# Performance Analysis of Parallel Programs

Lecture 9
Introduction to High Performance Computing
IN4049 TUDelft
2021/2022

# About Performance

- Performance metrics
- Performance analysis, estimation, and prediction
  - theoretical performance analysis
    - *speed-up*
    - *efficiency*
  - practical performance analysis
    - *hardware*
    - *software*

# Performance metrics

A pragmatic classification:

- Users :
  - How fast is my application?
    - *execution time*
- Developers:
  - How close to the absolute best can I be?
    - *estimate absolute best*
    - *compute performance gain (speed-up)*
- Budget-holders:
  - How much of my infrastructure am I using?
    - *efficiency*
    - *utilization*

# 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 analysis
  - estimate performance bounds
    - *performance bounds are typically worst-case, best-case, average-case scenarios*
  - platform and application are available/models
  - data-sets are available/models
- Performance prediction
  - estimate application behavior
  - platform and application are models
  - data-sets are real

# Theoretical performance analysis

# Performance Metrics

- 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: $T_o = 0$ (perfect linear speed-up)*

Total overhead relevant for dedicated parallel processing

- Speedup $S = \dfrac{T_{serial\_best}}{T_P}$

Overhead may **depend** on *p* !

Relative versus true speedup: using $T_P(P=1)$ instead of $T_{serial\_best}$

Superlinear speed-up is sometimes possible:
cache effects / memory sizes

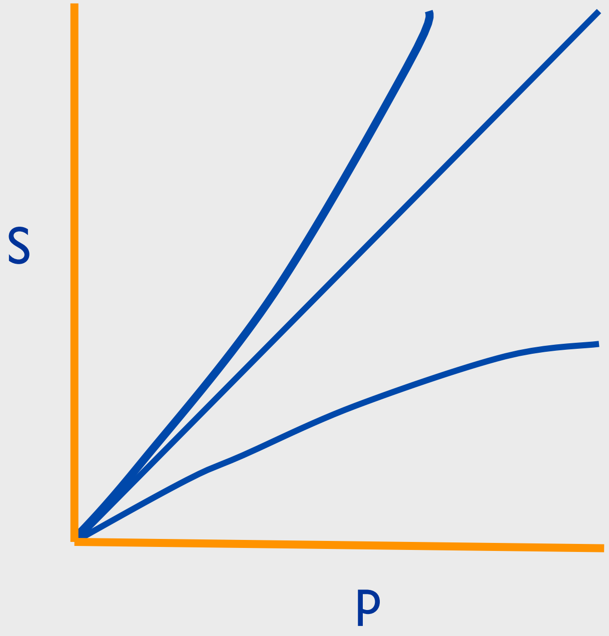
# Sub-/Super-/linear Speedup

$S_{linear} = p$ Linear Speedup

$S_{sublinear} < p$ Sublinear speedup

$S_{\mathrm{superlinear}} > p$ Superlinear speedup

E.g., a small partion of data can yield a higher cache hit rate



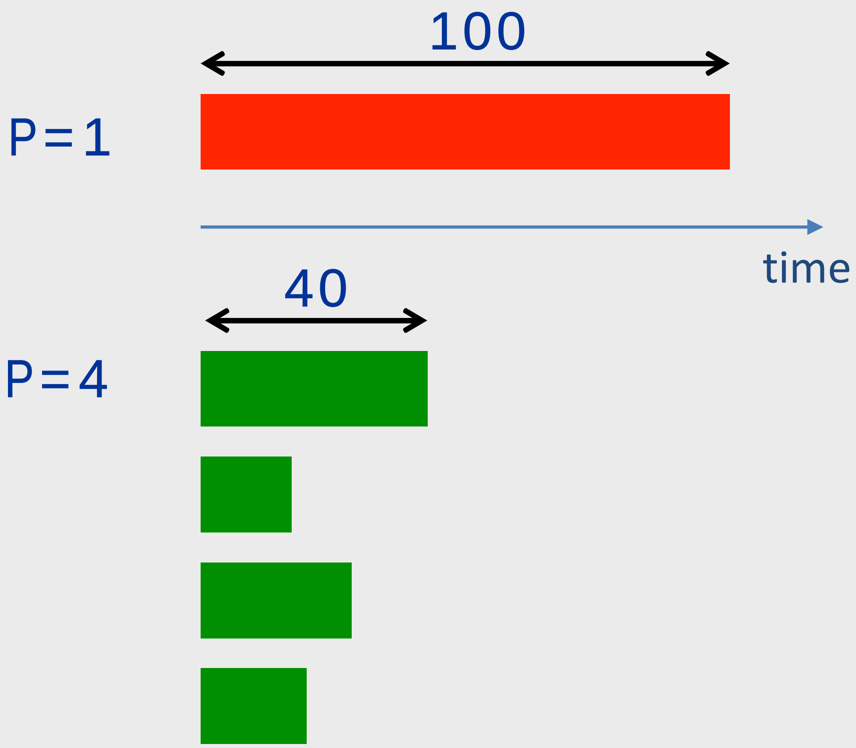
# Efficiency

$$\boxed{E = \frac{S}{p}}$$

$$\boxed{E = \frac{S}{p} = \frac{T_S}{p \cdot T_p}}$$

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

$$\boxed{E = \frac{1}{1 + \frac{T_o}{T_S}}}$$



S=2.5
E=2.5/4=0.625

# Sources of overhead in parallel programs

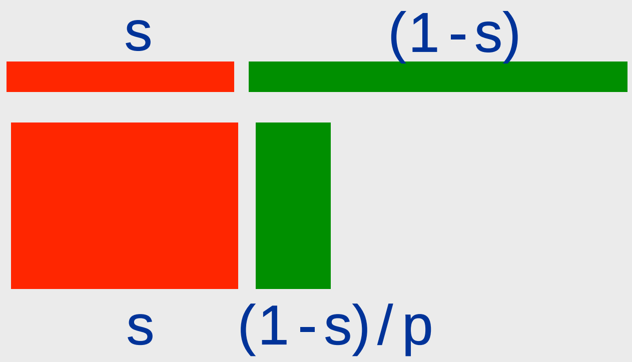
- Inter-process interaction
  - *communication => idling*
  - *synchronization => serialization*
- Load imbalance
  - *un-even workloads => idling*
- Additional computations, such as
  - *memory allocation*
  - *data partitioning*
  - *managing the parallelism*
  - *…*

# What is problem size?

- Intuitively: size of the input … but
  - *multiple inputs are a problem*
  - *does not characterize the dependency on the computation*
  - *example: MVP: A(m,n) * y(n)*
- Better definition:
  - *the problem size (W) is equal to the number of basic computation steps in the best sequential algorithm*
  - *example: MVP: $W=\Theta(m*n)$*
- We assume further that problem size W = $T_S$

# Recap: Amdahl’s law – fixed problem size

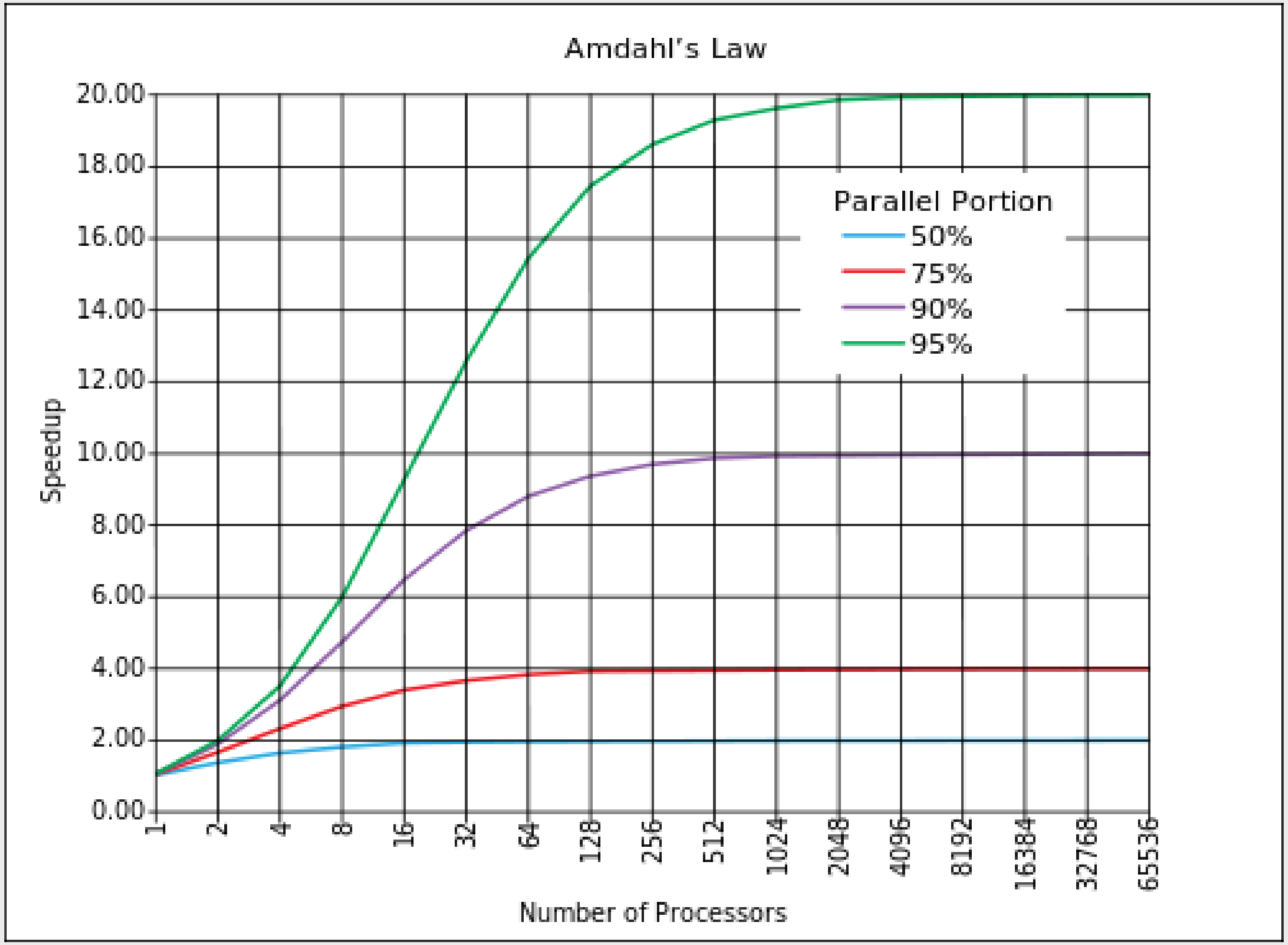
- Every application has an intrinsically sequential part
- Amdahl’s law:
  - *let s be the fraction of work that is sequential, then (1-s) is the fraction that is parallelizable*
  - *p = number of processors*
  - *S = Speedup*



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

*Speedup is bounded by the sequential fraction.*

# Amdhal’s Law: max speedup <1/s



# Isoefficiency function

What is a ‘good’ efficiency of a parallel program? →No simple answer

Emphasis on scalablity of a parallel algorithm: can a program retain its efficiency when #processors and problem size increase?

$$T_o = p \cdot T_P - W \qquad \text{(definition of overhead)}$$

and

$$S = \frac{W}{T_P} \quad \Rightarrow \quad S = \frac{W \cdot p}{W + T_o} \quad \Rightarrow \quad E = \frac{1}{1 + T_o/W}$$

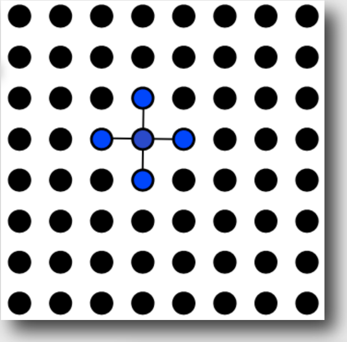
Conclusions:

1. E=1 when there is no overhead
2. E is fixed iff $T_o$/W is fixed

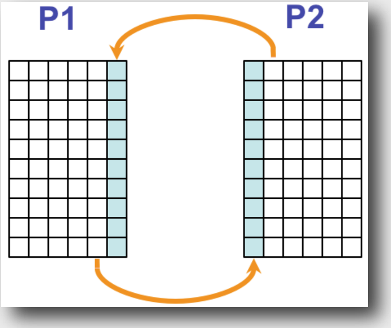
# Simple performance modeling

- Model computation
  - *count number of operations*
  - *assume flat memory model*
- Model communication
  - *typically simple model, linear with the number of data items communicated for large volumes of data*
  - *only model explicit the distant communication*
- Assume:
  - $T_s = number\_ops * t_{op}$
  - $T_p = (number\_ops/p) * t_{op} + T_{comm}$
  - $T_{comm} = number\_comm * t_{comm}$ $+ t_{setup}$ *(large setup time as will be seen later, we ignore it for simplicity now)*

# Example: stencil-type computations (1/3)



(n+2)x(n+2) grid
given boundary values



column-wise
data distribution

```
a[i,j]=0.25*(a[i,j-1]+a[i-1,j]+a[i,j+1]+a[i+1,j]);
```

$$T_S = 4t_{op} \cdot n^2$$

$$T_{comm} = 2n \cdot t_{data}$$
$$T_{calc} = 4t_{op} \cdot (n \cdot n/p) = 4t_{op}n^2/p$$

$$T_P = 4t_{op} \cdot n^2/p + 2n \cdot t_{data}$$

# Example: stencil-type computations (2/3)

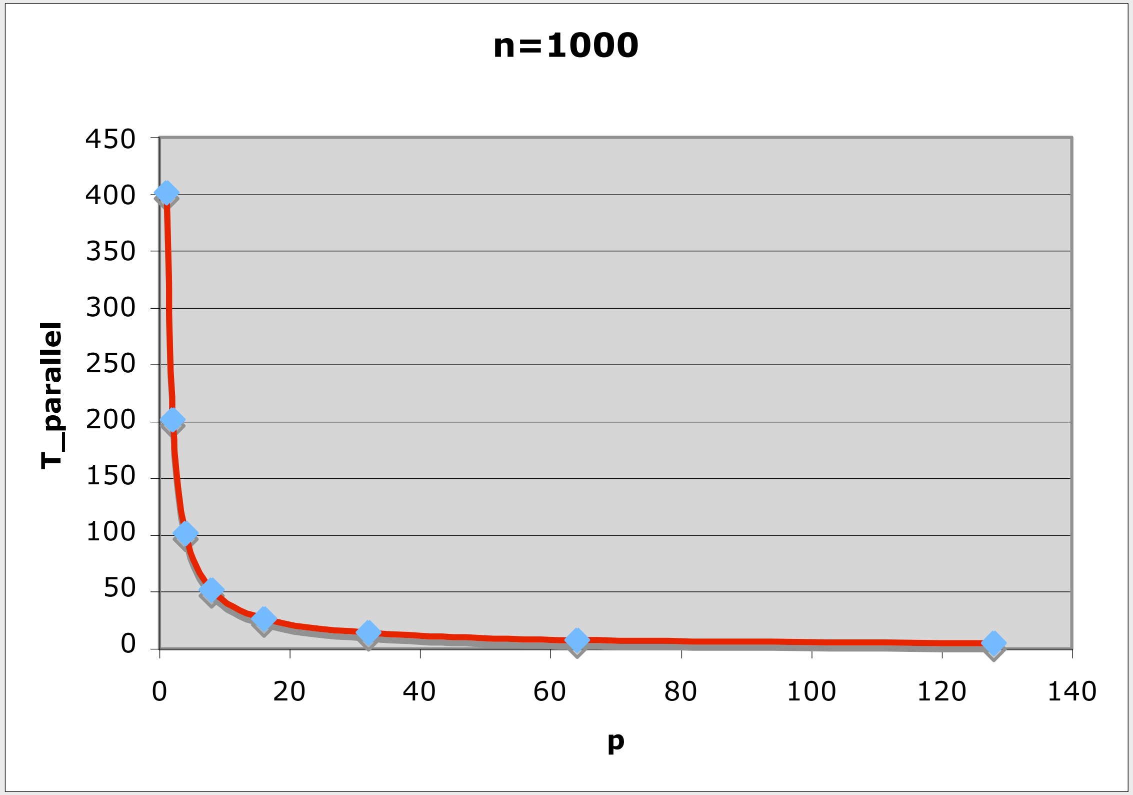
$$S = \frac{t_{op} n^2 \cdot p}{t_{op} n^2 + p \cdot n \cdot t_{data}/2}$$

$$E = \frac{S}{p} = \frac{1}{1 + (p/n)(t_{data}/2t_{op})}$$

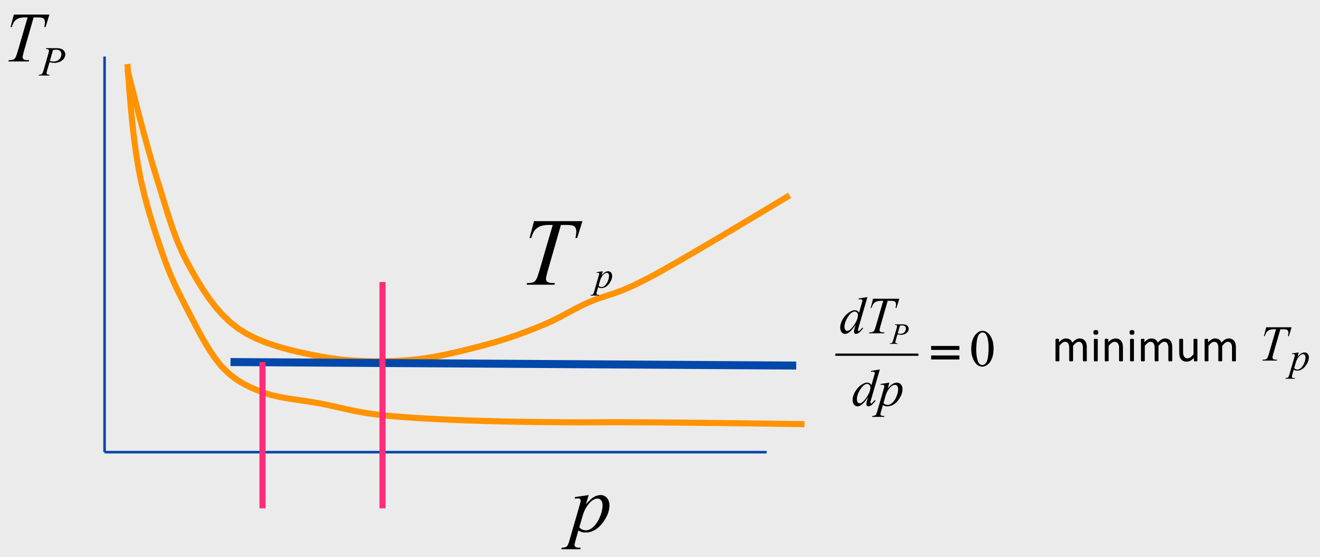
- So *p* must be small relative to *n* for efficiency
- Efficiency stays constant as long as p/n is constant

# Example: stencil-type computations (3/3)

Suppose $t_{data}=20*t_{op}$, n=1000



# Typical Performance Curve



$$E = 90\%$$

For stencil example
(n=1000): $p \approx 22$

# Performance evaluation: practical examples

# Hardware Performance metrics

- Clock frequency [GHz] = absolute hardware speed
    - *memories, CPUs, interconnects*
- Operational speed [GFLOPs]
    - *how many operations per cycle a machine can 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 …*
    - *FLOPs/Byte, FLOPs/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

2 chips * 4 cores * 4-way vectors * 2 FLOPs/cycle * 2.4 GHz = **154 GFLOPs**

- NVIDIA GTX 580 GPU

1 chip * 16 SMs * 32 cores * 2 FLOPs/cycle * 1.544 GHz = **1581 GFLOPs**

- ATI AMD Radeon HD 6970 GPU

1 chip * 24 SIMD engines * 16 cores * 4-way vectors * 2 FLOPs/cycle

* 0.880 GHz = **2703 GFLOPs**

# 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 B/s*

Examples:

| | |
|---|---|
| • Intel Core i7 DDR3: | 1.333 GHz * 2 * 64 = **21 GB/s** |
| • NVIDIA GTX 580 GDDR5: | 1.002 GHz * 4 * 384 = **192 GB/s** |
| • ATI HD 6970 GDDR5: | 1.375 GHz * 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 manufacturers 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 can achieve all of this*
  - *even difficult to write micro-benchmarks*
- hardware catalogue values are not realistic estimates of performance upper bounds

We need realistic estimates of what a platform can do when given a realistic workload => Roofline model

# Software metrics (3 P's)

***P**erformance metrics*

- Execution time
  - *Derive speed-up vs. best available sequential performance*
- Achieved GFLOPs:
  - *Count (FL)OPs, divide by execution time => FLOPS/s*
  - *Derive computational efficiency (i.e., utilization) =* $\frac{Achieved\ FLOPs}{Peak\ FLOPs}$
- Achieved GB/s:
  - *Count memory OPs, divide by execution time => B/s*
  - *Derive memory efficiency (i.e., utilization) =* $\frac{Achieved\ GB/s}{Peak\ GB/s}$

***P**roductivity and **P**ortability metrics*

- Programmability
- Production costs
- Maintenance costs

# Arithmetic intensity

- The (average) number of arithmetic (floating point) operations executed by a kernel per byte of memory accessed
- Ignore “overheads”
    - *loop counters*
    - *array index calculations*
    - *…*

Convert color into grayscale

```
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;
}}
```

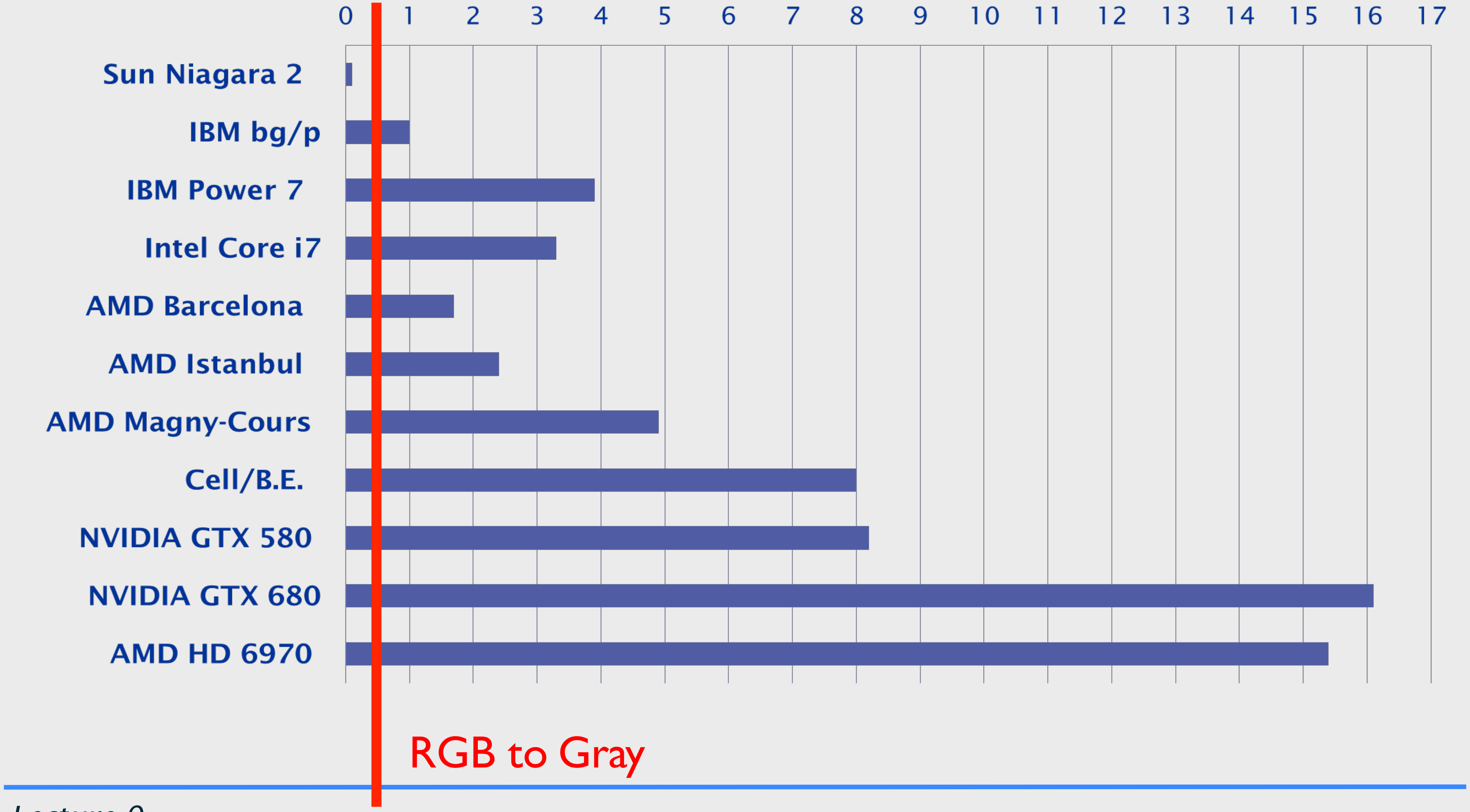
# Arithmetic intensity :

```
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;
}}
```

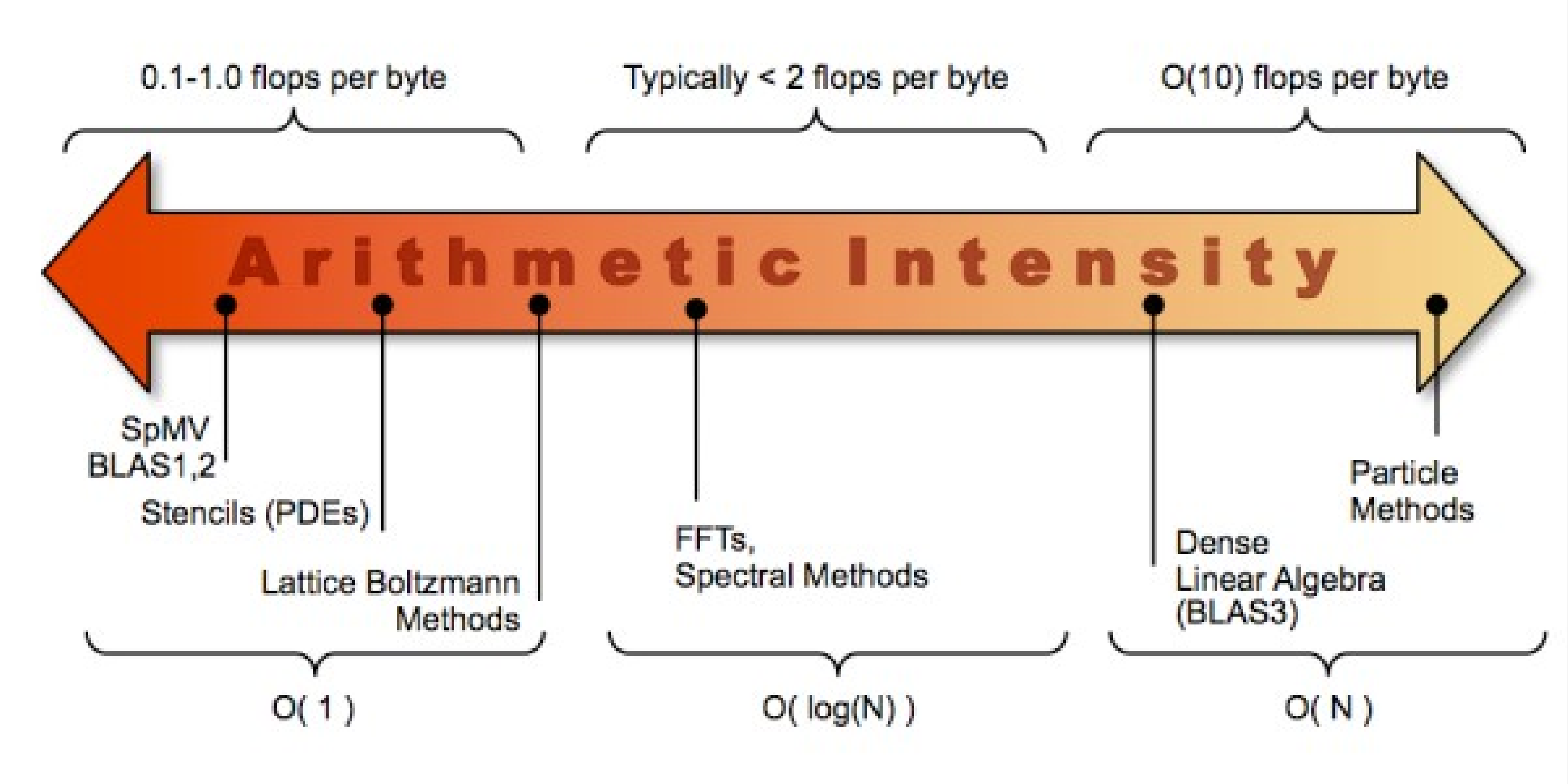
- What is the arithmetic intensity of the RGB-gray kernel?
- Is this a compute-intensive or a memory-intensive kernel?
- Is the kernel type (memory- or compute-intensive) dependent on the application, machine, or both?

# Compute or memory intensive?

Arithmetic intensity for several actual many-cores:



# AI for classes of applications



# Operational intensity

- An extension (i.e., refinement) of arithmetic intensity:

  The number of operations per byte of memory accessed

- Difference with Arithmetic Intensity
  - operations, not just arithmetic
  - caches are “skipped”
    - *count operations only*
    - *transfers not between processor and cache*
    - *but between cache and DRAM memory*

# Attainable performance

- Attainable GFlops/sec

= *min* ( Peak Floating-Point Performance,

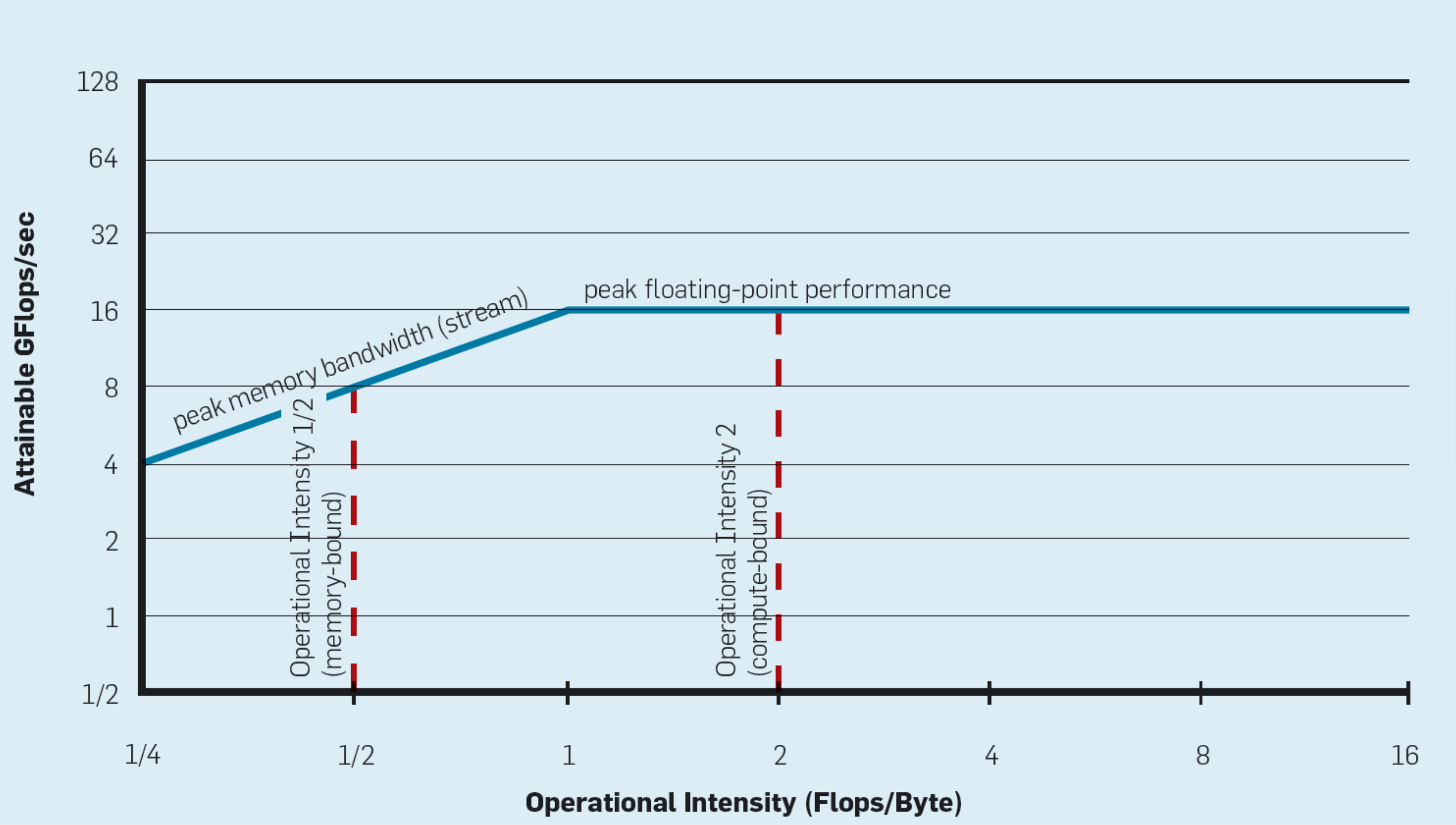
Peak Memory Bandwidth * Operational Intensity )

- To translate:
  - if an application is compute-bound =>

    performance is limited by peak performance
  - if an application is memory-bound =>

    performance is limited by the load it puts on the memory system

# The Roofline model



AMD Opteron X2 (two cores): 17.6 gflops, 15 GB/s, ops/byte = 1.17

# Attainable performance (cont'd)

- It is a measure of the performance an application can achieve
  - *more realistic because of the AI*
- 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)/$T_{exec\_A}$ (real execution)

AchievedBW(A,X) = MemOPs(A)/$T_{exec\_A}$ (real execution)

UtilizationCompute = AchievedCompute(A,X)/PeakCompute(X) < 1

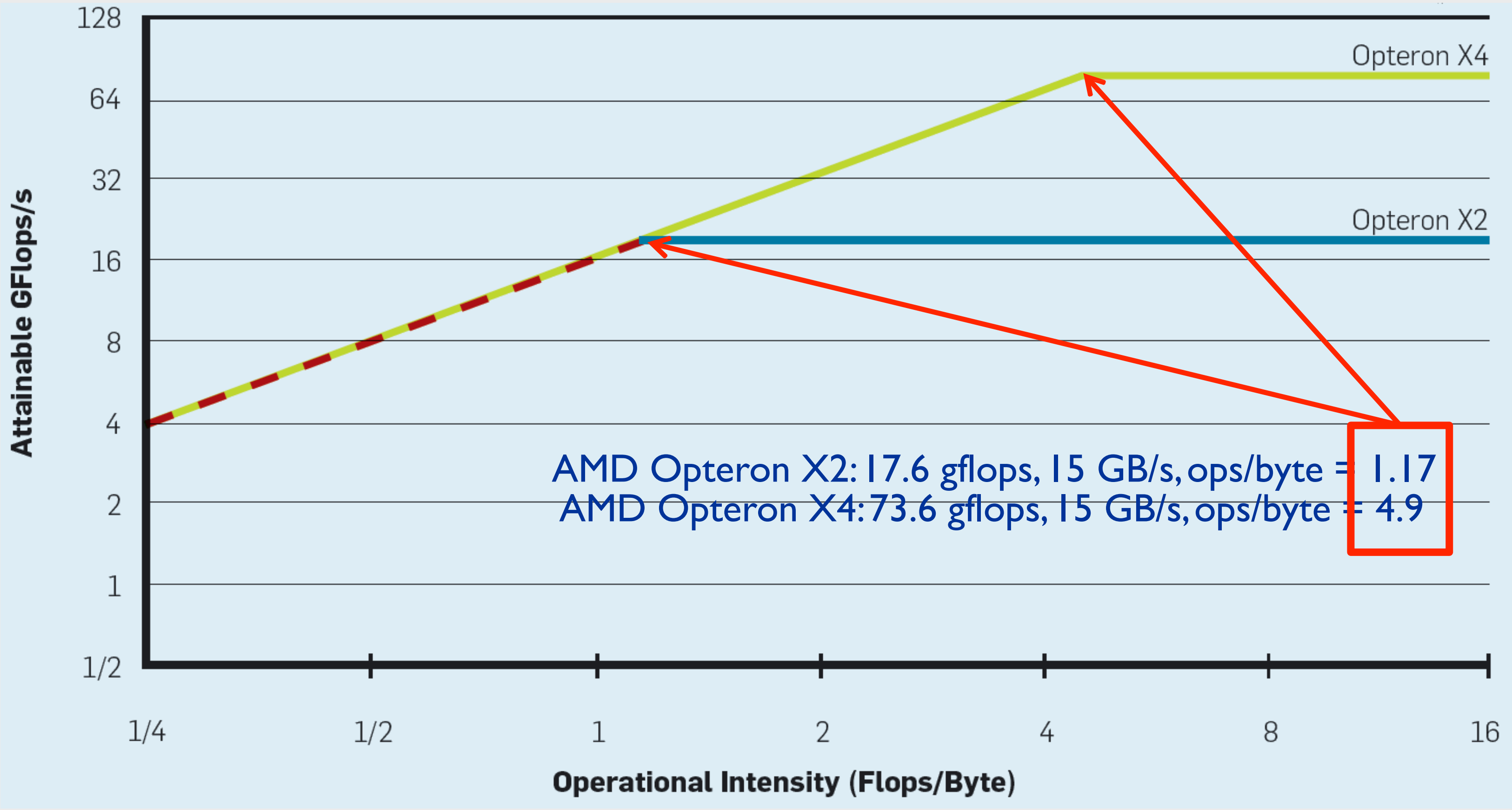
UtilizationBW=AchievedBW(A,X)/PeakBW(X) < 1

**Peak**$_{\mathbf{Compute}}$ **>= Roofline > Achieved**$_{\mathbf{Compute}}$

**Peak**$_{\mathbf{BW}}$ **? Achieved**$_{\mathbf{BW}}$

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

# Roofline: comparing architectures



AMD Opteron X2: 17.6 gflops, 15 GB/s, ops/byte = 1.17
AMD Opteron X4: 73.6 gflops, 15 GB/s, ops/byte = 4.9

# Use the Roofline model

Determine what to do first to gain performance

- increase memory streaming rate (fights mem-boundness)
  - *GPU: memory coalescing*
  - *CPUs: better caching*
- apply in-core optimizations (fights compute-boundness)
  - *vectorization*
- increase arithmetic intensity (fights mem-boundness)
  - *change your algorithm*
  - *think of new ways to reuse the data*