Performance

Sources of performance

- We assume a HPC machine to be a collection of interconnected nodes
- Performance depends on:
  - Application and parallelization
    - Application structure
    - Algorithm
    - Data set
    - Programming model
  - Single-node performance
  - Multi-node performance

Performance metrics

- A pragmatic classification:
- Users :
  - How fast is my application?
    - Execution time
    - Speed-up
- Developers :
  - How close to the absolute best can I be?
    - Estimate peak(s) and distance from peaks
- Budget-holders:
  - How much of my infrastructure am I using?
    - Utlization

Performance “actions”

- Performance measurement
  - Measure execution time
  - Derive metrics such as speed-up, throughput, bandwidth
  - Platform and application implementation are available
  - Data-sets are available

Performance “actions”

- Performance analysis
  - Estimate performance bounds
    - Performance bounds are typically worst-case, best-case, average-case scenarios.
  - Platform & application are real or close-to-real
    - Simplified models
  - Data-sets are models

Performance “actions”

- Performance prediction
  - Estimate application behavior
  - Platform & application are models
  - Data-sets are real

Speed-up

- Serial execution time T_S
- Parallel execution time T_P
- Overhead (p is # compute units)

$$T_O = p \cdot T_P - T_S$$

  - Ideal case: To = 0 (perfect linear speed-up)

- Speedup
  - Linear: S=p
  - Sublinear: S < p
  - Superlinear: S > p

$$S = \frac{T_{serial\_best}}{T_P}$$

$$T_S = 100$$

$$T_{P=1} = 200$$

$$T_{P=4} = 75$$

$$S_1 = \frac{200}{75} \approx 2.6$$

$$S_2 = \frac{100}{75} \approx 1.3$$

Is superlinear speed-up possible in theory?
In practice: cache effects + problem decomposition.

Superlinear speed-up: example

- T_cache=2 ns, T_mem=100 ns, T_penalty=0
- Problem: k FLOPs, 1 FLOP/mem access
- With 1 Core: Cache hit = 80%.

T_FLOP1 = 2 * 0.8 + 100 * 0.2 = 21.6 ns

- With 2 Cores: Cache hit rate improves to 90%

T_FLOP2 = 2 * 0.9 + 100 * 0.1 = 11.8 ns

- S = k * T_FLOP1 / (k/2) * T_FLOP2 ~ 3.67 !!

Speed-up

- Always use the best known sequential execution time
- Always vary the input sizes
  - Speed-up can depend on input data
- Always vary the machine "sizes"
  - Scalability is key

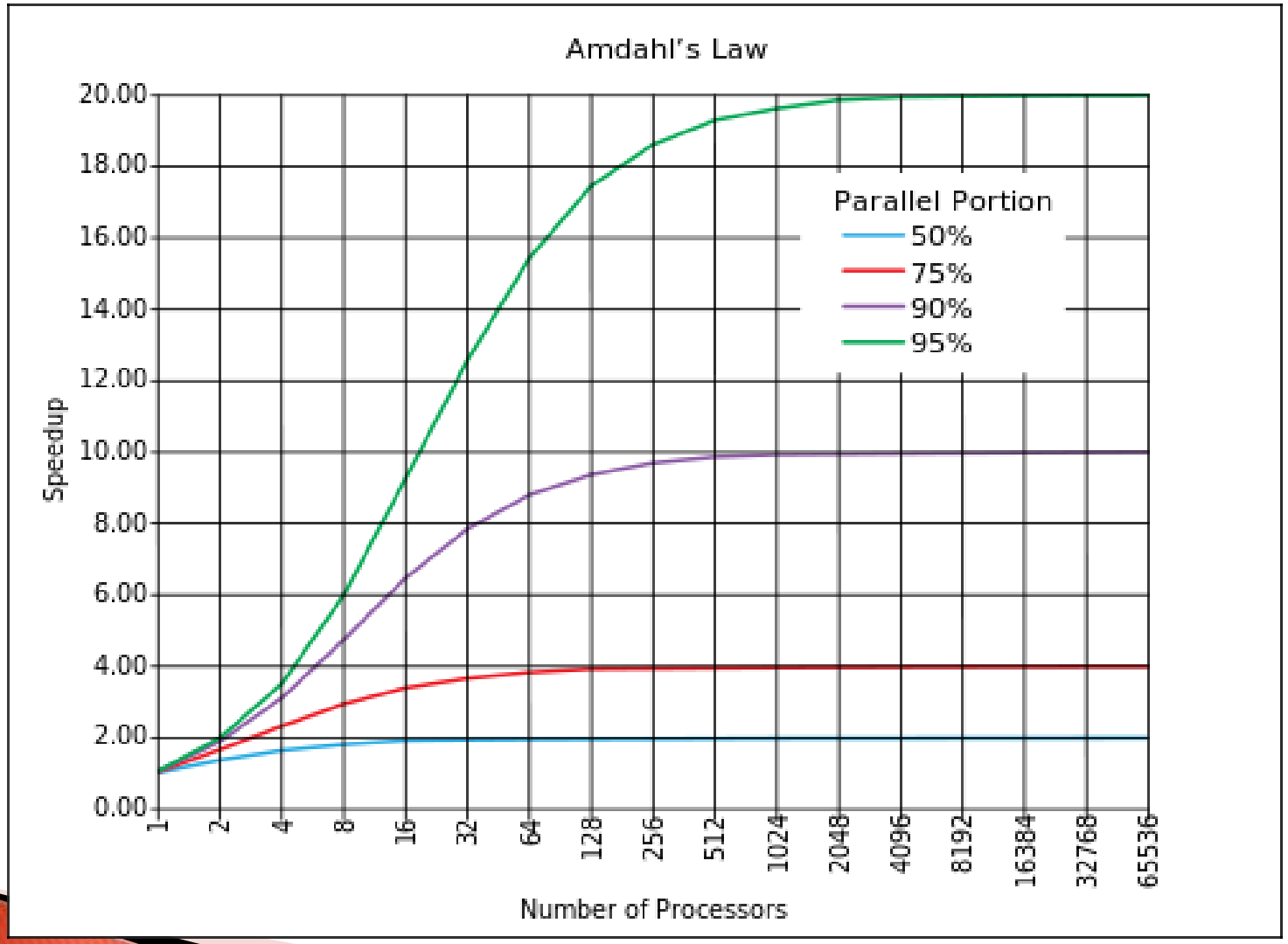
Performance estimation

- Amdahl's law:
  - s = sequential work
  - (1−s) = parallelizable work
  - p = number of processors
  - S = application speedup

$$
\begin{aligned}
S &= T_{seq}/T_{par} \\
&= 1/(s + (1-s)/p) \\
&\leq 1/s
\end{aligned}
$$

Speedup is bounded by the sequential fraction.

Amdhal's Law in pictures



Amdahl's law in the MC era*

- Design multi-cores to address *both* the parallel and sequential parts of the code
  - A lot of cores for high-throughput applications

and

  - Several cores to improve the performance of the serial part

*M.D.Hill and M.R.Marty, "Amdahl's Law in the Multicore Era", IEEE Computer, July 2008.

Efficiency

- Problem size: N, Processing elements: P

$$E(N,P) = \frac{T_{seq}(N)}{T_{par(N)} * P}$$

- It is a measure of the “goodness” of a parallel solution for a given application

Scalability

- A parallel solution is **scalable** if it maintains its **efficiency** (constant) when increasing P by increasing N.

- To translate:
  - Can you increase both P and N and keep efficiency the same => scalable application
  - Scalability is a measure of the parallelism in the application and its dependence on the problem size.

Simple performance modeling

- Model computation
  - Count number of operations
  - Assume flat memory model
- Model communication
  - Typically simple model, linear with the amount of data items communicated for large volumes of data
  - Only model explicit, distant communication
- Assume:
  - Ts = number_ops * t_op
  - Tp = (number_ops/p) * t_op + Tcomm
  - Tcomm = number_comm * t_comm

Practical Performance

Hardware Performance metrics

- Clock frequency [GHz] = absolute hardware speed
  - Memories, CPUs, interconnects
- Operational speed [GFLOPs]
  - How many operations per cycle can the machine do
- Memory bandwidth (BW) [GB/s]
  - Differs a lot between different memories on chip
  - Remember? Slow memory is large, fast memory is small …
- Power [Watt]

- Derived metrics
  - Normalized for comparison purposes …
  - FLOP/Byte, FLOP/Watt, …

Theoretical peak performance

Peak = chips * cores * vectorWidth * FLOPs/cycle * clockFrequency

- Cores = real cores, hardware threads, or ALUs, depending on the architecture

- Examples from DAS-4:
  - Intel Core i7 CPU = **154 GFLOPs**
    2 chips * 4 cores * 4-way vectors * 2 FLOPs/cycle * 2.4 GHz
  - NVIDIA GTX 580 GPU = **1581 GFLOPs**
    1 chip * 16 SMs * 32 cores * 2 FLOPs/cycle * 1.544 GHz
  - ATI HD 6970 GPU = **2703 GFLOPs**
    1 chip * 24 SIMD engines * 16 cores * 4-way vectors * 2 FLOPs/cycle * 0.880 GHz

DRAM Memory bandwidth (off-chip)

- Throughput = memory bus frequency * bits per cycle * bus width
  - Memory clock is not the CPU clock (typically lower)
  - Divide by 8 to get GB/s

- Examples:
  - Intel Core i7 DDR3: 1.333 * 2 * 64 = 21 GB/s
  - NVIDIA GTX 580 GDDR5: 1.002 * 4 * 384 = 192 GB/s
  - ATI HD 6970 GDDR5:1.375 * 4 * 256 = 176 GB/s

Memory bandwidths

- On-chip memory can be orders of magnitude faster
  - Registers, shared memory, caches, …
    - E.g., AMD HD 7970 L1 cache achieves 2 TB/s

- Other memories: depends on the interconnect
  - Intel's QPI (Quick Path Interconnect) : 25.6 GB/s
  - AMD's HT3 (Hyper Transport 3) : 19.2 GB/s
  - Accelerators: PCI-e 2.0 : 8.0 GB/s

Power

- Chip manufactures specify Thermal Design Power (TDP)
  - Some definition of maximum power consumption …
- We can measure dissipated power
  - Whole system
  - Typically (much) lower than TDP
- Power efficiency: FLOPS / Watt
  - Examples (with theoretical peak and TDP)
    - Intel Core i7: 154 / 160 = **1.0 GFLOPs/W**
    - NVIDIA GTX 580: 1581 / 244 = **6.3 GFLOPs/W**
    - ATI HD 6970: 2703 / 250 = **10.8 GFLOPs/W**

Summary

	Cores	Threads/ ALUs	GFLOPS	Bandwidth	**FLOPs/Byte**
Sun Niagara 2	8	64	11.2	76	**0.1**
IBM BG/P	4	8	13.6	13.6	**1.0**
IBM Power 7	8	32	265	68	**3.9**
Intel Core i7	4	16	85	25.6	**3.3**
AMD Barcelona	4	8	37	21.4	**1.7**
AMD Istanbul	6	6	62.4	25.6	**2.4**
AMD Magny-Cours	12	12	125	25.6	**4.9**
Cell/B.E.	8	8	205	25.6	**8.0**
NVIDIA GTX 580	16	512	1581	192	**8.2**
NVIDIA GTX 680	8	1536	3090	192	**16.1**
AMD HD 6970	384	1536	2703	176	**15.4**
AMD HD 7970	32	2048	3789	264	**14.4**

Absolute hardware performance

- Only achieved in the optimal conditions:
  - Processing units 100% used
  - All parallelism 100% exploited
  - All data transfers at maximum bandwidth
  - But
    - No application is like this
    - Even difficult to write micro-benchmarks
  - Hardware catalogue values are not realistic estimates of upper bounds of performance.

Arithmetic intensity

- The number of arithmetic (floating point) operations per byte of memory that is accessed.
- Ignore “overheads”
  - Loop counters
  - Array index calculations
  - …

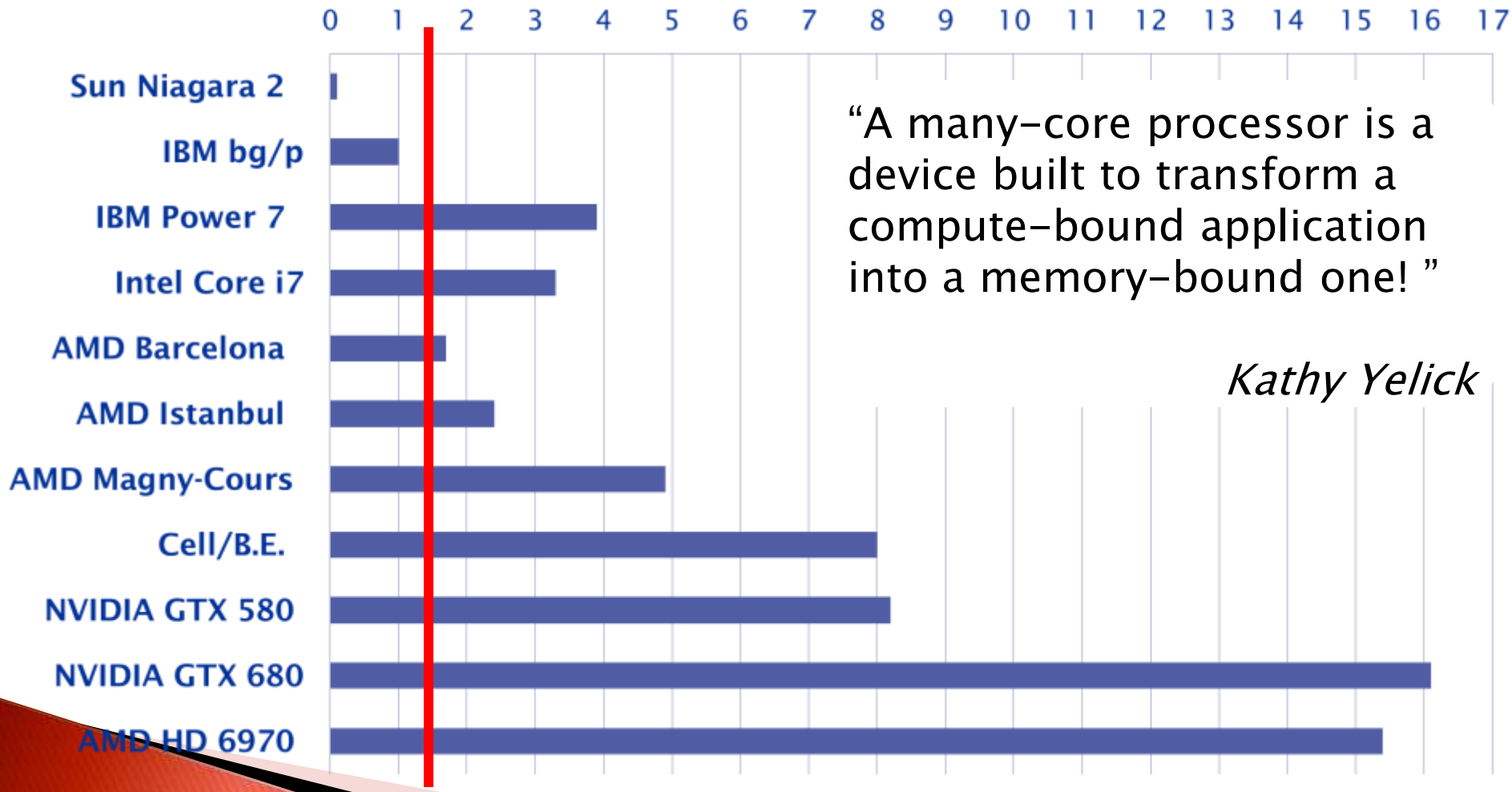
Arithmetic intensity

```
for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
                Pixel pixel = RGB[y][x];
                gray[y][x] =
                          0.30 * pixel.R
                        + 0.59 * pixel.G
                        + 0.11 * pixel.B; }}
```

- What is the arithmetic intensity of the RGB-gray kernel?
- Is this a compute-bound or a memory-bound kernel? Explain why.
- Does the kernel type (memory- or compute-bound) dependent on the application, machine, or both?

Compute or memory intensive?

Arithmetic intensity for several actual many-cores:

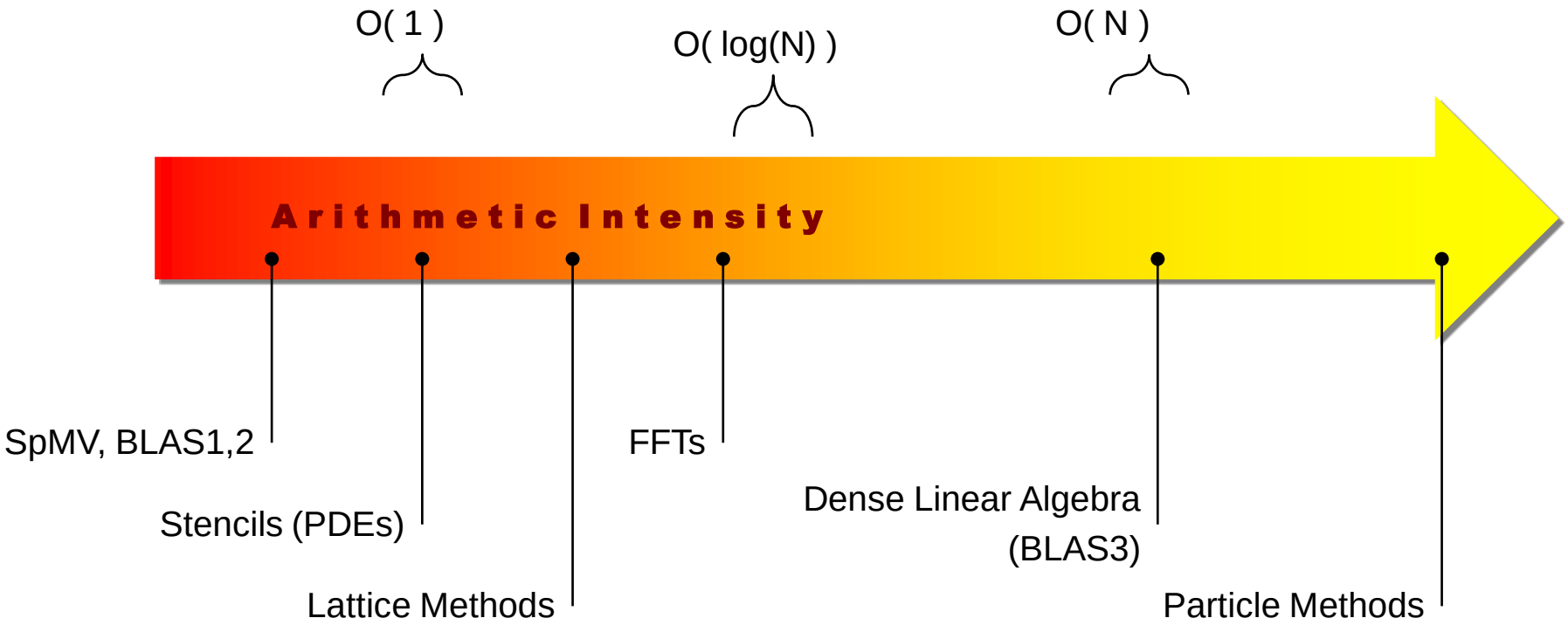


"A many-core processor is a device built to transform a compute-bound application into a memory-bound one! "

Kathy Yelick

RGB to Gray

AI for classes of applications



Red = high AI, Yellow = low AI

Attainable performance

- Attainable GFlops/sec

  min (Peak Floating-Point Performance,
  Peak Memory Bandwidth * Arithmetic Intensity)

- To translate:
  - If an application is compute-bound =>
    performance is limited by HW peak performance
  - If an application is memory-bound =>
    performance is limited by the load it puts on the memory system

Attainable performance (cont'd)

Typical case: application A runs on platform X in T_{exec_A} :

```
PeakCompute(X) = maxFLOP GLOPS/s                          (catalogue)
PeakBW(X) = maxBW GB/s                                    (catalogue)
RooflineCompute(A,X) = min(AI(A)*maxBW, maxFLOP)          (model)
AchievedCompute(A,X) = FLOPs(A)/Texec_A                   (real execution)
AchievedBW(A,X) = MemOPs(A)/Texec_A                       (real execution)
UtilizationCompute = AchievedCompute(A,X)/PeakCompute(X) < 1
UtilizationBW = AchievedBW(A,X)/PeakBW(X) ? 1
```

$Peak_{Compute}$ >= Roofline > $Achieved_{Compute}$

$Peak_{BW}$? $Achieved_{BW}$

Note: AchievedBW > PeakBW <=> faster memories on the chip play a role.

Take home message

- Application performance
  - Depends on algorithm, parallelization, programming model, machine
  - May depend on data-set
- Large variety of performance metrics
  - Depend on “end-users”
- Performance numbers
  - Used to understand application behavior – easy
  - Used to benchmark platforms – difficult
- Theoretical performance boundaries
  - Always too optimistic!
- Real performance boundaries
  - Important for understanding efficiency and/or utilization
  - Difficult to calculate => estimations