

COMPUTATIONAL MATH, SCIENCE AND ENGINEERING DEPARTMENT

MICHIGAN STATE
UNIVERSITY

MPI programming

Getting started

Motivation

Up to now we have been dealing with a thread view of parallel processing:

- low overhead executable element
- shared memory within a process
- no explicit way to communicate outside of a process with other threads
 - can be done but no direct support
 - support is in the process

Motivation(2)

Plenty of room for application of multi-threading, but if you want to step up to HPC you need to work with 100's, nay 1000's of processors

- can't be the thread model, because clearly we have to be outside of a single process
- need to work with many processors.



MPI (Message Passing Interface)

Industry and academia got together to provide a specification of how multiple processors, across different architectures, could be used for parallel processing

- MPI 1 (94), 1.1 (95), 1.2 (97)
- MPI 2 (97), 2.2(06)
- MPI 3 (12)



standard, not implementation

- MPI is a standard, created by a small group of individuals who (in all likelihood) will have to implement it
- not an IEEE standard (good and bad)
- as a result there are multiple implementations, each with their own quirks



MPI implementations

- mpich (<http://www.mpich.org/>)
- openmpi(<http://www.open-mpi.org/>)
- lam/mpi (<http://www.lam-mpi.org/>)
 - basically subsumed by openmpi now
- Intel MPI (<http://software.intel.com/en-us/intel-mpi-library>)
 - in HPC as IMPI, requires intel compilers

On HPC OpenMPI is default



Roughly 3 main goals

- creation of a topology of processes that are used in the computation
- a way for those processes to communicate with each other
- synchronization to coordinate their efforts.



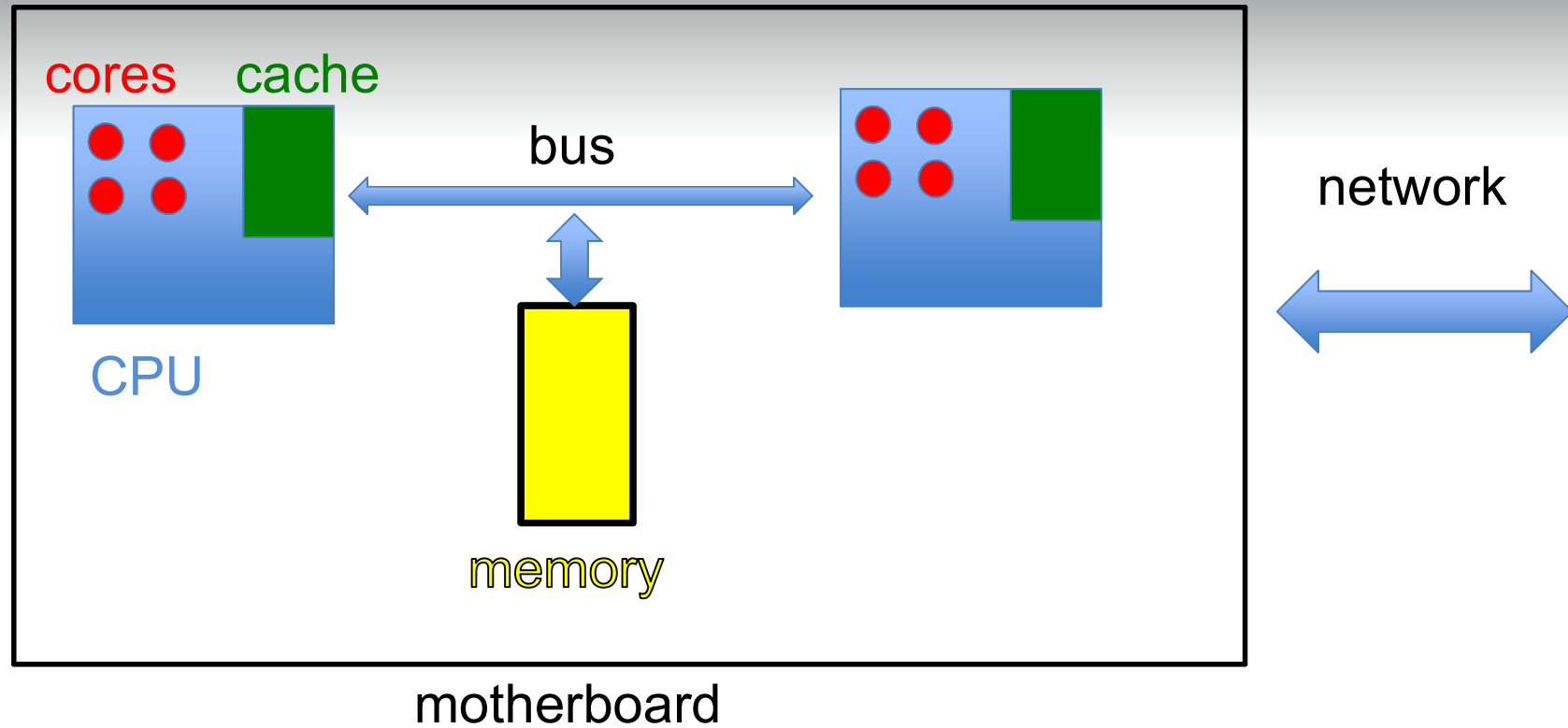
Process == computational element

Process does not necessarily mean Unix process.

- more like a computational element
- could be a thread, a node, another computing system on the network
- MPI does not (necessarily) take a stance on the kinds of things that count as a process

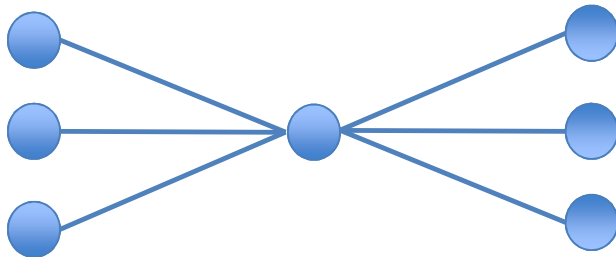


layout



typically distributed memory

- can create a virtual topology
interconnect between CE
 - remember, could be threads, could be chips on the same board, could be at UIUC Illinois



distributed, everybody gets a copy

The default approach is that every process gets a copy of the data

- distributed!

Ways around it, need more advanced features.

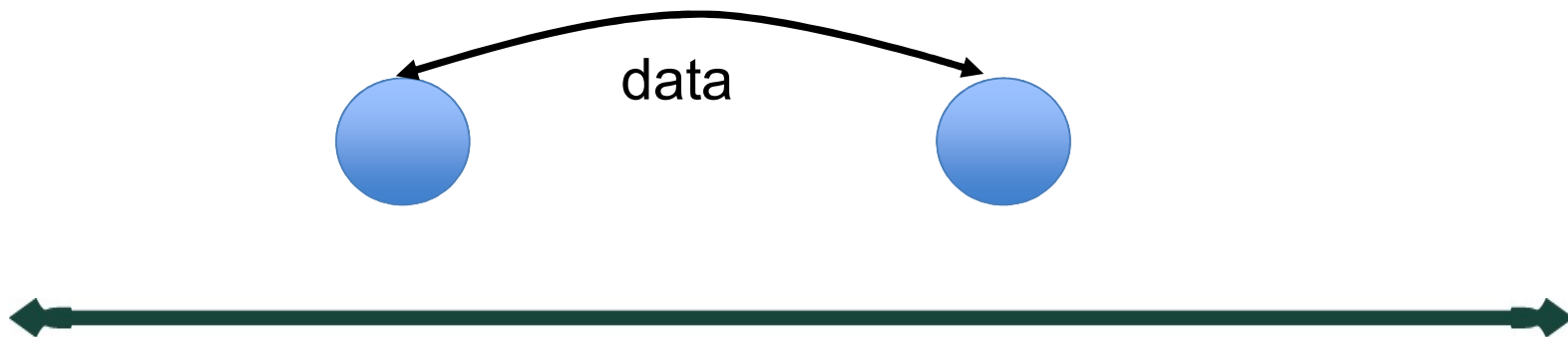
- we'll see later (kind of ugly actually)



various ways to exchange info

The fundamental property of MPI is the ability to exchange information between CE on the topology

- make this as seamless as possible so that, no matter the architecture of the CE, can move data



no shared memory

Even if the CE are on the same chip, the assumption is no shared memory

- have to pass info between CE as messages
- allows for a level of abstraction
- local implementation can make "local messages" faster



need synch

- Probably obvious, but we need synch to make all this stuff work



A warning

- The MPI standard supports C, C++ and Fortran
 - not really, in 2012 C++ bindings were deprecated with the intent of removing them
 - interesting story of "no one willing to do the work"
 - still in most implementations
 - some ways around this we will use



COMPUTATIONAL MATH, SCIENCE AND ENGINEERING DEPARTMENT

MICHIGAN STATE
UNIVERSITY*basic example*

```

5312 0000488B 15470B00 004889D6 4889C7E8 6E050000 488B1535 0B000048 89D64889 C7E85C05 0000488D 35790600 00488B05 140B0000 4889C7E8 7C050000 BE080000
5376 004889C7 E85D0500 00488B15 000B0000 4889D648 89C7E827 050000E8 10050000 4889C348 8D354A06 0000488B 05070A00 004889C7 E83F0500 004889D6 4889C7E8
5440 1C050000 488B15C5 0A000048 89D64889 C7E8EC04 0000E8CF 04000048 89C3488D 351E0600 00488B05 9C0A0000 4889C7E8 04050000 4889D648 89C7E8E1 04000048
5504 8B158A0A 00004889 D64889C7 E8B10400 00488B15 780A0000 4889D648 89C7E89F 04000048 8D35E705 0000488B 05570A00 004889C7 E8BF0400 000B0800 00004889
5568 C7E8A004 0000488B 15430A00 004889D6 4889C7E8 6A040000 488B1531 0A000048 89D64889 C7E85804 0000488D 35B70500 00488B05 100A0000 4889C7E8 78040000
5632 BE040000 004889C7 E8590400 00488B15 FC090000 4889D648 89C7E823 040000E8 F4030000 660F7EC3 488D3588 05000048 8B05D209 00004889 C7E83A04 0000660F
5696 6EC34889 C7E8A004 0000488B 15BF0900 004889D6 4889C7E8 E6030000 E8B10300 00660F7E C3488D35 5B050000 488B0595 09000048 89C7E8FD 03000066 0F6EC348
5760 89C7E8CD 03000048 8B158209 00004889 D64889C7 E8A90300 00488D35 36050000 488B0561 09000048 89C7E8C9 030000BE 18000000 4889C7E8 9E030000 488B154D
5824 09000048 89D64889 C7E87403 0000488B 153B0900 004889D6 4889C7E8 62030000 488D3509 05000048 8B051A09 00004889 C7E88203 0000BE08 00000048 89C7E863
5888 03000048 8B150609 00004889 D64889C7 E82D0300 00E8E602 00006648 0F7EC348 C050488B 05D80800 004889C7 E8430300 0066480F 6EC34889 C7E80603
5952 0000488B 15C70800 004889D6 4889C7E8 EE020000 E8A10200 0066480F 7EC3488D 35AC0400 00488B05 9C080000 4889C7E8 04030000 66480F6E C34889C7 E8C70200
6016 00488B15 88080000 4889D648 89C7E8AF 02000048 8D358704 0000488B 05670800 004889C7 E8CF0200 00BE3500 00004889 C7E8A402 0000488B 15530800 004889D6
6080 4889C7E8 7A020000 488B1541 08000048 89D64889 C7E86802 0000488D 355B0400 00488B05 20080000 4889C7E8 88020000 BE100000 004889C7 E8690200 00488B15
6144 0C080000 4889D648 89C7E89F 04000048 8B150609 00004889 D64889C7 E8A90300 00488D35 36050000 488B0561 09000048 89C7E8C9 030000BE 18000000 4889C7E8 9E030000 488B154D
6208 00004889 D648 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
6272 4889D648 89C7E8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
6336 8B060000 004883C4 485B5DC3 554889E5 4883EC10 897DFC89 75F88370 FC017531 817DF8FF FF000075 29488D3D F8070000 E87D0100 00488D15 04E7FFFF 488D35E5
6400 07000048 8B05F606 00004889 C7E86C01 0000C9C3 554889E5 BEFFFF00 00BF0100 0000E8A5 FFFFFF5D C3554889 E5B80080 FFFF5DC3 554889E5 B8FF7F00 005DC355
6464 4889E5B8 00000080 5DC35548 89E5B8FF FFFF7F5D C3554889 E5488B00 00000000 0000805D C3554889 E5488BFF FFFF7F5D C3554889 E58B0589 01000066
6528 0F6EC05D C3554889 E58B057D 01000066 0F6EC05D C3554889 E5488B00 00000000 00100066 480F6EC0 5DC35548 89E5488B FFFF7F5D C3554889 E58B0589 01000066
6592 4889E548 B8000000 00000000 00B8A010 00004889 45F08955 F8DB6D0F 5DC35548 89E548C7 C0FFFFF7 FFB8FE7F 00004889 45F08955 F8DB6D0F 5DC35548 89E548C7 C0FFFFF7
6656 FF252A06 0000FF25 2C060000 FF252E06 0000FF25 30060000 FF253206 0000FF25 34060000 FF253606 0000FF25 38060000 FF253A06 0000FF25 3C060000 FF253E06
6720 0000FF25 40060000 FF254206 0000FF25 44060000 FF254606 0000FF25 48060000 FF254A06 0000FF25 4C060000 FF254E06 0000FF25 50060000 FF255206 0000FF25
6784 54060000 4C8D1D95 05000041 53FF2585 05000090 68000000 00E9E6FF FFFF6819 000000E9 DCF7FFFF 682B0000 00E9D2FF FFFF683D 000000E9 E809D2FF
6848 00E9BEFF FFFF6861 000000E9 B4FFFFF7 68730000 00E9AAFF FFFF6885 000000E9 A0FFFFF7 68970000 00E996FF FFFF68B7 000000E9 8CFFFFF7 68F70000 00E982FF
6912 FFFF6800 00000000 FFFF77F7 FFFF77FF 53697A65 206F6620 73686F72 743A0053 6D616C6C 65737420 73686F72 743A004C 61726765 73742073 686F7274 3A005369
6976 7A65206F 6620696E 743A0053 6D616C6C 65737420 696E743A 004C6172 70696737A 3A005369 66206C6F 66E73A00 65737420 686F7274 3A005369 66206C6F
7040 673A004C 61726765 7374206C 6F6E673A 0053697A 65206F66 206C6F6E 67206C6F 6E672069 6E743A00 53697A65 206F6620 666C6F61 743A0053 6D616C6C 65737420
7104 666C6F61 743A004C 61726765 73742066 6C6F6174 3A004469 67697473 20696E20 6D617469 7376612C 20666C6F 61743A00 53697A65 206F6620 646F7562 6C653A00
7168 536D616C 6C657374 20646F75 626C653A 004C6172 67657374 20646F75 626C653A 00446967 69747320 696E206D 61746973 73612C20 646F7562 6C653A00 53697A65
7232 206F6620 6C6F6E67 20646F75 626C653A 00536D61 6C6C6573 74206C6F 6E67206A 6F75626C 653A004C 61726765 7374206C 6F6E6720 646F7562 6C653A00 00000000
7296 44696769 74732069 6E206D61 74697373 612C206C 6F6E6720 646F7562 6C653A00 14000000 00000000 017A5200 01781001 100C0708 90010000 34000000 1C000000
7360 69FCFFFF FFFFFFFF 0B000000 00000000 00040100 00000E10 86020403 00000000 06040600 00000C07 08000000 00000000 00000000 34000000 3CFCFFFF FFFFFFFF
7424 0B000000 00000000 00040100 00000E10 86020403 00000000 06040600 00000C07 08000000 00000000 34000000 00000000 00000000 00000000 00000000
7488 00040100 00000E10 86020403 00000000 06040600 00000C07 08000000 00000000 34000000 C4000000 E2FBFFFF FFFFFFFF 0B000000 00000000 00040100 00000E10
7552 86020403 00000000 06040600 00000C07 08000000 00000000 34000000 FC000000 B5FBFFFF FFFFFFFF 10000000 00000000 00040100 00000E10 86020403 00000000
7616 06040B00 00000C07 08000000 00000000 34000000 34010000 8DFBFFFF FFFFFFFF 10000000 00000000 00000000 00040100 00000E10 86020403 00000000 06040B00
7680 08000000 00000000 34000000 6C010000 65FBFFFF FFFFFFFF 10000000 00000000 00040100 00000E10 86020403 00000000 06040B00 00000C07 08000000 00000000
7744 34000000 A4010000 3DFBFFFF FFFFFFFF 10000000 00000000 00040100 00000E10 86020403 00000000 06040B00 00000C07 08000000 00000000 34000000 DC010000
7808 15FBFFFF FFFFFFFF 15000000 00000000 00040100 00000E10 86020403 00000000 06041000 00000C07 08000000 00000000 34000000 14020000 F2FAFFFF FFFFFFFF
7872 15000000 00000000 00040100 00000E10 86020403 00000000 06041000 00000C07 08000000 00000000 CFFAFFFF FFFFFFFF 1F000000 00000000 00000000
7936 00040100 00000E10 86020403 00000000 06041A00 00000C07 08000000 00000000 34000000 84020000 B6FAFFFF FFFFFFFF 1C000000 00000000 00040100 00000E10
8000 86020403 00000000 06041700 00000C07 08000000 00000000 34000000 BC020000 02FAFFFF FFFFFFFF 6A050000 00000000 00040100 00000E10 86020403 00000000
8064 06040500 00000303 04600500 000C0708 34000000 F4020000 34F9FFFF FFFFFFFF 48000000 00000000 00040100 00000E10 86020403 00000000 06044300 00000C07
8128 08000000 00000000 34000000 2C030000 44F9FFFF FFFFFFFF 15000000 00000000 00040100 00000E10 86020403 00000000 06041000 00000C07 08000000 00000000
8192 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
8256 BF190000 01000000 75190000 01000000 85190000 01000000 4A190000 01000000 3F190000 01000000 65190000 01000000 55190000 01000000 34190000 01000000
8320 29190000 01000000 9E1A0000 01000000 A31A0000 01000000 B21A0000 01000000 BC1A0000 01000000 C61A0000 01000000 D01A0000 01000000 D81A0000 01000000
8384 DA1A0000 01000000 E41A0000 01000000 EE1A0000 01000000 F81A0000 01000000 14190000 01000000 00000000 00000000 00000000 00000000 00000000
8448 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
8512 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
8576 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

```

CMSE 822, FS21, W.F. Punch

gotta do hello world, right

Let's do the hello world mpi style and
see what the basics look like



SPMD

- You can think of MPI runs as "single program multiple data"
- it is not unlike what you saw in OpenMP
 - you run a single program
 - each CE gets a copy of the program
 - distinguish each CE in some way



```

#include <iostream>
using std::cout; using std::endl;
#include <mpi.h>

int main(int argc, char **argv){
    int    comm_sz; // total procs
    int    my_rank; // my id

    // MPI from here
    MPI_Init(&argc, &argv);

    // get # of procs from communicator
    MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);

    // get my id from the communicator
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    if (my_rank == 0)
        cout << "I'm the master process 0" << endl;
    else
        cout << "I'm process:" << my_rank << ", out of:" << comm_sz << endl;

    // MPI finished here
    MPI_Finalize();
}

```

need
mpi.h

setup MPI

total CE
we have

my id

each CE does
work based on
their id

clean up

how to compile

MPI provides its own set of compilers and its own way to run the resulting compiled code

These are wrappers so they use the underlying compiler with the right flags.

Using these tools also makes sure you get the includes and libraries right.

compilers

This is on HPCC. I would stick to one of development machines for the moment.

- dev-intel14
- dev-intel16
- dev-intel18
- dev-amd20



versioning

If you want the latest and greatest, you need to load (in order):

- `module load GCC/11.1.0-cuda-11.4.0`
- `module load OpenMPI/4.1.1`




compiler (latest, greatest)

```
mpic++ my_code.cpp
```

```
mpic++ --showme
```

```
/opt/software/GCC/11.1.0-cuda-11.4.0/bin/c++ -  
I/opt/software/OpenMPI/4.1.1-GCC-11.1.0-cuda-  
11.4.0/include -pthread -  
L/opt/software/hwloc/2.4.1-GCCcore-11.1.0/lib -  
Wl,-rpath -Wl,/opt/software/hwloc/2.4.1-  
GCCcore-11.1.0/lib -Wl,-rpath -  
Wl,/opt/software/OpenMPI/4.1.1-GCC-11.1.0-cuda-  
11.4.0/lib -Wl,--enable-new-dtags -  
L/opt/software/OpenMPI/4.1.1-GCC-11.1.0-cuda-  
11.4.0/lib -lmpi
```



can use all the normal flags

```
mpic++ -Wall -g -std=c++11 helloWord.cpp
```

Just adds your commandline flags to the other flags



simple run

You can just run the job, but only 1 processor

```
> ./a.out
```

I'm the master process 0



run more processors

```
mpirun -n proc_cnt a.out  
-n, how many processors
```

```
>mpirun -n 4 a.out
```

```
I'm the master process 0
```

```
I'm process:1, out of:4
```

```
I'm process:2, out of:4
```

```
I'm process:3, out of:4
```



srun vs mpirun vs mpiexec

The startup of an MPI job can be confusing:

- mpirun: startup script, no standard for what it does. Most common
- mpiexec: MPI standard, often interchangeable with mpirun.
- srun: SLURM specific for MPI runs. mpirun probably calls this.



hostfile

You can create a file called the hostfile which lists the individual names of the machines you want to run on.

```
>cat my_hosts  
dev-intel14-k20  
dev-intel14-k20  
dev-intel14-k20  
dev-intel14-k20
```



run with the hostfile

```
>mpirun -hostfile my_hosts a.out
```

I'm the master process 0

I'm process:1, out of:4

I'm process:2, out of:4

I'm process:3, out of:4



batch run a job

You cannot list the set of hosts to run on since jobs are scheduled. However, you can request jobs in a script and, when the scheduler goes, the scheduler will create such a file so your job can run.



https://help.rc.ufl.edu/doc/Sample_SLURM_Scripts

```
#!/bin/bash
#SBATCH --job-name=mpi_job_test      # Job name
#SBATCH --mail-type=END,FAIL        # Mail events (NONE, BEGIN, END, FAIL, ALL)
#SBATCH --mail-user=email@msu.edu   # Where to send mail.
#SBATCH --ntasks=24                 # Number of MPI tasks (i.e. processes)
#SBATCH --cpus-per-task=1           # Number of cores per MPI task
#SBATCH --nodes=2                   # Maximum number of nodes to be allocated
#SBATCH --ntasks-per-node=12        # Maximum number of tasks on each node
#SBATCH --ntasks-per-socket=6       # Maximum number of tasks on each socket
#SBATCH --distribution=cyclic:cyclic # Distribute tasks cyclically first among nodes and
                                     # then among sockets within a node
#SBATCH --mem-per-cpu=600mb         # Memory (i.e. RAM) per processor
#SBATCH --time=00:05:00             # Wall time limit (days-hrs:min:sec)
#SBATCH --output=mpi_test_%j.log    # Path to the standard output and error files
relative to the working directory

echo "Date                = $(date)"
echo "Hostname            = $(hostname -s)"
echo "Working Directory = $(pwd)"
echo ""
echo "Number of Nodes Allocated    = $SLURM_JOB_NUM_NODES"
echo "Number of Tasks Allocated     = $SLURM_NTASKS"
echo "Number of Cores/Task Allocated = $SLURM_CPUS_PER_TASK"

mpirun -n $SLURM_NTASKS myExecutableName
```

What is required?

- ntasks: that's how many MPI CE you get.



submit

```
sbatch -constraint="[intel16|intel18]" hello.sbatch
```

- constraint not required but it does allow machine selection

