MPI lecture and labs 1

Victor Eijkhout

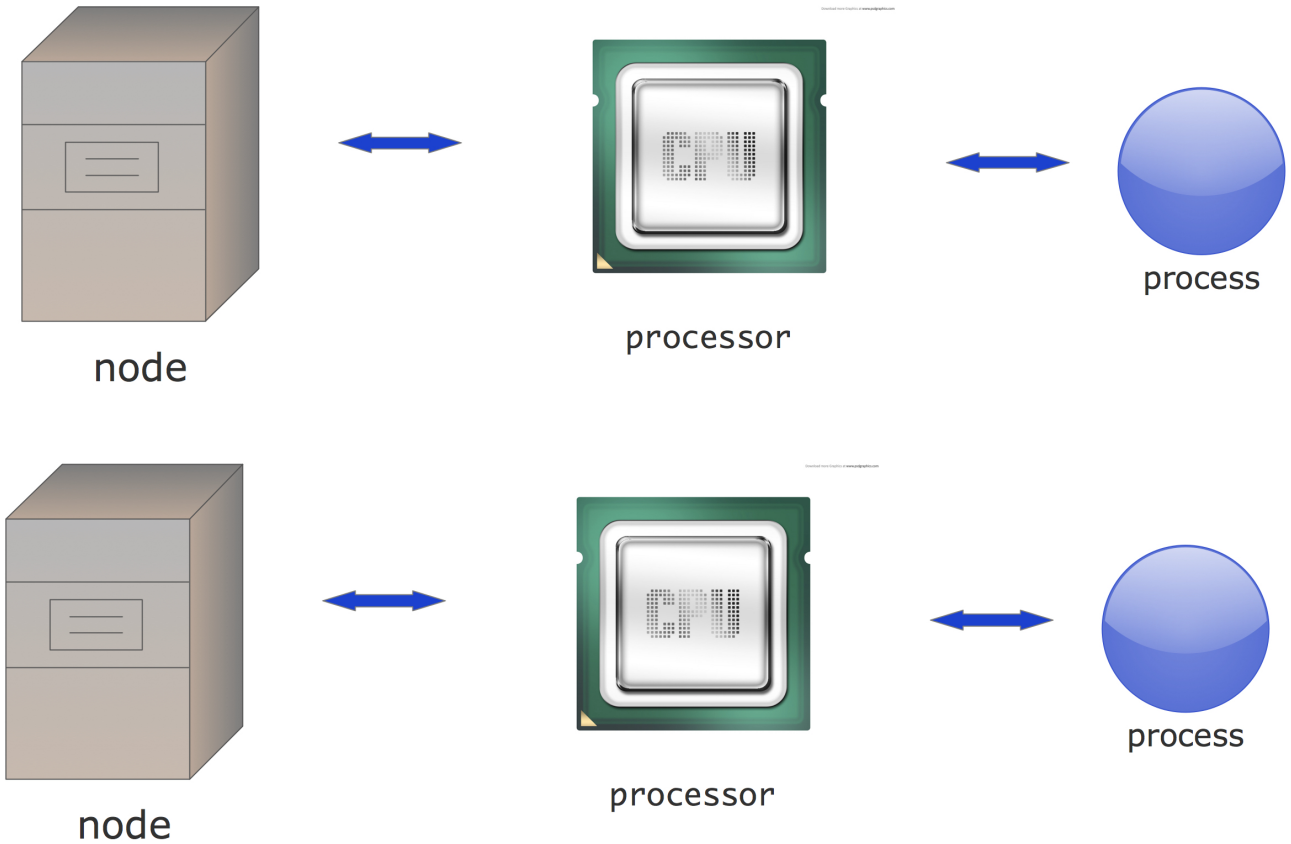
2016

Justification

The MPI library is the main tool for parallel programming on a large scale. This course introduces the main concepts through lecturing and exercises.

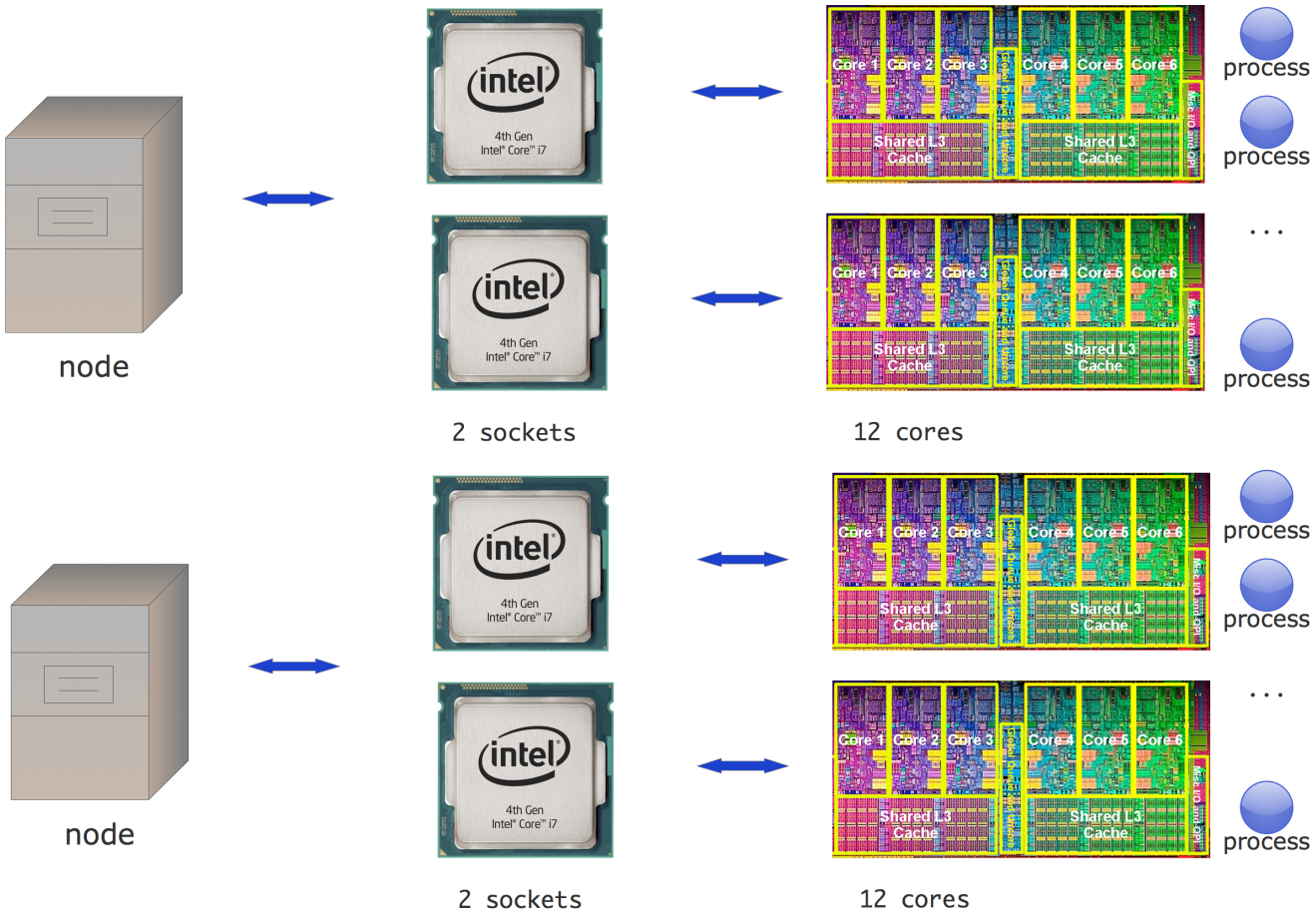
The SPMD model

Computers when MPI was designed



One process per node; all communication goes through the network.

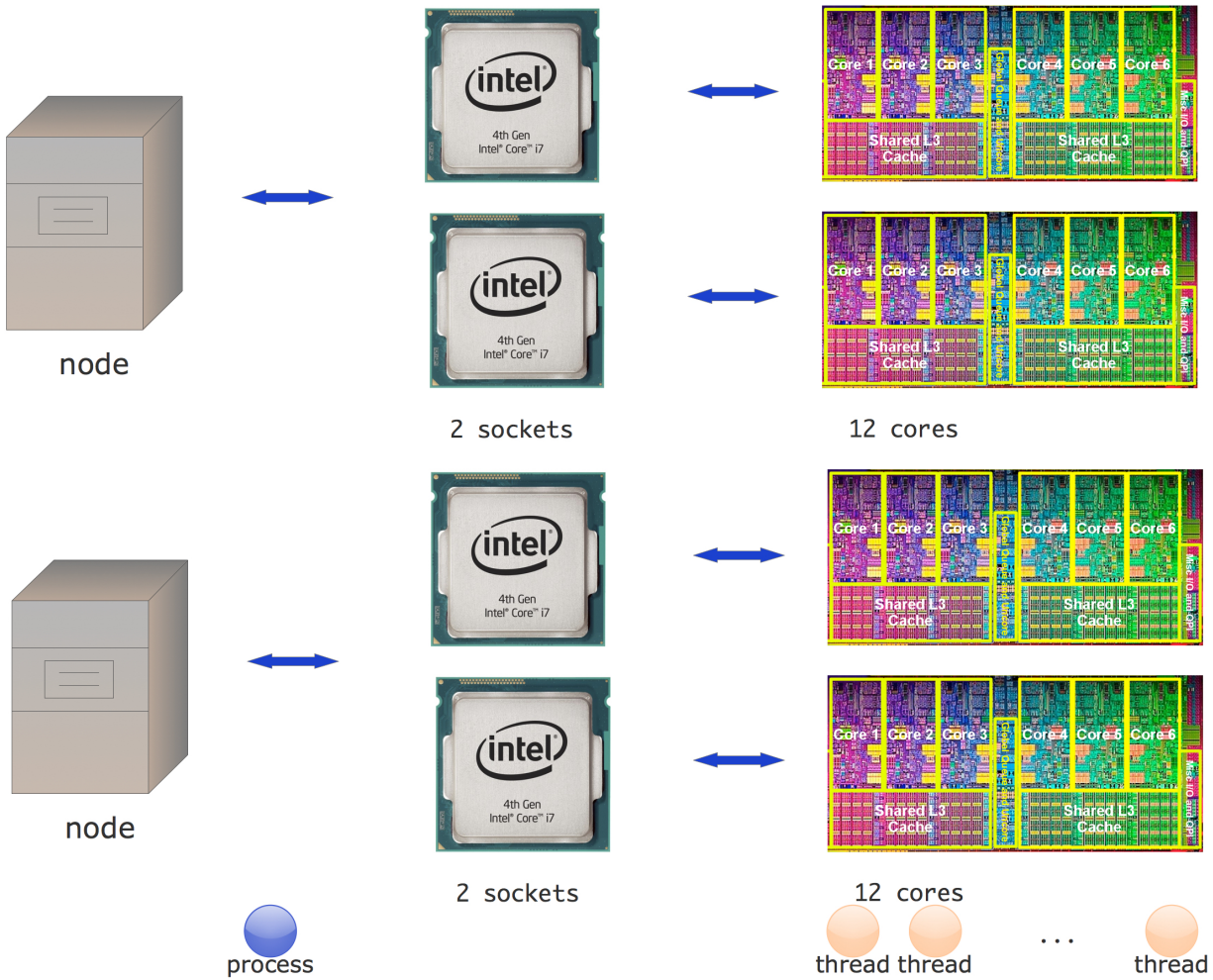
Pure MPI



A node has multiple sockets, each with multiple cores.

Pure MPI puts a process on each core: pretend shared memory doesn't exist.

Hybrid programming



Hybrid programming puts a process per node or per socket; further parallelism comes from threading.

Lab setup

Open two windows on stampede. In one window you will be editing and compiling, in the other, type `idev -N 2 -n 32 -t 2:0:0` which gives you an interactive session of 2 nodes, 32 cores, for the next 2 hours.

The C compiler is `mpicc`, Fortran is `mpif90`. To run (on a compute node!) type `ibrun yourprog`

Exercise 1

Write a ‘hello world’ program, without any MPI in it, and run it in parallel with `mpiexec` or your local equivalent. Explain the output.

MPI Init / Finalize

You need an include file:

```
#include "mpi.h" // for C
#include "mpif.h" ! for Fortran
```

Then put these calls around your code:

```
MPI_Init(&argc,&argv);
// your code
MPI_Finalize();
```

and for Fortran:

```
call MPI_Init(ierr)
! your code
call MPI_Finalize(ierr)
```

Python bindings

```
module load python-mpi
```

```
from mpi4py import MPI
```

No initialization needed.

Exercise 2

Now add the commands `MPI_Init` and `MPI_Finalize` to your code. Put three different print statements in your code: one before the init, one between init and finalize, and one after the finalize. Again explain the output.

Exercise 3

Now use the command `MPI_Get_processor_name` in between the init and finalize statement, and print out on what processor your process runs. Confirm that you are able to run a program that uses two different nodes.

TACC nodes have a hostname `cRRR-CNN`, where RRR is the rack number, C is the chassis number in the rack, and NN is the node number within the chassis. Communication is faster inside a rack than between racks!

C:

```
int MPI_Get_processor_name(char *name, int *resultlen)
```

Fortran:

```
MPI_Get_processor_name(name, resultlen, ierror)
CHARACTER(LEN=MPI_MAX_PROCESSOR_NAME), INTENT(OUT) :: name
INTEGER, INTENT(OUT) :: resultlen
INTEGER, OPTIONAL, INTENT(OUT) :: ierror
```

Python:

```
MPI.Get_processor_name()
```

Processor identification

Every processor has a number (with respect to a communicator)

```
int MPI_Comm_rank( MPI_Comm comm, int *rank )
int MPI_Comm_size( MPI_Comm comm, int *size )
```

For now, the communicator will be `MPI_COMM_WORLD`.

Note: mapping of ranks to actual processors and cores is not predictable!

Exercise 4

Write a program where each process prints out message reporting its number, and how many processes there are.

Write a second version of this program, where each process opens a unique file and writes to it. *On some clusters this may not be advisable if you have large numbers of processors, since it can overload the file system.*

Exercise 5

Write a program where only the process with number zero reports on how many processes there are in total.

Functional Parallelism

Parallelism by letting each processor do a different thing.

Example: divide up a search space.

Each processor knows its rank, so it can find its part of the search space.

Exercise 6

Is the number $N = 2,000,000,111$ prime? Let each process test a range of integers, and print out if they find a factor. You don't have to test all integers $< N$: any factor is at most $\sqrt{N} \approx 45,200$.