Parallel programming for *supercomputing*

Daniel Muñoz-Santiburcio
Univ. Politécnica de Madrid

Parallel programming for *supercomputing*

Daniel Muñoz-Santiburcio
Univ. Politécnica de Madrid

An Introduction course thinking in the user's needs:
Tips, tricks and good practices for programming in supercomputers

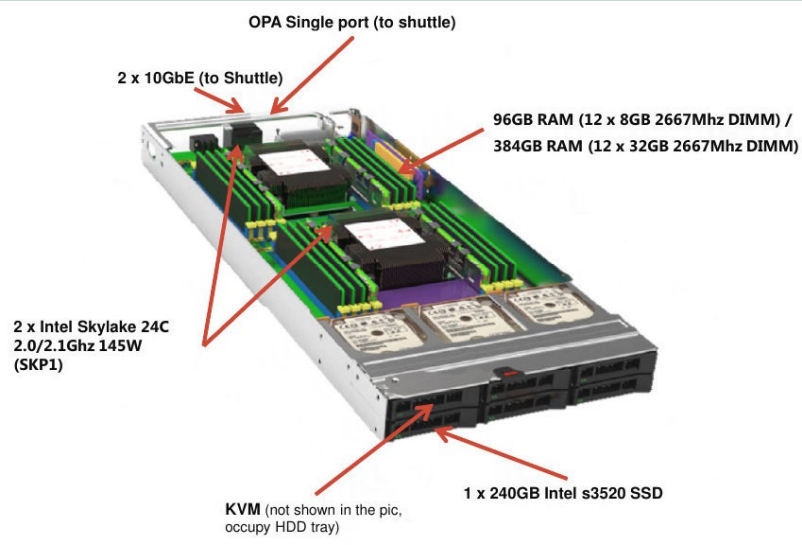
http://www.exeliz.com

Fundamentals

What is a supercomputer?

A supercomputer is usually composed of:

Racks
└ Nodes
└ Chips + Memory
└ Cores (CPUs)



Fundamentals

Important points to know about a supercomputer

- Know the hardware. E.g.: 1 node = 2 x (Intel Xeon silver 4316 @ 2.30 GHz, 20c)

- *Hyperthreading* (advertised as e.g. 2 threads/core) is *not necessarily* important

- Details like memory, interconnection (between nodes) and storage/filesystem are important

- A supercomputer is different from a *farm* (e.g. good vs. bad interconnection)

- Access details *may* be important (access via VPN or not, ssh keys, etc.)

- The Support team is *extremely* important

- Try to ask colleagues about experience in a given supercomputer before applying

Fundamentals

Important points to know about a supercomputer

- Details like {G/T/P/E}FLOPS are usually *not* important

- Sometimes ”CPU/processor” referes to the chip, sometimes to each core

- Computing/CPU time is almost always measured in **core-hours**

- Know the rules for accounting the computing time
  (e.g. using less than 1 node accounts like using 1 full node?)

- **Computing time is *precious*:** please be responsible

    - Try to run with the optimum number of nodes

    - Try to use the computer when it is idle

Fundamentals

Important points to know about a supercomputer

- The software is hugely important: know what is already installed and running

- There is a huge difference between using an off-the-shelf code and your own code

- Always know about compilers and libraries (what is installed vs. what you need)

- Particular versions and combinations of versions of compilers/libraries may be *crucial*

- <u>Always document</u> what you need

- Again, Support team is extremely important

Fundamentals

Do I need parallel computing?

- Parallel computing consists in taking advantage of the physical architecture of the computer to solve a problem quickly and efficiently

- Not all problems can be nicely parallelized. E.g. of best case scenario: matrix multiplication

- You must know the code and the nature of the problem

- Always RTFM (Read The Fine Manual): maybe there is (hidden) info about parallelization

- You may need a supercomputer not because of the cores, but because of the *memory*

Fundamentals

Some experience-based info about parallelization

- Usually, 1 node is the *sweetspot* for efficiency (performance always measured w.r.t. 1 node)

- In general, try to avoid using less than 1 node. Why:

  a) Possible performance problems because of different jobs/users in the same node
  b) Possible ”cascade” effects as a consequence of (*a*): jobs ”splitting” among nodes
  c) Possibly you will be accounted for the *full* node (remember: know the rules)

  - (*a*),(*b*) should not happen if the machine is well configured: again role of Support
  - Know whether the nodes are assigned fully/exclusively

- The parallelization within 1 node must be better than for > 1 node (intercommunication)
  … though it’s possible that you get almost perfect parallelization for more nodes

- Some machines have specially efficient combinations of nodes (e.g. multiples of 3)

Fundamentals

Some experience-based info about parallelization

- Using a lot of nodes is not always (actually almost never) better:

- Probability of crashes/errors (e.g. because of switch/node failures) increases with # of nodes: each crash leads to lost data and CPU time

    - ”Best case” scenario: crash kills immediately the job, dissapears of queue system
    - Bad case scenario: the executable stops running/printing, but stays in queue as ”Running”: blocks resources, and you’ll be accounted that computing time
    - Worst case scenario: job apparently ends fine but some error caused wrong results (rare but can happen)

- Always monitor your jobs for errors (read the *full* output!)

- Be careful with the Input/Output: too much writing slows down the job

Fundamentals

Tips for compiling and testing

- When you are compiling any code, document *everything* (versions of compilers and libraries, particular order of loaded modules, errors...)

- Document <u>especially well</u> the procedure that compiles the code successfully (in particular, store configuration files, module info and *Makefiles* like treasures)

- Prepare a ”pet job” for your code, which can run fast but demanding enough to check the parallelization (e.g. something that can run in 1-4 nodes in few minutes).

- Document the output and the timings of that job for every machine

- Always check the reproducibility of your ”pet job” in every machine (results and timings):

    - Discrepancies serve to discover configuration / compilation problems!

MPI and OpenMP

Basics

- There are two main frameworks for writing parallel code: MPI and OpenMP
  (do not confuse with ”OpenMPI”, which is an open-source implementation for MPI)

- In both, there are compilers for Fortran, C and C++

- If the code is not Fortran or C/C++, there may be other options:
  - Interpreted languages (e.g. Python, R, Matlab) with multithreading (over MPI/OpenMP)
  - CUDA (for GPUs)
  - Others

- Writing a parallel code is not at all straightforward: very different from the serial version

- There may (surely will) be parts that must be serial

- Coding for MPI and OpenMP is quite different

OpenMP

Basics

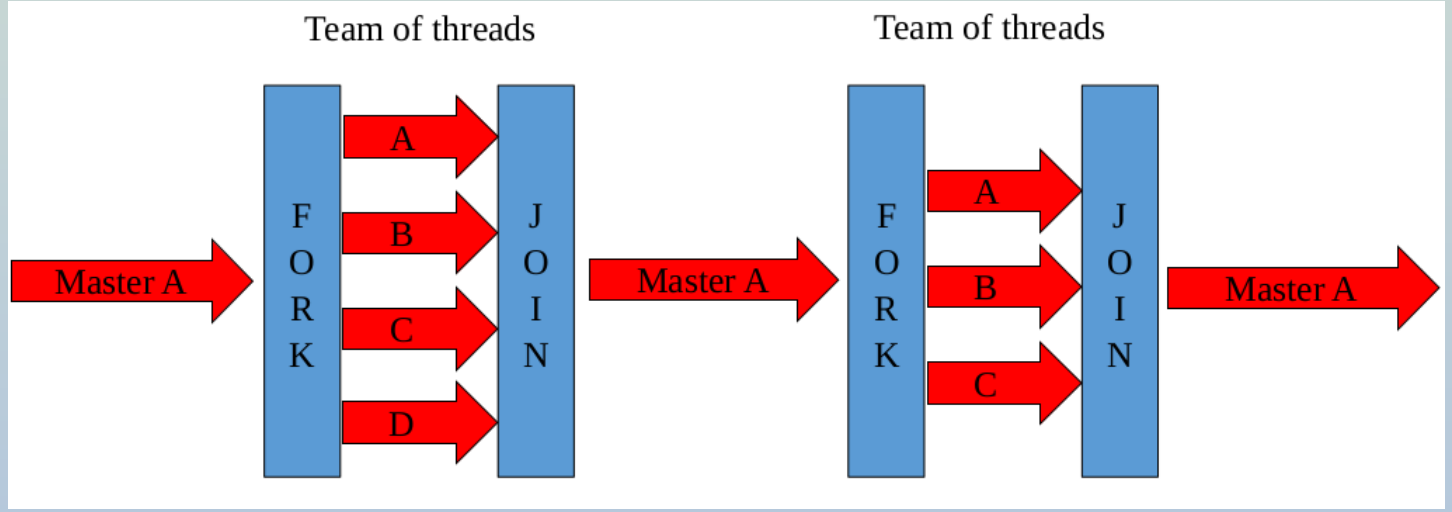
- OpenMP = ”Open Multi Processing”
- For **<u>shared memory</u>** machines: it can use up to 1 node
- For beginners seems easier and more natural than MPI
- The computation is distributed among ”threads” (e.g. each core runs 1 thread)
- All threads have access to the same memory:
    - All threads can read/write to the same variables!
    - You can enforce ”private” variables for each thread

OpenMP

Basics

- ”Fork-join” model: seems a natural way of tackling a problem

1) program starts as serial execution: 1 master thread
2) create more threads when arriving to a parallel region
3) when finished, the other threads disappear and master keeps running



OpenMP

Hello World in OpenMP (Fortran90 example)

```
program hello

use omp_lib

implicit none


write(*,*) " Hello! I am thread number ", OMP_get_thread_num()
write(*,*) " We could be up to ", OMP_get_max_threads() ," more threads"
write(*,*) " Currently there are ", OMP_get_num_threads() ," threads in total"


!$omp parallel

write(*,*) " I am thread number ", OMP_get_thread_num(),," out of ", OMP_get_num_threads(), "  threads in total"

!$omp end parallel


write(*,*) " Hello! I am thread number ", OMP_get_thread_num()
write(*,*) " We could be up to ", OMP_get_max_threads() ,"more threads"
write(*,*) " Currently there are ", OMP_get_num_threads() ," threads in total"

end program hello
```

Hello World in OpenMP: set the OMP_NUM_THREADS variable

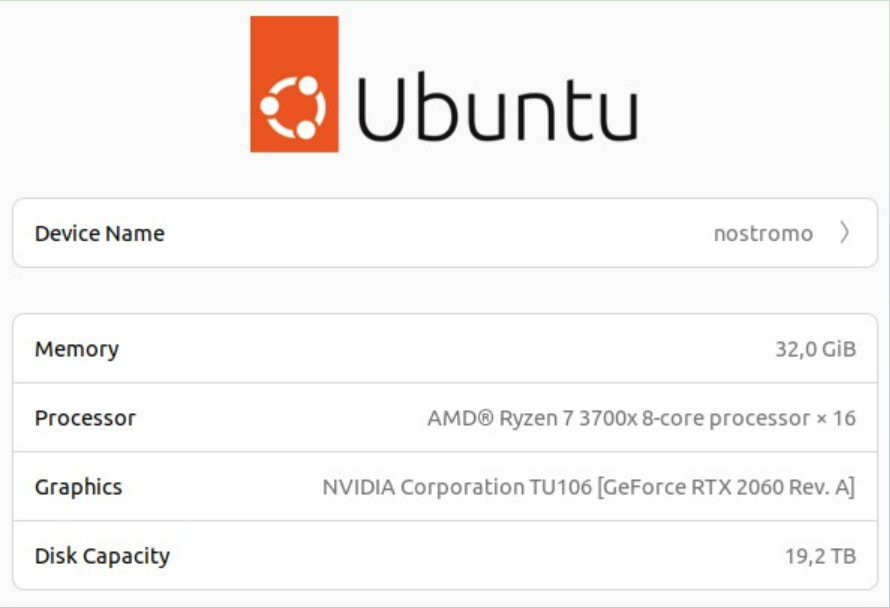
```
daniel@nostromo:~/WORK/curso_UCA/code$ gfortran -fopenmp hello_omp.f90 -o hello_omp.exe
daniel@nostromo:~/WORK/curso_UCA/code$ export OMP_NUM_THREADS=4
daniel@nostromo:~/WORK/curso_UCA/code$ ./hello_omp.exe
  Hello! I am thread number            0
  We could be up to            4  more threads
  Currently there are            1  threads in total
  I am thread number            2  out of            4   threads in total
  I am thread number            0  out of            4   threads in total
  I am thread number            1  out of            4   threads in total
  I am thread number            3  out of            4   threads in total
  Hello! I am thread number            0
  We could be up to            4 more threads
  Currently there are            1  threads in total
daniel@nostromo:~/WORK/curso_UCA/code$
daniel@nostromo:~/WORK/curso_UCA/code$ export OMP_NUM_THREADS=6
daniel@nostromo:~/WORK/curso_UCA/code$ ./hello_omp.exe
  Hello! I am thread number            0
  We could be up to            6  more threads
  Currently there are            1  threads in total
  I am thread number            0  out of            6   threads in total
  I am thread number            4  out of            6   threads in total
  I am thread number            3  out of            6   threads in total
  I am thread number            1  out of            6   threads in total
  I am thread number            2  out of            6   threads in total
  I am thread number            5  out of            6   threads in total
  Hello! I am thread number            0
  We could be up to            6 more threads
  Currently there are            1  threads in total
daniel@nostromo:~/WORK/curso_UCA/code$
```

OpenMP

Hello World in OpenMP: set the OMP_NUM_THREADS variable

```
daniel@nostromo:~/WORK/curso_UCA/code$ echo $OMP_NUM_THREADS

daniel@nostromo:~/WORK/curso_UCA/code$ ./hello_omp.exe
  Hello! I am thread number            0
  We could be up to           16  more threads
  Currently there are            1  threads in total
  I am thread number           13  out of           16   threads in total
  I am thread number            0  out of           16   threads in total
  I am thread number            2  out of           16   threads in total
  I am thread number            9  out of           16   threads in total
  I am thread number            4  out of           16   threads in total
  I am thread number           11  out of           16   threads in total
  I am thread number            1  out of           16   threads in total
  I am thread number            5  out of           16   threads in total
  I am thread number            3  out of           16   threads in total
  I am thread number           12  out of           16   threads in total
  I am thread number           14  out of           16   threads in total
  I am thread number           15  out of           16   threads in total
  I am thread number            8  out of           16   threads in total
  I am thread number            6  out of           16   threads in total
  I am thread number            7  out of           16   threads in total
  I am thread number           10  out of           16   threads in total
  Hello! I am thread number            0
  We could be up to           16 more threads
  Currently there are            1  threads in total
daniel@nostromo:~/WORK/curso_UCA/code$ export OMP_NUM_THREADS=2
daniel@nostromo:~/WORK/curso_UCA/code$ echo $OMP_NUM_THREADS
2
daniel@nostromo:~/WORK/curso_UCA/code$ ./hello_omp.exe
  Hello! I am thread number            0
  We could be up to            2  more threads
  Currently there are            1  threads in total
  I am thread number            0  out of            2   threads in total
  I am thread number            1  out of            2   threads in total
  Hello! I am thread number            0
  We could be up to            2 more threads
  Currently there are            1  threads in total
daniel@nostromo:~/WORK/curso_UCA/code$
```



MPI

Basics

- MPI = ”Message Passing Interface”

- For **<u>distributed memory</u>** machines: can use whatever number of nodes/cores

- The computation is distributed among ”processes” (e.g. each core runs 1 process)

- All the processes start as soon as MPI is initialized

- By default, each process has its own private memory

- Each process must communicate with the others in order to work

- We must say which process does whatever operation (otherwise *all* do it)

MPI

Hello World in MPI (Fortran90 example)

```
program hello

implicit none

include 'mpif.h'

  integer id
  integer ierr
  integer num_procs

  call MPI_Init ( ierr )
  call MPI_Comm_rank ( mpi_comm_world, id, ierr )
  call MPI_Comm_size (  MPI_COMM_WORLD, num_procs, ierr )

  write(*,*) " I am process number ", id, " of a total of ", num_procs

  if ( id .eq. 0 )  write(*,*) " I am the master process with id = ", id

  call MPI_Finalize ( ierr )

end program hello
```

MPI

Hello World in MPI (Fortran90 example)

```
daniel@nostromo:~/WORK/curso_UCA/code$ mpif90 hello_mpi.f90 -o hello_mpi.exe
daniel@nostromo:~/WORK/curso_UCA/code$ mpirun -n 2 hello_mpi.exe
  I am process number            0  of a total of            2
  I am the master process with id =            0
  I am process number            1  of a total of            2
daniel@nostromo:~/WORK/curso_UCA/code$ mpirun hello_mpi.exe
  I am process number            5  of a total of            8
  I am process number            6  of a total of            8
  I am process number            0  of a total of            8
  I am the master process with id =            0
  I am process number            1  of a total of            8
  I am process number            3  of a total of            8
  I am process number            4  of a total of            8
  I am process number            7  of a total of            8
  I am process number            2  of a total of            8
daniel@nostromo:~/WORK/curso_UCA/code$
```

MPI and OpenMP

Compiling / running with MPI and OpenMP in supercomputers

- OpenMP is usually supported by the ”regular” compilers (e.g. gfortran in a regular Ubuntu):

```
> gfortran  -fopenmp  program.f90  -o  program.exe

> ifort  -qopenmp  program.f90  -o program.exe
```

- MPI requires a specific compiler (e.g. OpenMPI, Intel compiler suite)

```
> mpif90  program.f90 -o  program.exe
```

  - ”mpif90” can be either OpenMPI or Intel
  - Always know which modules are loaded!

MPI and OpenMP

Compiling / running with MPI and OpenMP in supercomputers

```
upm84318@login2:~> module list

Currently Loaded Modules:
  1) intel/2017.4   2) impi/2017.4   3) mkl/2017.4   4) bsc/1.0



upm84318@login2:~> which mpif90
/apps/INTEL/2017.4/impi/2017.3.196/bin64/mpif90
upm84318@login2:~> module purge
remove mkl/2017.4 (LD_LIBRARY_PATH)
remove impi/2017.4 (PATH, MANPATH, LD_LIBRARY_PATH)
upm84318@login2:~> module load openmpi
Lmod has detected the following error:  Cannot load module "openmpi/1.10.7". At least one of these module(s) must be loaded:
   gcc/7.1.0 gcc/4.9.4 intel

While processing the following module(s):
    Module fullname  Module Filename
    ---------------  ---------------
    openmpi/1.10.7   /apps/modules/modulefiles/environment/openmpi/1.10.7

upm84318@login2:~> module load gcc/7.1.0
Set GNU compilers as MPI wrappers backend
upm84318@login2:~> module load openmpi
load openmpi/1.10.7 (PATH, MANPATH, LD_LIBRARY_PATH)
upm84318@login2:~> which mpif90
/apps/OPENMPI/1.10.7/GCC/7.1.0/bin/mpif90
upm84318@login2:~>
```

MPI and OpenMP

Compiling / running with MPI and OpenMP in supercomputers

- Some compilers / executables may produce different results
  - mpif90 vs mpiifort ?
  - mpirun vs mpiexec ?

```
upm84318@login2:~> module list

Currently Loaded Modules:
  1) intel/2017.4   2) impi/2017.4   3) mkl/2017.4   4) bsc/1.0



upm84318@login2:~> mpi
mpicalc            mpiexec            mpifc              mpiicpc            mpi-selector-menu
mpicc              mpiexec.hydra      mpigcc             mpiifort           mpitune
mpicleanup         mpif77             mpigxx             mpirun             mpivars.csh
mpicxx             mpif90             mpiicc             mpi-selector       mpivars.sh
upm84318@login2:~>
```

MPI and OpenMP

Hybrid MPI+OpenMP codes

- Idea: use several nodes with MPI parallelization between the nodes, and OpenMP parallelization within *each* node

- Seems the natural way of doing it: mimicks the physical structure of the supercomputer

- Usually assumed as the most efficient strategy

- However, in practice it happens quite often that pure MPI is more efficient