MinoTauro User Guide
http://www.bsc.es/user-support/mt.php

Remember to include the necessary modules in your job scripts if needed.

Hands-on 0 Data transfer.

1. Log in with your user into one of the login nodes ($ ssh nct0000X@mt01.bsc.es or mt02.bsc.es).
2. Copy the file PATC2015_MPI.tar.gz from /gpfs/projects/nct01/ to your home in order to work later in MinoTauro. Use the cp command to make the copy:

$ cp /gpfs/projects/nct00/nct00001/PATC2015_MPI.tar.gz .

Once the copy has finished successfully, you will have the file in your home. Extract the contents to access the tutorials:

$ tar xzvf PATC2015_MPI.tar.gz

This will create a folder named PATC2015_MPI in the directory where the .tar.gz file is located. Within the folder you will find several subfolders with the exercises:

```
[nct01020@nvb127 ~]$ tar xzvf PATC2015_MPI.tar.gz
PATC2015_MPI/
PATC2015_MPI/1-hello_world/
PATC2015_MPI/1-hello_world/hello_world.c
PATC2015_MPI/1-hello_world/Makefile
PATC2015_MPI/1-hello_world/job.sh
PATC2015_MPI/2-message/
PATC2015_MPI/2-message/message.c
PATC2015_MPI/2-message/Makefile
PATC2015_MPI/2-message/job.sh
PATC2015_MPI/3-mxm/
PATC2015_MPI/3-mxm/Makefile
PATC2015_MPI/3-mxm/job.sh
PATC2015_MPI/3-mxm/mxm.c
PATC2015_MPI/README
```

Hands-on 1 Compilers

Exercise 1: Compilation with ICC and GCC

1. Go to **PATC2015_MPI/1-hello_world** directory
2. You can see a source file hello_world.c, job.sh and a Makefile
3. You can to compile this program with icc and gcc

4. See README for hints
    a. module list
       mpicc -o hello_world_icc hello_world.c
    b. module switch intel gcc
       module load gnu
       module li
       mpicc -o hello_world_gnu hello_world.c
5. Once compiled, execute both binaries and compare the results: $ ./hello_world_XX

For ease of use, there is also a Makefile which contains the needed commands to compile, build and submit a job. You can see its contents by issuing $ cat Makefile

You have several options: 'make' 'make clean' 'make submit' and 'make hello_world'. Study the different Make targets and the commands that get executed when you execute the commands above.

To get an interactive shell in MinoTauro, with up to six cores (half a node) you can use the command

$ mnsh2 –n 6 (n=1…6)

This will open up a bash in which you can start MPI programs interactively, using up to six MPI instances (CPU cores).

Note: Due to issues with the script, your session will be closed 60s after the first *srun application* finished.

Hands-on 2 MPI

Exercise 1: different MPI implementations

Go to 2-message/

In this exercise you have to compile and execute the same MPI test program with different MPI implementations. To do so, you have to load the correct environment before compile and execute.

$ module list will show you the available modules in MinoTauro

1. bullxmpi-icc (example)

If you have the default module environment, you just type 'make bullxmpi-icc'. This will generate the first bullxmpi-icc binary. Try to execute it doing:

srun -n 4 ./msg-bullxmpi-icc

2. openmpi-icc

Now you have to switch the modules environment to compile with Openmpi and GCC. Once this is done, execute: make openmpi-icc

srun -n 4 ./openmpi-icc

3. impi-icc
4. bullxmpi-gcc

Hands-on 3 SLURM

```
# @ job_name         = hello_world
# @ initialdir       = .
# @ output           = hello_world-%j.out
# @ error            = hello_world-%j.err
# @ total_tasks      = 64
# @ tasks_per_node   = 12
# @ gpus_per_node    = 2
# @ cpus_per_task    = 1
# @ wall_clock_limit = 00:05:00

srun ./hello_world
```

In this exercise you have to submit the same job with different configurations of tasks and tasks per node. In the directory **1-hello_world** you can find a hello_world.c program. This program is reporting for each task your MPI rank, MPI size and the node where the process is executed.

First compile the program with your preferred MPI implementation (Intel MPI, BullxMPI, OpenMPI,…).

If you execute interactively, you can see this in the output:

```
$ srun -n 4 ./hello_world
Hello world! I'm process 1 of 4 on login1
Hello world! I'm process 2 of 4 on login1
Hello world! I'm process 3 of 4 on login1
Hello world! I'm process 0 of 4 on login1
```

Exercise 1

Edit job.sh file in order to execute 32 MPI tasks using 8 tasks per node.

How many nodes did your job use?

Exercise 2

Edit job.sh to execute 27 MPI tasks using 8 tasks per node.

How many nodes did your job use? Have all the nodes the same MPI task allocation?

Hands-on 4 Matrix Multiplication

Exercise 1: Running the application

1. Go to 3-mxm/ directory
2. You can see a source file mxm.c, a Makefile and a job.sh
3. Execute “make && mnsubmit job.sh” to build the executable and submit the job. Use mnq to see the queue. This program multiplies two matrices with each other, computing the result in parallel.
4. Once the job has finished, look at the results. Just like in the previous exercise, change the number of tasks and tasks per node to see differences in the solving time.

Exercise 2: Variable Problem size

1. Edit the source file mxm.c

2. Change the following line to change the matrix size:
   #define N 5000 /* number of rows and columns in matrix */
3. Execute 'make' and 'make submit' to build and submit the job
4. How does the execution time change for different matrix sizes?
5. Try to find an efficient sweet spot and observe how execution time rises if you use too many processes for too small a problem.

Exercise 3: Change the number of MPI tasks

1. Edit the job file job.sh
2. Change the number of total tasks and/or tasks per node
3. Run the example by issuing 'make submit' or 'mnsubmit job.sh' and observe the difference in execution time for different numbers of computing tasks.

Additional information

ICC Optimization Flags

Option	Description	C/C++	F
-I, -L, -l, -lm, -c, -o	As usual		
-ansi_alias **-no-ansi_alias**	Can help performance. Directs the compiler to assume the following: • -Arrays are not accessed out of bounds. • -Pointers are not cast to non-pointer types, and vice-versa. • -References to objects of two different scalar types cannot alias. If your program does not satisfy one of the above conditions, this flag may lead the compiler to generate incorrect code. For Fortran, conformance is according to Fortran 95 Standard type aliasability rules. C/C++ Default = -no-ansi_alias (off) Fortran Default = -ansi_alias (on)		
-assume *keyword* **-assume buffered_io**	Specifies assumptions made by the compiler. One option that may improve I/O performance is `buffered_io`, which causes sequential file I/O to be buffered rather than being written to disk immediately. See the ifort man page for details.		
-auto **-automatic** **-nosave**	Places variables, except those declared as SAVE, on the run-time stack. The default is -auto_scalar (local scalar of types INTEGER, REAL, COMPLEX, or LOGICAL are automatic). However, if you specify -recursive or -openmp, the default is -auto.		
-save **-noauto** **-noautomatic**	Places variables, except those declared as AUTOMATIC, in static memory. However, if you specify -recursive or -openmp, the default is -auto.		

-autodouble Defines real variables to be REAL(KIND=8). Same as specifying -r8.

-convert *keyword* Specifies the format for unformatted files, such as big endian, little endian, IBM 370, Cray, etc.

-D*name*[=value] Defines a macro *name* and associates it with a specified *value*. Equivalent to a #define preprocessor directive.

-fast Shorthand for several combined optimization options: `-03, -ipo -static`

-fpp
-cpp Invoke Fortran preprocessor. -fpp and -cpp are equivalent.

-fpe[n] Specifies the run-time floating-point exception handling behavior:

- 0 - Floating underflow results in zero; all other floating-point exceptions abort execution.
- 1 - Floating underflow results in zero; all other floating-point exceptions produce exceptional values (signed Infinities or NaNs) and execution continues.
- 3 - All floating-point exceptions produce exceptional values (signed Infinities, denormals, or NaNs) and execution continues. This is the default.

-g Build with debugging symbols. Note that -g does not imply -O0 in the Intel compilers; -O0 must be specified explicitly to turn all optimizations off.

-help Print compiler options summary

-ip Enable single-file interprocedural optimizations.

-ipo Enable multi-file interprocedural optimizations.

-mcmodel=medium Required if executable is greater than 2 GB. The program is linked in the lower 2 GB of the address space but symbols can be located anywhere in the address space. Programs can be statically or dynamically linked, but building of shared libraries are not supported with the medium model.

-mp 'Maintain precision' - favor conformance to IEEE 754 standards for floating-point arithmetic.

-mp1 Improve floating-point precision - less speed impact than -mp.

-O0 Turn off optimizer - recommended if using -g for debugging.

-O, -O1, -O2, -O3 Optimization levels. (O,O1,O2 are essentially equivalent). -O3 is the most aggressive optimization level. Note that Intel compilers perform optimization by default.

-openmp Enable OpenMP

-opt_report **-opt_report_file *filename*** **-opt_report_level [min|med|max]** **-openmp_report[0|1|2]** **-par_report[0|1|2|3]**	Various reporting options on optimization, OpenMP, or auto-parallelization. See man pages for more information.
-p, -pg	Enables function profiling with the `gprof` tool. Same as `-qp`
-parallel	Enable auto-parallelizer to generate multi-threaded code for eligible loops.
-prof_gen **-prof_file** **-prof_use**	Used for profile guided optimization.
-pthread, -lpthread	Link with Pthreads library
-r8 **-r16** **-real_size 64** **-real_size 128**	Different ways to specify the default size of real and/or double-precision numbers.
-shared	Create a shared object (.a, .so)
-static	Enables linking to shared libraries (.so) statically.
-V	Display compiler version information
-w	Disable all warning messages
-w[0|1|2]	Increasing levels of warning message reporting. Default=1
-Wall (C/C++) **-warn (Fortran)**	Enable all warning messages
-vec	Utilize streaming SIMD instructions - turns on auto-vectorizer. This is the default.
-xavx	Utilize the advanced vector extension (AVX) streaming SIMD instructions. Sandy Bridge processor only.