PHAROS
THE GREEK AI FACTORY

Submitting OpenMP Jobs on SLURM

Training Series - Course 2 "Introduction to HPC"

Nikolaos Triantafyllis (ntriantafyl@admin.grnet.gr)

grnet

What is HPC?

- High-Performance Computing (HPC) is the ability to perform sophisticated calculations at high speeds.
- An HPC cluster consists of hundreds or thousands of compute servers, so-called nodes. The nodes in each cluster work in parallel with each other.
- HPC solves large problems in science, engineering, or business, that are too complex for a PC. On typical PC it might take e.g. hours, days, weeks to perform the computations, but if you use an HPC Cluster, it might only take minutes, hours, days, respectively.

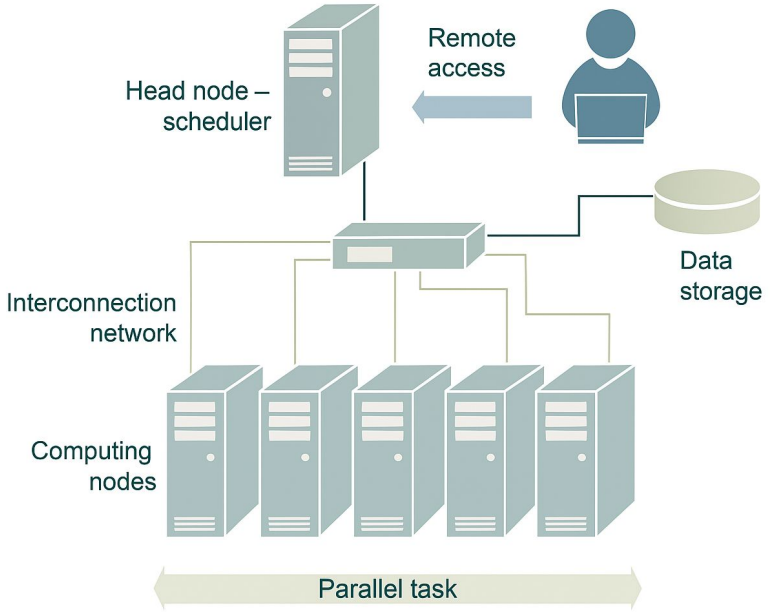
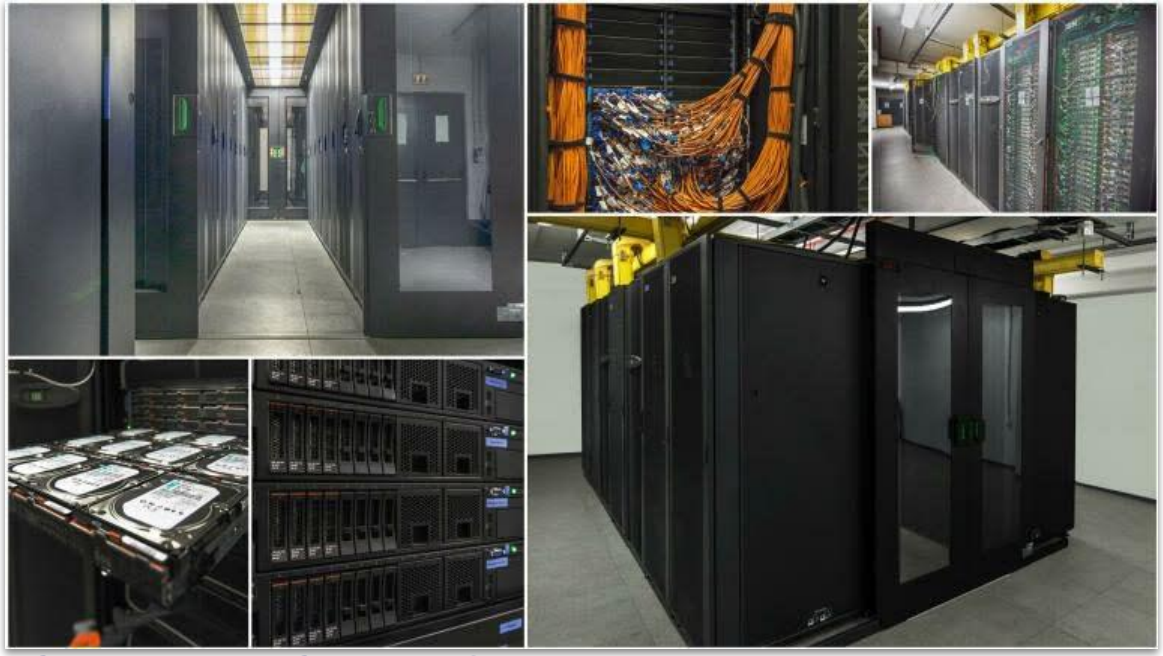


*image source: https://miro.medium.com/v2/resize:fit:720/format:webp/1*zOQ3LwOBQc9xnW9Pzm7U5A.png (regenerated)*



GRNET HPC - ARIS

(https://www.hpc.grnet.gr)

Supports the Greek and global research community with advanced HPC infrastructure for scientific exploration.

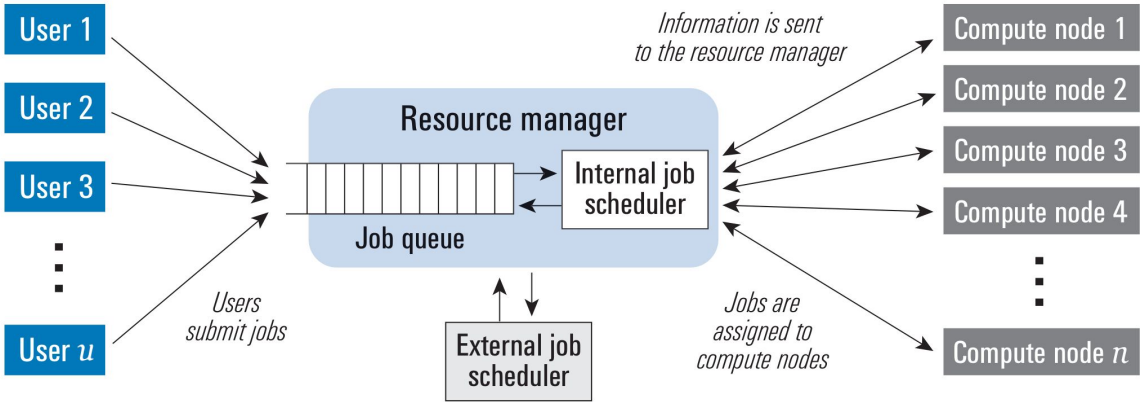


Image source: Esquema de Manejador de Recursos de LSF (Iqbal, Gupta, & Fang, 2005)

Simple Linux Utility for Resource Management

software stack that runs on HPC infrastructure and operates resource management, job scheduling and accounting

Typical HPC/SLURM infrastructure

- User executes SLURM client commands such as job submissions (sbatch) [Blue area]
- SLURM handles the received jobs and orchestrates operations [Purple area]
- SLURM passes user's jobs to compute nodes [Yellow area]
- User receives job's results back to their working dir

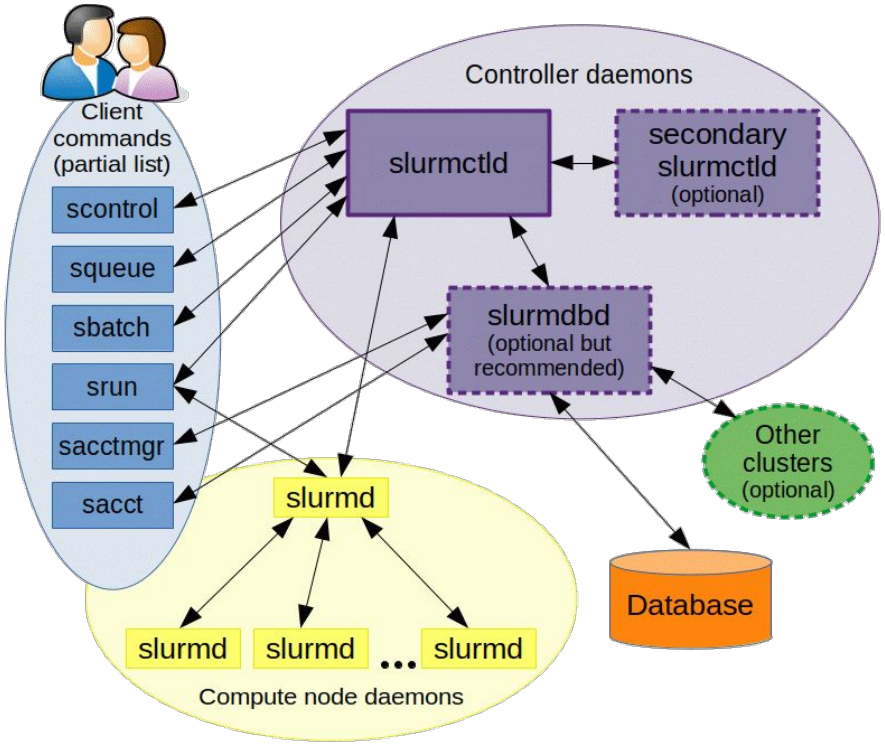


Image source: https://slurm.schedmd.com/arch.gif

A Job's Life Cycle

1. **Submission**: User access HPC and submits a job using sbatch
2. **Pending (PD)**: Job waits in queue for resources to become available
3. **Scheduling**: SLURM assigns resources based on priority and availability
4. **Running (R)**: Job executes on allocated resources
5. **Completion (CD)**: Job finishes successfully or fails
6. **Failure/Preemption (F)/(PR)**: Job may fail due to errors or get preempted by higher-priority jobs
7. **Cleanup**: SLURM releases resources, logs results

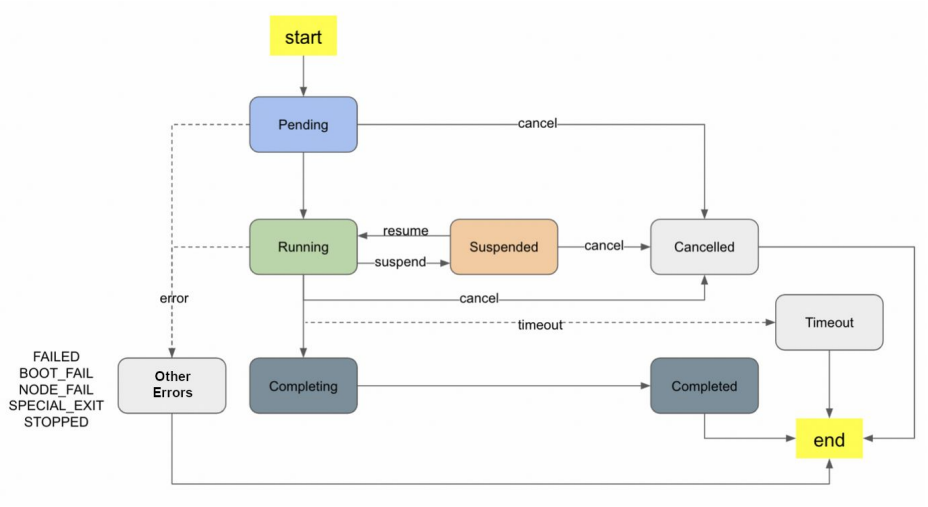


Image source: https://cos.twcc.ai/SYS-MANUAL/uploads/upload_8ead0a8bf1bd623e1a1c2ed8ab609677.png

SLURM Useful Commands

- **sacct** is used to report job accounting information
- **sbatch** is used to submit a job script for later execution
- **scancel** is used to cancel a pending or running job
- **sinfo** reports the state of partitions and nodes managed by SLURM
- **squeue** reports the state of jobs
- **srun** usually is executed inside the job script to run apps after job submission

More info: **https://slurm.schedmd.com**

HPC Modules

Managing Environment with Modules

- Modules control environment variables such as PATH, LD_LIBRARY_PATH
- Use **module** command to load, unload, and list modules

Module Commands

- **module avail**: List all available modules
- **module load <module>**: Load a module
- **module unload <module>**: Unload a module
- **module list**: List loaded modules
- **module purge**: Remove all modules

Example of loading a module with a certain version:

> module load gnu/15.2

> gcc –version

SLURM commands in action

- $ **sbatch script.sh**
  Submitted batch job 12345

- $ **squeue -j 12345**
  ```
  JOBID  PARTITION  NAME      USER  ST  TIME  NODES  NODELIST(REASON)
  12345  batch     my_job    user1 PD  0:00  20     (Resources)
  ```

- $ **squeue -j 12345**
  ```
  JOBID  PARTITION  NAME      USER  ST  TIME  NODES  NODELIST(REASON)
  12345  batch     my_job    user1 R         1:20  20     node[01-20]
  ```

- $ **sacct -j 12345 –format=JobID,JobName,Partition,Account,AllocCPUS,State,ExitCode**
  ```
  JobID    JobName  Partition      Account  AllocCPUS      State ExitCode
  ------ ----- ----- ----- ----- -----
  ---
  --
  12345    my_job         compute        my_acc         4       COMPLETED    0:0
  12345.batch     batch   compute        my_acc         4       COMPLETED    0:0
  12345.0         task1   compute        my_acc         2       COMPLETED    0:0
  12345.1         task2   compute        my_acc         2       COMPLETED    0:0
  ```

- $ **scancel 12345**

HPC Usage Report (ARIS-based)

$ **mybudget**

```
==========================================================================
Core Hours Allocation Information for account : testproj
==========================================================================
Allocated Core Hours  :          2400000.00
Consumed  Core Hours  :          15.00
Percentage of Consumed :         0.00
==========================================================================
```

$ **myreport**

```
--------------------------------------------------
-
Cluster/Account/User Utilization 2015-04-07T00:00:00 - 2015-10-07T23:59:59 (15897600 secs)
Time reported in CPU Hours
--------------------------------------------------
 Cluster         Account         Login    Proper Name     Used    Energy
----- -------- ----- -------- ----- --------------------
 aris      testproj username        User Name        15         110
```

SLURM #SBATCH directives

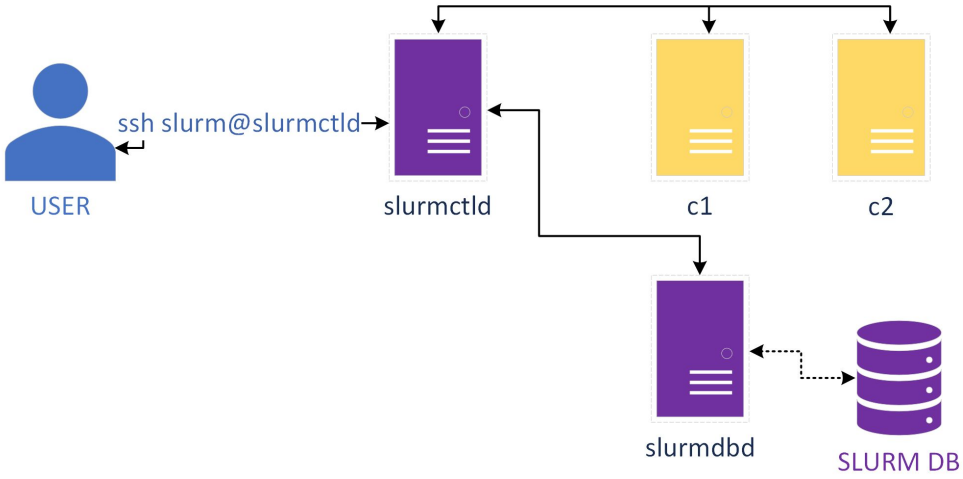
Command	Example Value	Description
#SBATCH –account	myaccount	Specifies which account is associated with.
#SBATCH –partition	compute	Specifies which partition/queue to use.
#SBATCH –nodes	1	Always use a single node for OpenMP jobs.
#SBATCH –ntasks-per-node	1	Number of tasks to launch per node (set to 1 for pure OpenMP jobs).
#SBATCH –cpus-per-task	20	Number of CPU threads per task (matches OMP_NUM_THREADS).
#SBATCH –time	01:00:00	Maximum wall time for the job.
#SBATCH –mem	56G	Total memory requested for the node.

SLURM Tutorial under containers

- In this tutorial you will deploy a typical HPC infrastructure using the SLURM resource manager under containers
- Submit OpenMP jobs:
  - A “Hello world” app,
  - A Monte Carlo PI estimation,
  - Train multiple Random Forest Regressor models simultaneously
- View example’s output

HPC/SLURM facility

- 5 containers:
  - 1 MySQL Server instance to store SLURM accounting
  - 1 node as the DB controller
  - 1 login node as the SLURM controller and user's login endpoint
  - 2 compute nodes for calculations
- Each compute node contains 2 CPUs of 2 cores each
- All 4 nodes operate Debian-based Linux OS



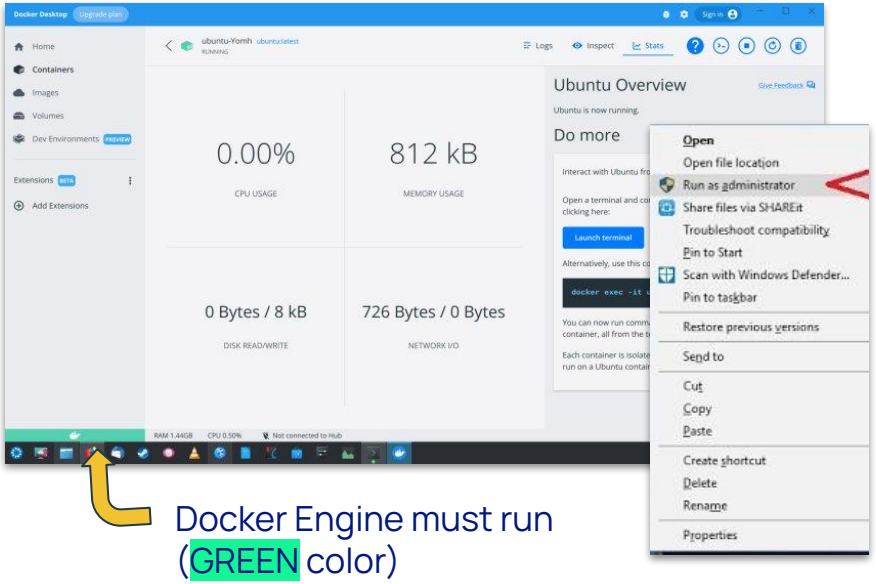
Prerequisites (for WINDOWS users)

1. Download Docker Desktop from:
   https://docs.docker.com/desktop/install/windows-insta ll

2. Follow step-by-step instructions here:
   https://www.linkedin.com/pulse/step-guide-how-install-docker-windows-1011-shashank-abhishek

Prerequisites (for WINDOWS users)

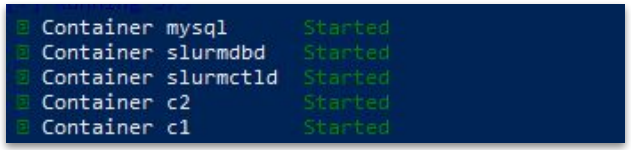
1. Use **default** options in installation
2. Your PC must be **restarted**
3. If docker engine does not start, you might need to close the Docker Desktop and run it in **administration** mode



Docker Engine must run (GREEN color)

Prerequisites (for WINDOWS users)

1. Make sure that Docker Desktop is initiated (GREEN color)
2. Download the Docker recipe to setup the virtual infrastructure of SLURM under containers:
   https://github.com/nikosT/slurm-docker-cluster/archive/refs/heads/master.zip
3. Extract content at some folder e.g. C:\...\slurm-docker-cluster-master
4. Open Windows PowerShell (in search button type PowerShell)
5. In Windows PowerShell terminal type:
   `cd C:\...\slurm-docker-cluster-master`
   `powershell -ExecutionPolicy Bypass`
   `. .\alias.ps1` *# load environment*
   `wstart` *# start the virtual cluster (~2.5 GB images' size)*
   When wstart is completed, you should view this



6. Then, type:
   `ssh slurm@slurmctld` *# access the login node*
7. `cd openmp_examples` *# change dir to the OpenMP examples*
8. `sbatch <file>.sh` *# submit your OpenMP job*
9. `ls` *# view the outputs of your submission*
10. `exit` *# logout from login node*
11. `wstop` *# stop the virtual cluster*

Prerequisites (for LINUX users)

In terminal type:

```
sudo apt-get install git docker docker.io docker-compose docker-compose-v2 # install docker
git clone https://github.com/nikosT/slurm-docker-cluster # get docker recipe
cd slurm-docker-cluster # change dir to the appropriate one
chmod -R 777 slurm #set appropriate permissions to the folder
source alias # load environment
wstart # start the virtual cluster (~2.5 GB images' size)
exit # logout from login node
wstop # stop the virtual cluster
```

Prerequisites (for WINDOWS/LINUX users)

- If you've run it before, make sure to delete the related volumes and images to start with a clean setup.

  In terminal type:

  ```
  docker volume rmi <image_id> #intergallactic/slurm-docker-main
  docker volume rm slurm-docker-cluster_etc_munge
  docker volume rm slurm-docker-cluster_etc_slurm
  docker volume rm slurm-docker-cluster_slurm_jobdir
  docker volume rm slurm-docker-cluster_var_lib_mysql
  docker volume rm slurm-docker-cluster_var_log_slurm
  ```

- If your computer has fewer than **8 cores**, you will need to modify the **/etc/slurm/slurm.conf** file from **within** the container (`ssh root@slurmctld`) and then run `wstop && wstart`.
  - e.g. setting 4 cores: `NodeName=c[1-2] CPUs=`**`4`**` Sockets=2 CoresPerSocket=`**`2`**` ThreadsPerCore=1 RealMemory=`**`2000`**` State=UNKNOWN` *# remember DefMemPerCPU=500*

#1 Hello OpenMP World example

Run in **terminal**:

```
ssh slurm@slurmctld
$> cd openmp_examples
$> gcc -fopenmp -O3 openmp_hello.c -o openmp_hello
$> sbatch run_openmp_hello.sh
```

View **run_openmp_hello.sh** file:

```
#!/bin/bash
#SBATCH –job-name=my_openmp_hello      # Job name
#SBATCH –output=my_openmp_hello_%j.out  # Output file name (%j expands to jobID)
#SBATCH –error=my_openmp_hello_%j.err   # Error file name (%j expands to jobID)
#SBATCH –partition=normal       # Partition name
#SBATCH –nodes=1              # Number of nodes
#SBATCH –ntasks=1             # Number of tasks
#SBATCH –ntasks-per-node=1      # Number of tasks per node
#SBATCH –cpus-per-task=8       # Number of tasks per node
#SBATCH –time=00:01:00          # Time limit (HH:MM:SS)
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
$HOME/openmp_examples/openmp_hello
```

#1 Hello OpenMP World example

Run in **terminal**:

```
ssh slurm@slurmctld
$> cd openmp_examples
$> gcc -fopenmp -O3 openmp_hello.c -o openmp_hello
$> sbatch run_openmp_hello.sh
```

View **my_openmp_hello_<job_id>.out** file:

```
Hello from thread 0 of 8 on c1 (pid: 55) I CPU: 10 I Socket: 0
Hello from thread 2 of 8 on c1 (pid: 55) I CPU: 9 I Socket: 0
Hello from thread 6 of 8 on c1 (pid: 55) I CPU: 1 I Socket: 0
Hello from thread 1 of 8 on c1 (pid: 55) I CPU: 2 I Socket: 0
Hello from thread 3 of 8 on c1 (pid: 55) I CPU: 11 I Socket: 0
Hello from thread 7 of 8 on c1 (pid: 55) I CPU: 8 I Socket: 0
Hello from thread 4 of 8 on c1 (pid: 55) I CPU: 7 I Socket: 0
Hello from thread 5 of 8 on c1 (pid: 55) I CPU: 5 I Socket: 0
```

Monte Carlo Method for Approximating π

1. Estimate π by simulating random points in a square (r=1) enclosing a quarter circle.
2. Generate N random points (x, y) where 0 ≤ x, y ≤ 1.
3. Count points inside the quarter circle: $x^2 + y^2 \leq 1$.
4. Compute: Ratio = Points inside circle / N
5. Approximate: π ≈ 4 × Ratio

More points → closer approximation to π.

Code:
https://github.com/nikosT/slurm-docker-cluster/blob/main-pull/slurm/openmp_examples/montecarlo.c

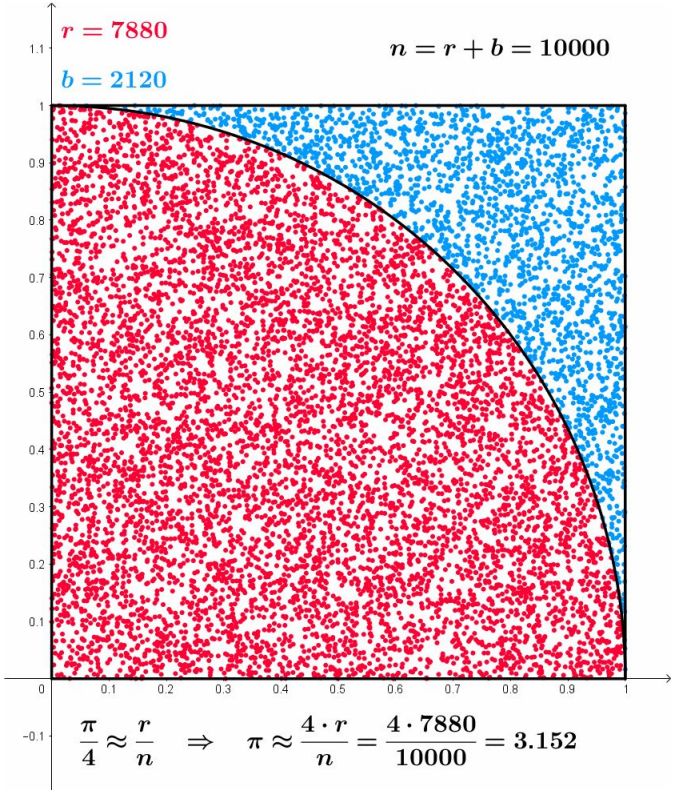


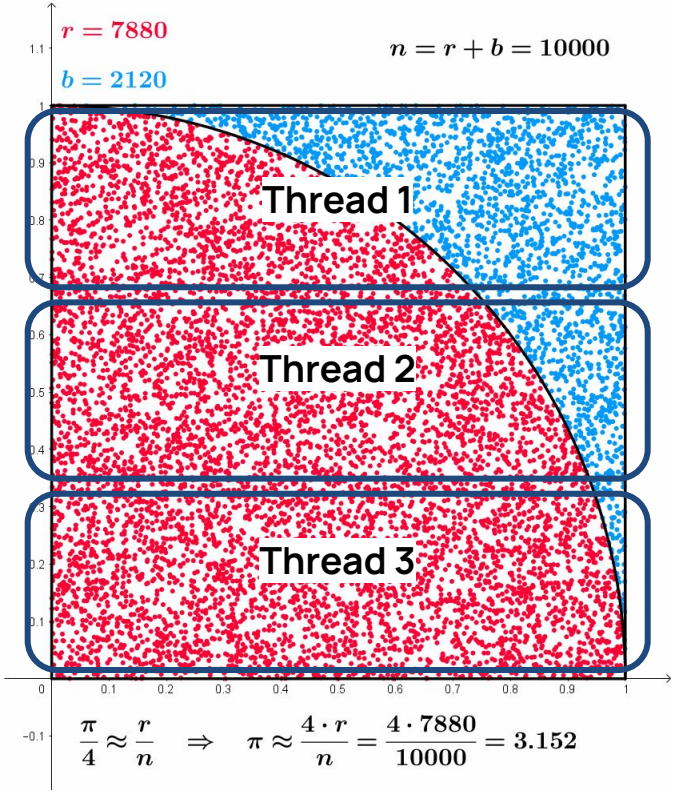
Image source: https://en.wikipedia.org/wiki/File:Pi_monte_carlo_all.gif

Monte Carlo Method for Approximating π

1. Estimate π by simulating random points in a square (r=1) enclosing a quarter circle.
2. Generate N random points (x, y) where 0 ≤ x, y ≤ 1.
3. Count points inside the quarter circle: $x^2 + y^2 \leq 1$.
4. Compute: Ratio = Points inside circle / N
5. Approximate: π ≈ 4 × Ratio

More points → closer approximation to π.

Code:
https://github.com/nikosT/slurm-docker-cluster/blob/main-pull/slurm/openmp_examples/montecarlo.c



Use OpenMP to split the workload into parallel chunks.

#2 Monte Carlo PI OpenMP example

Run in **terminal**:

```
ssh slurm@slurmctld
$> cd openmp_examples
$> gcc -fopenmp -O3 montecarlo.c -o montecarlo
$> sbatch run_montecarlo.sh
```

View **run_montecarlo.sh** file:

```
#!/bin/bash
#SBATCH –job-name=my_montecarlo     # Job name
#SBATCH –output=my_montecarlo_%j.out  # Output file name (%j expands to jobID)
#SBATCH –error=my_montecarlo_%j.err   # Error file name (%j expands to jobID)
#SBATCH –partition=normal       # Partition name
#SBATCH –nodes=1              # Number of nodes
#SBATCH –ntasks=1             # Number of tasks
#SBATCH –ntasks-per-node=1       # Number of tasks per node
#SBATCH –cpus-per-task=4       # Number of tasks per node
#SBATCH –time=00:01:00          # Time limit (HH:MM:SS)
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
$HOME/openmp_examples/montecarlo
```

#2 Monte Carlo PI OpenMP example

Run in **terminal**:

```
ssh slurm@slurmctld
$> cd openmp_examples
$> gcc -fopenmp -O3 montecarlo.c -o montecarlo
$> sbatch run_montecarlo.sh
```

View **my_montecarlo_<job_id>.out** file:

```
Thread 0: Time = 2.028424 seconds
Thread 3: Time = 2.030026 seconds
Thread 1: Time = 2.031826 seconds
Thread 2: Time = 2.039390 seconds

Estimated value of π: 3.141605
Total execution time: 2.039475 seconds
Number of threads used: 4
```

Random Forest Regressor Tuning

- RandomForestRegressor: **Scikit-learn** ensemble model averaging multiple decision trees for robust regression predictions. Reduces overfitting by combining diverse trees for continuous numerical outputs.
- **Hyperparameter Tuning**:`min_samples_leaf (int/float, default=1)`: Minimum samples required at a decision tree's leaf node.
- Tuning Process:
  - Trains model with `RandomForestRegressor(n_jobs=1, min_samples_leaf=msf)`
  - Fits on training data with `model.fit(Xtrain, ytrain)`
  - Predicts on test data with `model.predict(Xtest)`
  - Evaluates performance using Pearson correlation `np.corrcoef(ytest, pred)[0,1]`
- Parallel Execution:
  - Splits tuning of 40 min_samples_leaf values (1–40) across 4 processes.
  - Each process handles 10 values:
    - Process 1: [1–10]
    - Process 2: [11–20]
    - Process 3: [21–30]
    - Process 4: [31–40]
- Code: **https://github.com/nbakas/HighPerformanceComputing/blob/main/07-ScalabilityTuningRF.py**

#3 RF Tuning Multi-Process example

Run in **terminal**:

```
ssh slurm@slurmctld
$> cd openmp_examples
$> sbatch run_tuning.sh
```

View **run_tuning.sh** file:

```
#!/bin/bash
#SBATCH –job-name=my_tuning      # Job name
#SBATCH –output=my_tuning_%j.out  # Output file name (%j expands to jobID)
#SBATCH –error=my_tuning_%j.err   # Error file name (%j expands to jobID)
#SBATCH –partition=normal        # Partition name
#SBATCH –nodes=1              # Number of nodes
#SBATCH –ntasks=4       # Number of tasks per node
#SBATCH –cpus-per-task=1       # Number of tasks per node
#SBATCH –time=00:01:00          # Time limit (HH:MM:SS)

$HOME/openmp_examples/ScalabilityTuningRF.py
```

#3 RF Tuning Multi-Process example

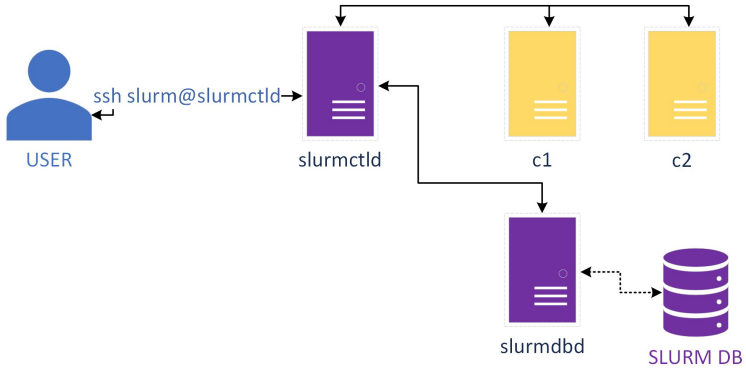
Run in **terminal**:

```
ssh slurm@slurmctld
$> cd openmp_examples
$> sbatch run_tuning.sh
```

View **my_tuning_<job_id>.out** file:

```
Number of existing processes: 4, Number of used processes: 4
Process 3 started
Process 3, min_samples_leaf: 31, Pearson: 0.7677726194945116
...
Process 2 started
Process 2, min_samples_leaf: 21, Pearson: 0.7940901197229328
...
Process 1 started
Process 1, min_samples_leaf: 11, Pearson: 0.8453098573872617
...
Process 0 started
Process 0, min_samples_leaf: 1, Pearson: 0.9060490977747907
...
Parallel Execution Time: 2250.66590 milliseconds
```

Homework



Familiarization with SLURM

Try accessing resources **interactively** in SLURM:

1. Open 2 terminals, change to the **slurm-docker-cluster-master** directory and load the **environment**
2. From both terminals access the **login node** (slurmctld), then:
3. In Terminal #1, type: `localhost` *(what's the node's name and why?)*
4. In Terminal #1, type: `srun –nodes=1 –time=00:10:00 –pty bash` *(what do you think this command does?)*
5. In Terminal #1, type: `localhost` *(what's the node's name and why?)*
6. In Terminal #2, type: `squeue` *(is there any job running?)*
7. In Terminal #1, type: `exit` *(what happened?)*
8. In Terminal #2, type: `squeue` *(is there any job running?)*
9. In Terminal #1, type: `localhost` *(what's the node's name and why?)*

Any Questions?

EuroHPC Joint Undertaking

Co-funded by the European Union

AI Factories PHAROS GREECE

ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Υπουργείο Ψηφιακής Διακυβέρνησης και Τεχνητής Νοημοσύνης