Parallel and distributed computing with PhysiCell

In this practical session, we will explore PhysiCell's capabilities to run simulations in parallel. In particular, we will take advantage of the different cores that a single machine could have, and subsequently, the different nodes available at a supercomputer such as MN4.

Note: This tutorial was delivered by José Carbonell Caballero and Thalia Diniaco during the course *Introduction to HPC for Life Scientists* organised by PerMedCoE, BioExcel and PATC in Barcelona on 7 and 8 March 20223. The practical was run in MareNostrum 4 at the Barcelona Supercomputing Centre. To run it in other supercomputers, some of the commands might need to be modified.

1- Using different cores

To start the hands-on session you have to copy the PhysiCell's source code into your home directory.

```
> cp -r /gpfs/projects/nct00/nct00029/physicell_x .
> cd physicell_x
> ls -l
```

Then, the first step you have to do is the source code compilation, thereby getting the executable file that we will use in this section.

```
> module load gcc/8.1.0
> module switch impi openmpi/3.1.1
> make heterogeneity-sample-mpi
> make
> ls -l
```

After compilation, you should find an executable file called "heterogeneity.exe". This executable could be used to directly run PhysiCell on the machine we are logged. However, since we are working in a shared environment we should use the queuing system that the supercomputer is providing us. But, before execution, we have to apply a small modification in the PhysiCell's configuration file, aim to reduce the number of iterations, and therefore, the computation time needed for each run.

```
> vim config/PhysiCell_settings.xml
```

```
...
```

```
        <overall>
                <max_time units="min">120</max_time> <!-- 5 days * 24 h * 60 min →
                <time units>min</time units>
                <space_units>micron</space_units>

                <dt_diffusion units="min">0.01</dt_diffusion>
                <dt mechanics units="min">0.1</dt mechanics>
                <dt_phenotype units="min">6</dt_phenotype>
        </overall>

…
```

Now you are ready to send a PhysiCell job to the queuing system. To do so we will create a batch file including the fields required by Slurm to properly specify the number of needed resources. For this test, we will use 4 cores.

```
> vim batch_test
```

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=4
#SBATCH -t 00:10:00
#SBATCH -o test_1_4-%j.out
#SBATCH -e test 1 4-%j.err

mpiexec --map-by ppr:1:node:pe=4 ./heterogeneity.exe ./config/PhysiCell settings.xml
test_c4 4
```

Once created, we can finally send the job by using the “sbatch” command.

```
> sbatch --reservation=Intro23-day1 batch_test
```

Then, we can check the status of the simulation by using the following command.

```
> squeue
```

If you have been able to reach this point, congratulations, you now know how to launch a real PhysiCell job in the queueing system of MN4. Therefore, is time to finish our benchmark. In particular, we have to fill this table, which will give us an interesting overview of the use of parallel computing capabilities in this tool.

# Cores	Expected time	Observed time
1		

2		
4		
8		
16		

Finally, try to answer the following questions:

- What are the main conclusions you get from the obtained computing times?
- Is it useful to count on OpenMP capabilities on PhysiCell?
- Are the observed times expected? If not, explain why.
- Is there room for improvement?
- If you would send several times the same job, Would you obtain the same time?

Extra Tip 1 (automatizing the batch file creation):

Working in a cluster is all about being practical, and this implies identifying tasks that are repeatedly performed manually, and therefore need to be automated. In our case, we are in charge of performing a benchmark with PhysiCell. With this aim, we have to prepare a series of batch files corresponding to the execution of the tool with each number of cores we want to explore. These files are very similar, except for the main parameter (number of cores) that is repeated in different fields such as the job name, or the output folder. To automate the generation of this file we are going to generate the following shell script that takes only one input argument, representing the number of cores :

```
> vim create_batch
```

```
#!/bin/bash

nc=$1

name="test_n${nc}"


echo "#!/bin/bash"
echo "#SBATCH -n $nc"
echo "#SBATCH -t 10:00"
echo "#SBATCH -J $name"
echo "#SBATCH -o $name.out"
echo "#SBATCH -e $name.err"
echo "#SBATCH -D ."

echo "mpiexec --map-by ppr:1:node:pe=$nc ./heterogeneity.exe config/PhysiCell_settings.xml
$name $nc"
```

Remember that before running the script you need to grant it permissions to execute

```
> chmod +x create_batch
```

And then, we are ready to use it

```
> ./create_batch 4 > batch_test_c4
```

To check if we have done a good job we can check the content of the batch file we have generated.

```
> cat batch_test_c4
```

```
#!/bin/bash
#SBATCH -n 4
#SBATCH -t 10:00
#SBATCH -J test_n4
#SBATCH -o test_n4.out
#SBATCH -e test n4.err
#SBATCH -D .

./heterogeneity.exe config/PhysiCell_settings.xml test_n4 4
```

So now its time to use our new shell script to prepare all needed batch files:

```
> mkdir my_batches
> ./create_batch 1 > my_batches/batch_test_c1
> ./create_batch 2 > my_batches/batch_test_c2
> ./create_batch 4 > my_batches/batch_test_c4
> ./create_batch 8 > my_batches/batch_test_c8
> ./create_batch 16 > my_batches/batch_test_c16
```

As we have created all the files in the same folder (*my_batches*), we can submit our jobs directly using a simple for loop

```
> for i in my_batches/*; do echo $i; sbatch --reservation=Intro23-day1 $i; done
> squeue
```

```
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
          20387465 sequentia  test_n1 nct00007 PD       0:00      1 (None)
          20387466 sequentia test_n16 nct00007 PD       0:00      1 (None)
          20387467 sequentia  test_n2 nct00007 PD       0:00      1 (None)
          20387468 sequentia  test_n4 nct00007 PD       0:00      1 (None)
          20387469 sequentia  test_n8 nct00007 PD       0:00      1 (None)
```

2- Using different nodes

The PhysiCell tool integrates MPI technologies to implement distributed computing. In this case, we have to include additional fields into our batch test file to run the simulation as a MPI execution.

```
> vim batch_mpi_test
```

```
#!/bin/bash
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=8
#SBATCH -J test_c8_n2
#SBATCH -t 10:00:00
#SBATCH -o test c8 n2.out
#SBATCH -e test_c8_n2.err

mpiexec --map-by ppr:1:node:pe=8 ./heterogeneity.exe config/PhysiCell_settings.xml
test_n8_2 8
```

Once created, we can finally send the job:

```
> sbatch  --reservation=Intro23-day1 batch_mpi_test
```

Now, we can complete our benchmark, including more than one node.

# Nodes	# Cores	Expected time	Observed time
1	1		
1	2		
1	4		
2	1		
2	2		
2	4		
2	8		

Finally, try to answer the following questions:

- What are the main conclusions you get from the obtained computing times?

- Is it useful to count on MPI capabilities on PhysiCell?
- Are the observed times expected? If not, explain why.
- Are the obtained computing times with MPI comparable with the OpenMP ones using the same amount of resources?

You can also use this script to generate the different sbatch scripts by introducing first the number of cores and the the number of cpus.

First open the file

```
> vim batch_mpi_test
```

Then **replace** the code with the following:

```
#!/bin/bash
nc=$1
name="test n${nc} $2"


echo "#!/bin/bash"
echo "#SBATCH --nodes=$2"
echo "#SBATCH --ntasks-per-node=1"
echo "#SBATCH --cpus-per-task=$nc"
echo "#SBATCH -t 10:00"
echo "#SBATCH -J $name"
echo "#SBATCH -o $name.out"
echo "#SBATCH -e $name.err"
echo "#SBATCH -D ."


echo "mpiexec --map-by ppr:1:node:pe=$nc ./heterogeneity.exe config/PhysiCell_settings.xml
$name $nc"
```

Remember, we need to give execution rights to the file.

```
> chmod +x batch_mpi_test
```

For example if you wish to generate the sbatch file witch will run on 2 nodes and 24 cores:

```
> ./batch_mpi_test 24 2> test_24_2
```

Then submit the job to slurm.

```
> sbatch --reservation=Intro23-day1 test_24_2
```