Parallel programming tools and Portable, flexible and parallel I/O (HDF5)

Cecilia Jarne

cecilia.jarne@unq.edu.ar

CONICET

Universidad Nacional de Quilmes

WTPC
Workshop en técnicas de programación científica

High Performance Computing

Summary:

- The basic ideas.
- Parallel architectures.
- Software Implementations.

What is High Performance Computing?

- HPC = High Performance Computing = Efficiency.
- HPC: I care how quickly I get an answer.
- HPC: High productivity.
- HPC: Old Software + New Hardware.

What is High Performance Computing?

¿Where?

- Smartphone
- Desktop/laptop
- Clúster
- Supercomputer
- In the cloud

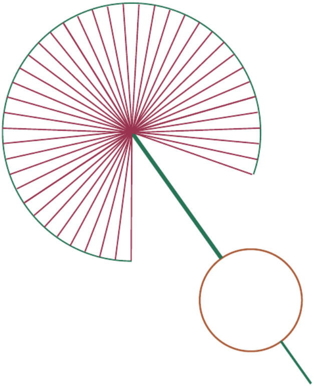
¿When?

- Take advantage of the hardware we have.
- Decide what hardware to buy.
- Obtain results from extreme simulations.

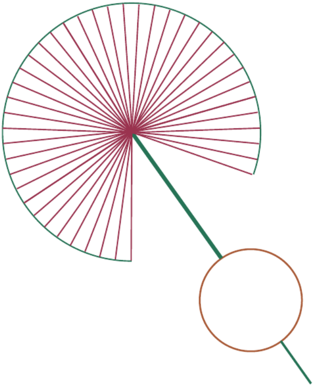
What we resign?

Friendly interfaces, reusable and portable software ...

HPC on a desktop



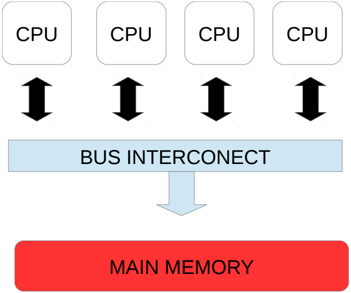
Processors (no time to lose).
Connections (key).
OS choose how it connects.



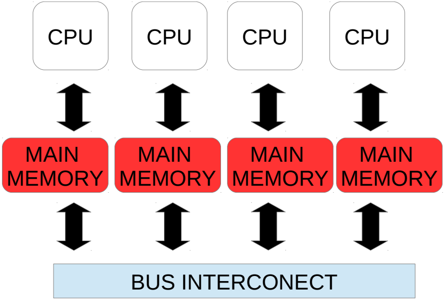
Computers (no time to waste).
Connections (key).
Master node chooses how to connect.

Architecture of a cluster

Symmetric MultProcessors



NonUniform Memory Access



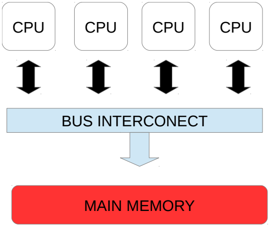
Shared memory

Advantages:

- Easy for the programmer.
- Sharing data is faster and more direct.

Disadvantages:

- Poor scalability.
- Synchronization by the programmer.
- More difficult and expensive to design and produce machines with shared memory as the number of processors increases.



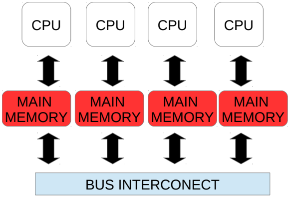
Distributed memory

Advantages

- Memory scale with the number of processors.
- Each processor quickly accesses its own local memory without interference and without overhead.
- Obtener hardware off-the-shelf with a reasonable *performance*.

Disadvantages

- Data communication between processes by the programmer.
- Complicated adapting existing code.



- Access time to the data is not uniform (and varies a lot!)

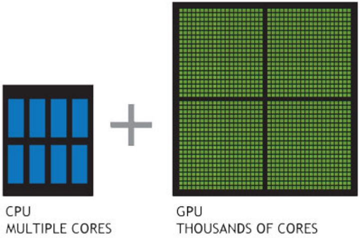
WTPC Workshop en técnicas de programación científica

Architecture of a cluster

How does it impact the design of the software?

- Massive parallelism.
- Increasing complexity.
- Less efficiency for old software.
- Little predictability.

We have to think in the hardware when coding!

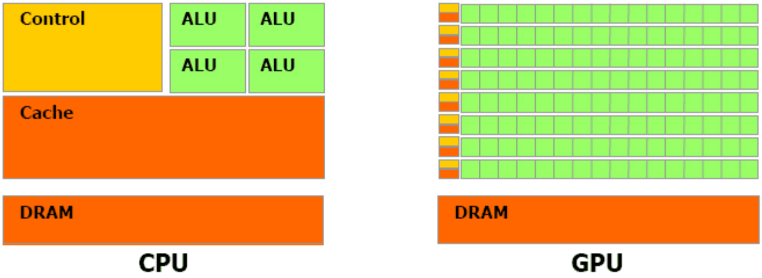


GPU (*graphics processing unit*)

Workshop en técnicas de programación científica

Accelerators

Schematically:



Accelerators

Main idea is:

less ctrl **less caché** **more ALUs**

$\downarrow$

Massive parallelism

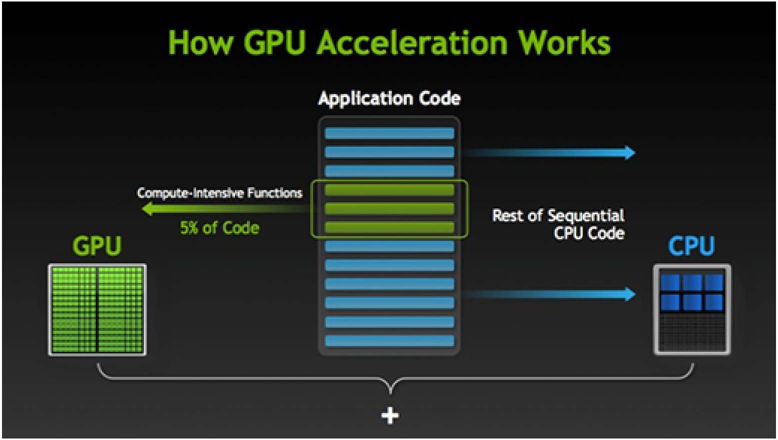
(to feed so many ALUs)

$\downarrow$

Data parallelism

(enough to hide latency)

Schematically:



Accelerators

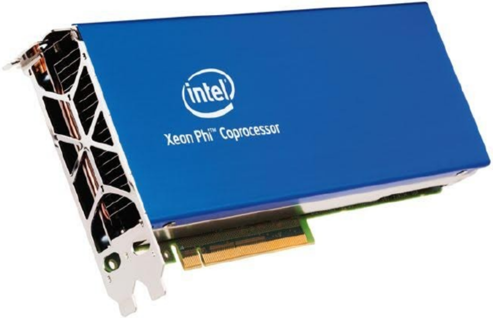
Why does it accelerate?

- Very scalable design.
- A lot of bandwidth.
- Many low frequency processors.
- Ideal for massive data processing.

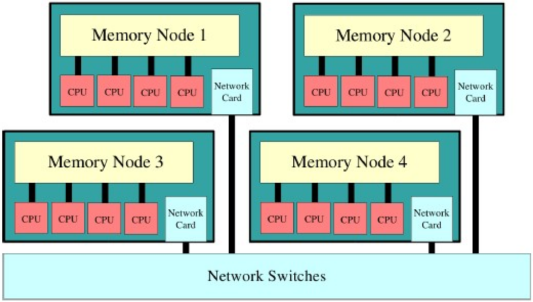
It does not always accelerate

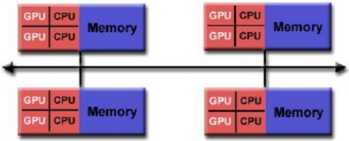
- Hyou have to pass the information to the board.
- Difficult to synchronize the processors.
- Serial execution VERY slow.

Accelerators



Hybrid architectures





HPC in a Supercomputer

Parallel application performance

What does the execution time of a parallel program depend on?

$$\boxed{T = T_{PROC} + T_{COM} + T_{IDLE}}$$

Parallel application performance

$$T_{PROC}$$

It depends on:

- Complexity and dimension of the problem.
- Number of tasks used.
- Characteristics of the processing elements (hardware, heterogeneity, non-dedication).

Parallel application performance

$$T_{COM}$$

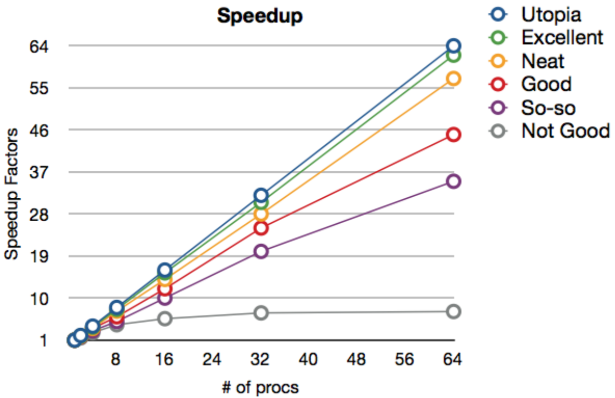
Depends on the location of processes and data (inter and intra-processor communication, communication channel).

$$T_{IDLE}$$

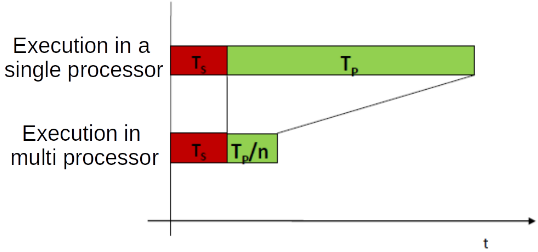
Due to non-determinism in execution, minimizing it is a design objective.

Speed Up

$$S_N = T_1 / T_N$$



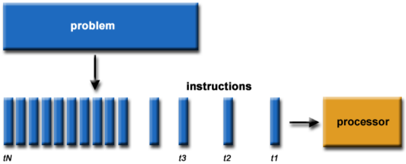
Amdhal's Law


Execution in a single processor
T_S
T_P
Execution in multi processor
T_S
T_P/n
t

Serial model

Traditionally, software has been written for serial computation:

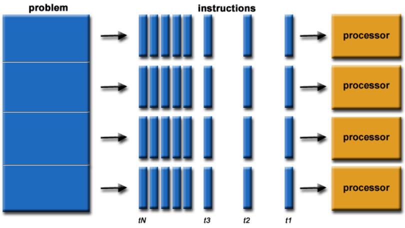
- A problem is broken into a discrete series of instructions.
- Instructions are executed sequentially one after another.
- Executed on a single processor.
- Only one instruction may execute at any moment in time.



Parallel Model

Parallel computing is the simultaneous use of multiple compute resources to solve a computational problem

- A problem is broken into discrete parts that can be solved concurrently.
- Each part is further broken down to a series of instructions.
- Instructions from each part execute simultaneously on different processors.
- An overall control/coordination mechanism is employed.



Parallel model

The computational problem should be able to:

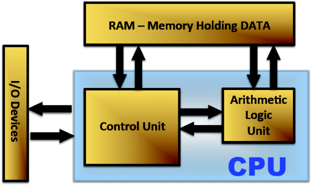
- Be broken apart into discrete pieces of work that can be solved simultaneously.
- Execute multiple program instructions at any moment in time.
- Be solved in less time with multiple compute resources than with a single compute resource.

The compute resources are typically:

- A single computer with multiple processors/cores.
- An arbitrary number of such computers connected by a network.

Von Neumann

Four main components:

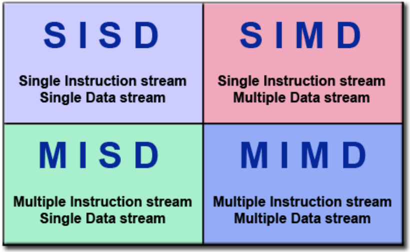


- Memory.
- Control Unit.
- Arithmetic Logic Unit.
- Input/Output.

- Read/write, random access memory is used to store both program instructions and data.
- Program instructions are coded data which tell the computer to do something.
- Data is simply information to be used by the program.
- Control unit fetches instructions/data from memory, decodes the instructions and then sequentially coordinates operations to accomplish the programmed task.
- Arithmetic Unit performs basic arithmetic operations.
- Input/Output is the interface to the human operator.

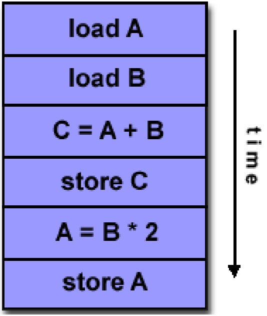
Parallel model: Concepts and Terminology

Flynn's Classical Taxonomy



Serial model

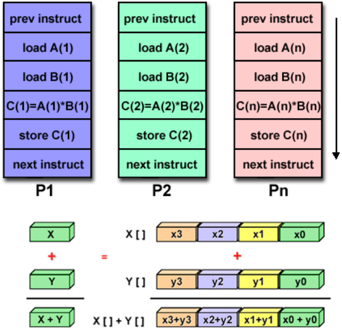
Single Instruction, Single Data (SISD):



- A serial (non-parallel) computer.
- Single Instruction: Only one instruction by the CPU during any one clock cycle.
- Single Data: Only one data stream is being used during any one clock cycle.
- This is the oldest type of computer.
- Examples: older generation mainframes, minicomputers, workstations and single processor/core PCs.

Parallel model: SIMD

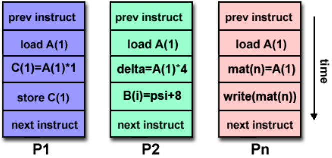
Single Instruction, Multiple Data (SIMD): A type of parallel computer



- Single Instruction: All processing units execute the same instruction at any given clock cycle.
- Multiple Data: Each processing unit can operate on a different data element.
- Best suited for specialized problems characterized by a high degree of regularity, such as graphics/image processing.
- Synchronous (lockstep) and deterministic execution.
- Two varieties: Processor Arrays and Vector Pipelines.

Parallel model: MISD

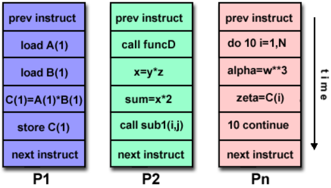
Multiple Instruction, Single Data (MISD) other type of parallel computer:



- Multiple Instruction: Each processing unit operates on the data independently via separate instruction streams.
- Single Data: A single data stream is fed into multiple processing units.
- Few (if any) actual examples of this class of parallel computer have ever existed.

Parallel model

Multiple Instruction, Multiple Data (MIMD):



- Multiple Instruction: Every processor may be executing a different instruction stream.
- Multiple Data: Every processor may be working with a different data stream.
- Execution can be synchronous or asynchronous, deterministic or non-deterministic.
- Currently, the most common type of parallel computer - most modern supercomputers fall into this category.
- Examples: most current supercomputers, networked parallel computer clusters and "grids", multi-processor SMP computers, multi-core PCs.

Parallel model: Concepts and Terminology

- **Node:**
  A standalone çomputer in a box". Usually comprised of multiple CPUs/processors/cores, memory, network interfaces, etc.
- **CPU / Socket / Processor / Core:**
  This varies, depending upon who you talk to. In the past, a CPU (Central Processing Unit) was a singular execution component for a computer. Then, multiple CPUs were incorporated into a node. Then, individual CPUs were subdivided into multiple çores", each being a unique execution unit.
- **Task:**
  Typically a program or program-like set of instructions executed by a processor. A parallel program consists of multiple tasks running on multiple processors.
- **Pipelining:**
  Breaking a task into steps performed by different processor units, with inputs streaming through.

WTPC
Workshop en técnicas de programación científica

- **Shared Memory:**
  Computer architecture where all processors have direct access to common physical memory. In a programming sense a model where parallel tasks all have the same "picture.ºf memory and can directly address and access the same logical memory locations regardless of where the physical memory actually exists.
- **Distributed Memory:**
  In hardware, refers to network based memory access for physical memory that is not common. As a programming model, tasks can only logically "see" local machine memory and must use communications to access memory on other machines where other tasks are executing.
- **Communications:**
  Parallel tasks typically need to exchange data. There are several ways this can be accomplished.
- **Granularity:**

  In parallel computing, granularity is a qualitative measure of the ratio of computation to communication.

Parallel model

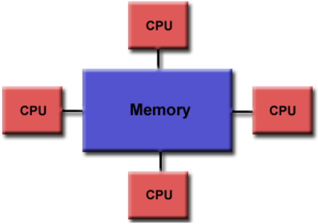
- **Embarrassingly Parallel:**
  Solving many similar, but independent tasks simultaneously; little to no need for coordination between the tasks.
- **Scalability:**
  Refers to a parallel system's (hardware and/or software) ability to demonstrate a proportionate increase in parallel speedup with the addition of more resources. Factors that contribute to scalability include:
  - Hardware - particularly memory-cpu bandwidths and network communication properties.
  - Application algorithm.
  - Parallel overhead related.
  - Characteristics of your specific application.

Workshop en técnicas de programación científica

Parallel Computer Memory Architectures

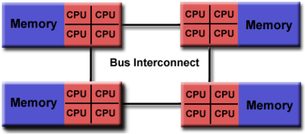
Shared Memory:

Uniform Memory Access (UMA):



- Identical processors.
- Equal access and access times to memory.

Non-Uniform Memory Access (NUMA):



- Not all processors have equal access time to all memories.
- Memory access across link is slower.

Parallel Programming Models

There are several parallel programming models in common use:

- Shared Memory (without threads).
- Threads.
- Distributed Memory / Message Passing.
- Data Parallel.
- Hybrid.
- Single Program Multiple Data (SPMD).
- Multiple Program Multiple Data (MPMD).

Parallel model: Threads Model

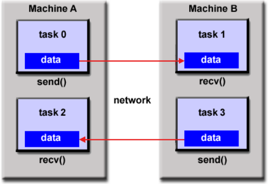
This programming model is a type of shared memory programming. From a programming perspective, threads implementations commonly comprise:

- A library of subroutines that are called from within parallel source code.
- A set of compiler directives embedded in either serial or parallel source code. In both cases, the programmer is responsible for determining the parallelism (although compilers can sometimes help).
- An interesting library is: Multiprocessing is a package that supports spawning processes using an API similar to the threading module.

  `https://stackoverflow.com/questions/2846653/how-to-use-threading-in-python`

Parallel model: Distributed Memory / Message Passing Model

This model demonstrates the following characteristics:



- A set of tasks that use their own local memory during computation. (Multiple tasks can reside on the same physical machine and/or across an arbitrary number of machines.)
- Tasks exchange data through communications by sending and receiving messages.
- Data transfer usually requires cooperative operations to be performed by each process. (For example, a send operation must have a matching receive operation.)
- **Implementations: Message Passing Interface (MPI)**

HDF5 and h5py

The h5py package is a Pythonic interface HDF5 binary data format.
https://www.h5py.org/

- We can save different kinds of information.
- We can save our trained neural network models and reload.
- To read the files:

```
import h5py
filename = 'file.hdf5'
f = h5py.File(filename, 'r')
```

- To save into this files:

```
#!/usr/bin/env python
import h5py

# Create random data
import numpy as np
data_matrix = np.random.uniform(-1, 1, size=(10,
      3))

# Write data to HDF5
data_file = h5py.File("file.hdf5", "w")
data_file.create_dataset("group_name", data=
      data_matrix)
data_file.close()
```

HDF5 and h5py

Alternatives

- JSON: Nice for writing human-readable data; VERY commonly used (read & write)
- CSV: Super simple format (read & write).
- pickle: A Python serialization format (read & write).
- MessagePack (Python package): More compact representation (read & write).
- HDF5 (Python package): Nice for matrices (read & write).
- XML: exists too *sigh* (read & write).

HDF5

For your application, the following might be important:

- Support by other programming languages.
- Reading / writing performance.
- Compactness (file size).

Credits:

- Blaise Barney, Lawrence Livermore National Laboratory
  https://computing.llnl.gov/tutorials/parallel_comp/
- ICTP Introductory School on Parallel Programming and Parallel Architecture for High-Performance Computing:
  http://indico.ictp.it/event/7659/overview
- WTPC 2017, Graciela Molina (FACET -UNT).
  https://wtpc.github.io/clases/2017/11_MPI.pdf
- MPI4PY:
  https://www.howtoforge.com/tutorial/distributed-parallel-
- http://mpi-forum.org/docs/