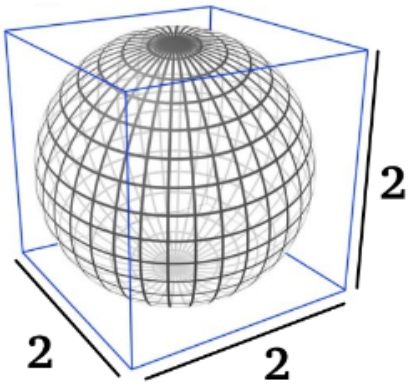
Python Scalability Hands On

This hands on tutorial is aimed to provide the user the skills to decide which kind of algorithm or parallelization method use /implement depending on the requirements (Time, efficiency or precission). Please download and upload to the cluster the hands-on files *here*.



Sphere into cube

Consist in the integration of an Hypersphere(4-D) using two integration algorithms: Rectangles and Montecarlo:

- The *Rectangle integration* is the "direct" implementation of the integral definition *b locked URL*. The definition can be reformulated as a sum over all the "space" of a Volume Differential *blocked URL*. For al x,y,z,t verifying *blocked URL*

- Otherwise, the *Montecarlo Inegration*(one of the possible implementations) consist in the random exploration of a finite "space" (trying random ponts and check if verify *blocked URL*) . Then, we determine the ratio "points inside sphere/Total points" *blocked URL* . The precission of this method is proportional to the square root of the number of tested points.

First steps

1. Copy the hands-on folder to your /scratch directory.

2. Enter the folder and check the contents. There are 3 python scripts (.py files) and 2 slm files. The python scripts consist in "rectangle" integral, mpi "rectangle" integral, and a MonteCarlo Integral. All of them are the integration of a 4D sphere (The theoretical result is around 4.93480....).

3. Launch the file integral.slm and wait for the result. You can check your job joining the node (ssh node) and typeing the "top" command (top -u user)

4. Open the file and modify the requirements to avoid the error(for exaple doubling some value) and finish the job normally.

5. Launch again the slm script and wait for the end. Note the total Walltime.

Exploring alternatives

We don't accept the total walltime because guess is to much. So we have two alternatives: Paralelize the Rectangle algorithm or change the algorithm(Montecarlo):

1. Assuming that we have parallelized our initial script, now called mpi_tr_integral.py (you can check it if you want), we have created a slm script to perform a scalability test (mpi_test.slm). Launch int and wait for the results (output file).

2. Alternativelly, we have coded a Montecarlo Inegration algorithm (which is sequential) called mc_integral.py to compute the same integral.
    a. Generate a slm file similar to "integral.slm" to run the mc_integral.py and launch it to the queue.

3. Check the results of both jobs (mpi_test and montecarlo integration). Compare it in some ways (WallTime, Numerical result, Monetary consumption,...) and decide which alternative fits to your requirements or you can propose a better alternative. Maybe you could propose another alternative.

Solutions

First Steps

Solution 3

```
sbatch integral.slm
```

Solution 4

```
#SBATCH -t 10
```

Solution 5

```
sacct -l
```

Exploring Alternatives

Solution 2

```
#!/bin/bash

#SBATCH -J Python_Montecarlo_integration
#SBATCH -p std
#SBATCH -o mc_integral.out
#SBATCH -e mc_integral.err
#SBATCH -n 1
#SBATCH -t 12

module load tools/python/gnu/2.7.14

srun python ./mc_integral.py
```

Solution 3

```
1) Focusing on WallTime, the fastest way is to launch the mpi rectangle integration with the most number
of cores(But is the most expensive).
2) In terms of Monetary efficiency is cheaper use the montecarlo integration, but losing precission in
the result.
3) Looking only the numerical result, the rectangular integration is more precise( 2 digits more than
MC).
4) An alternative good solution could be parallelize the Montecarlo algorithm and increase the number of
points to improve the precission without sacrifice Walltime.
```