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

Performance Metrics

- Serial execution time: T_S
- Parallel execution time: T_P
- Overhead (p is # compute units) $T_O = p \cdot T_P - T_S$

Total overhead relevant for dedicated parallel processing

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

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

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

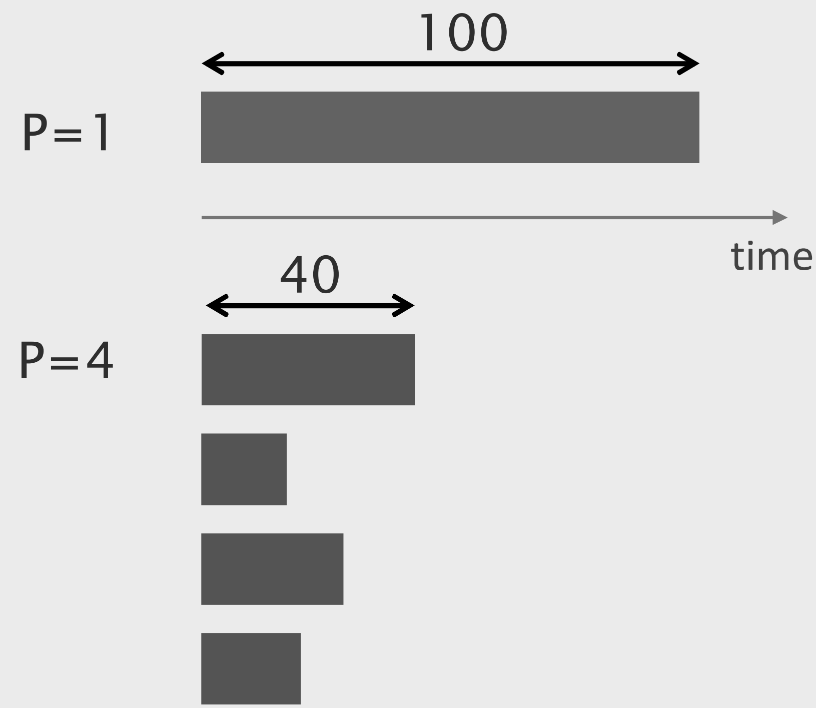
Efficiency

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

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

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

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



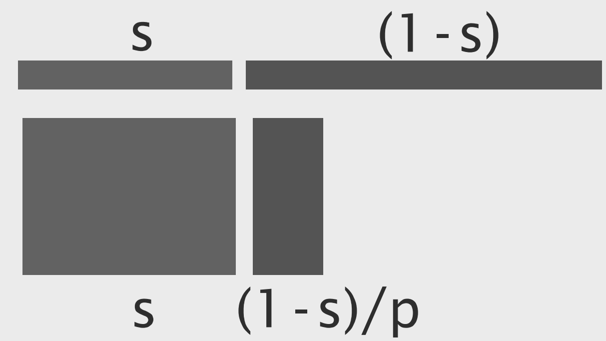
S=2.5
E=2.5/4=0.625

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*
  - *…*

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.

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?

$T_o = p \cdot T_P - W$ (definition of overhead)

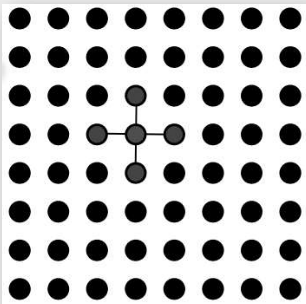
and

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

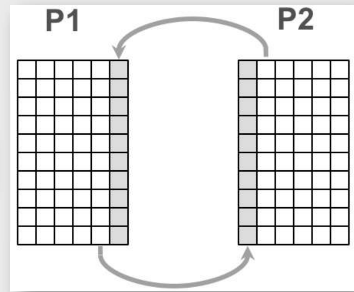
Conclusions:

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

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

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**

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**

Software metrics (3 P's)

Performance 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) =* $\frac{Achieved\ GB/s}{Peak\ GB/s}$

Productivity and ***P***ortability metrics

- Programmability
- Production costs
- Maintenance costs

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

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*