# Programming with OmpSs - Single node hands-on

## Hardware Configuration and Login Information

| -- MareNostrum III -- | |
|---|---|
| - Peak Performance of 1,1 Petaflops<br>- 100.8 TB of main memory<br>- Homogeneous Nodes<br>    - 3,056 compute nodes<br>    - 2x Intel SandyBridge-EP E5-2670/1600 20M 8-core at 2.6 GHz<br>    - 8x4GB DDR3-1600 DIMMS (2GB/core)<br>- Heterogeneous Nodes<br>    - 42 heterogeneous compute nodes<br>    - 2x Intel SandyBridge-EP E5 -2670/1600 20M 8-core at 2.6 GHz<br>    - 2x Xeon Phi 5110 P<br>    - 8x8GB DDR3-1600 DIMMS (4GB/core)<br>- 2 PB of disk storage<br>- Interconnection networks:<br>    - Infiniband FDR10<br>    - Gigabit Ethernet<br>– Operating System: Linux-SuSe Distribution | Change password node:<br>**- dl01.bsc.es**<br><br>Login nodes:<br>**- mn3.bsc.es**<br><br>username: **nct010[01-30]**<br>passwd: ******<br><br>(*) Last five digits and password will be provided during the hands-on session. |

## Getting the exercises (and configuring OmpSs)

Connect to a MareNostrum *Login Node* (see information above about available login nodes) using your *username* and your *password*. It is important to enable X11 forwarding when starting your ssh session due we will use visualization tools.

Copy the heat-ompss.tar.gz file from ~nct00002/OmpSs directory to your home and unpack it (using **tar**). It will create a new directory called ompss-exercises-MN. Inside the directory you will find a configure script file. Make sure you execute **source** on that file to configure your environment in order to use OmpSs tools.

```
$ ssh -X username@login-node
login as: nct99999
nct99999@login-node's password: ******
Last login: Thu Jan 01 00:00:00 1970 from 00.00.00.00
nct99999@login-node:~>tar -xvf ~nct00002/heat-ompss.tar.gz
nct99999@login-node:~>cd heat-ompss
nct99999@login-node:~>source env.sh
```

All OmpSs exercises come with a makefile (***Makefile***). Some of them are also configured to compile different versions for each program: sequential version, OmpSs version, and instrumented version.

Each exercise may have several job scripts. For example, a script with suffix “-i” will generate a tracefile and a script with suffix “-g” will generate the task dependence graph. Before submitting any job, make sure all environment variables have the values you expect to. The job would be submitted using: “bsub < <job_script>”. While the jobs are queued you can check their status using the command “bjobs” (it may take a while to start executing). Once a job has been executed you will get two files. One for *console standard output* (with .out extension) and other for *console standard error* (with .err extension).

## Heat diffusion

During the hands-on we will do several exercises of increasing difficulty with the heat diffusion example. A plain sequential code and an initial OmpSs code are provided. A Makefile is also provided that enables to build the sequential version, the OmpSs version and the instrumented OmpSs version. In the text, actions required from your side are indicated with bullets (▪).

(a) Compiling and Executing OmpSs Programs

The first solver of the heat diffusion example has been parallelized with OmpSs (Jacobi). The strategy used has been to generate inline tasks for the two inner loops of the computation (the ones that process a block of the matrix).

- Analyse the proposed OmpSs parallelization by opening the heat-ompss.c and solver-ompss.c files.
- Compile the sequential, OmpSs and instrumented OmpSs versions using the provided Makefile.
- Execute and generate the task-dependence graph by using the run-ompss-g_mn.sh script. Use the queues (“ > bsub < run-ompss-g_mn.sh” ). Visualize and observe the graph.
- Execute and generate the Paraver tracefile by using the run-ompss-i_mn.sh (submiting the script to the queuing system). Visualize the tracefile. Analyze the execution at least with the following configuration files: “task.cfg”, “task_number.cfg”, “nb_tasks_in_execution.cfg”
- Execute a performance run using the run_ompss_mn.sh (using the queueing system). This will take a while, so launch the script and continue with next exercise. Afterward, analyse the results and draw execution and speed-up charts.

### *Parallelizing the Red-Black solver*

Propose an OmpSs parallelization for the Red-Black solver (you can follow the Jacobi strategy or think of another if you consider more appropriate).

- Write the solution.
- Compile and check that the solution is running correctly. To run the redblack solver, you should modify the “test*.dat” files to select this algorithm.
- Generate the graph by modifying the testgrind3.dat to select the redblack algorithm.
- Generate the tracefile by modifying the testgrind2.dat to select the redbalck algorithm.
- Compare the Jacobi and Red-Black solutions, both the graph and different Paraver views.

- Execute a performance run. Again, generate the execution and speed-up charts and compare with the Jacobi solution.
- What differences do you observe?

## *Parallelizing the Gauss-Seidel solver*

Propose an OmpSs parallelization for the Gauss-Seidel solver.

- Write the solution.
- Compile and check that the solution is running correctly. To run the redblack solver, you should modify the "test*.dat" files to select this algorithm.
- Generate the graph by modifying the testgrind3.dat to select the redblack algorithm.
- Generate the tracefile by modifying the testgrind2.dat to select the redbalck algorithm.
- Compare the Gauss-Seidel, Jacobi and Red-Black solutions, both the graph and different Paraver views.
- Execute a performance run. Again, generate the execution and speed-up charts and compare with the previous solutions.
- What differences do you observe?
- Can you improve your solution (i.e., by overlapping the execution of different iterations?)