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

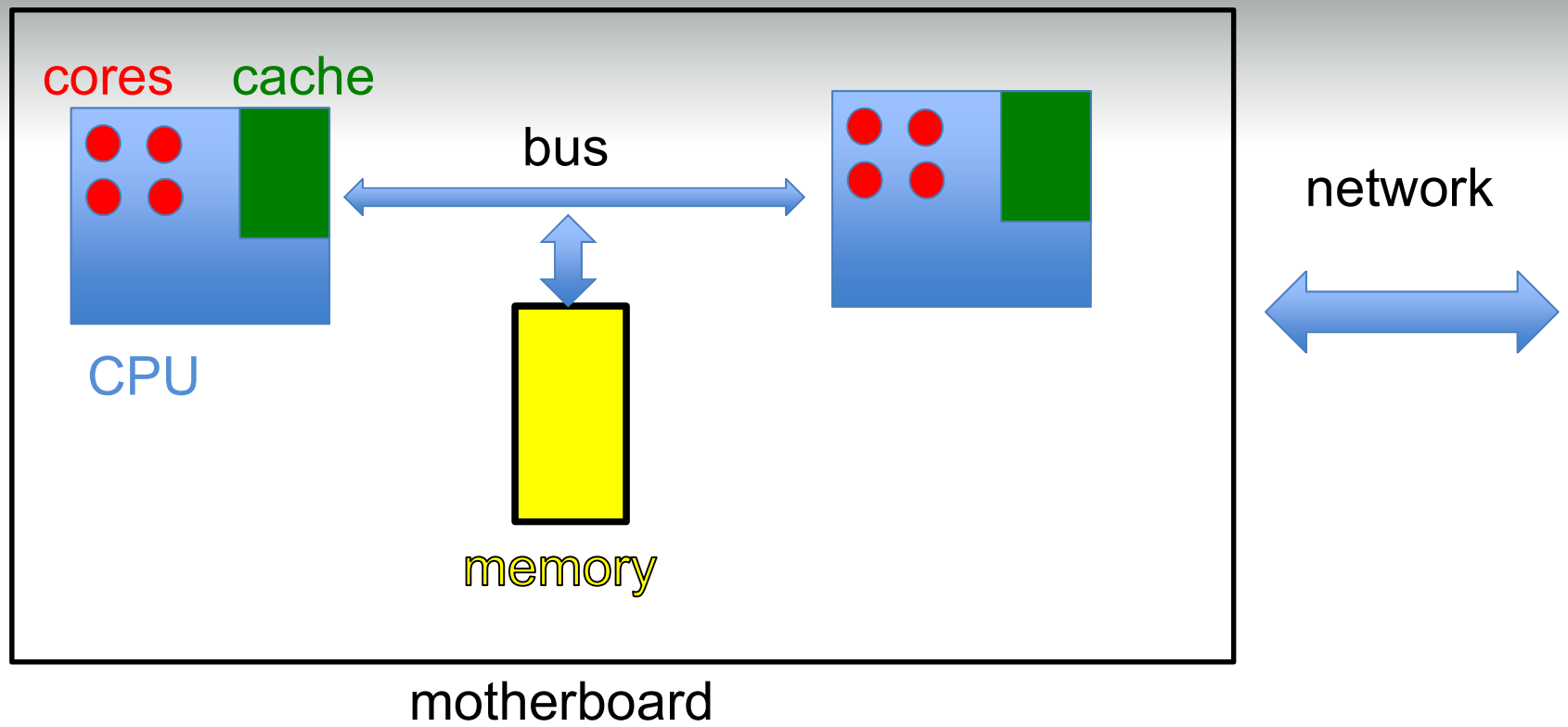
- creation of a topology of processes that are used in the computation
- a way for those processes to communicate with each other
- synchronization to coordinate their efforts.

Process == computational element

Process does not necessarily mean Unix process.

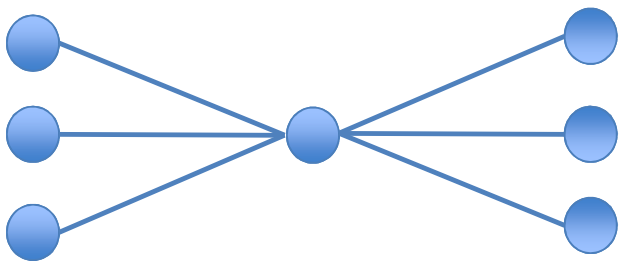
- more like a computational element
- could be a thread, a node, another computing system on the network
- MPI does not (necessarily) take a stance on the kinds of things that count as a process

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!

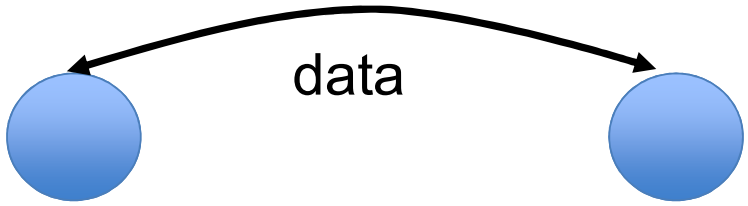
Ways around it, need more advanced features.

- 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

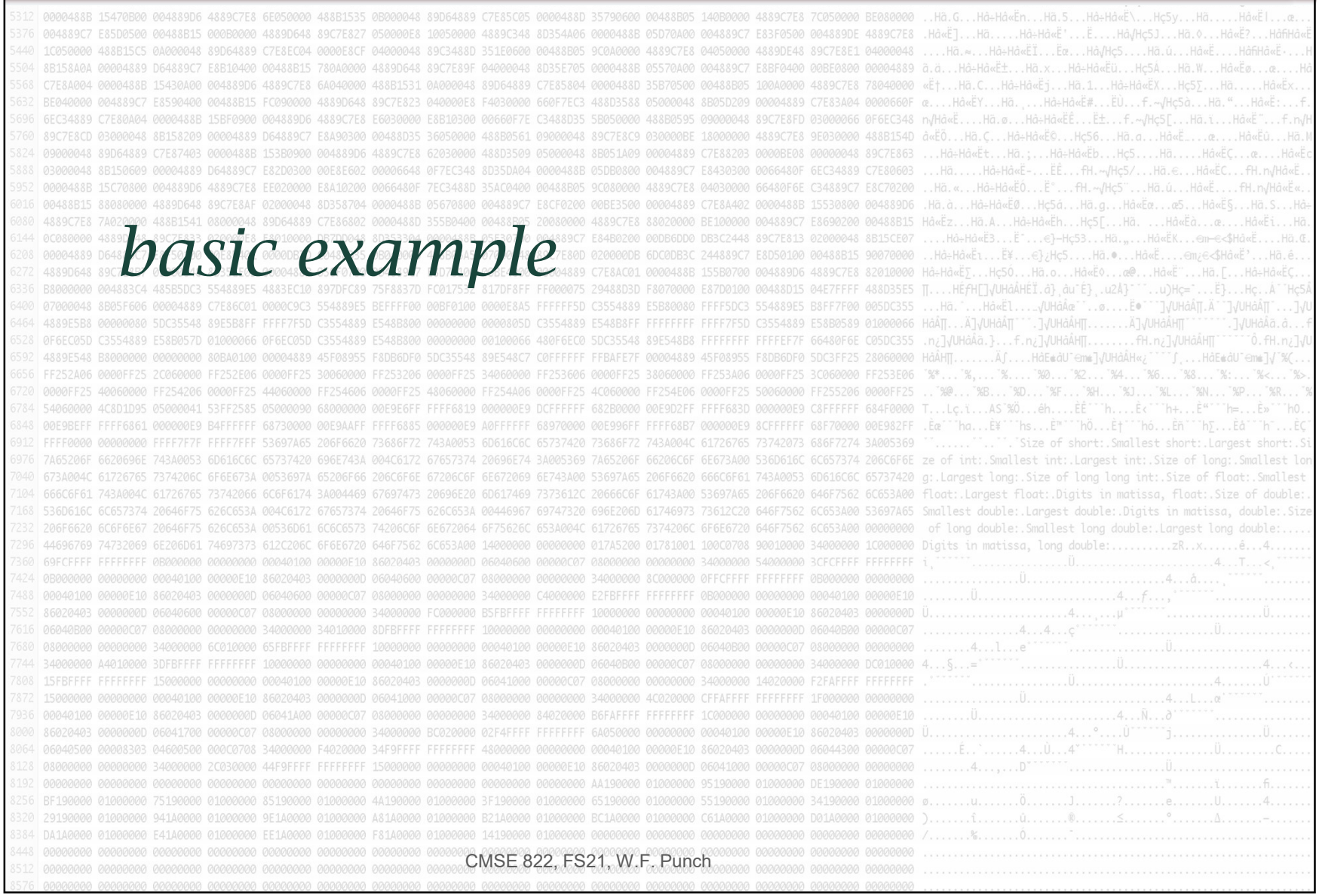
need synch

- 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

basic example



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