

Introduction to HPC

Lecture 7

Jakub Gałecki



Zakupiliśmy i uruchomiliśmy 1000 maszyn wyposażonych w procesory serwerowe, etc.

Jak możemy udostępnić je do użytkowania?



- \cdot slurm
- PBS
- TORQUE
- Grid Engine
- LSF



Simple Linux Utility for Resource Management

https://slurm.schedmd.com/

- Przydziela zasoby obliczeniowe (sprawiedliwie)
- Pozwala na prowadzenie księgowości CPUh (i innych)
- Skalowalny
- Zarządza kolejką zadań
- Integracja z MPI



- squeue stan kolejki
- sinfo informacje o stanie zasobów
- · salloc alokacja zasobów na zadanie
- srun uruchomienie zadania
- sbatch dodanie zadania do kolejki
- scancel anulowanie zadania
- scontrol*
- sacct*
- sreport*

Więcej na:

https://kdm.icm.edu.pl/Tutorials/HPC-intro/slurm_intro/



- Node (węzeł) jednostka obliczeniowa (maszyna), posiadająca CPU i pamięć
- Partition (partycja) zbiór węzłów o konkretnych własnościach, bądź wydzielony do osobnej grupy
- Job (zadanie) zlecona praca, np. skrypt
- Step (krok) podzbiór zadania, często zadanie składa się tylko z jednego kroku
- Task (...) jeden z procesów biorących udział w kroku/zadaniu (typowo rząd MPI)



Slurm dodaje do środowiska uruchamianych zadań zmienne, zawierające istotne informacje

Przykładowo:

- SLURM_CPUS_ON_NODE
- SLURM_JOB_ID
- SLURM_NNODES
- SLURMD_NODENAME



- Komendą **srun** uruchamiamy dany skrypt/program **srun** jest *blokujący* **srun** jest uruchamiany w ramach zaalokowanych zasobów
- (salloc, sbatch), jeżeli takowe istnieją
- srun należy używać zamiast mpirun/mpiexec



srun [...] --pty bash -l

Efektywnie logowanie ssh na węzeł (z ewentualnym ograniczeniem zasobów)

Przydatne do niewielkich eksperymentów



Zleca wykonanie skryptu/programu do kolejki

sbatch działa asynchronicznie

Typowy workflow zlecanego skryptu:

- 1. Ładujemy potrzebne moduły
- 2. Inicjalizujemy środowisko
- 3. Uruchamiamy skalowalny program komendą **srun**
- 4. Finalizujemy zadanie

∰ <u>≷</u>

- · -A/--account
- \cdot -p/--partition
- · -n/--ntasks
- -N/--nodes
- --ntasks-per-X
- --time
- --mem
- · -J/--job-name
- -o/--output
- \cdot --dependency

Opcje możemy także zaszyć w skryptach zlecanych **sbatch**, wymieniając je w komentarzach na początku skryptu:

```
#SBATCH --opcja wartość
```



W przypadku, kiedy chcemy zlecić dużą liczbę zadań, różniących się jedynie parametrami, slurm daje nam do dyspozycji tablicę(?) zadań – job array.

Zlecamy jedynie komendą **sbatch**, przy pomocy opcji --array, np.:

- --array=0-31
- --array=1,3,5,7
- --array=1-7:2

Poszczególne elementy tablicy zadań będą mogły sprawdzić swój indeks przy pomocy zmiennej środowiskowej SLURM_ARRAY_TASK_ID

Więcej na: https://slurm.schedmd.com/job_array.html



Aplikacje na maszynach HPC są udostępniane przez moduły:

- module av pokaż dostępne moduły
- module load [module] załaduj moduł
- module unload [module] wyłącz moduł
- module show [module] pokaż co dokładnie robi dany moduł

Na maszynach ICM moduły dostępne są na węzłach obliczeniowych (konieczna flaga **-1** dla powłoki)



Logowanie: ssh username@hpc.icm.edu.pl Kopiowanie na serwer: scp /local/path host:host/path Kopiowanie z serwera: scp host:host/path /local/path Flagi scp analogiczne jak cp

Więcej informacji: https://kdm.icm.edu.pl/Tutorials/Logowanie/ssh.pl/

Zadania



Zapoznaj się z obecnym stanem dostępnych partycji. Jakie zadania znajdują się obecnie w kolejce?

Zaloguj się na węzeł obliczeniowy w sesji interaktywnej, wykorzystując jeden rdzeń. Wydrukuj nazwę węzła, do którego uzyskałeś/aś dostęp.



Zbadaj przydzielone zasoby, zlecając do kolejki program:

```
#include <mpi.h>
#include <omp.h>
#include <cstdlib>
#include <iostream>
#include <string>
int main(int argc, char *argv[]) {
 MPI_Init(&argc, &argv);
 const auto node id = std::getenv("SLURM NODEID");
 const auto node str = node id ? node id : "not launched via slurm";
 int comm sz. rank:
 MPI Comm size(MPI COMM WORLD, &comm sz);
 MPI Comm rank(MPI COMM WORLD, &rank);
 const int num_threads = omp_get_max_threads();
  const auto msg = std::string("Rank ") + std::to_string(rank) + " of " +
                   std::to string(comm sz) +
                   ". threads available: " + std::to string(num threads) +
                   ". slurm node ID: " + node str + "\n":
  std::cout << msg;</pre>
 MPI_Finalize();
```

Kompilacja (po załadowaniu modułu MPI):

```
$ mpic++ -fopenmp kod.cpp -o program.x
```



Uruchom 5 zadań, każde z których:

- 1. Oblicza trajektorię wzdłuż systemu Lorentza
 - od punktu [1, 1, 1] + mała, losowa perturbacja
 - w interwale czasowym [0,5]
 - wykorzystuje do obliczeń Pythona
- 2. Zapisuje wynik do pliku

Następnie w trybie interaktywnym wczytaj wszystkie wyniki i narysuj je na jednym wykresie (innymi kolorami). Zapisz wykres do pliku i obejrzyj wyniki – chaotyczne trajektorie wzdłuż atraktora Lorentza.

Do napisania potrzebnego kodu w Pythonie wykorzystaj GAI.