Introduction to Parallel Computing

ING2-GSI-MI Architecture et Programmation Parallèle

Juan Angel Lorenzo del Castillo

juan-angel.lorenzo-del-castillo@cyu.fr

CY Cergy Paris Université

2023-2024

Table of Contents

Table of Contents

Introduction

- The world is highly **complex**
- For centuries **science** has sought to understand the world:
  - Theory
  - Experiments
  - Simulation

Theory

Theory (from Greek *theorein*, « observe, inspect »)

A (scientific) theory is a well-substantiated explanation of some aspect of the natural world, based on a body of facts that have been repeatedly confirmed through observation and experiment. Such fact-supported theories are not "guesses" but reliable accounts of the real world. Source : Wikipedia

- Theories are developed to understand the way things behave
  - Check that it correctly describes known situations
  - Used to predict what will happen in new situations
- In the physical sciences, most theories comprise a set of mathematical equations.
- Typical predictions :
  - *The bridge will collapse if the load exceeds 100 tonnes*
  - *Planets will rotate around the sun in elliptical orbits*
  - *Earth's climate will warm up if enough CO 2 is produced*

Theory

Theory (from Greek *theorein*, « observe, inspect »)

A (scientific) theory is a well-substantiated explanation of some aspect of the natural world, based on a body of facts that have been repeatedly confirmed through observation and experiment. Such fact-supported theories are not "guesses" but reliable accounts of the real world. Source : Wikipedia

- Theories are developed to understand the way things behave
  - Check that it correctly describes known situations
  - Used to predict what will happen in new situations
- In the physical sciences, most theories comprise a set of mathematical equations.
- Typical predictions :
  - *The bridge will collapse if the load exceeds 100 tonnes*
  - *Planets will rotate around the sun in elliptical orbits*
  - *Earth's climate will warm up if enough CO 2 is produced*

Experiments

- Do something in the real world...
  - take measurements
  - make sure it is repeatable
- ... that may be used to validate theory
  - check against predictions from theory
- ... or to develop a theory
  - collect lots of data and look for patterns

(Computational) Simulations

- **Computer science** is the study of computers
- **Computational science** is performing scientific studies in many disciplines using computers

Why do simulations? :

- to deal with problems which are
  - too large
  - too small
  - too complex
  - too expensive
  - too dangerous
  - ...
- To validate theories (by obtaining predictions)
- Use computers to get the results from a known theory (no need for experiments)

Simulations in real life

Most of the time we will use **models**

Model

- *A model is a task-driven, purposeful simplification and abstraction of a perception of reality, shaped by physical, legal, and cognitive constraints.*
- *Representation of the reality that enables it to be processed by computing tools* Source : Wikipedia

Limitation : How to obtain an accurate representation of the model

- Models are never perfect
- Computer power is limited
- Accurate algorithms may be too time consuming
- Programming a sophisticated algorithm may be difficult

Simulations in real life

Most of the time we will use **models**

Model

- *A model is a task-driven, purposeful simplification and abstraction of a perception of reality, shaped by physical, legal, and cognitive constraints.*
- *Representation of the reality that enables it to be processed by computing tools* Source : Wikipedia

Limitation : How to obtain an accurate representation of the model

- Models are never perfect
- Computer power is limited
- Accurate algorithms may be too time consuming
- Programming a sophisticated algorithm may be difficult

Simulations in real life

Most of the time we will use **models**

Model

- *A model is a task-driven, purposeful simplification and abstraction of a perception of reality, shaped by physical, legal, and cognitive constraints.*
- *Representation of the reality that enables it to be processed by computing tools* Source : Wikipedia

Limitation : How to obtain an accurate representation of the model

- Models are never perfect
- Computer power is limited
- Accurate algorithms may be too time consuming
- Programming a sophisticated algorithm may be difficult

Simulations in real life

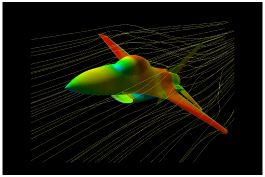
Analytical vs Computational

- The analytical method
  - reduces to a small number of variables or degrees of freedom
  - simplifies the problem (e.g. by linearisation)
  - studies situations close to known, exact solutions
- The computational method
  - uses a finite but large number of degrees of freedom
  - simulates a system for a large but finite length of time
  - generates many numerical realisations of the system
  - can repeat the same simulation many times

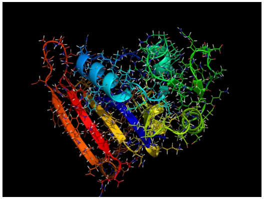
Simulations in real life

Discretisation

- We often have a continuous system to simulate
  - e.g. study flow of air around an aeroplane
  - e.g. dispersion of chemical elements emitted by a factory
  - e.g. weather forecast
- A computer requires a discrete model
  - to make evolve the system through discrete time steps
  - it stores a finite number of variables (e.g. velocity) as a finite number of points in a grid (samples).



Source : www.psc.edu



Source : www.gromacs.org

Simulations in real life

Advantages

- Any particular simulation can be replayed
  - can examine different features of exactly the same experiment
- The values of all variables are known
  - e.g. what is the temperature inside a working jet engine?
- Can study phenomena that are
  - too small, too large, too dangerous or too far away to experiment upon directly
  - e.g. the internal structure of the earth, the expansion of the universe.

Simulations in real life

Disadvantages

- Must sacrifice mathematical purity to solve real problems
  - No simple answers like distance = speed $\times$ time
- Introduces an extra level of approximation
  - the original theory is an approximation of reality
  - A computer model, its implementation is often an approximation of the theory
  - so it is crucial to understand any errors that this introduces
- It is an experimental method
  - how do we know that it is working correctly?
  - must test, calibrate and verify (like any piece of lab equipment)

Table of Contents

Parallelism

The idea of performing calculations in parallel was first suggested by Charles Babbage in the 19th Century... but was not technically possible at the time.

How can we improve the performance of a program?

- Design faster processors
  - Physic and economic limits
- Design algorithms that perform better
  - But always limited by the processor performance
- Ask for help :-)
  - Solve a problem by using several processors
  - Exploit the problem inherent parallelism

High Performance Computing (HPC)

- Parallel Architectures
- Parallel Algorithms
- Parallel Programming

Parallel Architectures

Flynn's Taxonomy

Classification of architectures by instruction stream and data stream (Michael Flynn, 1966)

- **SISD** : *Single Instruction Single Data*
- **SIMD** : *Single Instruction Multiple Data*
- **MISD** : *Multiple Instructions Single Data*
- **MIMD** : *Multiple Instructions Multiple Data*
  - Several independent processors capable of executing separate programs on different data

Parallel Architectures

Flynn's Taxonomy

Classification of architectures by instruction stream and data stream (Michael Flynn, 1966)

- **SISD** : *Single Instruction Single Data*
- **SIMD** : *Single Instruction Multiple Data*
- **MISD** : *Multiple Instructions Single Data*
- **MIMD** : *Multiple Instructions Multiple Data*
  - Several independent processors capable of executing separate programs on different data

Parallel Architectures

Flynn's Taxonomy

Classification of architectures by instruction stream and data stream (Michael Flynn, 1966)

- **SISD** : *Single Instruction Single Data*
- **SIMD** : *Single Instruction Multiple Data*
- **MISD** : *Multiple Instructions Single Data*
- **MIMD** : *Multiple Instructions Multiple Data*
  - Several independent processors capable of executing separate programs on different data

Parallel Architectures

Flynn's Taxonomy

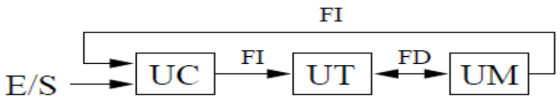
Classification of architectures by instruction stream and data stream (Michael Flynn, 1966)

- **SISD** : *Single Instruction Single Data*
- **SIMD** : *Single Instruction Multiple Data*
- **MISD** : *Multiple Instructions Single Data*
- **MIMD** : *Multiple Instructions Multiple Data*
  - Several independent processors capable of executing separate programs on different data

Parallel Architectures

Flynn's Taxonomy

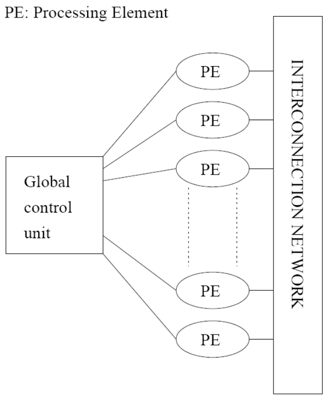
- **SISD** : *Single Instruction Single Data*
  - One CPU, one memory unit, no parallelism: Von Neumann's architecture.
  - Each operation is performed on a single piece of data at a time.
  - Classic sequential programming.
  - The control unit (UC), which receives its instruction stream (FI) from the memory unit (UM), sends the instructions to the processing unit (UT) which performs its operations on the data stream (FD) from the Memory unit.



Parallel Architectures

Flynn's Taxonomy

- **SIMD** : *Single Instruction Multiple Data*
  - Every processor synchronously executes the same instructions on different data
  - Each processor has its own memory where it keeps its data
  - Processors can communicate with each other
  - Usually thousands of simple processors.
  - Suitable for regular processing, such as matrix calculations on solid matrices or image processing
  - Vector processors, GPUs, Intel's MMX and SSE instructions.

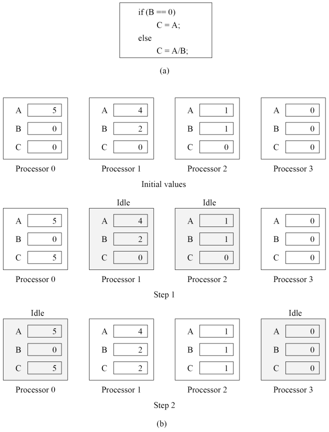


Grama, Gupta et al. Introduction to Parallel Computing 2nd ed. Addison-Wesley 2003.

Parallel Architectures

Flynn's Taxonomy

- **SIMD** : *Single Instruction Multiple Data*



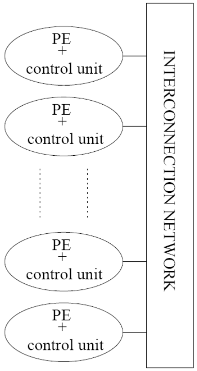
Executing a conditional statement on an SIMD computer with four processors: (a) the conditional statement; (b) the execution of the statement in two steps.

Grama, Gupta et al. Introduction to Parallel Computing 2nd ed. Addison-Wesley 2003.

Parallel Architectures

Flynn's Taxonomy

- Shared-memory **MIMD**
  - Each processor has access to the memory as a global address range
  - From the user's point of view, it looks like a single machine
  - Any modification made in memory is seen by all processors
  - Communications among processors are made via write/reads to memory (*coherent* cache memories)



Grama, Gupta et al. Introduction to Parallel Computing 2nd ed. Addison-Wesley 2003.

- Distributed-memory **MIMD**
  - Each processor has its own local memory and its own operating system
  - Processors communicate via explicit message passing
  - Highly scalable architecture

Parallel Architectures

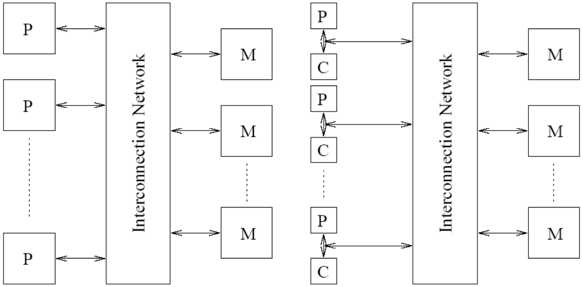
Shared-memory MIMD

- Types or shared-memory MIMD architectures:
  - **UMA** (*Uniform Memory Access*) :
    - Base for SMP : access to the whole memory
    - Cache coherence (*snoopy* protocols)
    - Limited number of processors (bus contention problems)
  - **NUMA / cc-NUMA** (*(cache-coherent) Non-uniform Memory Access*) :
    - Each processor has a local, low-latency memory
    - Each processor has access to every processor's memory units (remember it is shared memory)
    - Hence, the access time to remote memory units is higher than to its local memory

Parallel Architectures

Shared-memory MIMD

- **UMA** (*Uniform Memory Access*)

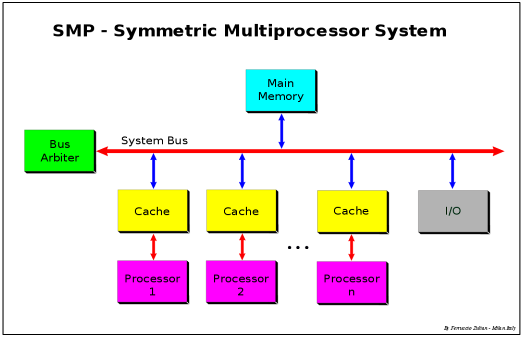


Grama, Gupta et al. Introduction to Parallel Computing 2nd ed. Addison-Wesley 2003.

Parallel Architectures

Shared-memory MIMD

- SMP (Symmetric MultiProcessing): A Shared-memory MIMD in which each processor has **equal** access to all parts of memory (also I/O, etc.)
  - Same latency and bandwidth

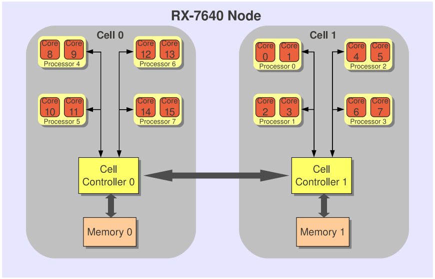


Source : Wikipedia

Parallel Architectures

Shared-memory MIMD

- **NUMA** (*Non-uniform Memory Access*)
  - Example: rx7640 node from the Finisterrae supercomputer (CESGA)

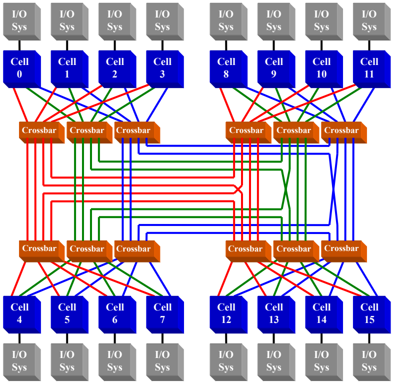


Source : Juan Angel Lorenzo, PhD thesis.

Parallel Architectures

Shared-memory MIMD

- **NUMA** (*Non-uniform Memory Access*)
  - Example: Superdome from the Finisterrae supercomputer (CESGA)



Source : Hewlett-Packard.

Parallel Architectures

Distributed-memory MIMD

- Each node has a processor with its own local memory and operating system (*Server cluster*)
- Processors are connected by some high-speed interconnect mechanism (Infiniband, Ethernet,...)
- Communication via explicit message passing (MPI, etc.), like sending emails to each other.
- Highly scalable architecture: allows **MPP** (*Massively Parallel Processing*)

Source : Beowulf Project.

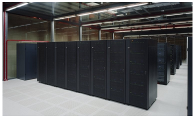
Parallel Architectures

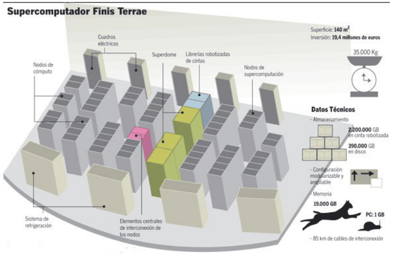
Hybrid Architectures

- Most of modern supercomputers are hybrid machines
  - Several SMP-UMA or NUMA nodes
  - Inter-connected by high-speed networks
  - The interaction between their components is not always obvious

HP Alphaserver

IBM BlueGene/L


HP FINISTERRAE


Supercomputador Finis Terrae
Cuadros eléctricos
Superdome
Librerías robotizadas de cintas
Nodos de cómputo
Nodos de supercomputación
Sistema de refrigeración
Elementos centrales de interconexión de los nodos
Superficie: 140 m²
Inversión: 19,4 millones de euros
35.000 Kg
Datos Técnicos
- Almacenamiento
2.200.000 GB en cinta robotizada
390.000 GB en disco
- Configuración modularizable y ampliable
- Memoria
19.000 GB
PC: 1 GB
- 85 km de cables de interconexión

Source : CESGA.

Table of Contents

Parallel Programming

SPMD paradigm (*Single Program Multiple Data*)

- **Shared memory**
  - In this course: OpenMP (*Open MultiProcessing*)
    - Multi-processing API
    - Parallelisation libraries and compiler directives
    - Very low overhead
- **Distributed memory**
  - Passing-message model
  - “Send” and “receive” primitives
  - Communications can be **synchronous** (an answer is expected) or **asynchronous** (*send and forget*)
  - In this course : MPI (*Message Passing Interface*)

Parallel Programming

Parallelisation steps

1. Decompose
   - Job 1, job 2, job 3,...
   - **Job:** piece of the total work
   - To be divided among the processors
   - **Load Balance**
2. Assign
   - *Process 0 ... Process N* or *Thread 0 ... Thread N*
   - How the jobs are divided
   - Together with decomposition called partitioning
   - Dynamic or static
   - Somewhat independent of the architecture
3. Coordinate
   - Messages, barriers, waits
4. Map
   - Implement on your system
   - Choose were to execute each process

Table of Contents

Performance Metrics

We want to measure the performance of our system (architecture, program, etc.) to quantify how much a modification that we made improved (or worsened) the performance.

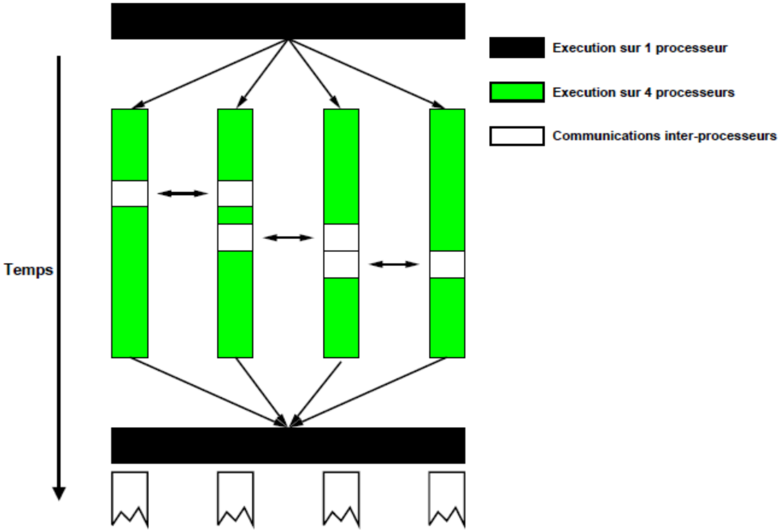
How to measure the performance

- Speed-up (Accélération).
- Efficiency (Efficacité).
- Load Balance.

Some concepts:

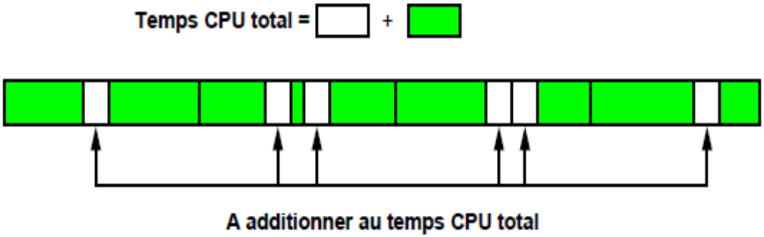
- Scalability.
- Latency and Bandwidth.
- T_1 (n): time to solve a problem of size n on 1 processor.
- T_p (n): time to solve the same problem on p processors.

Execution Time

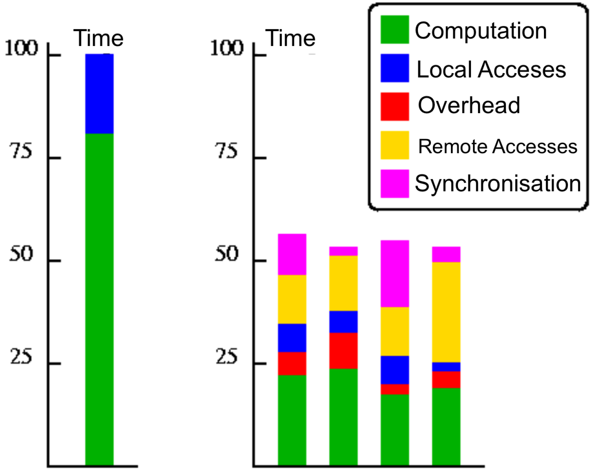

Execution sur 1 processeur
Execution sur 4 processeurs
Communications inter-processeurs
Temps

Execution Time

Communication overhead added to the CPU time



Execution Time



Execution Time

Measures of computer performance

- MIPS: Million Instructions Per Second
- FLOPS: Floating Point Operations per Second (GFLOPS)
- IPC: Instructions per cycle
- CPI: Cycles/Instructions.

Speedup (Accélération)

- How the performance changes when improvements are made to the program.
- **T_1 (n):** time to solve a problem of size n on 1 processor (before).
- **T_p (n):** time to solve the same problem on p processors (after).

$$\textbf{S}peedup = \frac{T_1(n)}{T_p(n)}$$

Ideal scenario: S = p

Efficiency (Efficacité)

- There are typically portions of a program inherently sequential. They cannot be parallelised.
- And there are usually some idle processors.

$$\mathbf{E}\mathit{fficiency} = \frac{S}{p}$$

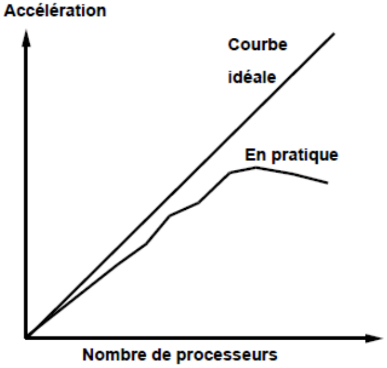
- Usually defined in %
- Ideal scenario: $S = p$ which means $E = 100\,\%$

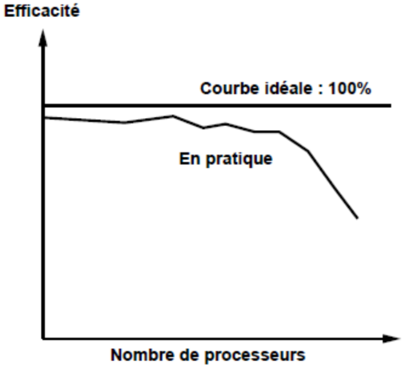
Speedup and Efficiency

A good speedup is not the same as good efficiency:

- $T_1(n) = 100\ sec.$
- $T_4(n) = 50\ sec. \rightarrow S_4 = 2 \rightarrow E_4(n) = 0{,}5$
- $T_8(n) = 33\ sec. \rightarrow S_8 = 3 \rightarrow E_8(n) = 0{,}38$

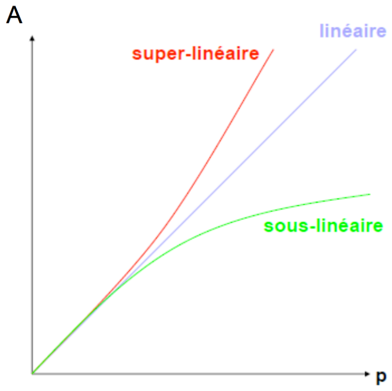
Speedup and Efficiency

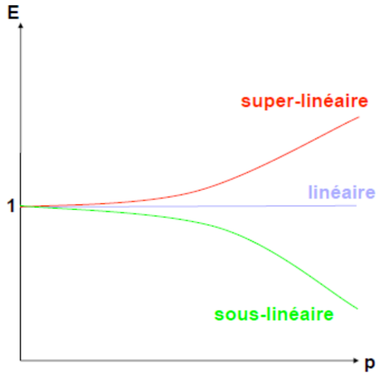




Speedup and Efficiency

Other Behaviours:





Superlinear speedup

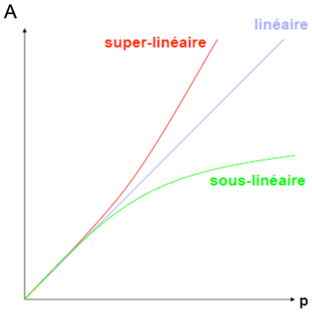
- Uncommon.
- Usually has something to do with the use of the memory hierarchy.
- Other example may be in loops. Iterate N elements using an index *i*:
  - In sequential, *i* is compared and incremented N times.
  - In a SIMD, *i* is compared and incremented less than N times.
  - Sequential: $N = 80 \rightarrow$ loop through 80 elements$\rightarrow 80 \times (i + 1)$
  - Vector size 8: $N = 80 \rightarrow$ loop through 10 elements$\rightarrow 10 \times (i + 1)$
- Computer does less work.

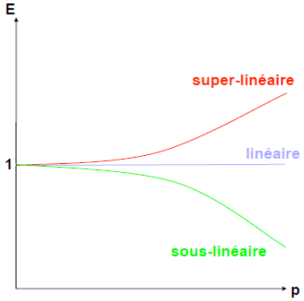
Load Balance

- Workload distribution
- In our case, need to keep processors busy.
- Two kinds of scheduling:
  - *Dynamic:* distribution changes during execution
  - *Static:* distribution is decided and fixed at the beginning

Scalability

- Capability of a system, network, or process to handle a growing amount of work.
- In our case, whether an algorithm keeps its speedup as *p* increases.





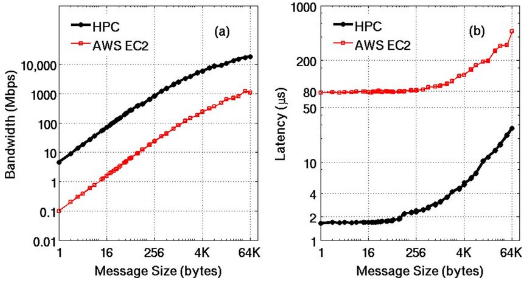
Scalability

How to improve scalability:

- Minimise communications.
- Exploit data locality.
- Balance the load.
- Minimise synchronisation.

Latency and bandwidth

- **Latency** is a time interval between the moment when the message is sent and a reply is received. Typically measured in seconds.
- **Bandwidth** (*bande passante*) measures the maximum throughput of a computer network. Typically measured in MBytes/sec.
- Latency is important when the messages are small
- Bandwidth is important for large messages



Source : Xiuhong Chen et al. Running climate model on a commercial cloud computing environment:

A case study using Community Earth System Model (CESM) on Amazon AWS

Amdahl's Law

The non-parallelisable part of an algorithm limits its performance and implies an upper limit to its scalability.

Gives the **theoretical speedup** in **latency** of the execution of a task at **fixed workload** that can be expected of a system whose resources are improved.

- More of a rule than a law.
- It considers that:
  - A program has a fraction F_p of parallelisable code.
  - The non parallelisable part is $1 - F_p$.

Amdahl´s Law

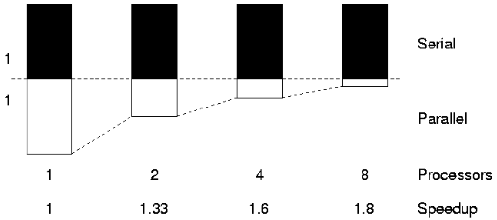
Example

A program has $F_p = 80\,\%$, with an ideal speedup of 1. Consider a sequential execution time of 100 seconds:

- $np = 1 \rightarrow T(1) = T_{parallel} + T_{sequential} \rightarrow 100 = 80 + 20$
- $np = p \rightarrow T(p) = T_{parallel} / S + T_{sequential} \rightarrow T(p) = 80/p + 20$

It does not matter how large p is, $T(p) > 20 = T_{sequential}$

- $S(p) = T(1)/T(p) = 100/(80/p + 20) < 5$
- $E(p) = S(p)/p < 5/p \rightarrow lim(E(p))_{p\rightarrow\infty} = 0$



Source : EPCC

Amdahl's Law

Equation

Amdahl's law can be formulated the following way:

$$S_{latency}(s) = \frac{1}{(1 - F_p) + \frac{F_p}{s}}$$

where

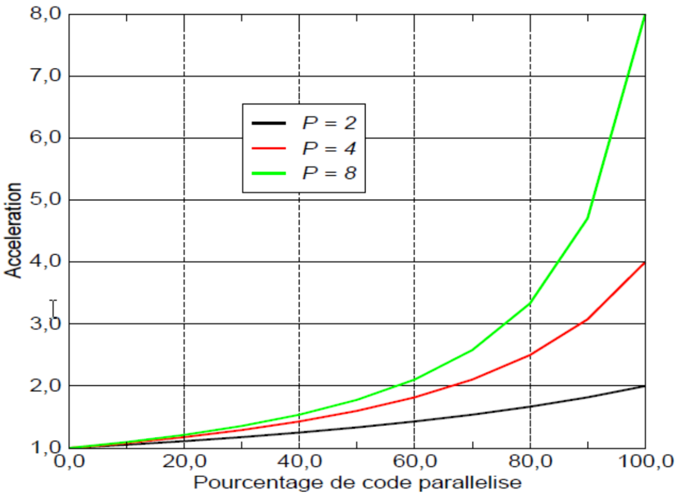
- $S_{latency}$ is the theoretical speedup of the execution of the whole task
- s is the speedup of the part of the task that benefits from improved system resources (the number of processors used)
- F_p is the proportion of execution time that the part benefiting from improved resources originally occupied

Furthermore,

$$S_{latency}(s) \leq \frac{1}{1 - F_p}$$

$$\lim_{s \to \infty} S_{latency}(s) = \frac{1}{1 - F_p}$$

Amdahl's Law


8,0
7,0
6,0
5,0
4,0
3,0
2,0
1,0
Acceleration
P = 2
P = 4
P = 8
0,0
20,0
40,0
60,0
80,0
100,0
Pourcentage de code parallelise

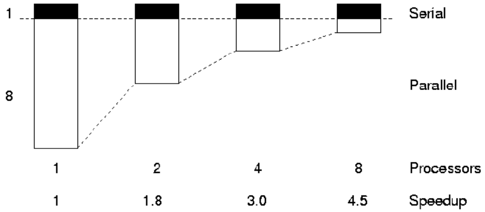
Amdahl’s Law

In many applications the sequential part is less than 0.1 %

Gustafson's Law

Similar to Amdahl's law, but it considers a fixed time. Need larger problems for larger numbers of CPUs. The parallelisation increases the problem size instead.

- Most people do not care about 1 minute.
- A sequential algorithm solves a problem in 1 minute.
- The parallel version of the algorithm solves the problem in 1 minute.
- But the parallel algorithm can solve the problem better, with more detail.



Source : EPCC

Gustafson´s Law

Equation

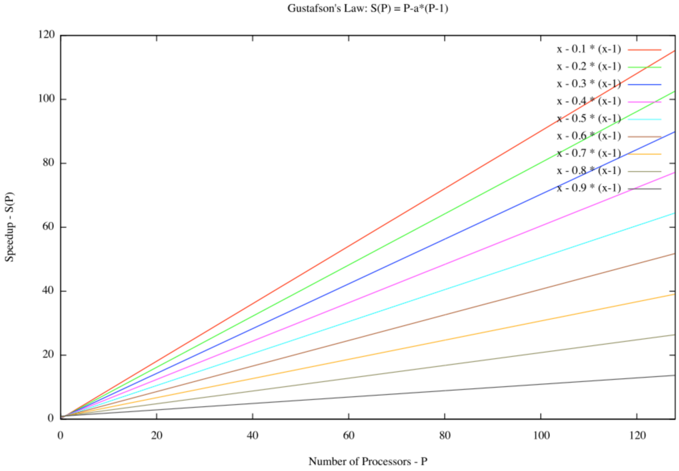
Gustafson’s law can be formulated the following way:

$$\begin{aligned} S_{latency}(s) &= (1 - F_p) + s \cdot F_p \\ &= 1 + (s - 1) \cdot F_p \end{aligned}$$

where

- $S_{latency}$ is the theoretical speedup of the execution of the task **with parallelism** (scaled speedup)
- s is the speedup of the part of the task that benefits from improved system resources (the number of processors used)
- F_p is the proportion of the execution **workload** of the whole task concerning the part that benefits from the improvement of the resources (parallelisation) of the system *before the improvement*

Gustafson´s Law


Gustafson's Law: S(P) = P-a*(P-1)
Speedup - S(P)
Number of Processors - P
x - 0.1 * (x-1)
x - 0.2 * (x-1)
x - 0.3 * (x-1)
x - 0.4 * (x-1)
x - 0.5 * (x-1)
x - 0.6 * (x-1)
x - 0.7 * (x-1)
x - 0.8 * (x-1)
x - 0.9 * (x-1)