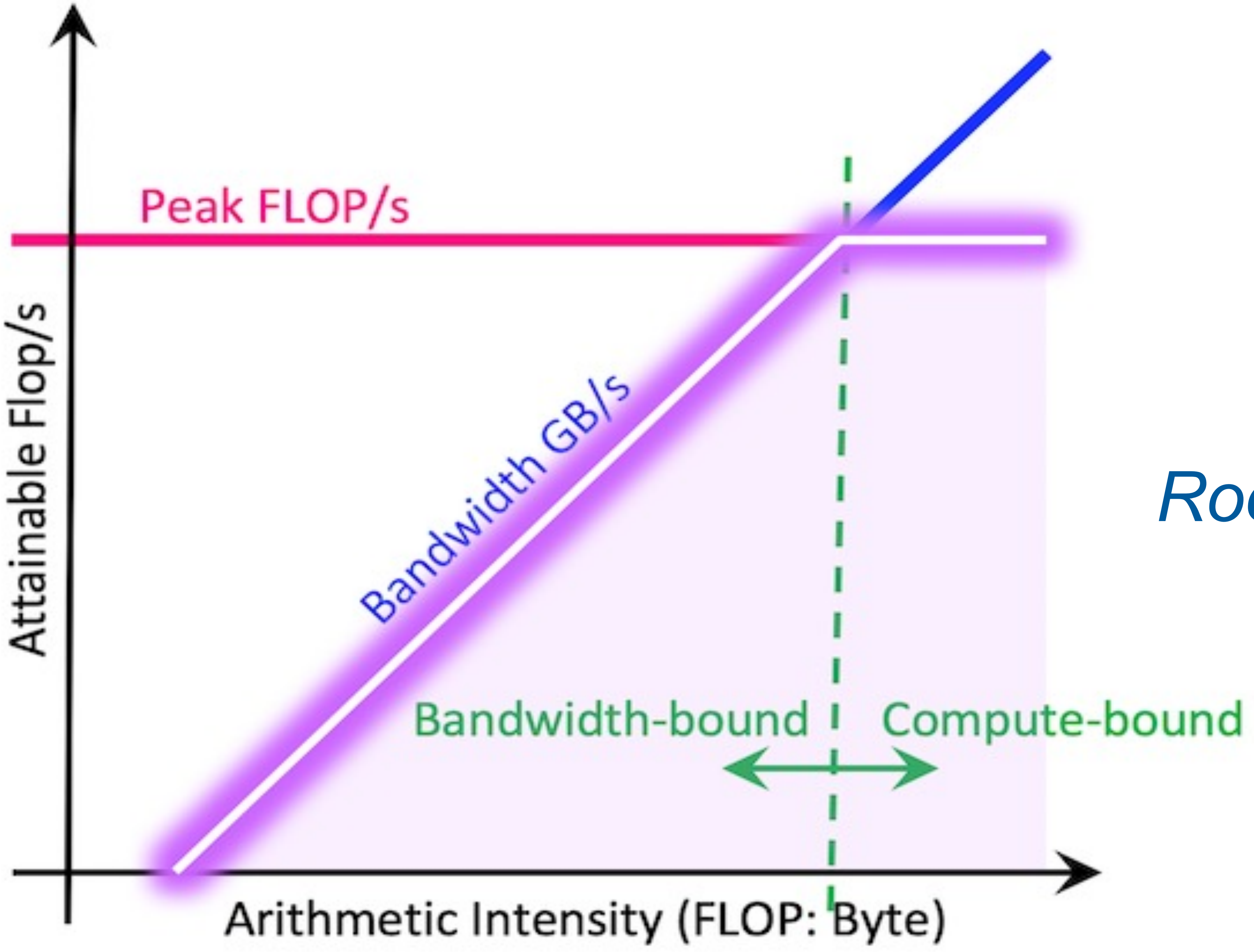


Roofline Model

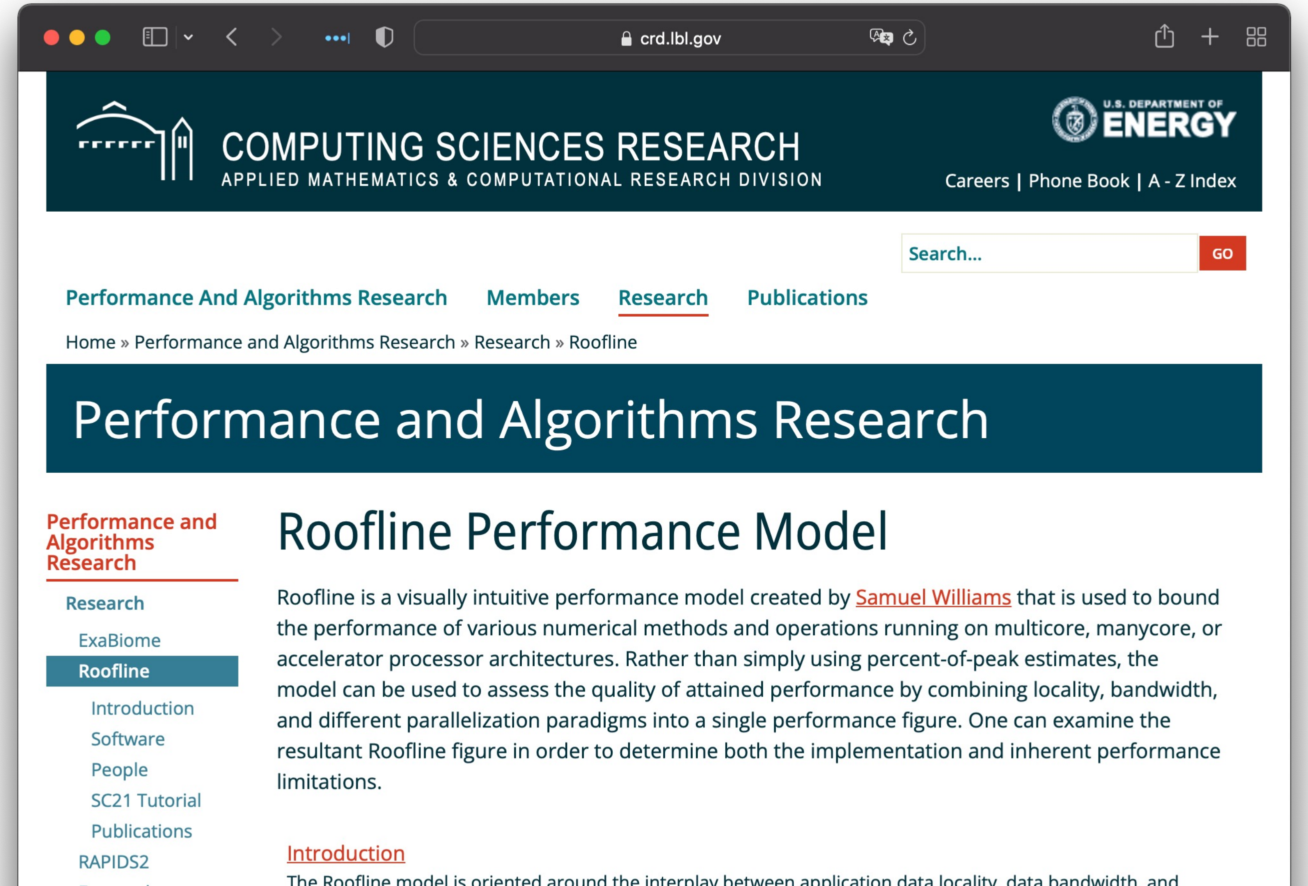
Why Use Performance Models or Tools?

- 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 / 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)
- Effectively latency hiding has resulted in a shift from a latency-limited computing regime to a **throughput-limited computing regime**

- The **Roofline Model** is a throughput-oriented performance model
  - Applies to x86, ARM, POWER, CPUs, GPUs, Google TPUs, FPGAs, etc.
  - Helps quantify **good performance**



- **Roofline: An Insightful Visual Performance Model for Multicore Architectures**

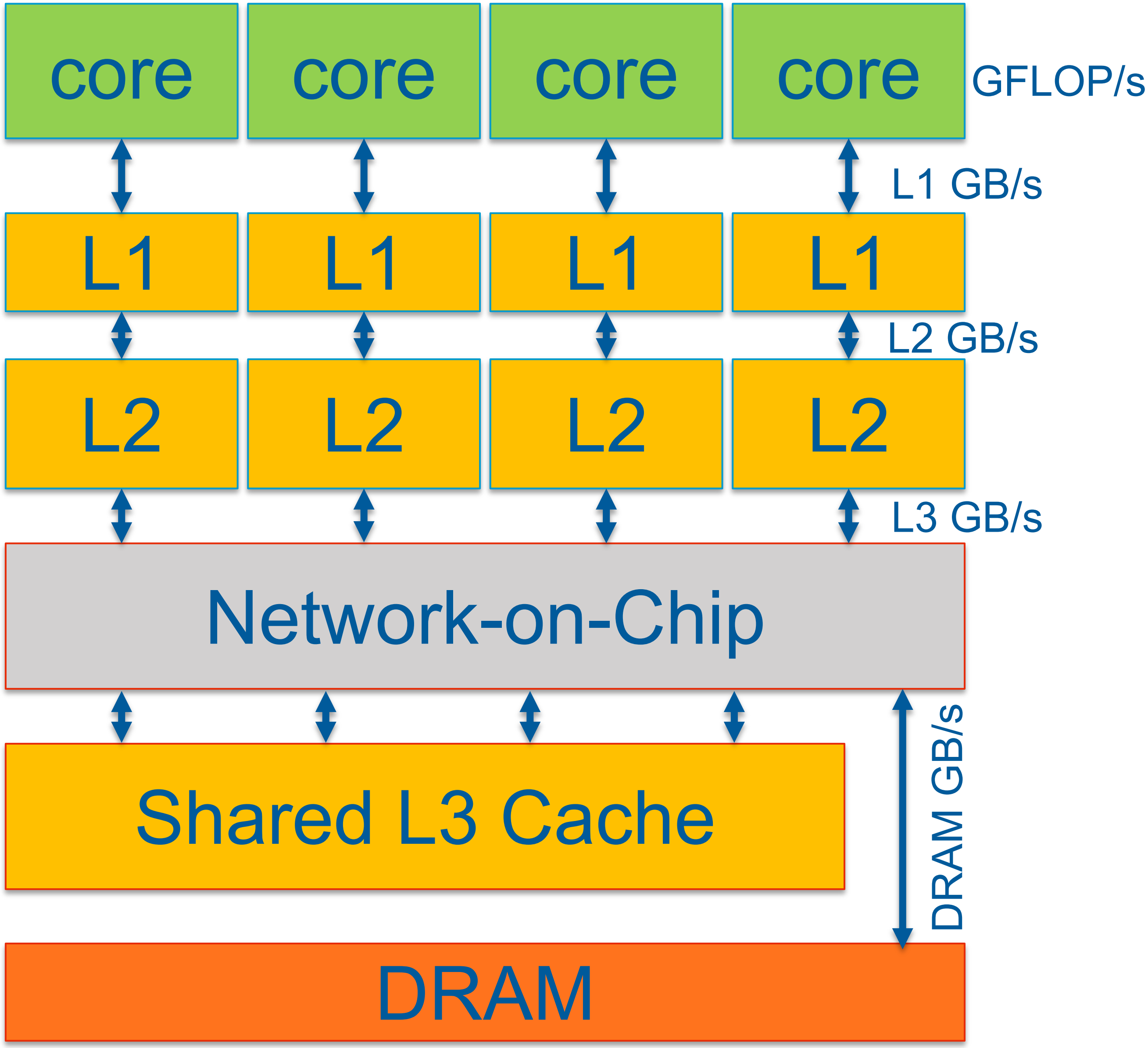
  S. Williams, A. Waterman, and D. Patterson

  Comm. of the ACM, 52(4), pp. 65–76, 2009

  https://doi.org/10.1145/1498765.1498785

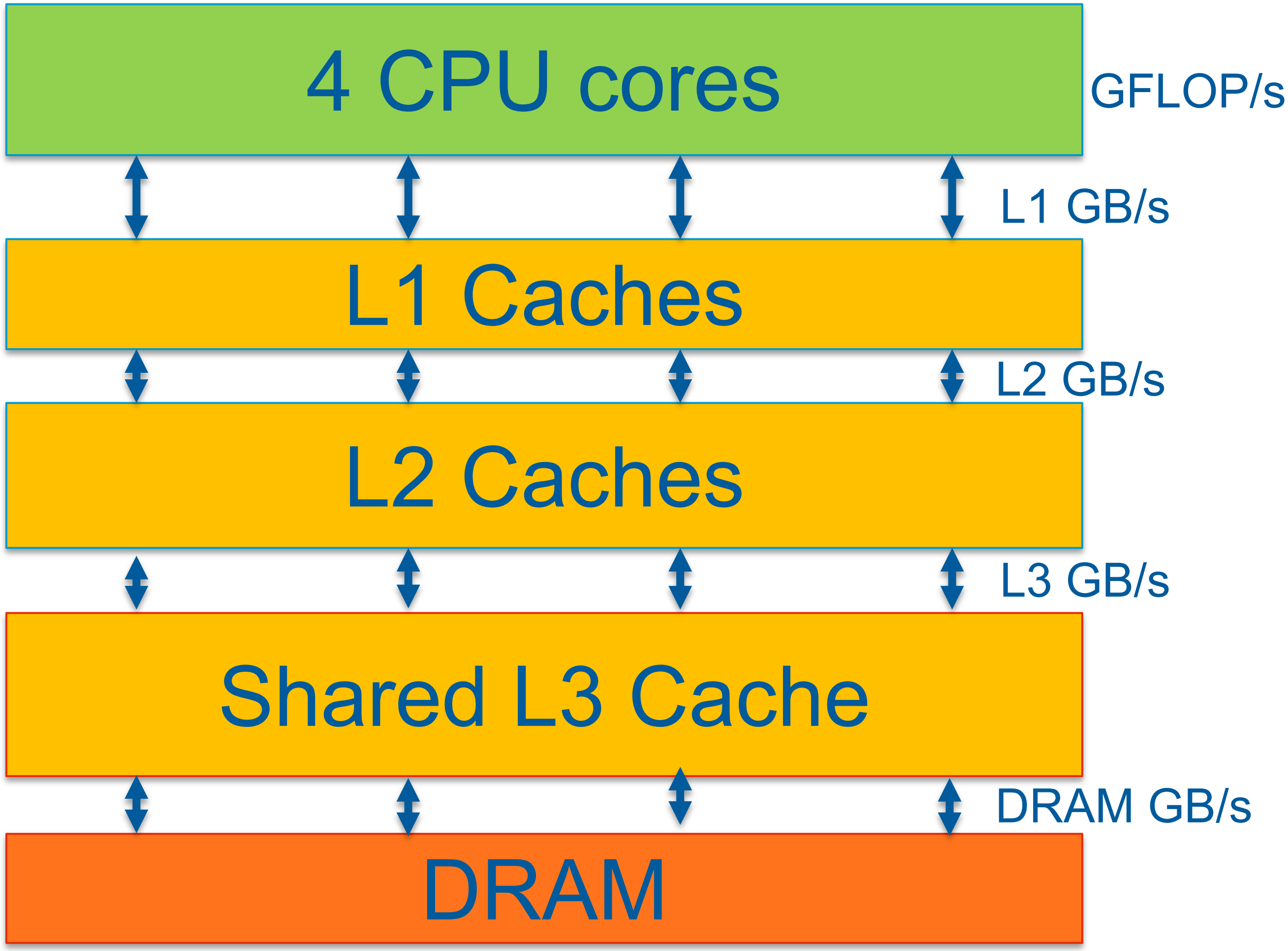
Reduced Model

- Modern architectures can be complex
- Don’t model / simulate full architecture
- Make assumptions on performance / usage
  - Peak GFLOP/s on data in L1
  - Load-balanced SPMD code



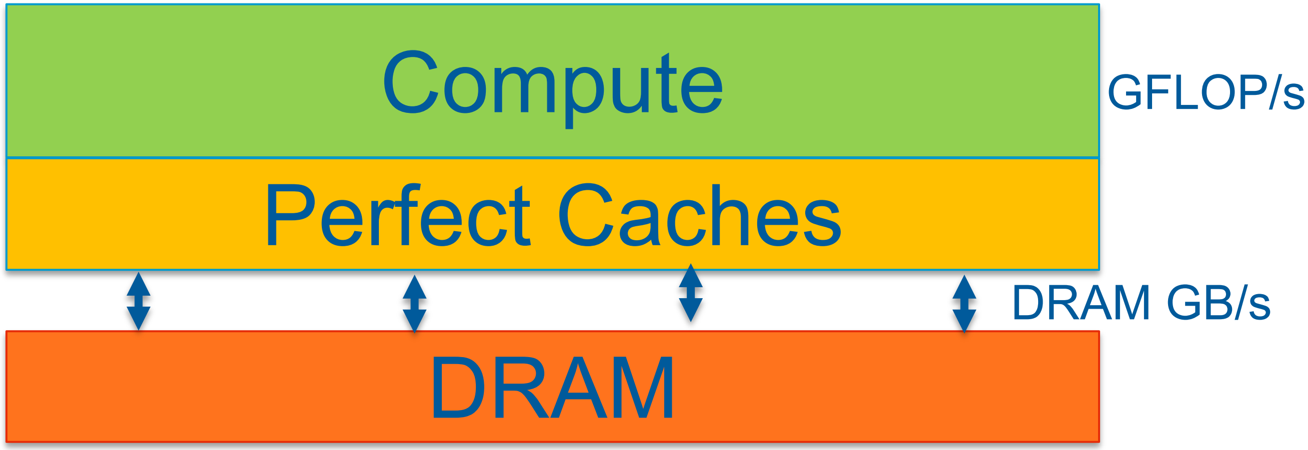
Reduced Model

- Modern architectures can be complex
- Don’t model / simulate full architecture
- Make assumptions on performance / usage
  - Peak GFLOP/s on data in L1
  - Load-balanced SPMD code
  - Sufficient cache bandwidth/capacity

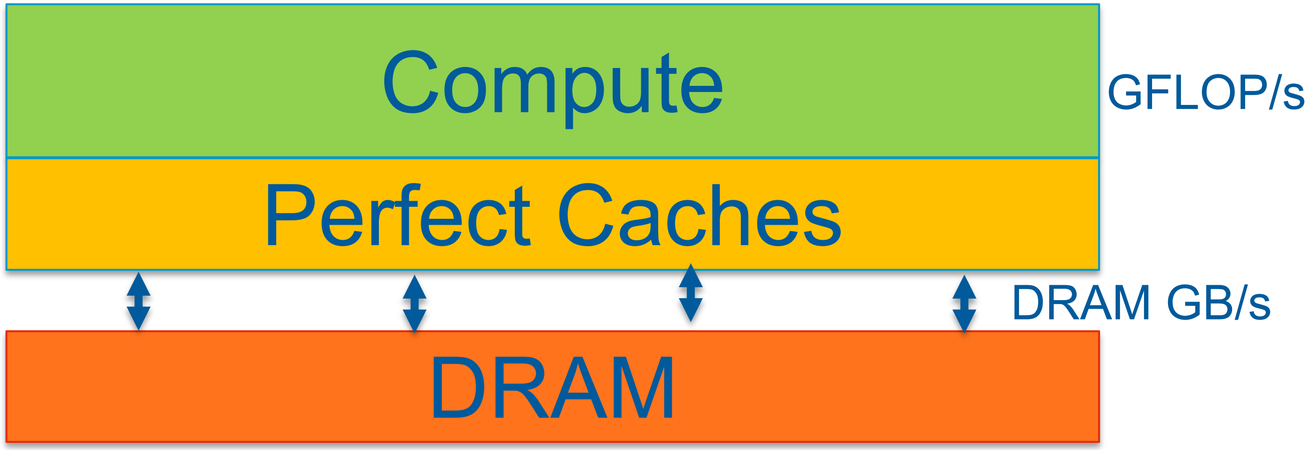


Reduced Model

- Modern architectures can be complex
- Don’t model / simulate full architecture
- Make assumptions on performance / usage
  - Peak GFLOP/s on data in L1
  - Load-balanced SPMD code
  - Sufficient cache bandwidth/capacity
  - Basis for DRAM Roofline Model



Roofline Model



- Any given loop nest will perform
  - Computation (e.g., FLOPs)
  - Communication (e.g., moving data to/from DRAM)
- With perfect overlap of communication and computation
  - Runtime is determined by whichever is greater

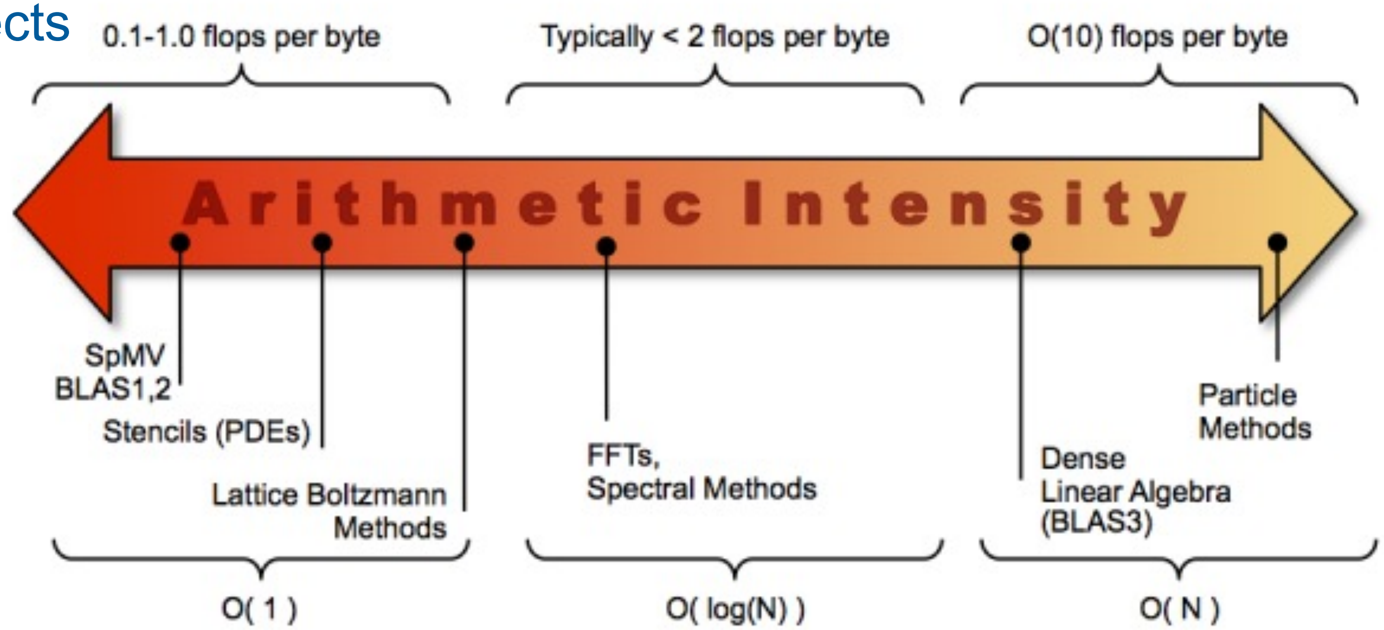
$$time = \max \begin{cases} \#FLOPs\ /\ Peak\ GFLOP/s \\ \#Bytes\ /\ Peak\ GB/s \end{cases} \qquad GFLOP/s = min \begin{cases} Peak\ GFLOP/s \\ AI * Peak\ GB/s \end{cases}$$

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

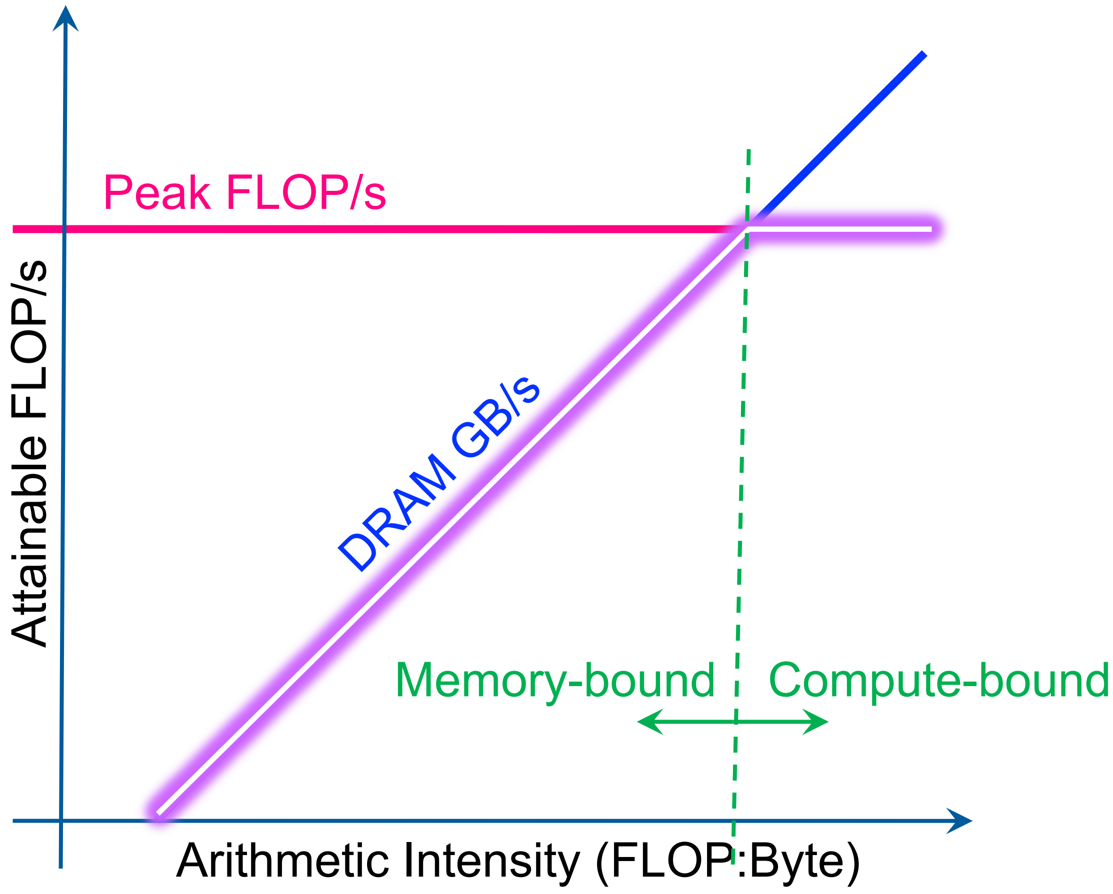
    total loads/stores (bytes requested)
  - Equal to ratio of sustained GFLOP/s

    to sustained GB/s (time cancels)



(DRAM) Roofline Model

- $GFLOP/s = min \begin{cases} Peak\ GFLOP/s \\ AI * Peak\ GB/s \end{cases}$
- AI = FLOPs / Bytes presented to DRAM
- Plot roofline bound using Arithmetic Intensity (AI) as the x-axis
- **Log-log scale** makes it easy to doodle, extrapolate performance along Moore's Law, etc.



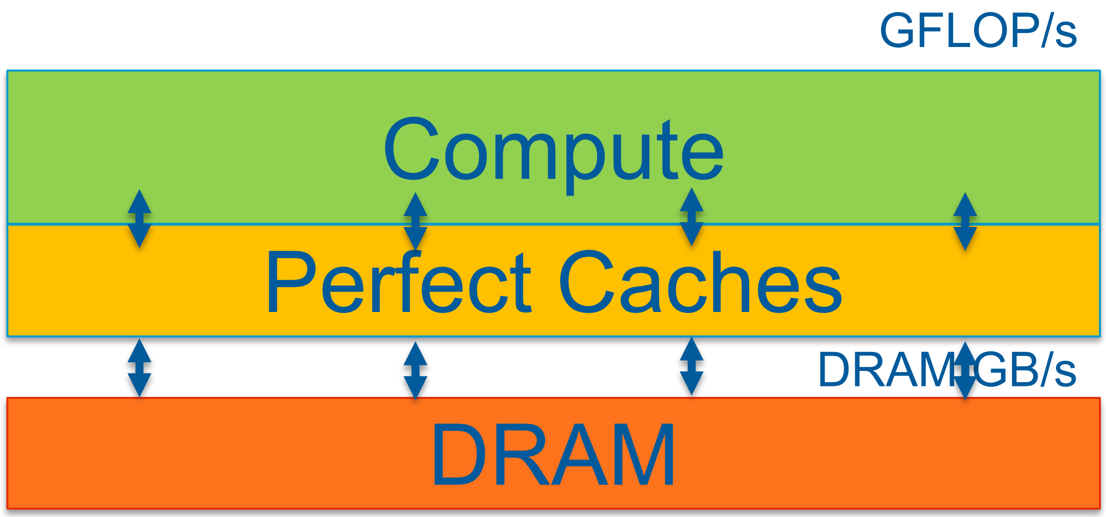
Roofline Model Example #1

- 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 (DAXPY)

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

  - 2 FLOPs per iteration
  - Transfer 24 bytes per iteration

    (read X[i], Y[i], write Y[i])
  - AI = 0.083 FLOPs per byte == memory bound



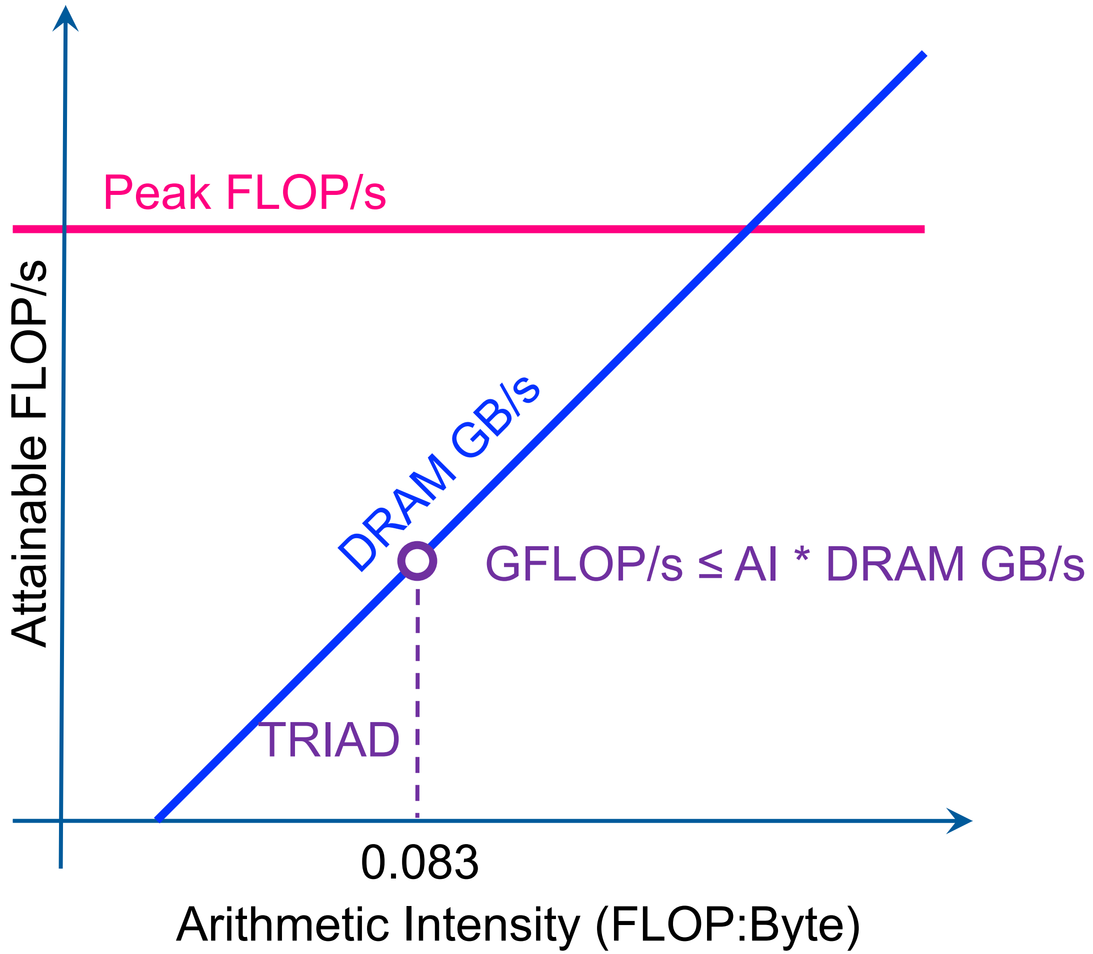
Roofline Model Example #1

- 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 (DAXPY)

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

  - 2 FLOPs per iteration
  - Transfer 24 bytes per iteration

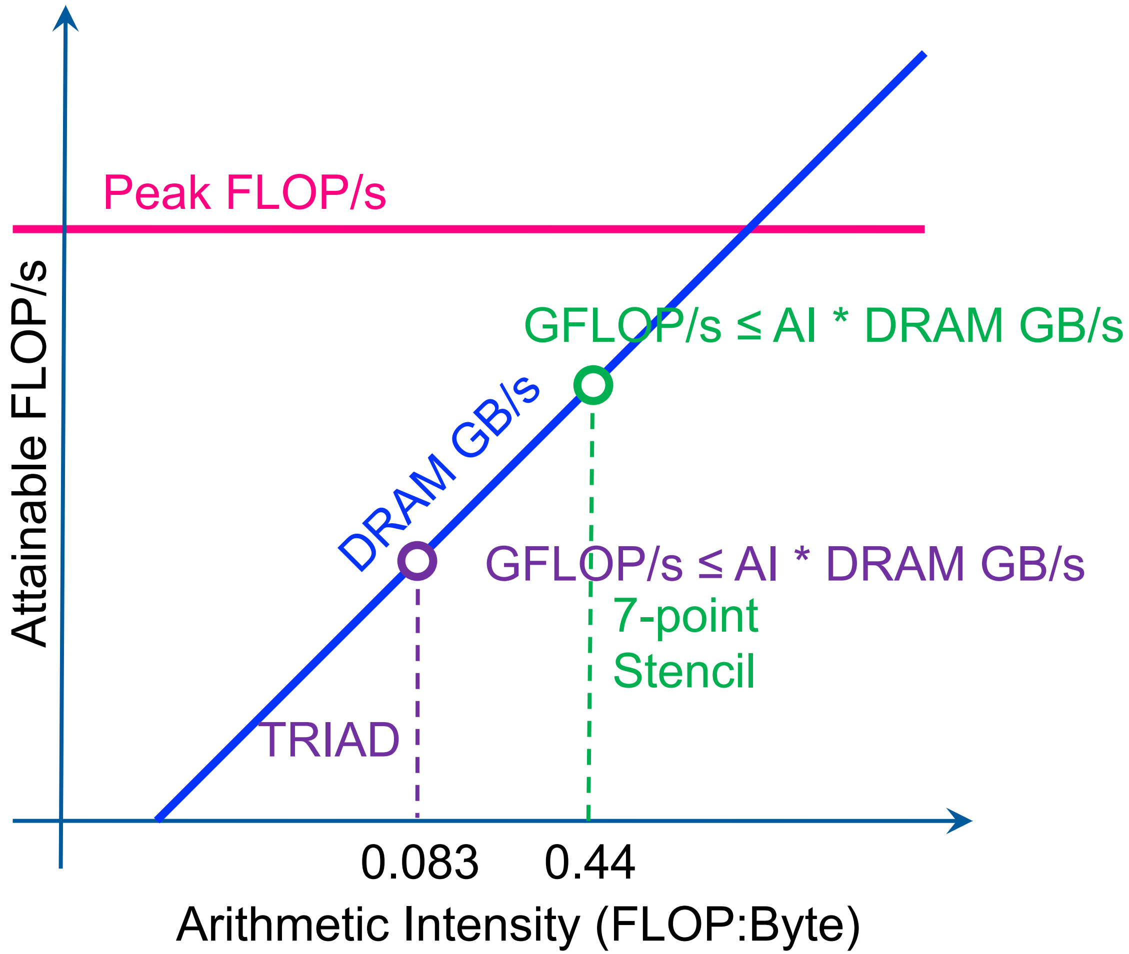
    (read X[i], Y[i], write Y[i])
  - AI = 0.083 FLOPs per byte == memory bound



Roofline Model 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 (int k=0; k<N; ++k)
  for (int j=0; j<N; ++j)
    for (int i=0; i<N; ++i)
      new[k][j][k] = -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  ];
```

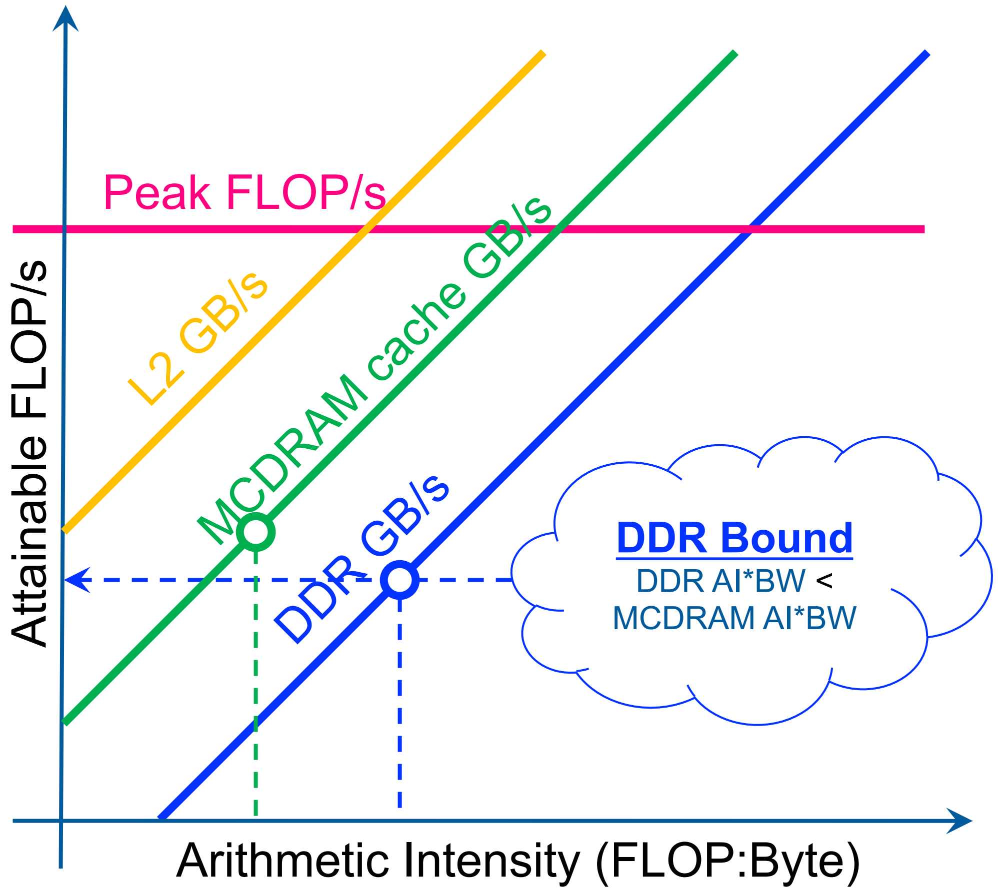


Hierarchical Roofline Model

- 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

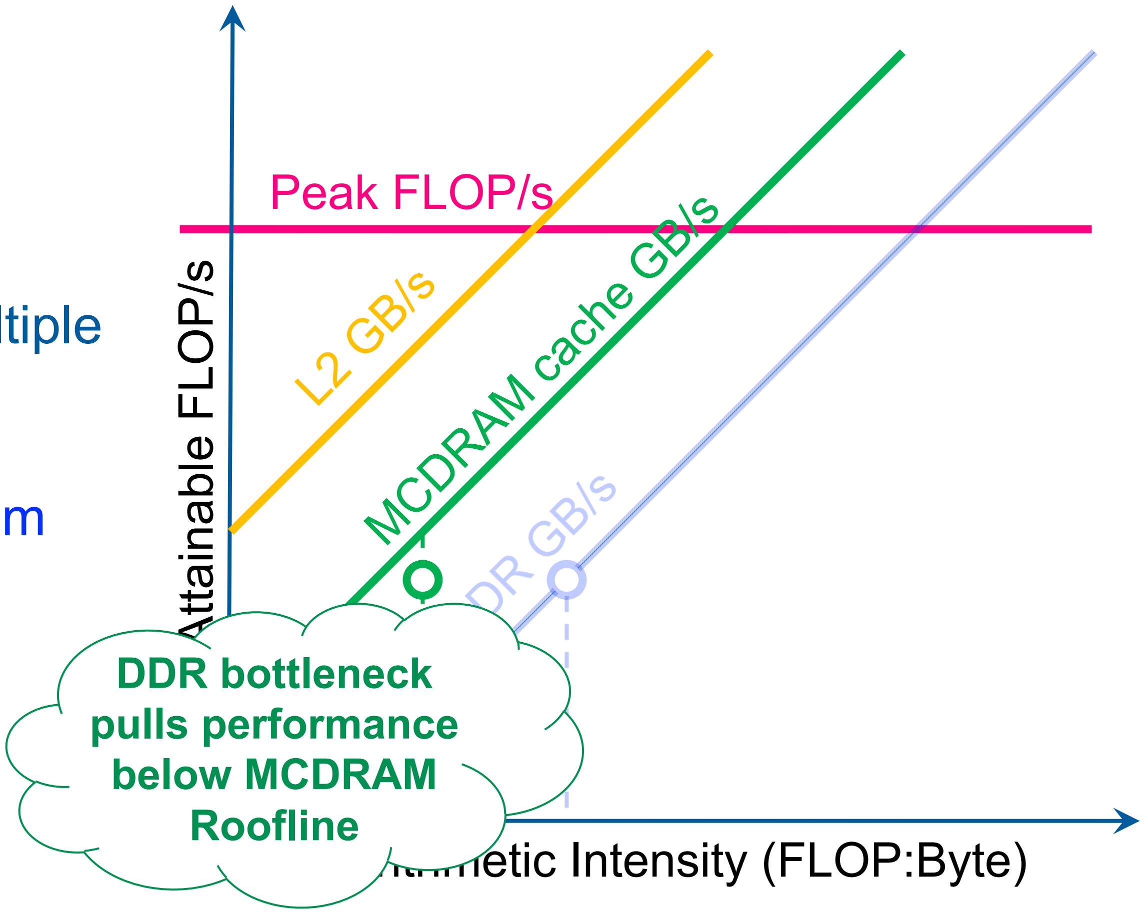
Hierarchical Roofline Model

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



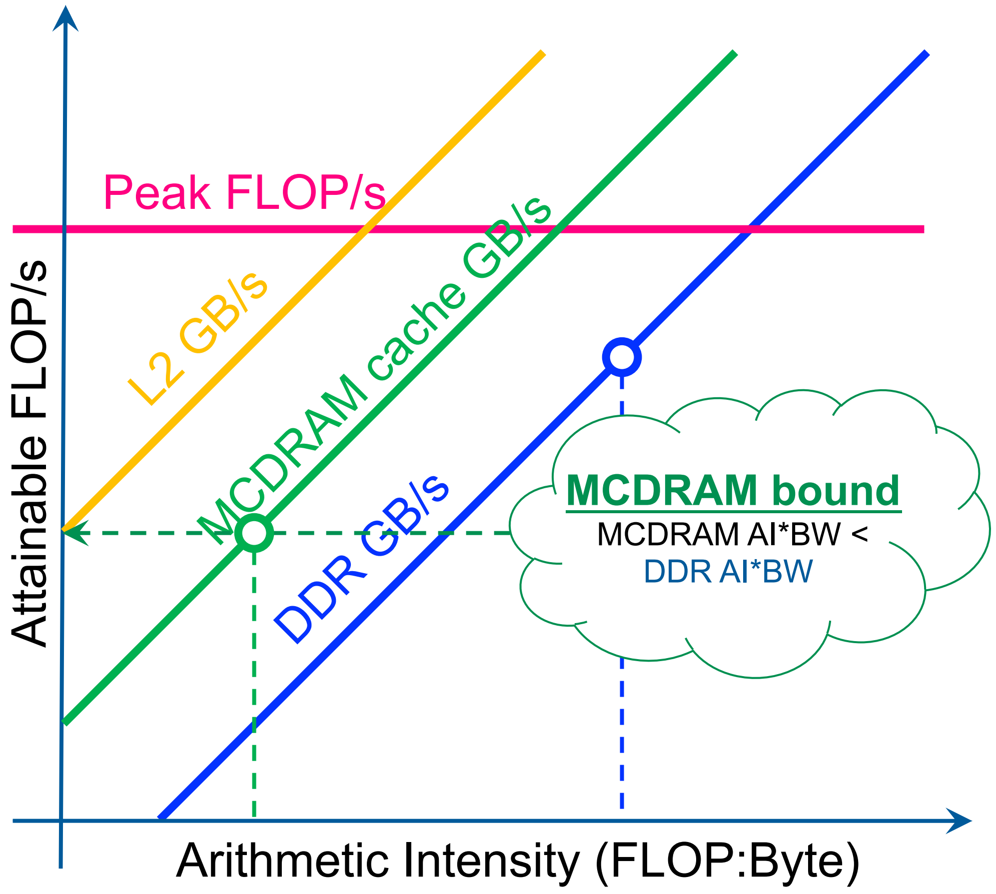
Hierarchical Roofline Model

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



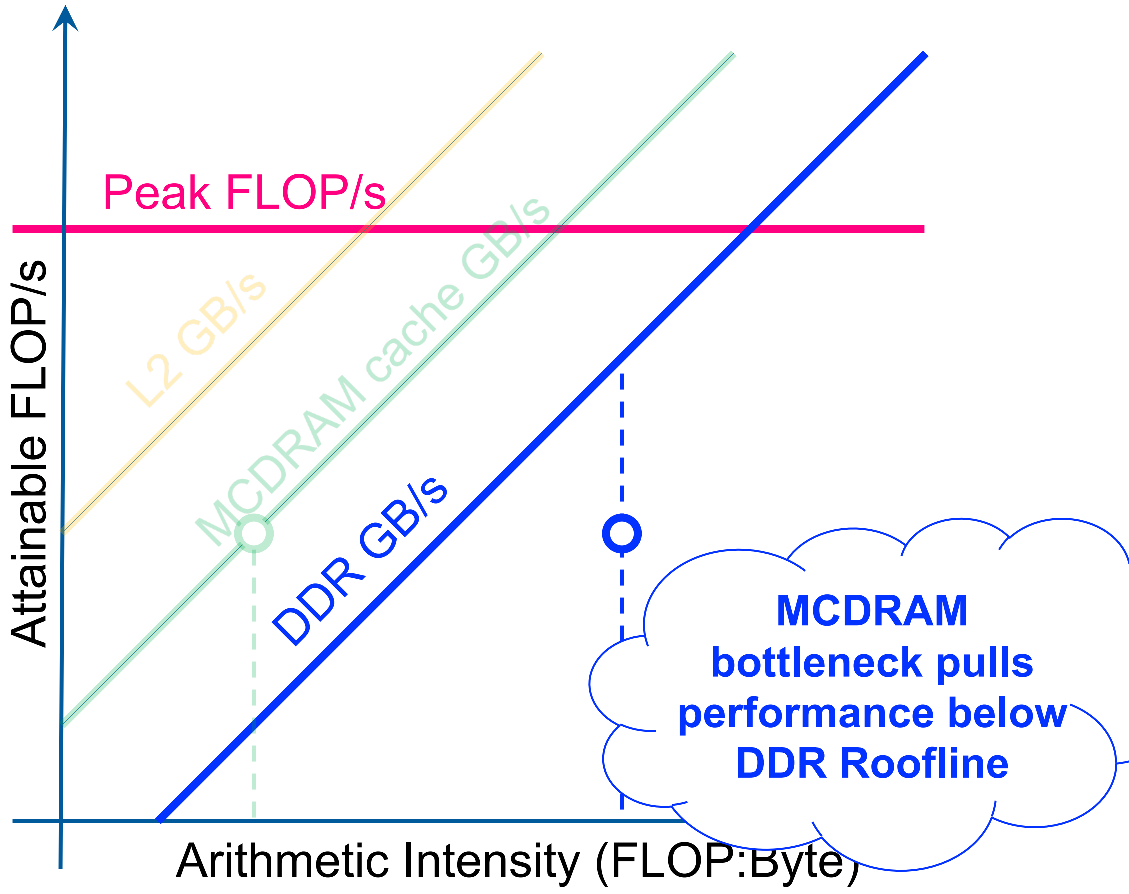
Hierarchical Roofline Model

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



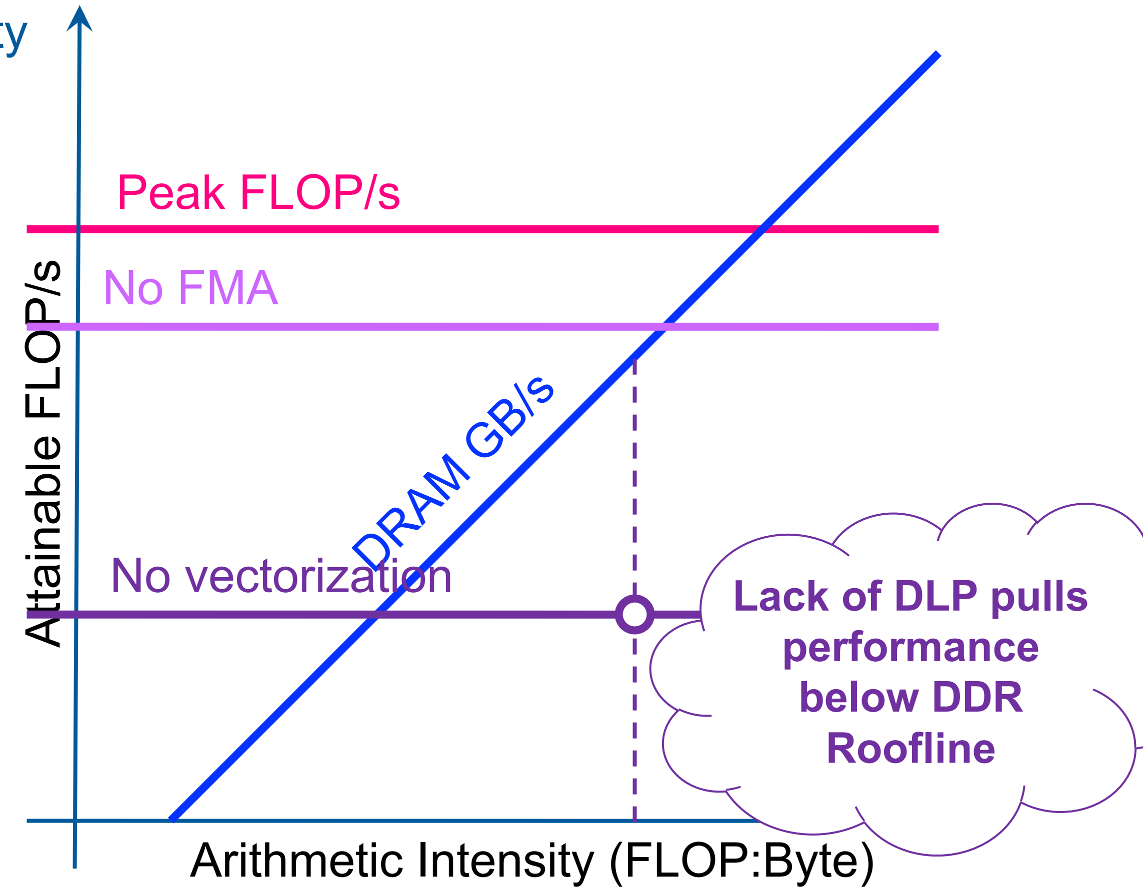
Hierarchical Roofline Model

- Construct superposition of Rooflines
  - Measure a bandwidth
  - Measure AI for each level of memory
  - Loop nest may have multiple AI’s and multiple bounds (FLOPs, L1, L2, … DRAM)
  - BUT performance is bound by the minimum

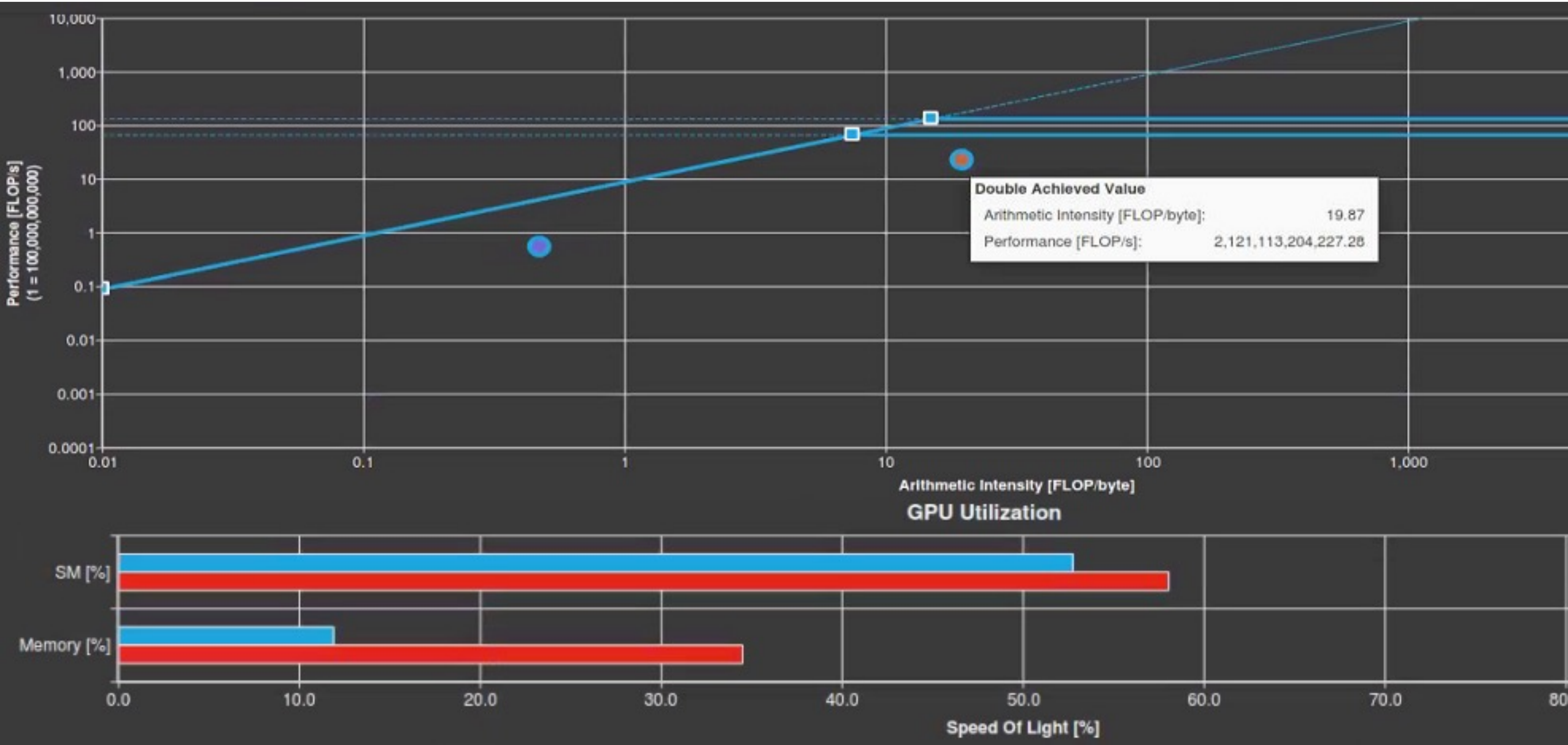


Data-, Instruction-, Thread-Level Parallelism

- We assumed one can attain peak FLOPs with high locality
- In reality, this is premised on sufficient
  - Use special instructions (e.g., fused multiply-add)
  - Vectorization (16 FLOPs per instruction)
  - Unrolling, out-of-order execution (hide FPU latency)
  - OpenMP across multiple cores
- Without these,
  - Peak performance is not attainable
  - Some kernels can transition from memory-bound to compute-bound
- In reality, DRAM bandwidth is often tied to DLP and TLP (single core can't saturate BW w/scalar code)



Nsight Compute: Roofline Analysis



Performance Model for Memory-Bound Kernels

- Simple performance model for memory-bound kernels
  - Operations are for free
  - All data comes from cache if possible
- With this, we can estimate the execution time based on
  - bytes: #memory transactions per thread
  - N: Total problem size (launch configuration)
  - b: Memory bandwidth of the GPU
  - $Time = \frac{N \cdot bytes}{b}$

Performance Model for Memory-Bound Kernels

- Example: 3x3 blur filter
  - bytes: #memory transactions per thread
    9 reads, 1 store: assume that one middle pixel needs to be read
    → 1 read + 1 store, 32bit floating point ➔ 8 bytes
  - N: Total problem size (launch configuration)
    image of 4096x4096 pixels ➔ 16.777.216
  - b: Memory bandwidth of the GPU
    GeForce GTX 970 peak memory bandwidth: 192 GB/s
    GeForce GTX 970 memcpy memory bandwidth: 138 GB/s
    - $Time = \frac{N \cdot bytes}{b} = \frac{16777216 \cdot 8\ bytes}{192GB/s} = 0.000699\ s = 0.699\ ms$
    - $Time = \frac{N \cdot bytes}{b} = \frac{16777216 \cdot 8\ bytes}{138GB/s} = 0.000973\ s = 0.973\ ms$
    - Measured time: $0.904\ ms$