ICHEC
Irish Centre for High-End Computing

Parallel Performance

Basic Concepts

- FLOP: Floating Point Operation (addition, multiplication, square root, and so on)
- FLOP/sec, FLOP/s, FLOPS: a common performance metric
- FLOPS units
  - megaFLOPS (MFLOPS) 10^6
  - gigaFLOPS (GFLOPS) 10^9 single CPU performance
  - teraFLOPS (TFLOPS) 10^12 small-middle scaled supercomputers
  - petaFLOPS (PFLOPS) 10^15 we are here right now
  - exaFLOPS (EFLOPS) 10^18 the next milestone
- The most powerful supercomputers nowadays operate in petaFLOPS range (one quadrillion calculations per sec)
- The TOP500 table shows the 500 most powerful commercially available computer systems worldwide: https://www.top500.org/lists/top500/

Basic concepts

- **Peak Performance:**
  - The theoretical maximum performance
  - Little indication on how the system will perform in practice, can never be reached.
  - $[\#FLOPs/cycle_{max}] \times clock\ rate\ [Hz] \times \#\ cores \times \#\ sockets \times \#\ nodes$

- **Terminology:**
  - The **cores** perform FLOPs.
  - A **core** can do a certain number of **FLOPs** every time its internal clock ticks. These ticks are called cycles and measured in Hertz (Hz).
  - A **processor** contains one or more **cores**.
  - A **socket** holds one **processor**.
  - A **node** contains one or more **sockets**.
  - A "**chassis**" houses one or more **nodes**.

Execution time

- The CPU time spent by the system executing a particular task
- For compute intensive problems, the work needs to be distributed over multiple CPUs
- Therefore steps need to be done in parallel and the CPUs need to exchange information between each other rapidly

Speedup

The goal of speedup is to use P processors to solve a problem P times faster

Relative Speedup: compare the same (parallel) algorithm on one and multi-processor system

$$S_r(p) = \frac{T_{par}(1)}{T_{par}(p)}$$

Absolute Speedup: best sequential algorithm for the one processor system is compared to the best parallel algorithm for the multi-processor system

$$S_a(p) = \frac{T_{seq}}{T_{par}(p)}$$

Speedup

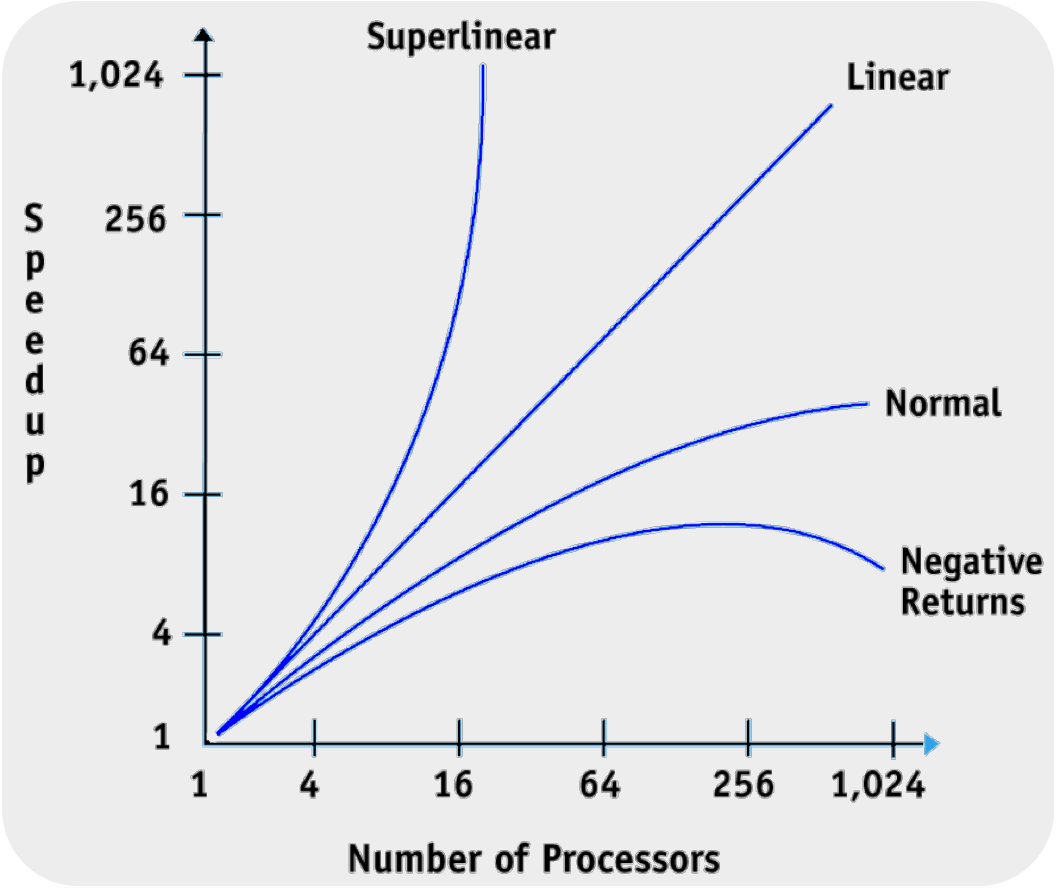
The execution time depends on what the program does!

A parallel program spends time in:

- Work
- Synchronization
- Communication
- Extra work (overheads)

A program implemented for a parallel machine is likely to do extra work (than a sequential program) even when running in a single processor machine!

Speedup


Superlinear
Linear
Normal
Negative Returns
Speedup
1,024
256
64
16
4
1
1
4
16
64
256
1,024
Number of Processors

Super Linear speedup

Sometimes P processors can achieve a speedup > P

- usually the result of improving an inferior sequential algorithm
- can legitimately occur because of cache and memory effects
  - More processors typically also provide more memory/cache
  - Total computation time decreases due to more page/cache hits

Limits to speedup and Amdahl's Law

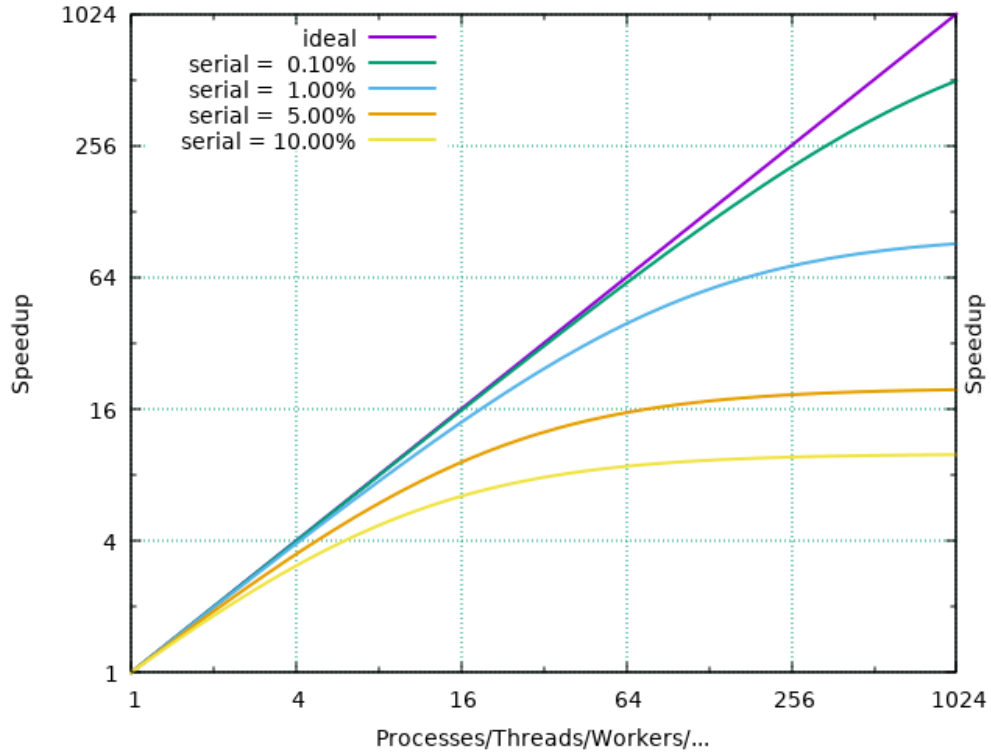
If I use two processors, shouldn't my program run twice as fast?

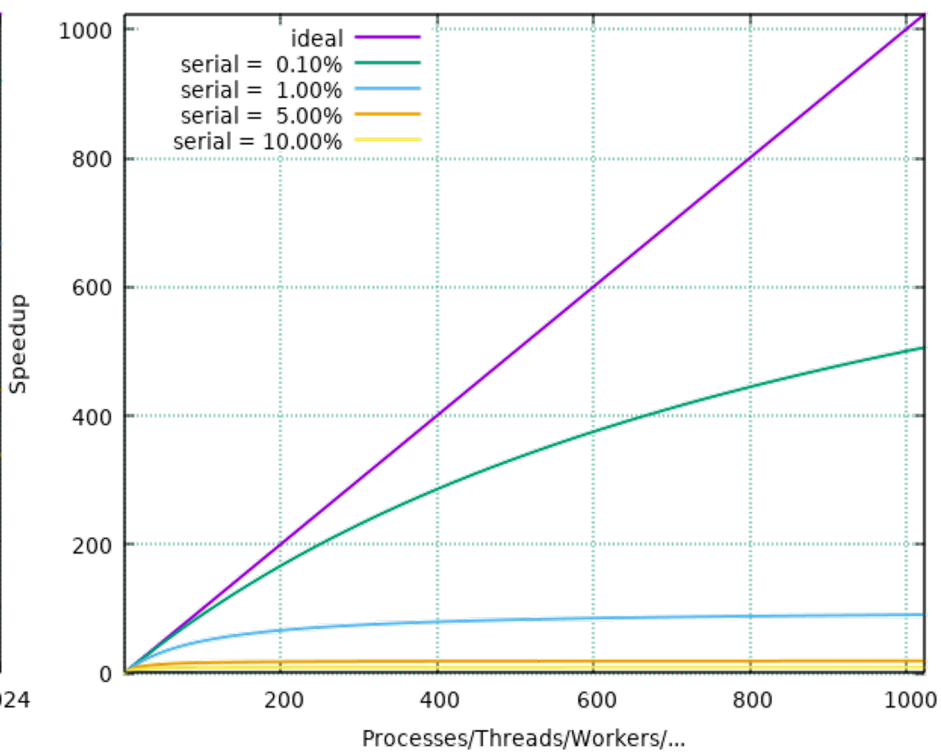
The speedup of a program using multiple processors in parallel computing is limited by the sequential fraction of the program.

Amdahl's Law: Let *f* be the fraction of operations in a computation that must be performed sequentially, where 0 <= *f* <= 1. The maximum speedup achievable by a parallel computer with *p* processor:

$$S_{max}(f,p) = \frac{1}{f + \frac{1-f}{p}}$$

Amdahl's Law


ideal
serial = 0.10%
serial = 1.00%
serial = 5.00%
serial = 10.00%
Speedup
Processes/Threads/Workers/...
1
4
16
64
256
1024


ideal
serial = 0.10%
serial = 1.00%
serial = 5.00%
serial = 10.00%
Speedup
Processes/Threads/Workers/...
0
200
400
600
800
1000

Limits to speedup

- Serial sections limit the parallel performance
- Parallel Overhead: the time required to coordinate parallel tasks and communicate information between processors
  - cost of starting a thread or process
  - cost of communicating shared data
  - cost of synchronizing
  - cost of extra (redundant) computation

…the effort expended on achieving high parallel processing rates is wasted unless it is accompanied by achievements in sequential processing rates of very nearly the same magnitude.
— Gene Amdahl

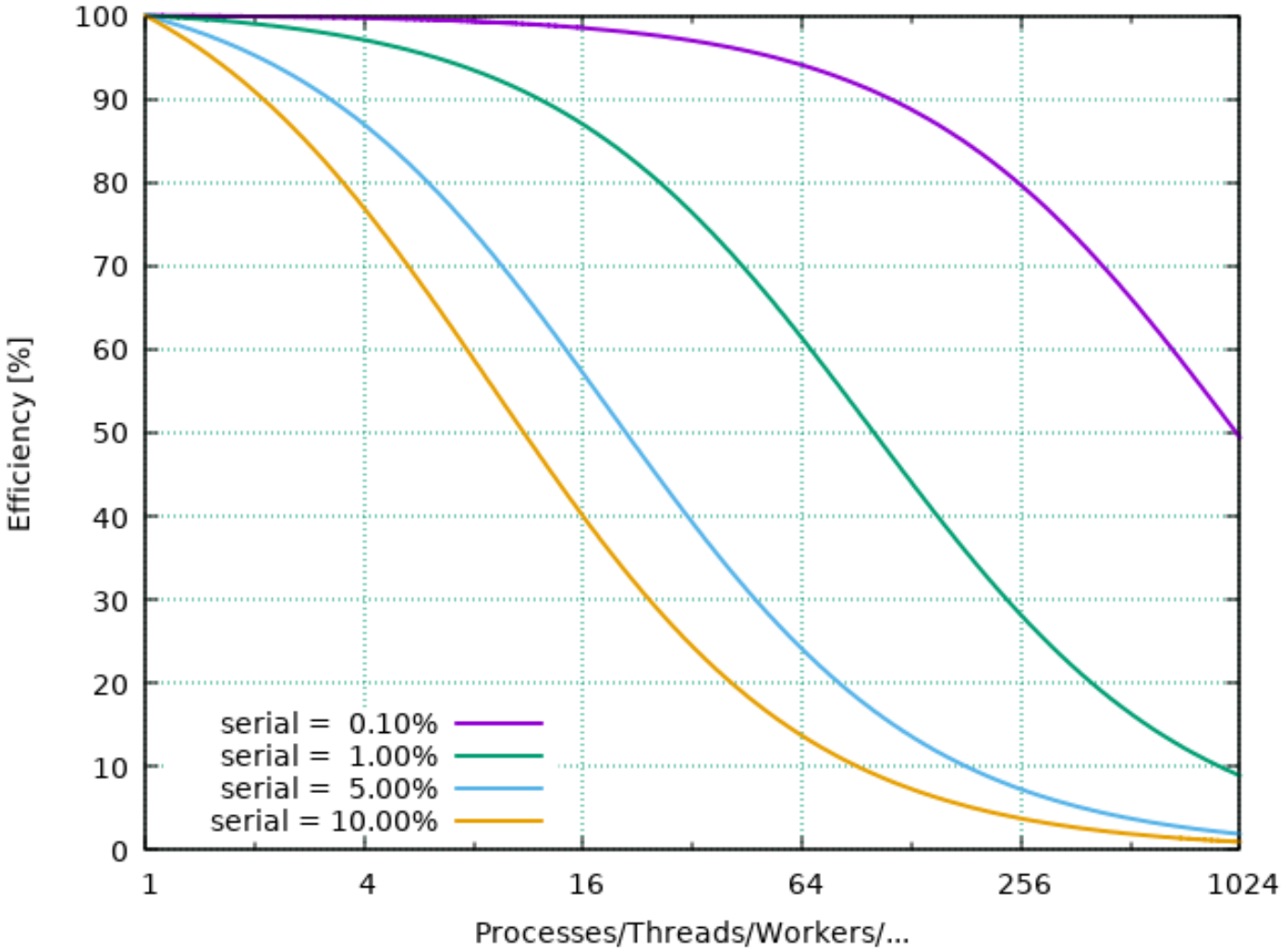
Efficiency

- A measure of how well-utilized the processors are in solving the problem, compared to how much effort is wasted in communication and synchronization
- $\mathrm{E(p)} <= 1$
- $\mathrm{E(p)} = 1$ if there is linear speedup

$$E_r(p) = \frac{S_r(p)}{p}$$

$$E_a(p) = \frac{S_a(p)}{p}$$

Efficiency


Efficiency [%]
100
90
80
70
60
50
40
30
20
10
0
1
4
16
64
256
1024
Processes/Threads/Workers/...
serial = 0.10%
serial = 1.00%
serial = 5.00%
serial = 10.00%

Scalability

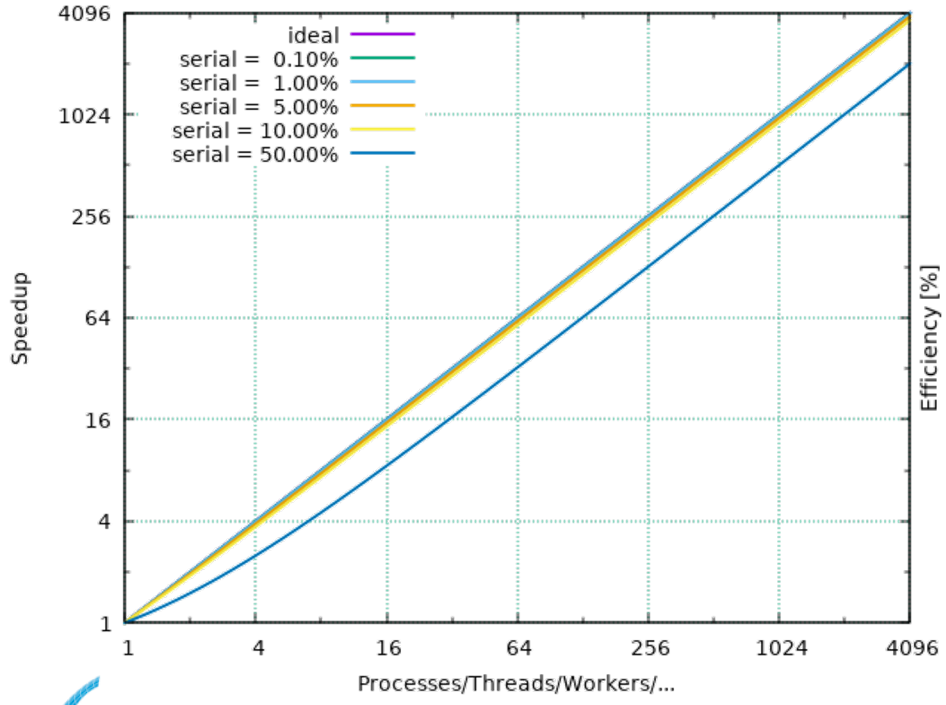
- The performance when the size of the problem or the number of processors changed.
- Assume a problem of size N using P processors takes time T
  - Strong Scaling: As N remains fixed, kP processors solve the problem in T/k time.
  - Weak Scaling: For a problem of size kN, kP processors solve the problem in time T.

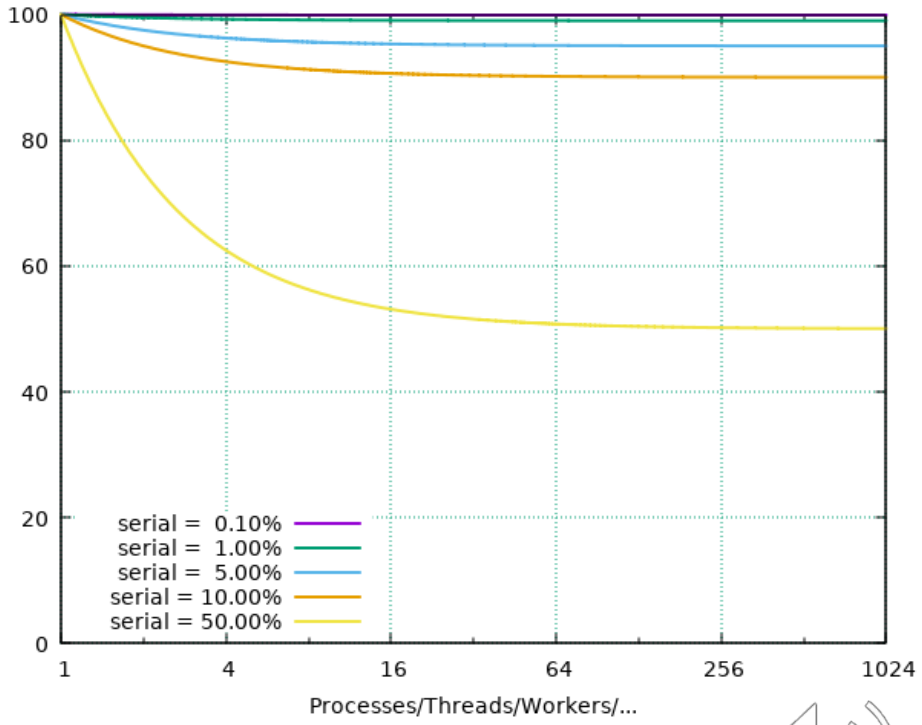
Gustafson-Barsis' Law

…speedup should be measured by scaling the problem to the number of processors, not by fixing the problem size.

— John Gustafson

$$S(f, p) = f + p*(1 - f)$$





ICHEC
Irish Centre for High-End Computing

Communication

For some problems, increasing the number of processors will:

- Decrease execution time attributable to computation
- May increase execution time attributable to communication

Time required for communication is dependent upon system communication bandwidth parameters.

The time (T) required to send W words between any two processors is given by:

$$T = L + \frac{W}{B}$$

where

- L = Latency: the fastest time in which a single byte can be sent
- B = Bandwidth: how much data can be sent per unit time

Load Imbalance

- Amdahls and Gustafson's laws assume that all processors are equally busy
  - Not always true

Core	1	2	3	4	Total
# tasks	6	1	3	2	12

  - Takes 6 minutes to complete as other cores waiting for core 1 to finish
  - If work distributed evenly → 3 minutes

- Make best use of processors at hand before increasing number of processors

Quantifying Load Imbalance

- Define Load Imbalance Factor

$$LIF = maximum\ load\ /\ average\ load$$

  - For perfectly balanced problems $\mathrm{LIF} = 1.0$, as expected
  - Usually; $\mathrm{LIF} > 1.0$
  - LIF tell you how much faster your calculation could be with a balanced load
- Core tasks
  - $\mathrm{LIF} = 6/3 = 2$
  - Initial time = 6 mins
  - Best time = $\mathrm{LIF}/2 = 3$ mins

Parallel performance factors