Parallelism

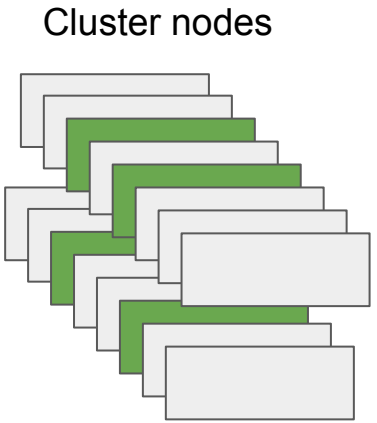
Week 1
The setup

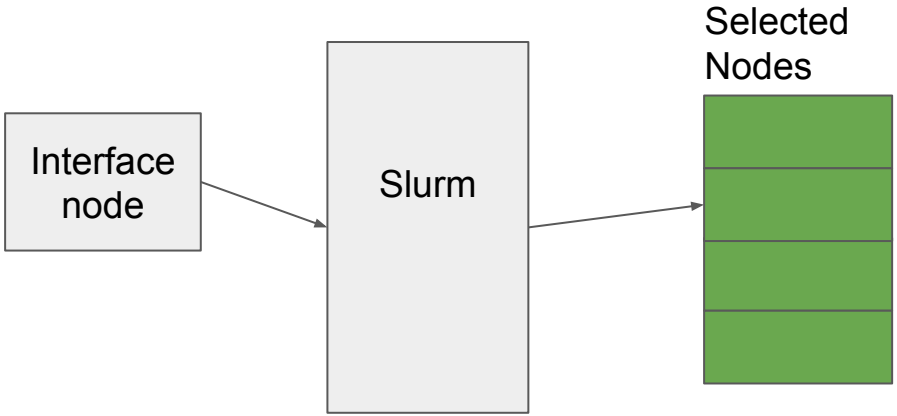
Getting HPC access

Please visit : https://catalogue-si.unige.ch/en/hpc

What is the cluster and how to use.

https://doc.eresearch.unige.ch/hpc/start





What is Slurm ?

Slurm is an open source, fault-tolerant, and highly scalable cluster management and job scheduling system for large and small Linux clusters. Slurm has three key functions.

- *it allocates exclusive and/or non-exclusive access to resources (compute nodes) to users for some duration of time so they can perform work.*
- *it provides a framework for starting, executing, and monitoring work (normally a parallel job) on the set of allocated nodes.*
- *it arbitrates contention for resources by managing a queue of pending work.*

https://doc.eresearch.unige.ch/hpc/slurm#what_is_slurm

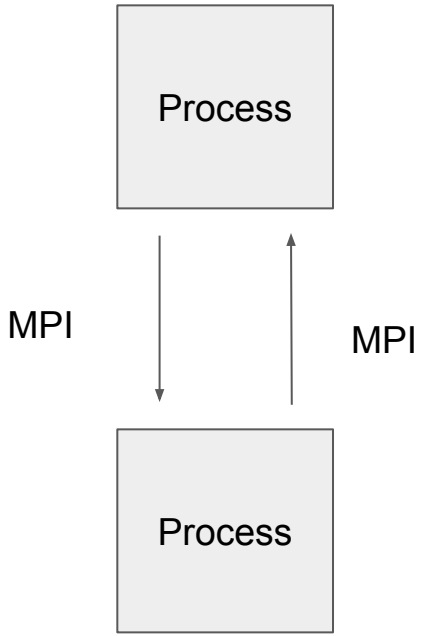
Parallel Programming with MPI

The Message-Passing Model : A process is (traditionally) a program counter and address space.

→ Processes may have multiple threads (program counters and associated stacks) sharing a single address space.

→MPI is for communication among processes, which have separate address spaces.

→Inter-process communication consists of – synchronization – movement of data from one process's address space to another's.



Installing OpenMPI on your PC

Linux :Use packagemanager to install “openmpi”

$ sudo apt install openmpi-bin libopenmpi-dev

Mac : use brew install

$ brew install openmpi

Windows :

If you use WSL2 just use linux commands

Hello world

```
#include <mpi.h>
#include <iostream>

int main() {
    // Initialize the MPI environment
    MPI_Init(NULL, NULL);

    // Get the number of processes
    int world size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size );

    // Get the rank of the process
    int world rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank );

    // Get the name of the processor
    char processor_name[MPI_MAX_PROCESSOR_NAME] ;
    int name len;
    MPI_Get_processor_name(processor_name, &name_len);

    // Print off a hello world message
    std::cout <<"Hello world from processor " <<processor name << ",
rank"<< world_rank << "out of " <<  world_size << "processors \n" ;

    // Finalize the MPI environment.
    MPI_Finalize();
}
```

During `MPI_Init`, all of MPI's global and internal variables are constructed

`MPI_Comm_size` returns the size of a communicator. In our example, `MPI_COMM_WORLD` (which is constructed for us by MPI) encloses all of the processes in the job, so this call should return the amount of processes that were requested for the job.

`MPI_Comm_rank` returns the rank of a process in a communicator. Each process inside of a communicator is assigned an incremental rank starting from zero.

`MPI_Finalize` is used to clean up the MPI environment. No more MPI calls can be made after this one.