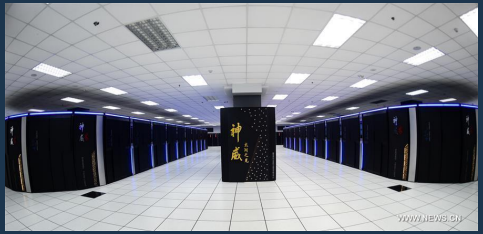
LAPP
Laboratoire d'Annecy de Physique des Particules

Introduction to High Performance Computing

Pierre Aubert

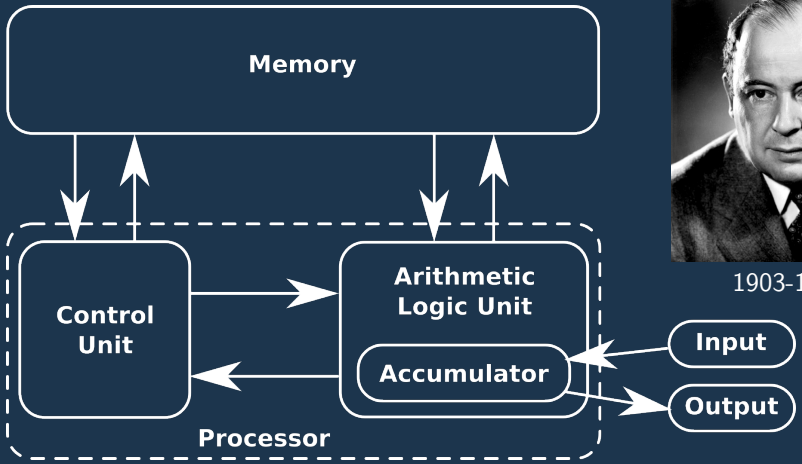
High Performance Computing (HPC)

- Part of the computer science
- Get the best performances by using the right algorithms on the right architectures



Computing Processing Unit (CPU)

John Von Neumann architecture (1945)



1903-1957

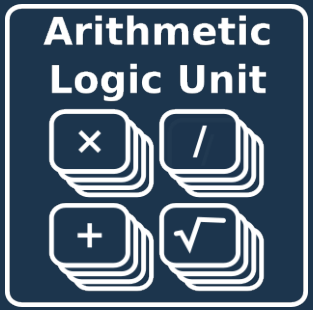
Arithmetic Logic Unit (ALU)

ALU

Arithmetic Logic Unit (ALU)

ALU

ALU Vectorized



HPC Libraries

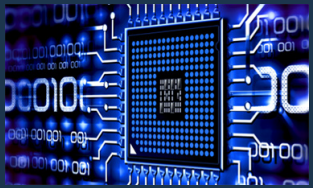
- C :
  - MKL, Atlas, BLAS, Lapack
- C++ :
  - TBB, Eigen, Armadillo, HPX
- Python :
  - Numpy
  - Numba (JIT, Just In Time)

Numba

HPX
STE||AR GROUP

Aim of this tutorial

- How HPC libraries work
- How to measure performances of a function

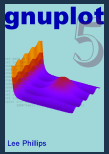


- Focus on **float** computation (simple precision)
  - Sufficient in most cases and get very good speed up

- Precision of the Computation :
  - Optimized version is **MORE** precise than scalar version !!!
    - So, asking to have exactly the same result as scalar version is a non sense !!!
    - To clame scalar was check is not a plea because errors can compensate each other
  - Optimized **float** version can reach same precision as scalar **double** version

Prerequisites for this tutorial

- Tools for compilation :
  - **GCC/G++** : version **7.2** (I do not know what is going on with the version 8)
  - **CMake** : version $\geq$ **3.0**
  - **Make** : version $\geq$ **4.0**
- Versioning Tool :
  - **Git** : version $\geq$ **2.14.1**
- Tool for drawing plot :
  - **Gnuplot** : version $\geq$ **5.0**
- Optional Tools :
  - **hwloc-ls**
  - **jupyter-notebook**
  - **anaconda**



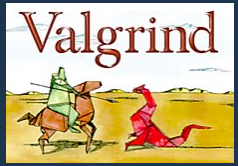
Outline of the tutorial

- Warm up
- Creation of a HPC/Timer library
- Optimisation of Hadamard product (+ python wrapper)
- Optimisation of saxpy (homework)
- Optimisation of a vector reduction
- Application/exercice : Optimisation of barycentre computation (homework)
- Optimisation of Dense Matrix-Matrix multiplication
- What about branching ? (bonus)
- Conclusion

How to evaluate performances ?

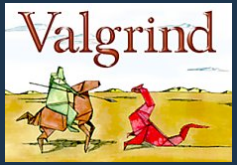
Basically with a timer.

- Instrumenting the code
  - GProf
  - Perf
- Emulate the binary
  - Valgrind (`http://www.valgrind.org/`)
  - Maqao (`http://www.maqao.org/`)
- Python :
  - cprofile (+ snakeviz)
  - time



How to evaluate time spent in a function ?

- Tools :
  - **GProf**
  - **Perf**
  - **Valgrind**
  - **Maqao**



- Functions :
  - **clock** : to get a time in seconds (not very precise).
  - **rdtsc** : to get a time in cycles (very precise).

- Method :
  - To evaluate N calls of the function and then to average the results.

The Kernel approach

- What is a kernel ?
  - The function which does the computation and which does not call any other function. So a pure mathematic function.
- Elapsed time of compilation :
  - GCC always tries to make a short compilation (typically 1 second per file).
  - It is the same if the file has 100 000 lines or not.
  - So, short files implies better optimisations.

Where to get the tutorial ?

Web Tutorial : `https://lappweb.in2p3.fr/~paubert/ASTERICS_HPC/index.html`

Minimal repository : `https://lappweb.in2p3.fr/~paubert/ASTERICS_HPC/ressource/build/Correction/ExampleMinimal.tar.gz`

Correction : `https://lappweb.in2p3.fr/~paubert/ASTERICS_HPC/ressource/build/Correction/Examples.tar.gz`