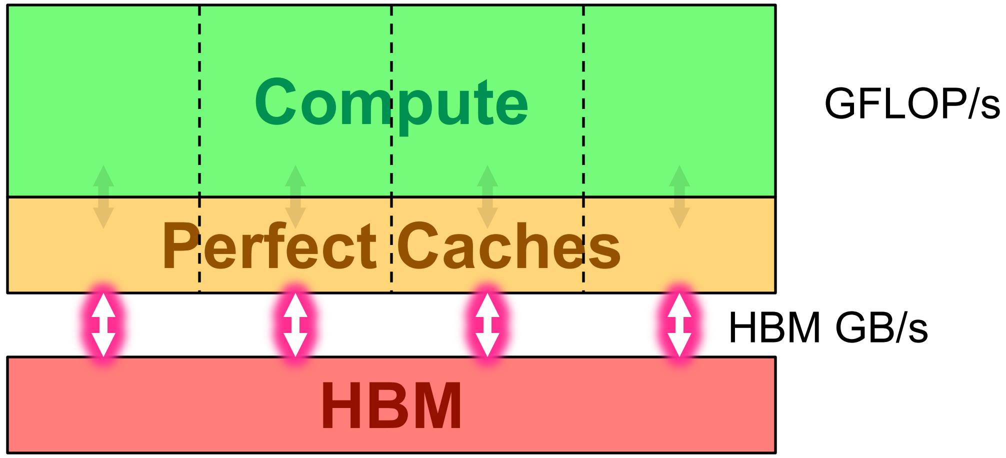


```
#pragma omp parallel for
for(k=1;k<dim+1;k++){
for(j=1;j<dim+1;j++){
for(i=1;i<dim+1;i++){
  new[k][j][i] = -6.0*old[k  ][j  ][i  ]
                    + old[k  ][j  ][i-1]
                    + old[k  ][j  ][i+1]
                    + old[k  ][j-1][i  ]
                    + old[k  ][j+1][i  ]
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

Roofline Example #2

- Conversely, 7-point constant coefficient stencil…
  - 7 FLOPs
  - 8 memory references (7 reads, 1 store) per point
  - Ideally, cache will filter all but 1 read and 1 write per point
  - **7 / (8+8) = 0.44 FLOPs per byte (DRAM)**

```
#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  ];
}}}
```

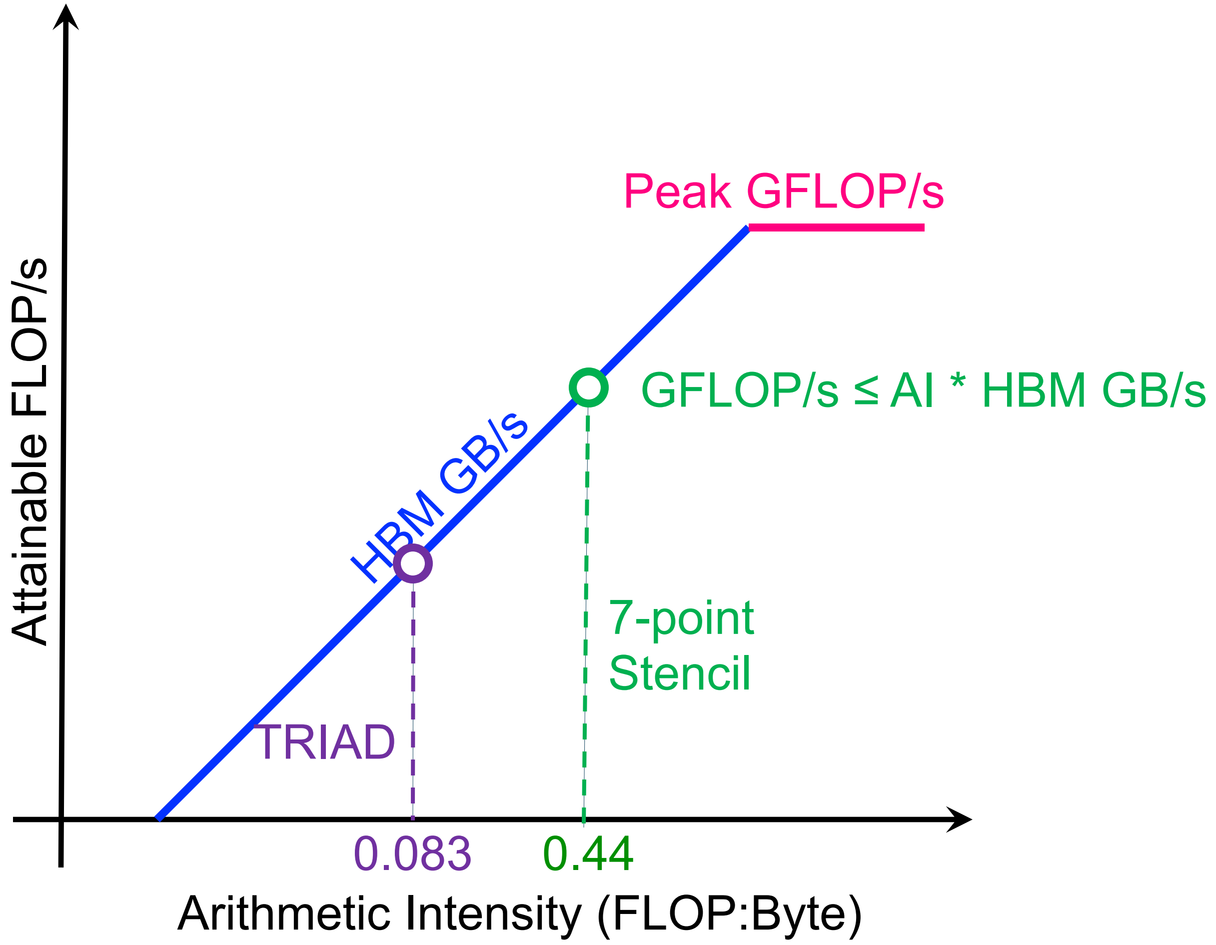


Roofline Example #2

- Conversely, 7-point constant coefficient stencil…
  - 7 FLOPs
  - 8 memory references (7 reads, 1 store) per point
  - Ideally, cache will filter all but 1 read and 1 write per point
  - ➢ **7 / (8+8) = 0.44 FLOPs per byte (DRAM)**

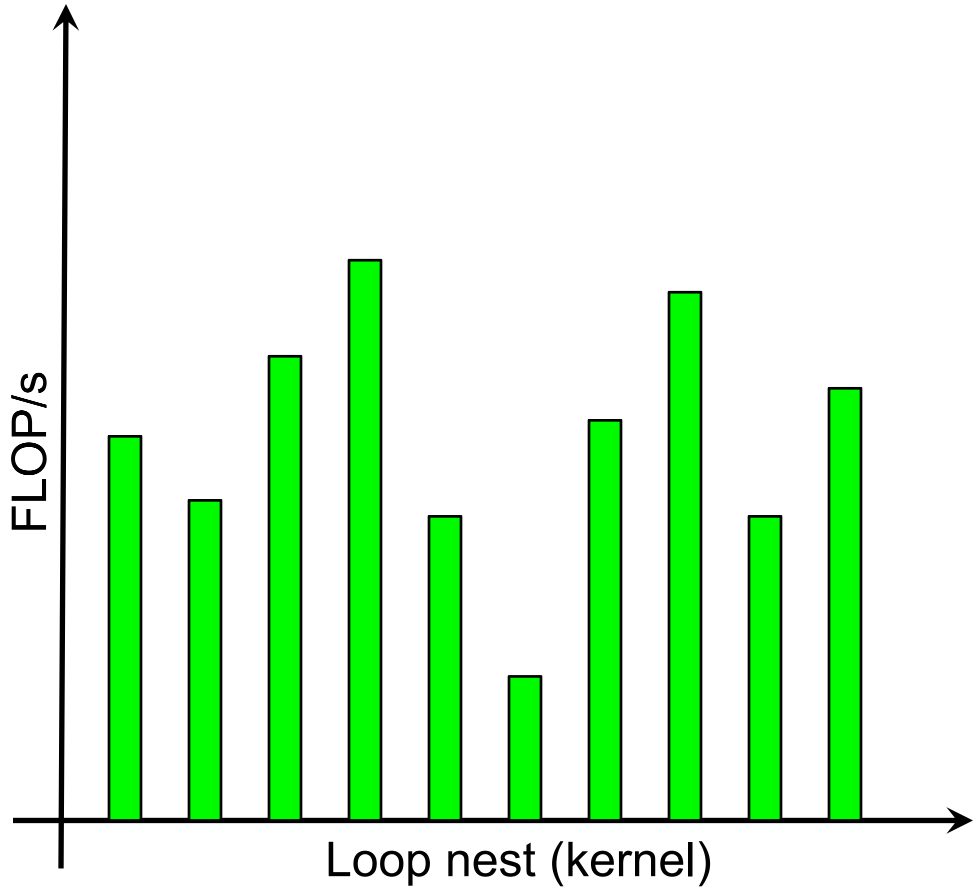
    == memory bound, but 5x the FLOP rate as TRIAD

```
#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  ];
}}}
```



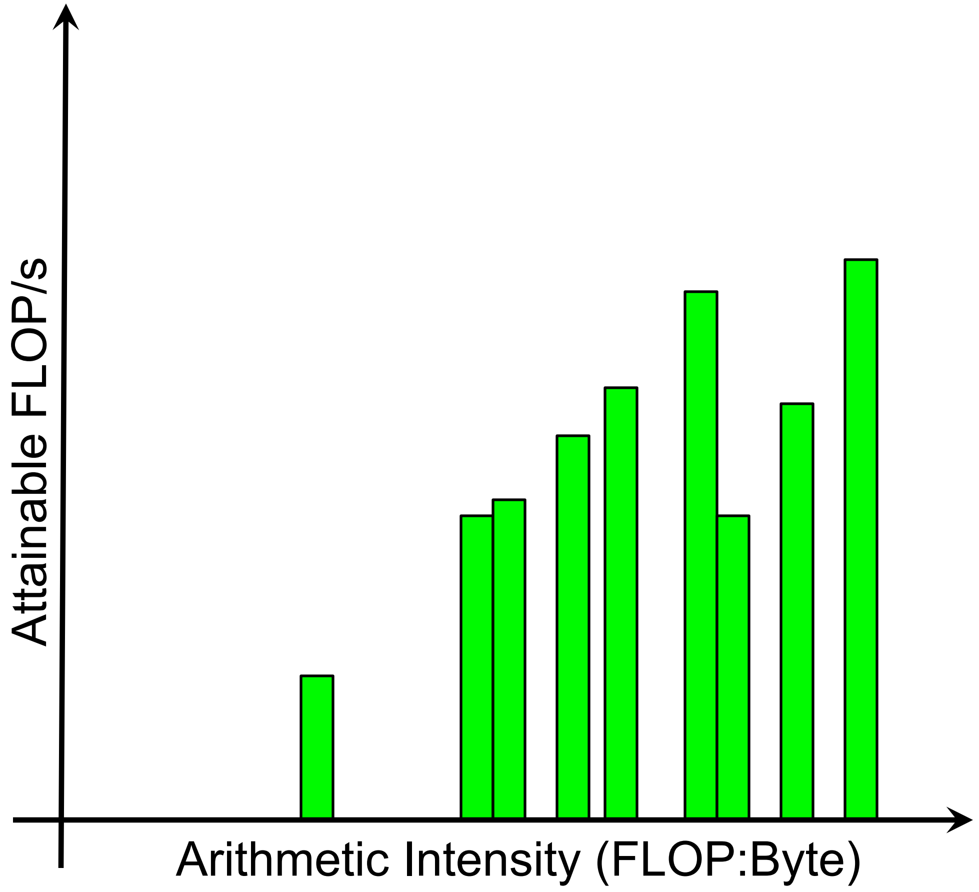
What is "Good" Performance?

- Think back to our mix of loop nests…



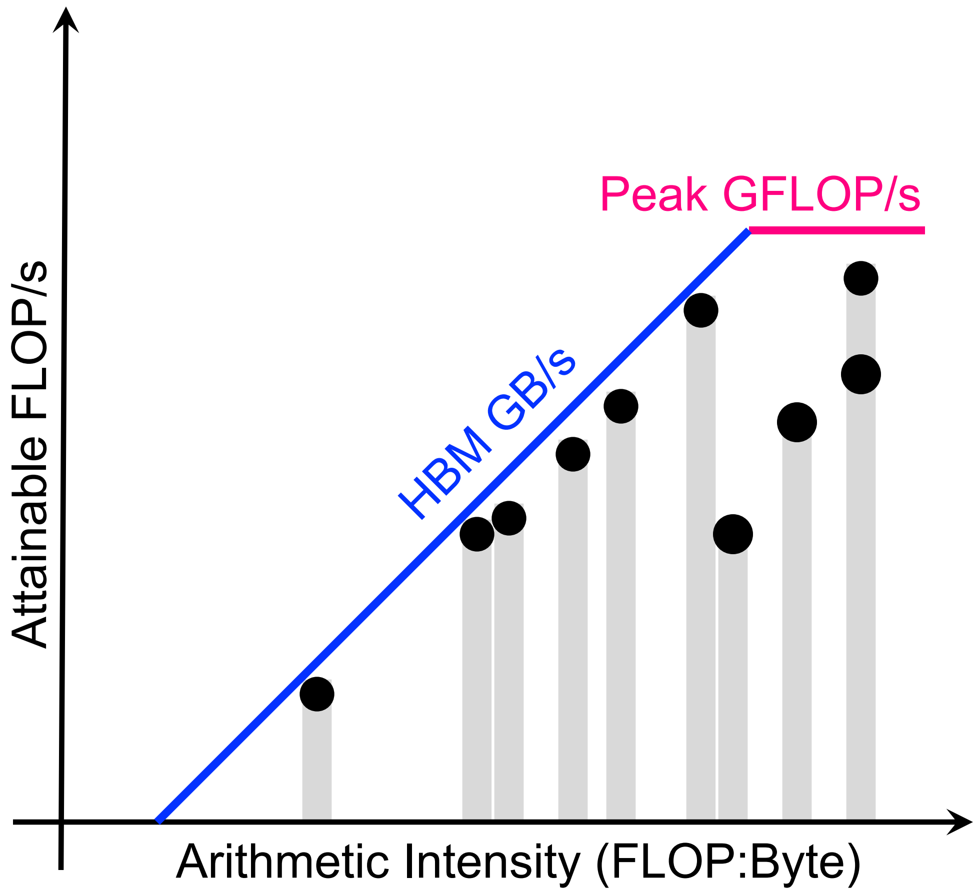
What is “Good” Performance?

- We can sort kernels by arithmetic intensity…



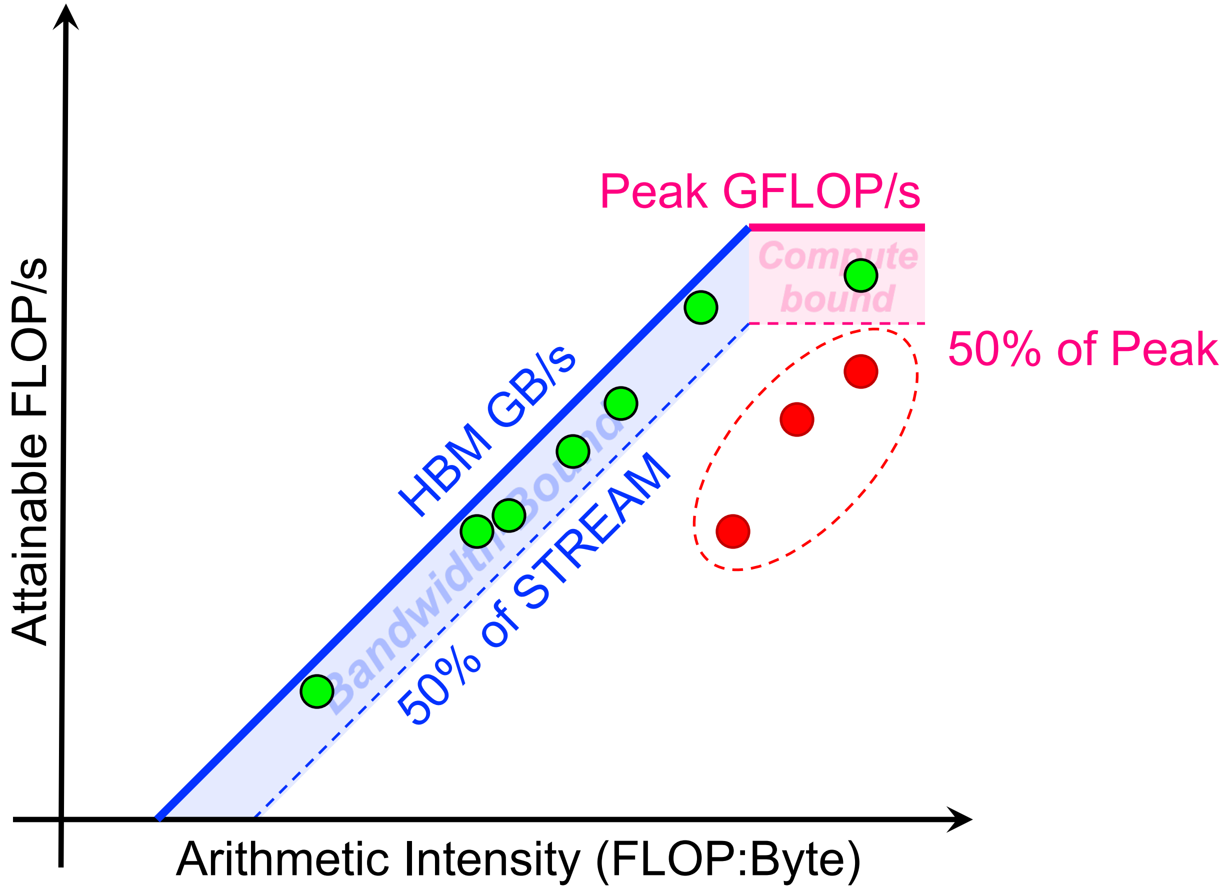
What is “Good” Performance?

- We can sort kernels by arithmetic intensity…
- … and compare performance relative to machine capabilities



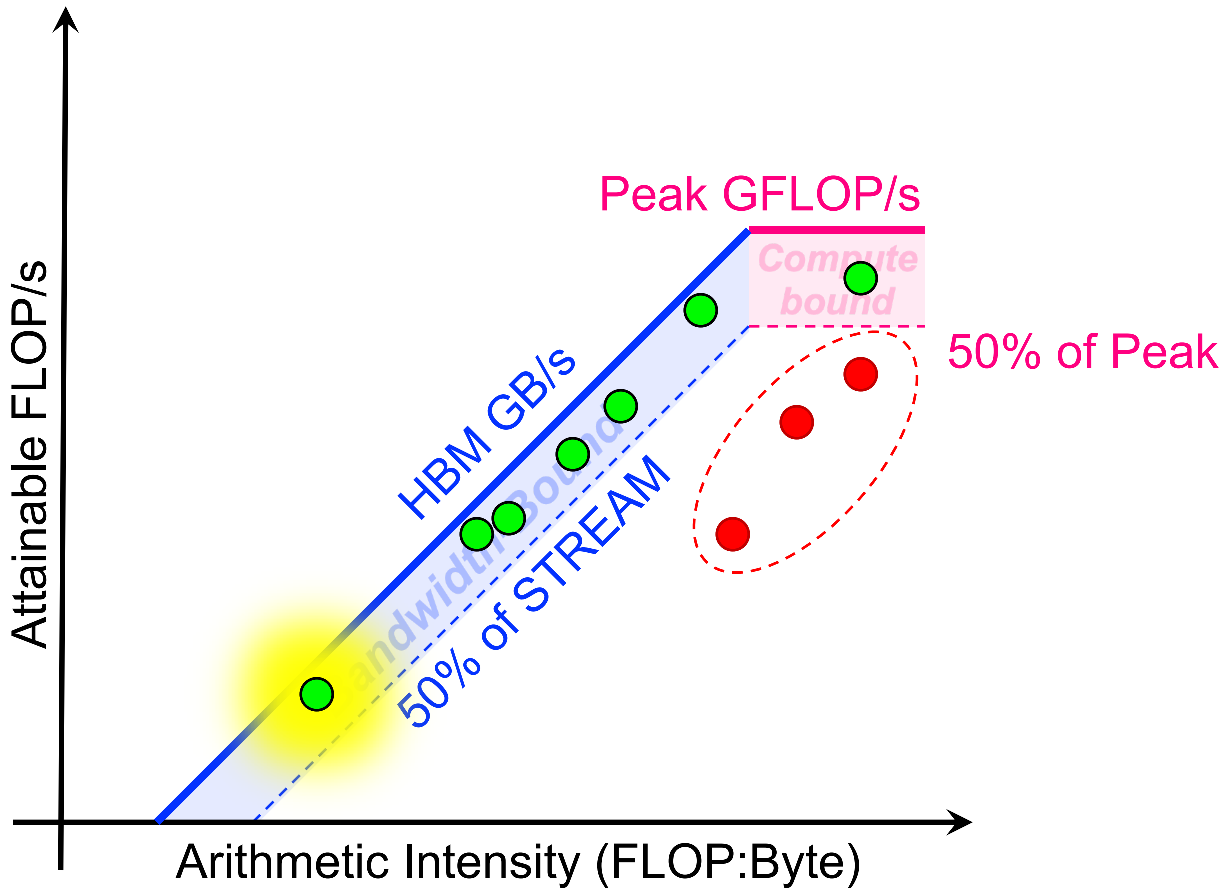
What is “Good” Performance?

- Kernels near the roofline are making **good use** of computational resources



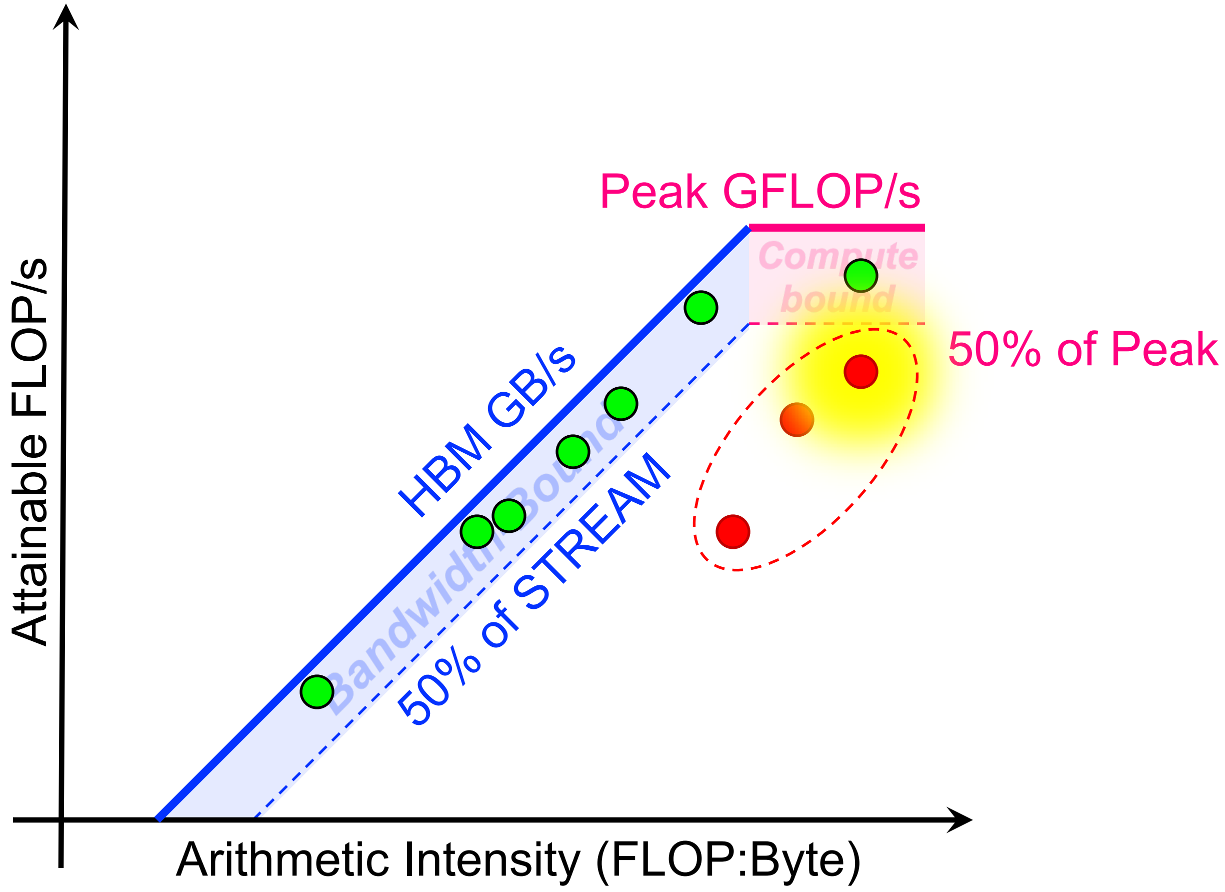
What is “Good” Performance?

- Kernels near the roofline are making **good use** of computational resources
  - kernels can have **low performance** (GFLOP/s), but make **good use** (%STREAM) of a machine



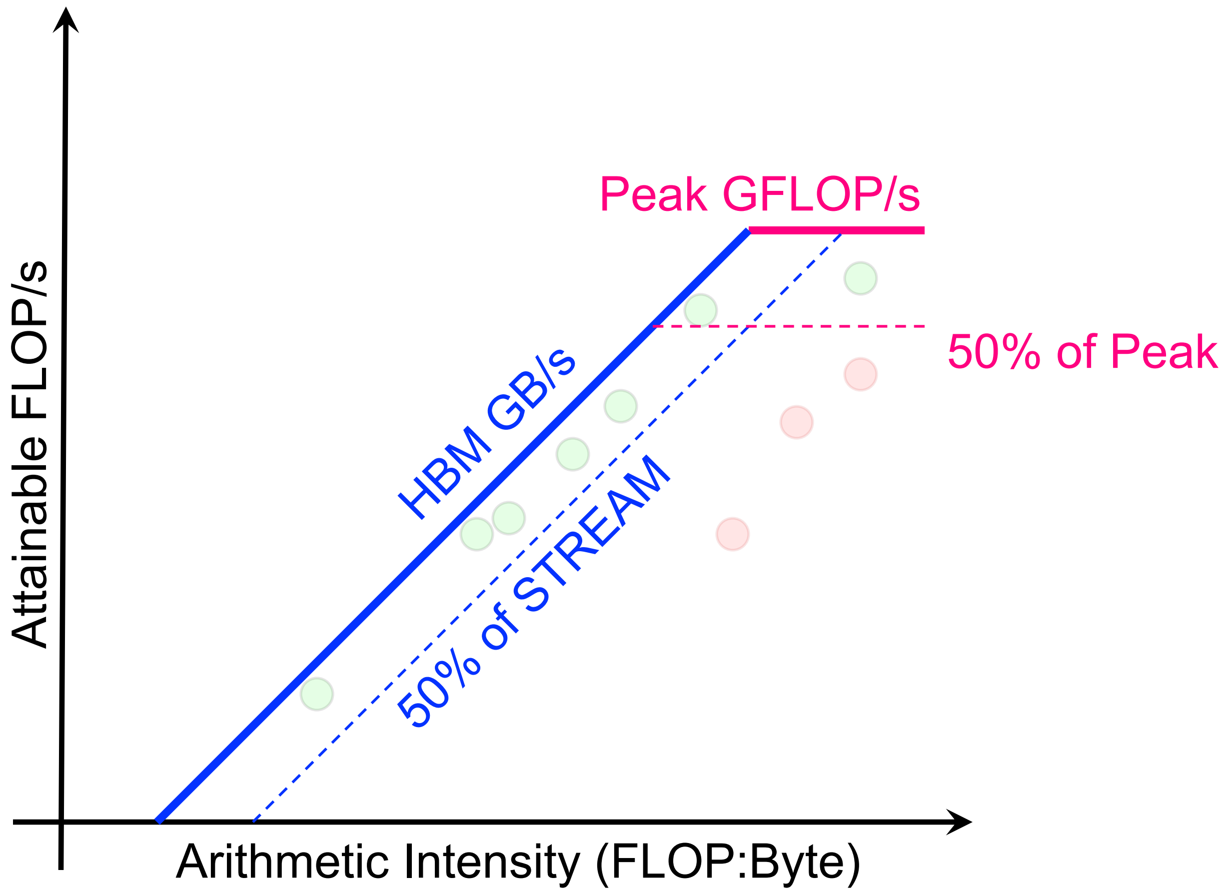
What is “Good” Performance?

- Kernels near the roofline are making **good use** of computational resources
  - kernels can have **low performance** (GFLOP/s), but make **good use** (%STREAM) of a machine
  - kernels can have **high performance** (GFLOP/s), but still make **poor use** of a machine (%peak)



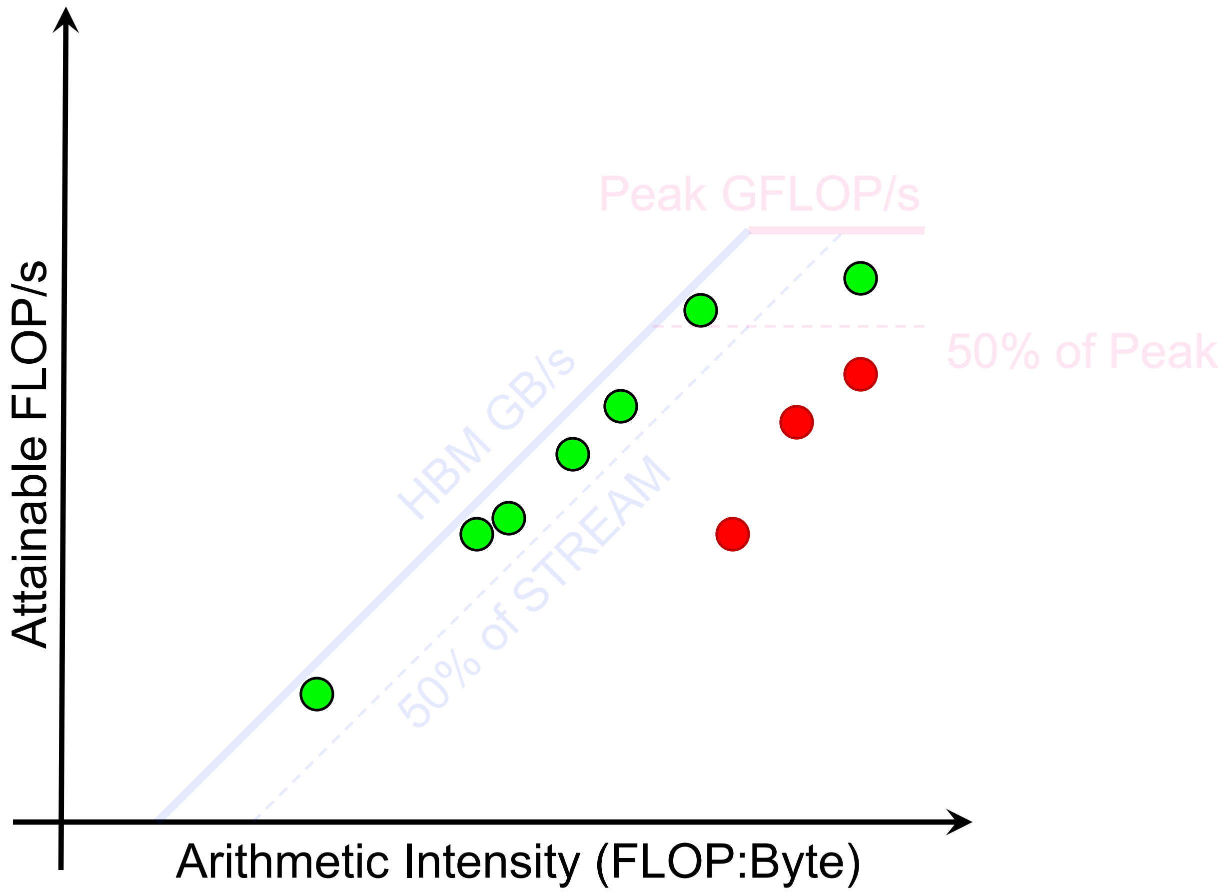
Roofline is made of two components

- Machine Model
  - Lines defined by peak GB/s and GF/s (**Benchmarking**)
  - Unique to each architecture
  - Common to all apps on that architecture



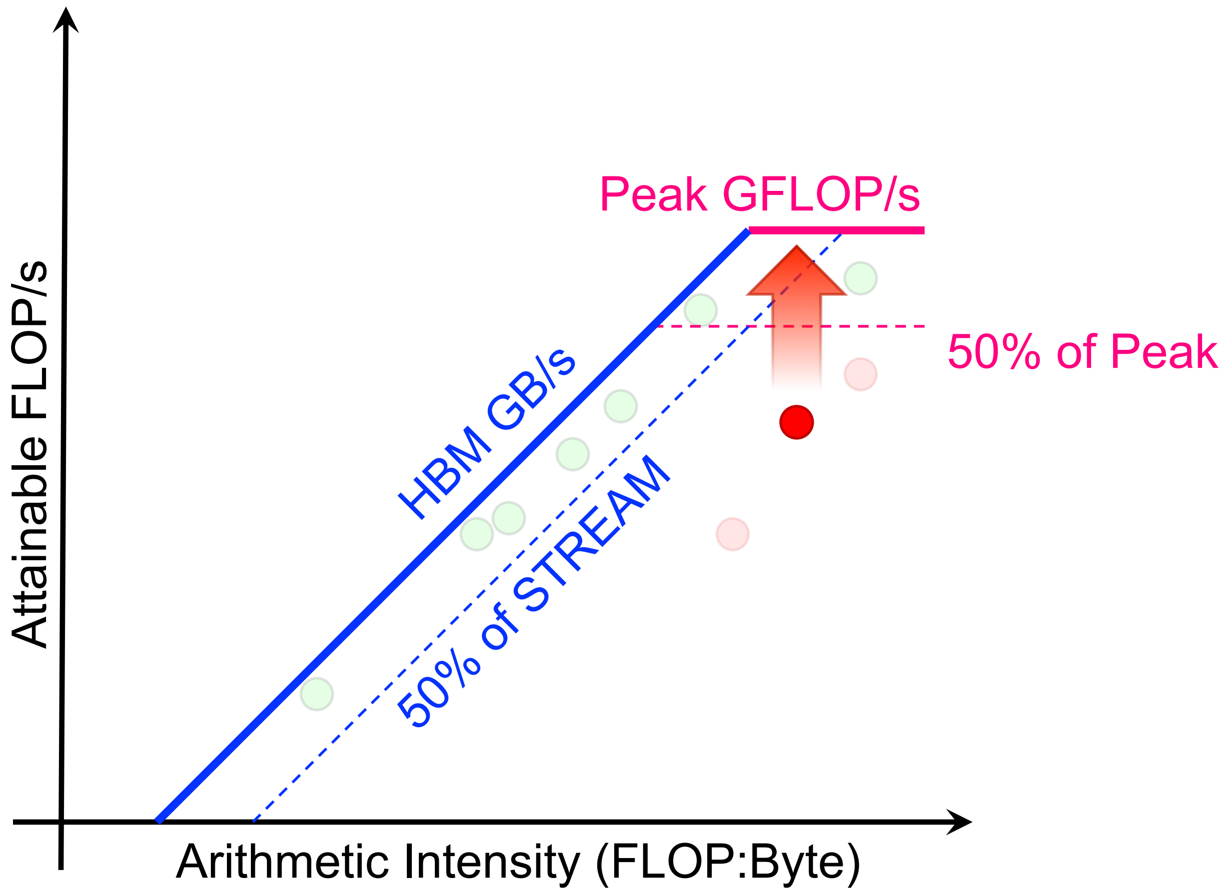
Roofline is made of two components

- Machine Model
    - Lines defined by peak GB/s and GF/s (**Benchmarking**)
    - Unique to each architecture
    - Common to all apps on that architecture
- Application Characteristics
    - Dots defined by application GFLOP's and GB's (**Application Instrumentation**)
    - Unique to each application
    - Unique to each architecture



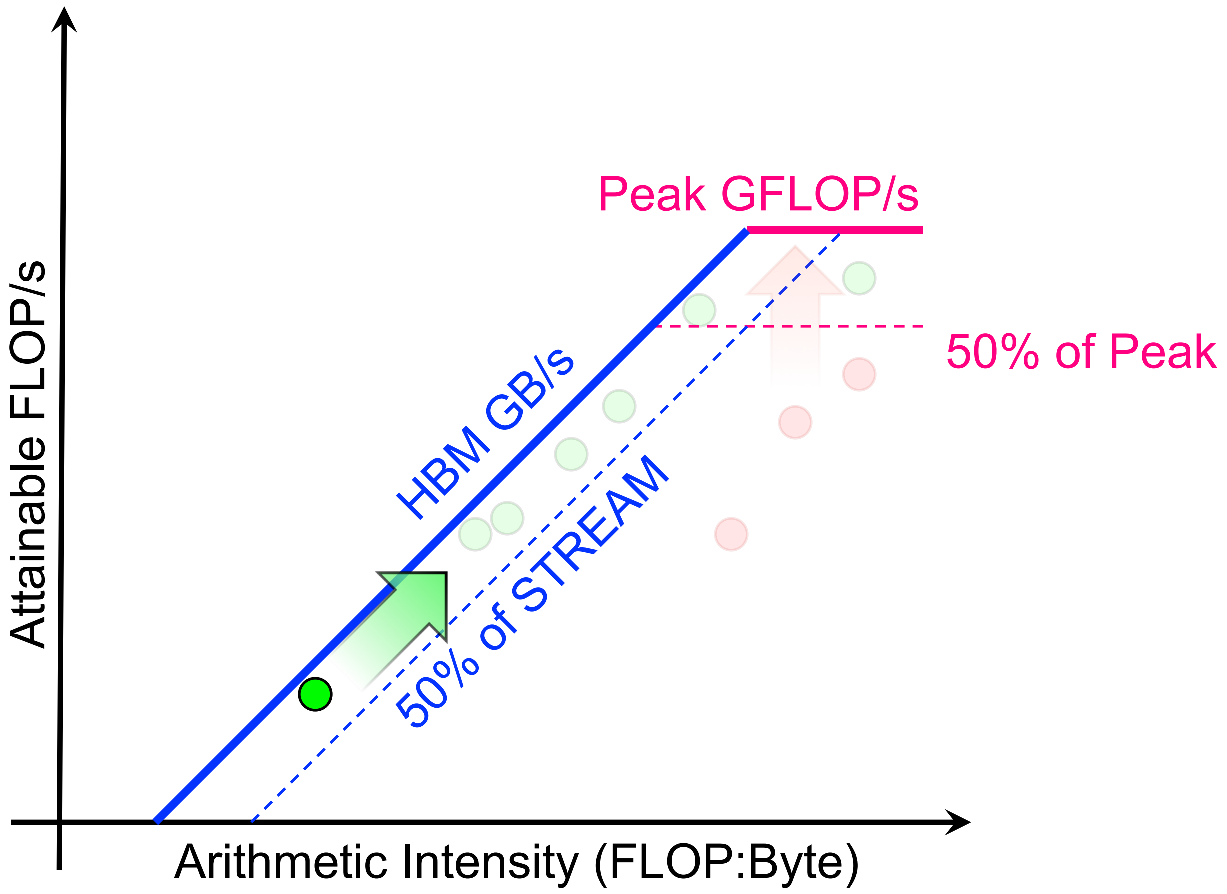
General Performance Optimization Strategy

- Get to the Roofline



General Performance Optimization Strategy

- Get to the Roofline
- Increase Arithmetic Intensity when bandwidth-limited
  - Reducing data movement increases AI

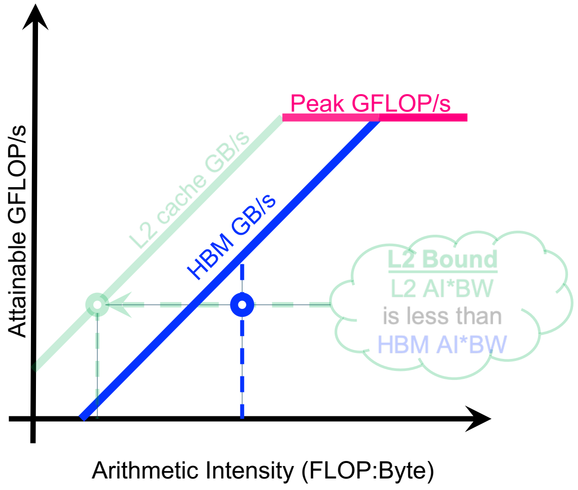


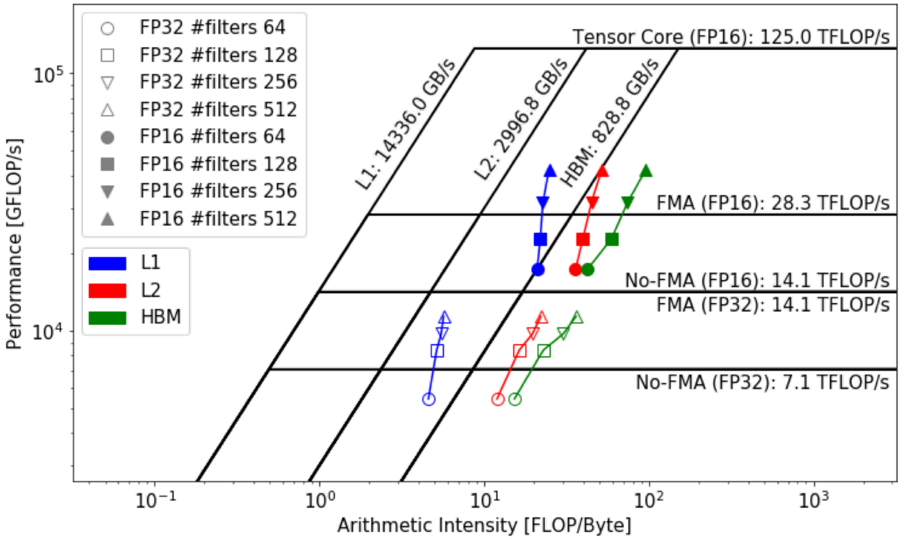
How can performance ever be below the Roofline?

Performance Below the Roofline?

Hierarchical Roofline Model

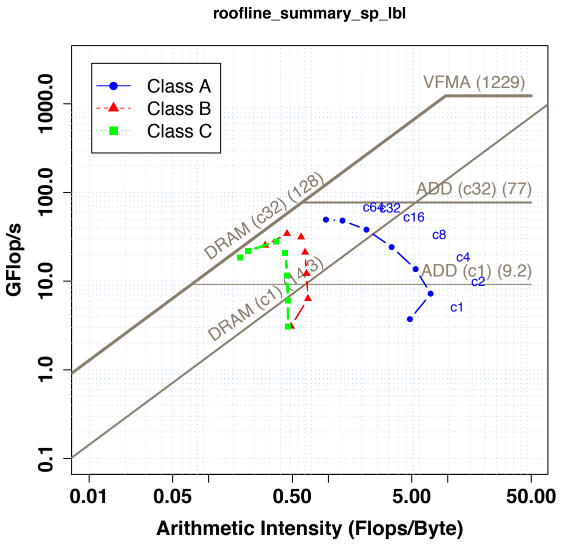
Charlene Yang, Thorsten Kurth, Samuel Williams, "Hierarchical Roofline analysis for GPUs: Accelerating performance optimization for the NERSC-9 Perlmutter system", Concurrency and Computation: Practice and Experience (CCPE), August 2019.





Additional FP Ceilings

Charlene Yang, Thorsten Kurth, Samuel Williams, "Hierarchical Roofline analysis for GPUs: Accelerating performance optimization for the NERSC-9 Perlmutter system", Concurrency and Computation: Practice and Experience (CCPE), August 2019.

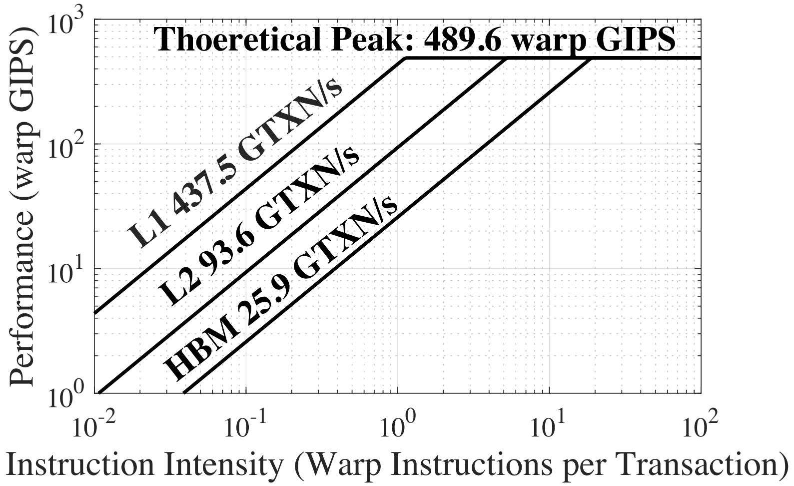


Roofline Scaling Trajectories

Khaled Ibrahim, Samuel Williams, Leonid Oliker, "Performance Analysis of GPU Programming Models using the Roofline Scaling Trajectories", International Symposium on Benchmarking, Measuring and Optimizing (Bench), BEST PAPER AWARD, November 2019.

Instruction Roofline Model

Nan Ding, Samuel Williams, "An Instruction Roofline Model for GPUs", Performance Modeling, Benchmarking, and Simulation (PMBS), BEST PAPER AWARD, November 2019.



Summary

Why We Use Roofline…

1. Determine when we're done optimizing code
    - Assess performance relative to machine capabilities
    - Track progress towards optimality
    - Motivate need for algorithmic changes

2. Identify performance bottlenecks & motivate software optimizations

3. Understand performance differences between Architectures, Programming Models, implementations, etc…
    - Why do some Architectures/Implementations move more data than others?
    - Why do some compilers outperform others?

4. Predict performance on future machines / architectures
    - Set realistic performance expectations
    - Drive for Architecture-Computer Science-Applied Math Co-Design

Take away

- Roofline helps understand application performance relative to machine capabilities
    - just the beginning of the optimization process
    - Other bottleneck- or architecture-specific tools can be used to refine the process

- Roofline helps frame the conversation between…
    - Application Developers
    - Computer Scientists
    - Applied Mathematicians
    - Processor Vendors

…providing a common mental model and optimization language

Questions