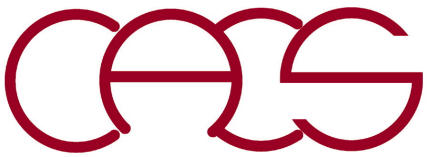
Message Passing Interface (MPI) Programming

Aiichiro Nakano

Collaboratory for Advanced Computing & Simulations
Department of Computer Science
Department of Physics & Astronomy
Department of Chemical Engineering & Materials Science
Department of Biological Sciences
University of Southern California

Email: anakano@usc.edu



How to Use USC HPC Cluster

System: Xeon/Opteron-based Linux cluster

http://hpcc.usc.edu/support/infrastructure/hpcc-computing-resource-overview

Node information

http://hpcc.usc.edu/support/infrastructure/node-allocation

Partition Names	Node Range	# of Nodes	Mem Size	Cores per Node	CPU Speed GHz	CPU Type	GPUs per Node	GPU Model	avx	Model	/tmp Size	Network
large main quick	hpc0965 – hpc0972	8	24 GB	12	3.0	xeon	–	–	–	sl160	110 GB	myri
large main quick	hpc4331 – hpc4388	58	128 GB	20	2.4	xeon	2	p100	avx avx2	xl190r	1.79 TB	IB

How to Use USC HPC Cluster

Log in

```
> ssh anakano@hpc-login3.usc.edu
```

hpc-login1: 32-bit i686 instruction-set codes

hpc-login2, hpc-login3: 64-bit x86_64 instruction-set codes

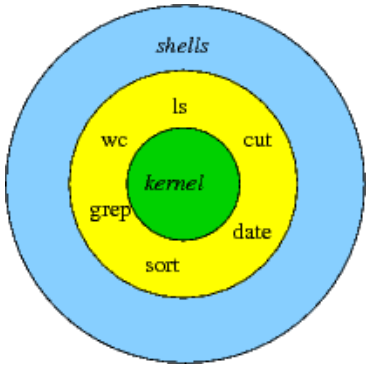
To use the MPI library:

if using C shell (or tcsh), add in .cshrc

```
source /usr/usc/openmpi/default/setup.csh
```

else if using bash, add in .bashrc

```
source /usr/usc/openmpi/default/setup.sh
```



`echo $0` to find which shell you are using

Compile an MPI program

```
> mpicc -o mpi_simple mpi_simple.c
```

Execute an MPI program

```
> srun -n 2 mpi_simple
```

```
[anakano@hpc-login3 ~]$ which mpicc
/usr/usc/openmpi/1.8.8/slurm/bin/mpicc
```

Submit a Slurm Batch Job

Prepare a script file, mpi_simple.sl

```
#!/bin/bash
#SBATCH --ntasks-per-node=2
#SBATCH --nodes=1
#SBATCH --time=00:00:10
#SBATCH --output=mpi_simple.out
#SBATCH -A lc_an1
WORK_HOME=/home/rcf-proj/an1/yourID
cd $WORK_HOME
srun -n $SLURM_NTASKS --mpi=pmi2 ./mpi_simple
```

Slurm (Simple Linux Utility for Resource Management): Open-source job scheduler that allocates compute resources on clusters for queued jobs

Total number of processors = ntasks-per-node × nodes

Submit a Slurm job

```
hpc-login3: sbatch mpi_simple.sl
```

Check the status of a Slurm job

```
hpc-login3: squeue -u anakano
Tue Aug 14 08:07:38 2018
JOBID   PARTITION NAME     USER    STATE   TIME TIME_LIMI NODES NODELIST(REASON)
1362179 quick     mpi_simp anakano RUNNING 0:03 1:00      1     hpc1118
```

Kill a Slurm job

```
hpc-login3: scancel 1362179
```

Sample Slurm Output File

```
hpc-login3: more mpi_simple.out
----------------------------------------
Begin SLURM Prolog Tue 14 Aug 2018 08:07:39 AM PDT
Job ID:         1362179
Username:       anakano
Accountname:    lc_an1
Name:           mpi_simple.sl
Partition:      quick
Nodes:          hpc1118
TasksPerNode:   2
CPUSPerTask:    Default[1]
TMPDIR:         /tmp/1362179.quick
SCRATCHDIR:     /staging/scratch/1362179
Cluster:        uschpc
HSDA Account:   false
End SLURM Prolog
----------------------------------------
n = 777
```

quick	hpc1118 – hpc1122	5	48 GB	24	2.3	opteron	–	–	–	dl165	890 GB	myri

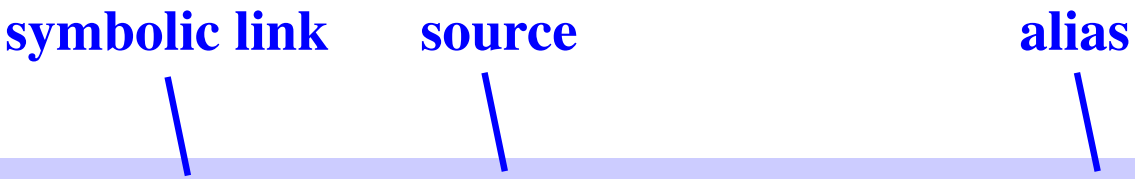
Interactive Job at HPC

Reserve 2 processors for 60 minutes

```
$ salloc -n 2 -t 60
salloc: Pending job allocation 1362924
salloc: job 1362924 queued and waiting for resources
salloc: job 1362924 has been allocated resources
salloc: Granted job allocation 1362924
salloc: Waiting for resource configuration
salloc: Nodes hpc1118 are ready for job
----------------------------------------
Begin SLURM Prolog Tue Aug 14 11:39:12 2018
Job ID:         1362924
Username:       anakano
Accountname:    lc_an1
Name:           sh
Partition:      quick
Nodes:          hpc1118
TasksPerNode:   2
CPUSPerTask:    Default[1]
TMPDIR:         /tmp/1362924.quick
SCRATCHDIR:     /staging/scratch/1362924
Cluster:        uschpc
HSDA Account:   false
End SLURM Prolog
----------------------------------------

[anakano@hpc1118 cs653]$ srun --mpi=pmi2 -n 2 ./mpi_simple
n = 777
```

Symbolic Link to Work Directory




```
[anakano@hpc-login3 ~]$ ln -s /home/rcf-proj/an1/anakano work653
[anakano@hpc-login3 ~]$ ls -l
total 2196
drwx------ 14 anakano m-csci    4096 Aug 30 14:37 course/
drwx------  2 anakano m-csci    4096 Aug 30 14:37 mail/
-rw-------  1 anakano m-csci   30684 Aug 30 14:37 mbox
drwx------ 16 anakano m-csci    4096 Aug 30 14:37 src/
...
lrwxrwxrwx  1 anakano m-csci      27 Aug 30 14:38 work653 ->
/home/rcf-proj/an1/anakano/
[anakano@hpc-login3 ~]$ cd work653
[anakano@hpc-login3 ~/work653]$ pwd
/auto/rcf-proj/an1/anakano
```

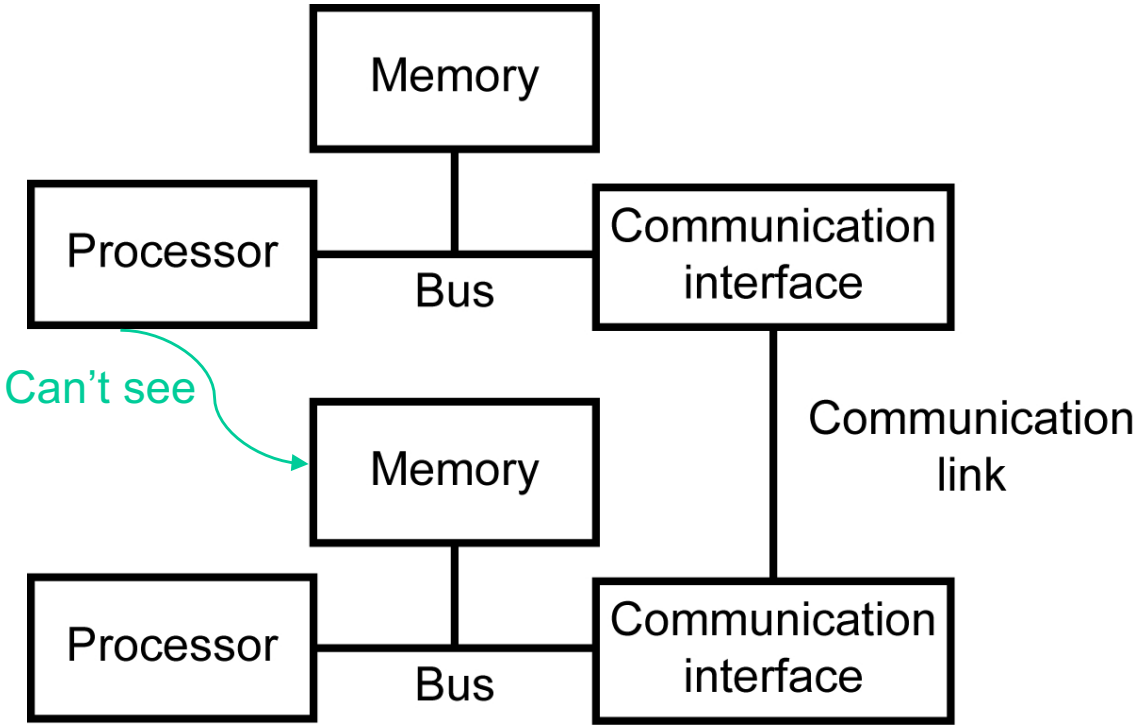
Note: **You are required to work in the class directory**
/home/rcf-proj/an1/*your_ID*

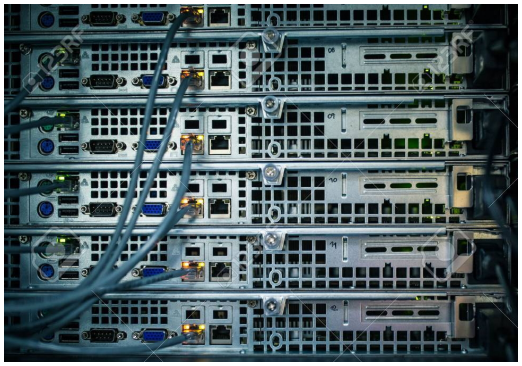
File Transfer

For file transfer to HPC, only use the dedicated file server:
`hpc-transfer.usc.edu`

```
macbook-pro $ sftp anakano@hpc-transfer.usc.edu
sftp> cd /home/rcf-proj/an1/anakano
sftp> put md.*
sftp> ls
md.c    md.h    md.in
sftp> exit
macbook-pro $
```

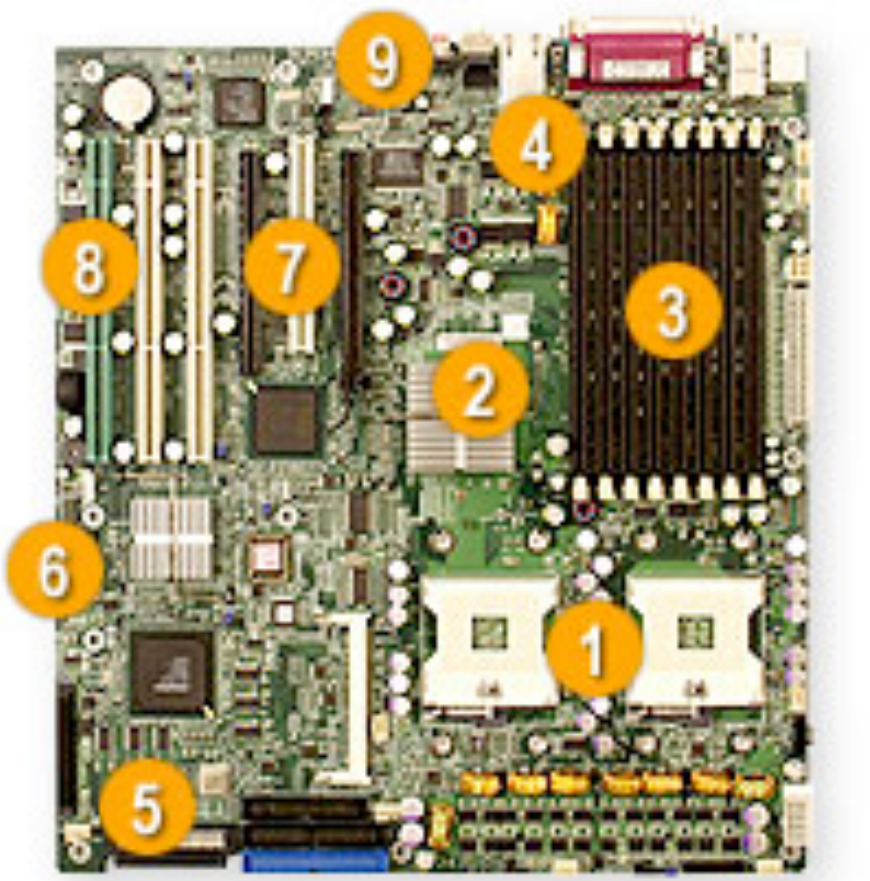
Parallel Computing Hardware





- **Processor**: Executes arithmetic & logic operations.
- **Memory**: Stores program & data.
- **Communication interface**: Performs signal conversion & synchronization between communication link and a computer.
- **Communication link**: A wire capable of carrying a sequence of bits as electrical (or optical) signals.

Motherboard



Key Features

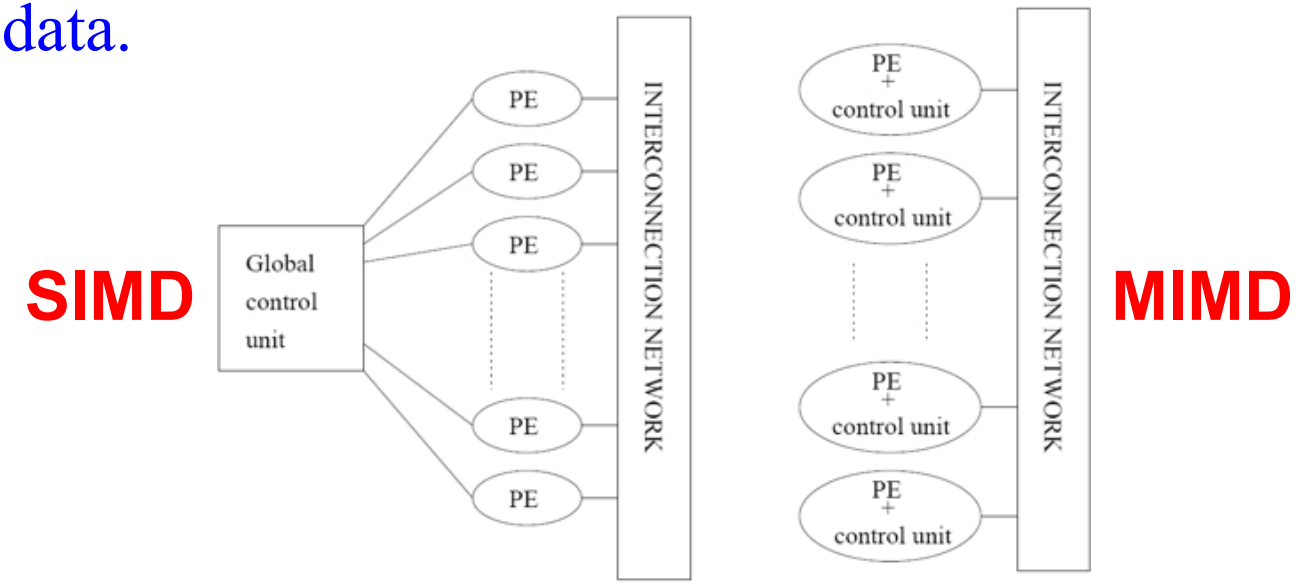
1. Dual Intel® Xeon™ EM64T Support up to 3.60 GHz
2. Intel® E7525 (Tumwater) Chipset
3. Up to 16GB DDRII-400 SDRAM
4. Intel® 82546GB Dual-port Gigabit Ethernet Controller
5. Adaptec AIC-7902 Dual Channel Ultra320 SCSI
6. 2x SATA Ports via ICH5R SATA Controller
7. 1 (x16) & 1 (x4) PCI-Express,
   1 x 64-bit 133MHz PCI-X,
   2 x 64-bit 100MHz PCI-X,
   1 x 32-bit 33MHz PCI Slots
8. Zero Channel RAID Support
9. AC'97 Audio, 6-Channel Sound

Supermicro X6DA8-G2

Parallel Computing Platforms (1)

Control structures

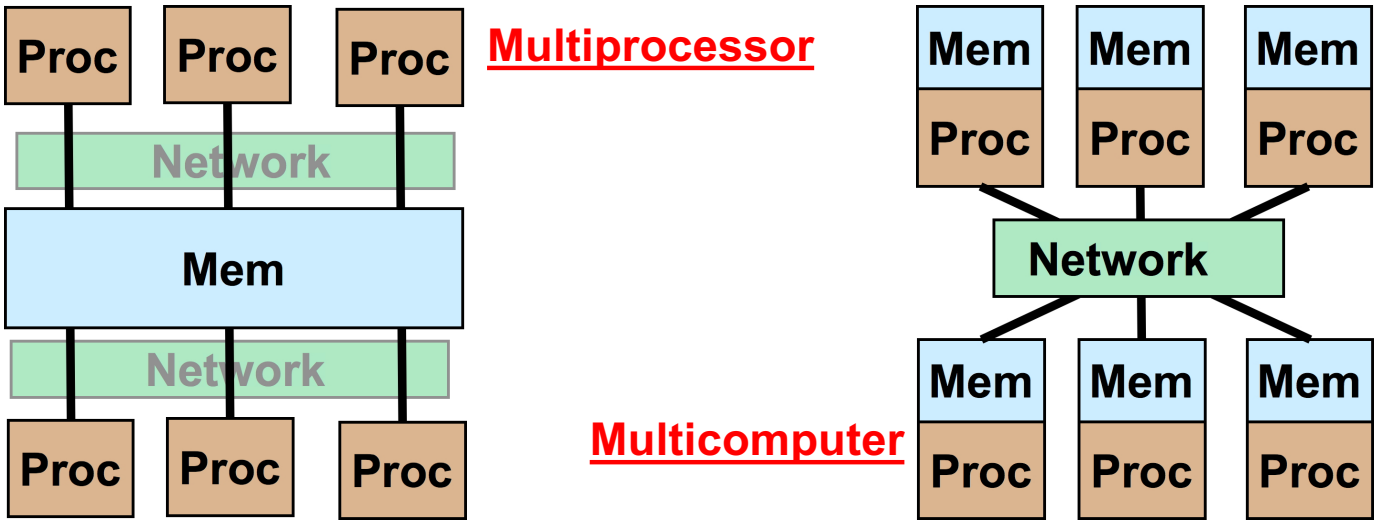
- **Single-instruction multiple-data (SIMD)**: A single control unit dispatches instruction to each processing element (PE).
- **Multiple-instruction multiple-data (MIMD)**: Different processing elements can execute different instructions on different data.
- **Single-program multiple-data (SPMD)**: A simple variant of MIMD; multiple instances of the same program execute on different data.



Parallel Computing Platforms (2)

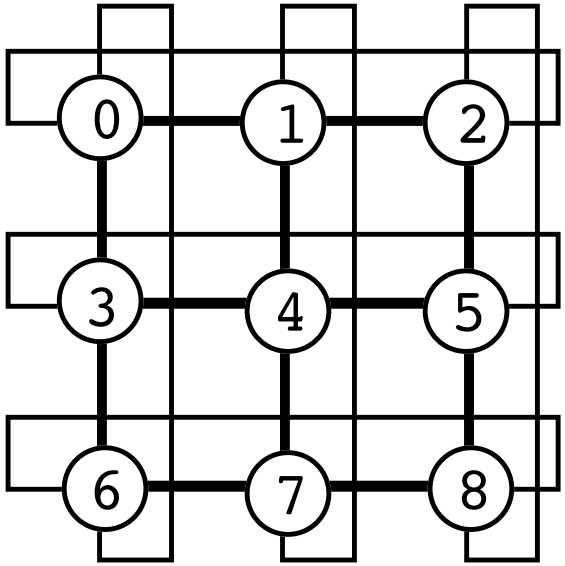
Communication model

- **Shared-address-space platform (multiprocessor)**: Supports a common data space that is accessible to all processors.
  – **Uniform memory access (UMA)**: Time taken by a processor to access any memory word is identical
  – **Nonuniform memory access (NUMA)**: Time taken to access certain memory words is longer than others.
- **Message-passing platform (multicomputer)**: Consists of multiple processing nodes each with its own address space.



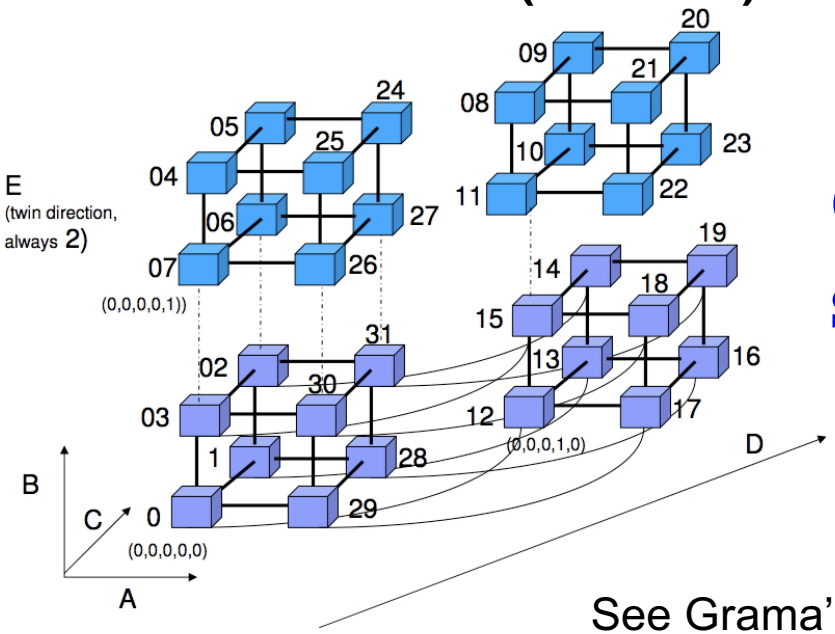
Communication Network

Mesh (torus)

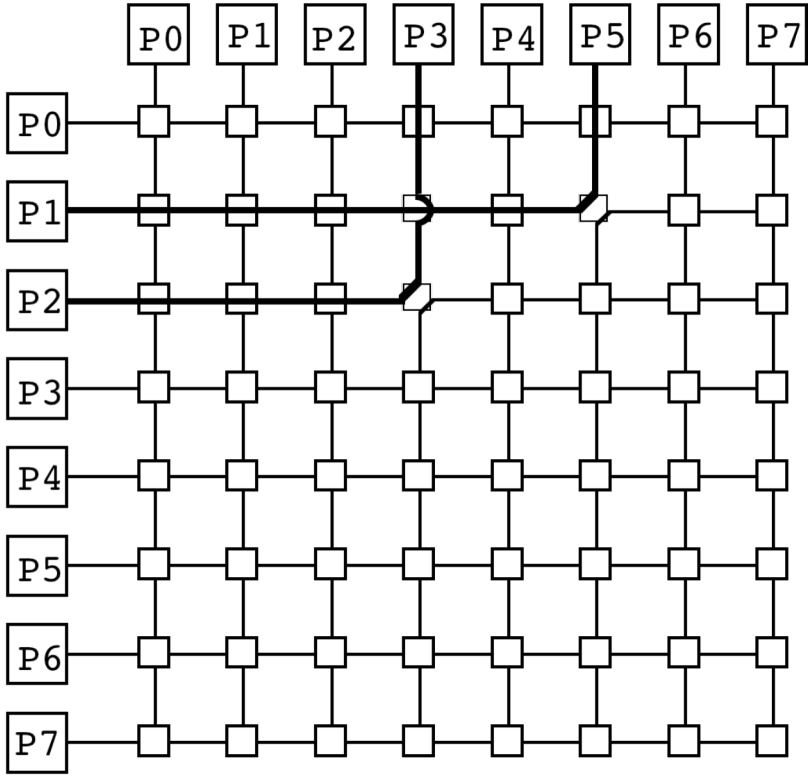


NEC Earth Simulator (640x640 crossbar)

IBM Blue Gene/Q (5D torus)



Crossbar switch



See Grama'03, Chap. 2

Parallel Programming

MPI: Message Passing Interface

- **Standard programming language for multicomputers based on message passing**
- **Review the rest of the slides & detailed notes**

  http://cacs.usc.edu/education/cs653/02MPI.pdf

```
MPI_Send(), MPI_Recv()
```

OpenMP: Open specifications for Multi Processing

- **Portable application program interface (API) for shared-memory parallel programming on multiprocessors based on multithreading by compiler directives**
- **Review the slides**

  http://cacs.usc.edu/education/cs653/02-02OpenMP-slide.pdf

```
#pragma omp parallel
```

Message Passing Interface

MPI (Message Passing Interface)

A standard message passing system that enables us to write & run applications on parallel computers

Download for Unix & Windows:

`http://www.mcs.anl.gov/mpi/mpich`

Compile

```
> mpicc -o mpi_simple mpi_simple.c
```

Run (srun is Slurm dialect)

```
> mpirun -np 2 mpi_simple
```

MPI Programming

`mpi_simple.c`: Point-to-point message send & receive

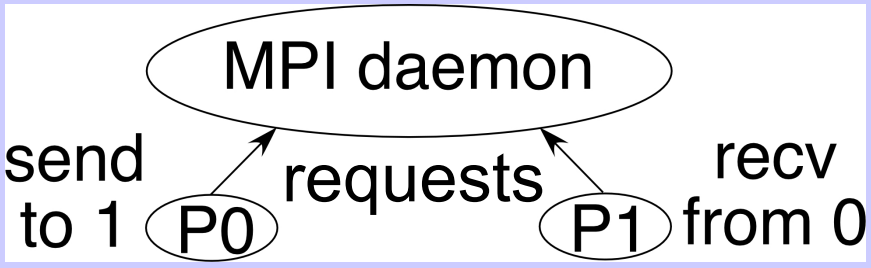
```
#include "mpi.h"
#include <stdio.h>
int main(int argc, char *argv[]) {
  MPI_Status status;
  int myid;
  int n;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &myid);
  if (myid == 0) {
    n = 777;
    MPI_Send(&n, 1, MPI_INT, 1, 10, MPI_COMM_WORLD);
  }
  else {
    MPI_Recv(&n, 1, MPI_INT, 0, 10, MPI_COMM_WORLD, &status);
    printf("n = %d\n", n);
  }
  MPI_Finalize();
  return 0;
}
```

MPI rank

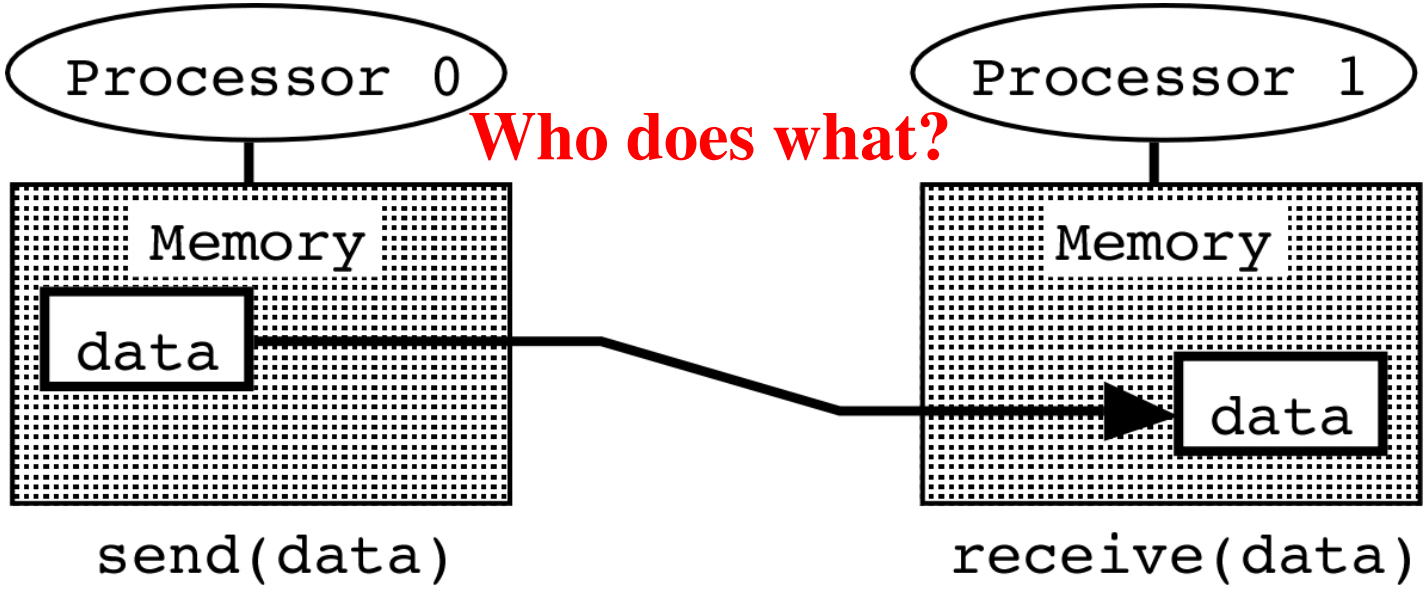
Matching message labels

Data triplet

To/from whom



Single Program Multiple Data (SPMD)



Process 0

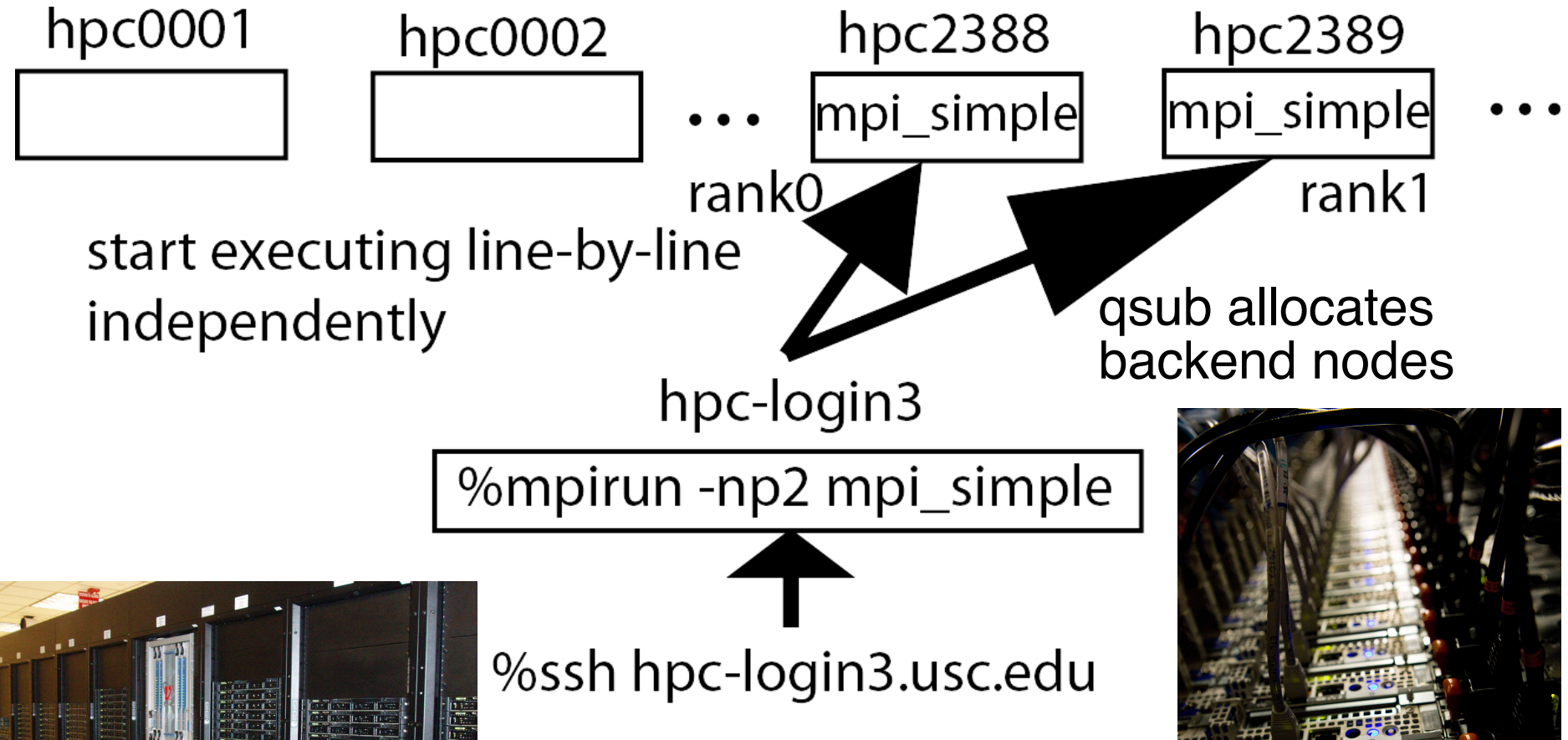
```
if (myid == 0) {
  n = 777;
  MPI_Send(&n,...);
}
else {
  MPI_Recv(&n,...);
  printf(...);
}
```

Process 1

```
if (myid == 0) {
  n = 777;
  MPI_Send(&n,...);
}
else {
  MPI_Recv(&n,...);
  printf(...);
}
```

Single Program Multiple Data (SPMD)

What really happens?



MPI Minimal Essentials

We only need `MPI_Send()` & `MPI_Recv()` within `MPI_COMM_WORLD`

```
MPI_Send(&n, 1, MPI_INT, 1, 10, MPI_COMM_WORLD);
MPI_Recv(&n, 1, MPI_INT, 0, 10, MPI_COMM_WORLD, &status);
```

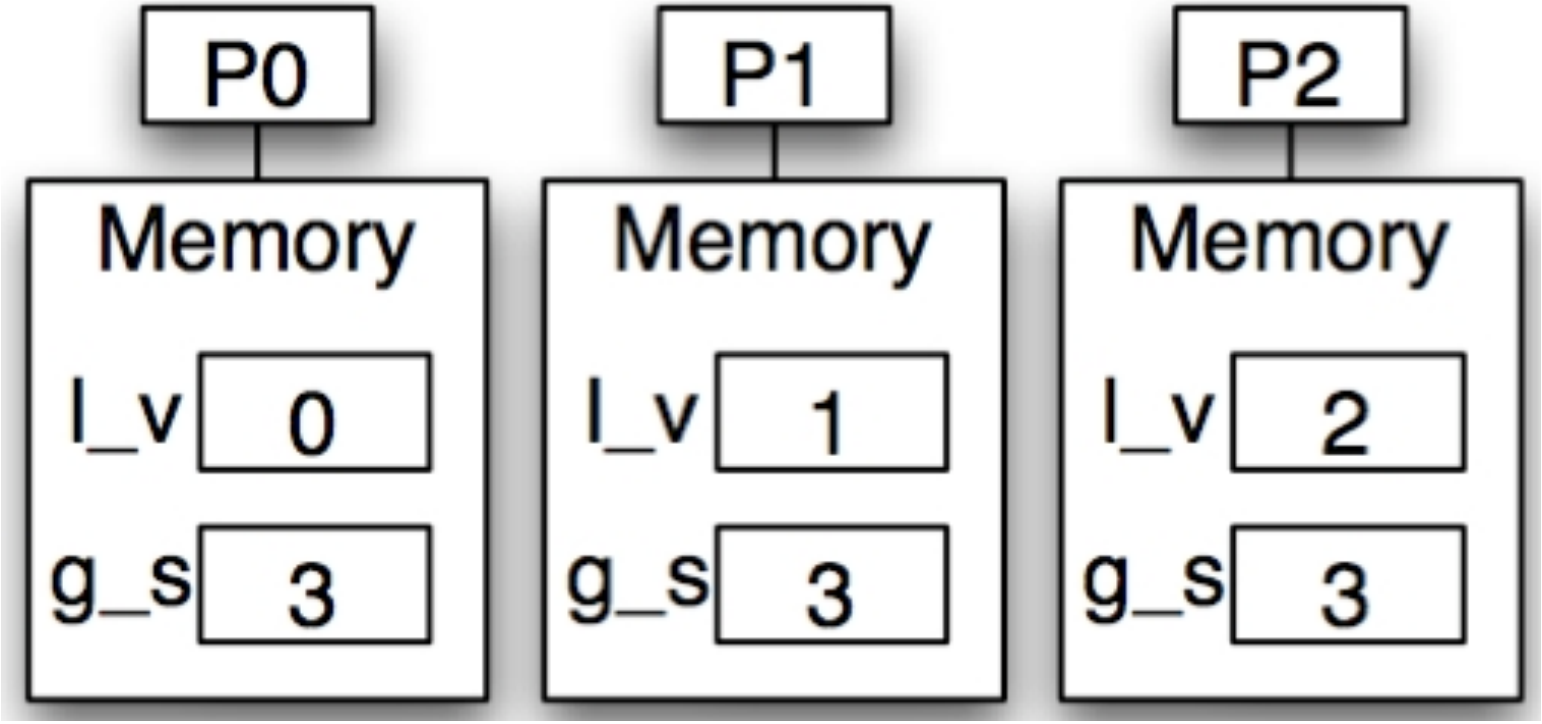
Data triplet — To/from whom — Information

Global Operation

All-to-all reduction: Each process contributes a partial value to obtain the global summation. In the end, all the processes will receive the calculated global sum.

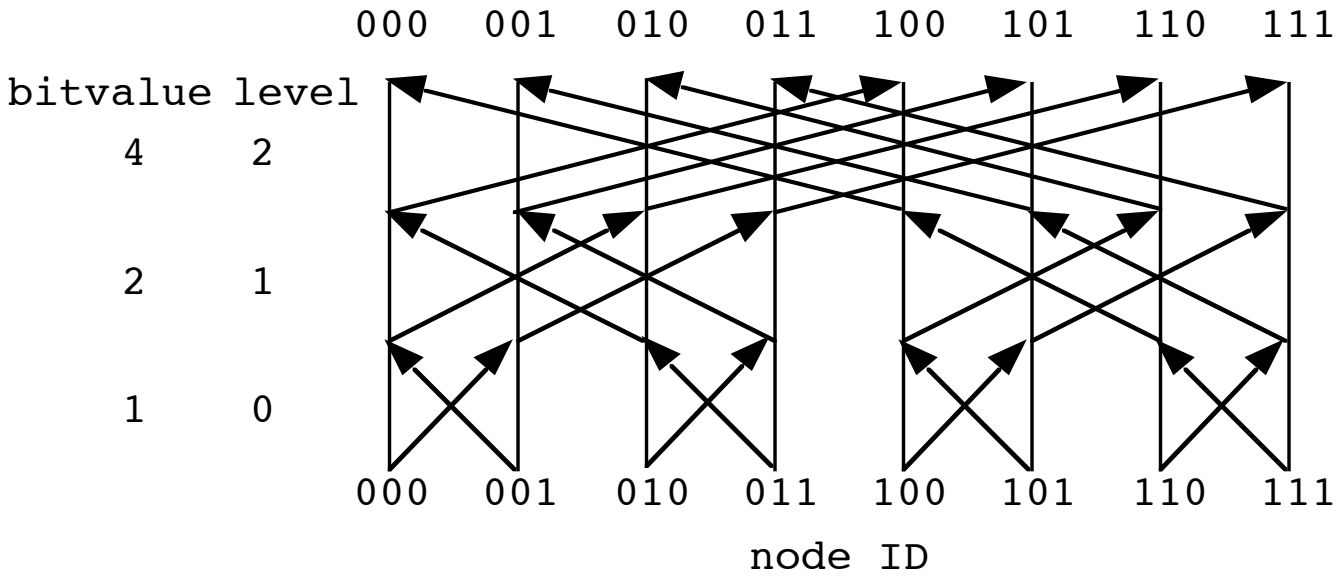
```
MPI_Allreduce(&local_value, &global_sum, 1, MPI_INT, MPI_SUM,
MPI_COMM_WORLD)
```

```
int l_v, g_s;  // local variable & global sum
l_v = myid;    // myid is my MPI rank
MPI_Allreduce(&l_v, &g_s, 1, MPI_INT, MPI_SUM, MPI_COMM_WORLD);
```



Hypercube Algorithm

Hypercube algorithm: Communication of a reduction operation is structured as a series of pairwise exchanges, one with each neighbor in a hypercube (**butterfly**) structure. Allows a computation requiring all-to-all communication among p processes to be performed in $\log_2 p$ steps.

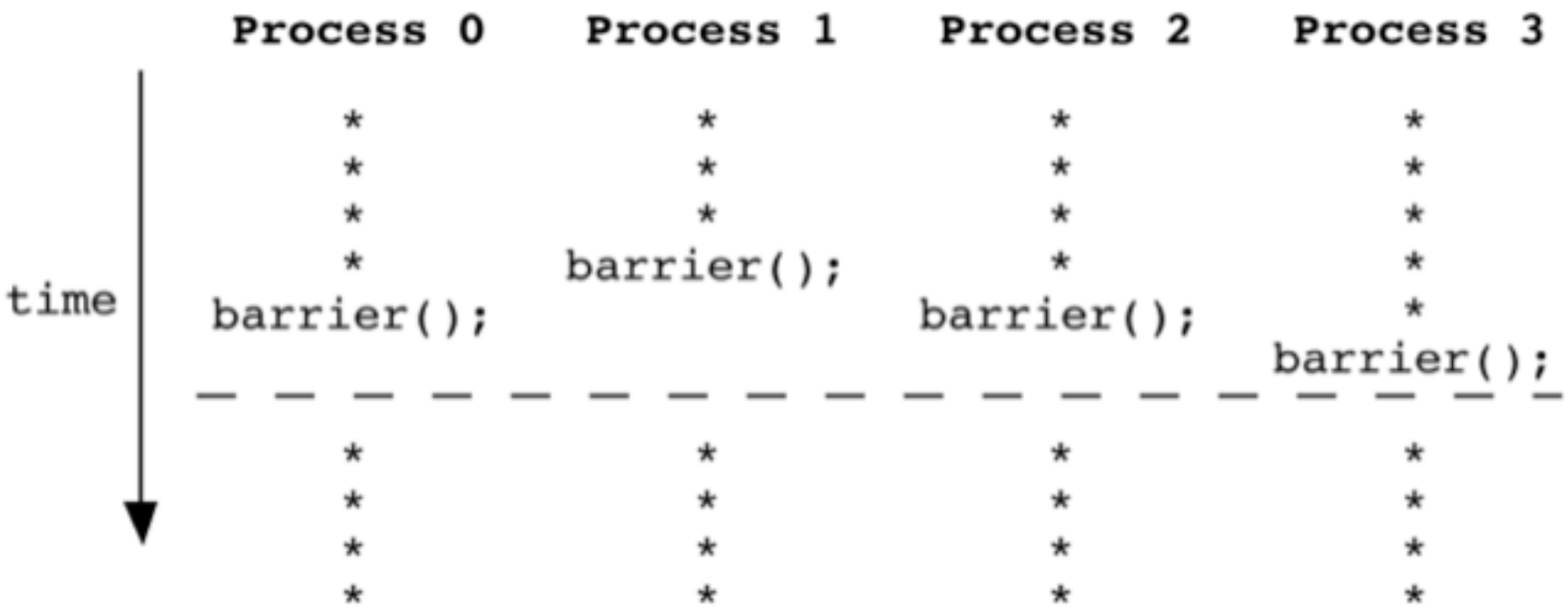


Butterfly network

```
a000 + a001 + a010 + a011 + a100 + a101 + a110 + a111
=  ((a000 + a001) + (a010 + a011))
+  ((a100 + a101) + (a110 + a111))
(2)                (1)      (0)
```

Barrier

```
<A>;
barrier();
<B>;
```



MPI_Barrier(MPI_Comm communicator)

MPI Communication

MPI communication functions:

1. **Point-to-point**

   `MPI_Send()`
   `MPI_Recv()`

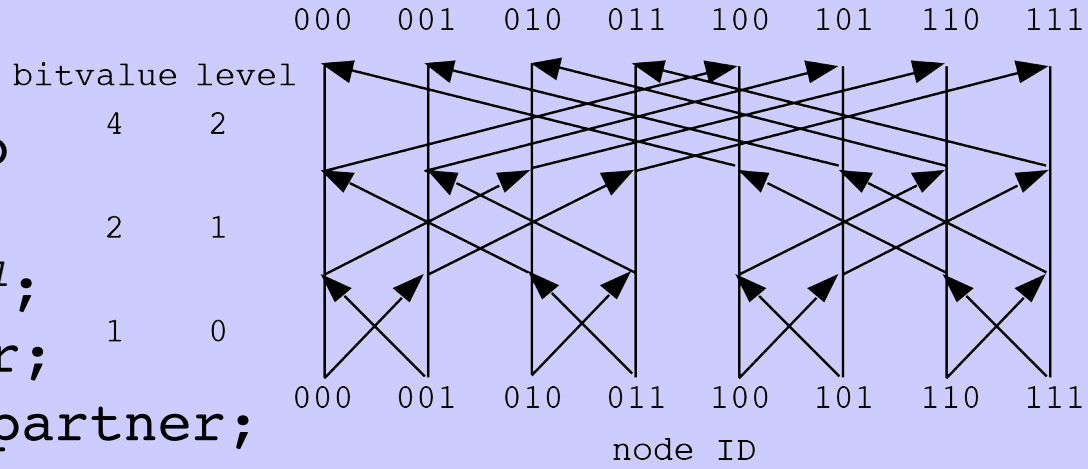
2. **Global**

   `MPI_Allreduce()`
   `MPI_Barrier()`
   `MPI_Bcast()`

See Grama'03, Chap. 6

Hypercube Template

```
procedure hypercube(myid, input, log2P, output)
begin
  mydone := input;
  for l := 0 to log2P-1 do
  begin
    partner := myid XOR 2^l;
    send mydone to partner;
    receive hisdone from partner;
    mydone = mydone OP hisdone
  end
  output := mydone
end
```



level l	2^l bitvalue
0	001
1	010
2	100

Associative operator (*e.g.*, sum, max)

Exclusive OR

a	b	a XOR b	
0	0	0	b
0	1	1	
1	0	1	$\bar{b}$
1	1	0	

abcdefg XOR 0000100 = abcd$\bar{e}$fg

In C, ^ (caret operator) is bitwise XOR applied to int

Driver for Hypercube Test

```
#include "mpi.h"
#include <stdio.h>
int nprocs;  /* Number of processors */
int myid;    /* My rank */

double global_sum(double partial) {
  /* Implement your own global summation here */
}

int main(int argc, char *argv[]) {
  double partial, sum, avg;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &myid);
  MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
  partial = (double) myid;
  printf("Rank %d has %le\n", myid, partial);
  sum = global_sum(partial);
  if (myid == 0) {
    avg = sum/nprocs;
    printf("Global average = %d\n", avg);
  }
  MPI_Finalize();
  return 0;
}
```

Sample Slurm Script

```
#!/bin/bash
#SBATCH --ntasks-per-node=4
#SBATCH --nodes=2
#SBATCH --time=00:00:59
#SBATCH --output=global.out
#SBATCH -A lc_an1

WORK_HOME=/home/rcf-proj/an1/anakano
cd $WORK_HOME

srun -n $SLURM_NTASKS --mpi=pmi2 ./global
srun -n               4 --mpi=pmi2 ./global
```

Total number of processors
= ntasks-per-node (4) × nodes (2) = 8

Output of `global.c`

- **4-processor job**

```
Rank 0 has 0.000000e+00
Rank 1 has 1.000000e+00
Rank 2 has 2.000000e+00
Rank 3 has 3.000000e+00
Global average = 1.500000e+00
```

- **8-processor job**

```
Rank 0 has 0.000000e+00
Rank 1 has 1.000000e+00
Rank 2 has 2.000000e+00
Rank 3 has 3.000000e+00
Rank 5 has 5.000000e+00
Rank 6 has 6.000000e+00
Rank 4 has 4.000000e+00
Rank 7 has 7.000000e+00
Global average = 3.500000e+00
```

Distributed-Memory Parallel Computing

hpc1001

MPI daemon

hpc1002

MPI daemon

Who does what?
No synchronization!

MPI daemon

MPI daemon

hpc1003

MPI daemon

hpc1004

Communicator

mpi_comm.c: Communicator = process group + context

```
#include "mpi.h"
#include <stdio.h>
#define N 64
int main(int argc, char *argv[]) {
  MPI_Comm world, workers;
  MPI_Group world_group, worker_group;
  int myid, nprocs;
  int server, n = -1, ranks[1];
  MPI_Init(&argc, &argv);
  world = MPI_COMM_WORLD;
  MPI_Comm_rank(world, &myid);
  MPI_Comm_size(world, &nprocs);
  server = nprocs-1;
  MPI_Comm_group(world, &world_group);
  ranks[0] = server;
  MPI_Group_excl(world_group, 1, ranks, &worker_group);
  MPI_Comm_create(world, worker_group, &workers);
  MPI_Group_free(&worker_group);
  if (myid != server)
    MPI_Allreduce(&myid, &n, 1, MPI_INT, MPI_SUM, workers);
  printf("process %2d: n = %6d\n", myid, n);
  MPI_Comm_free(&workers);
  MPI_Finalize();
  return 0;
}
```

Example: Ranks in Different Groups

World Rank	Institution*	Country /Region	National Rank	Total Score	Score on Alumni
1	Harvard University		1	100	100
2	Stanford University		2	72.1	41.8
3	Massachusetts Institute of Technology (MIT)		3	70.5	68.4
4	University of California-Berkeley		4	70.1	66.8
5	University of Cambridge		1	69.2	79.1
51	University of Southern California		33	31	31.7

```
MPI_Comm_rank(world, &usc_world);
MPI_Comm_rank(us, &usc_national);
```

Rank is relative in each communicator!

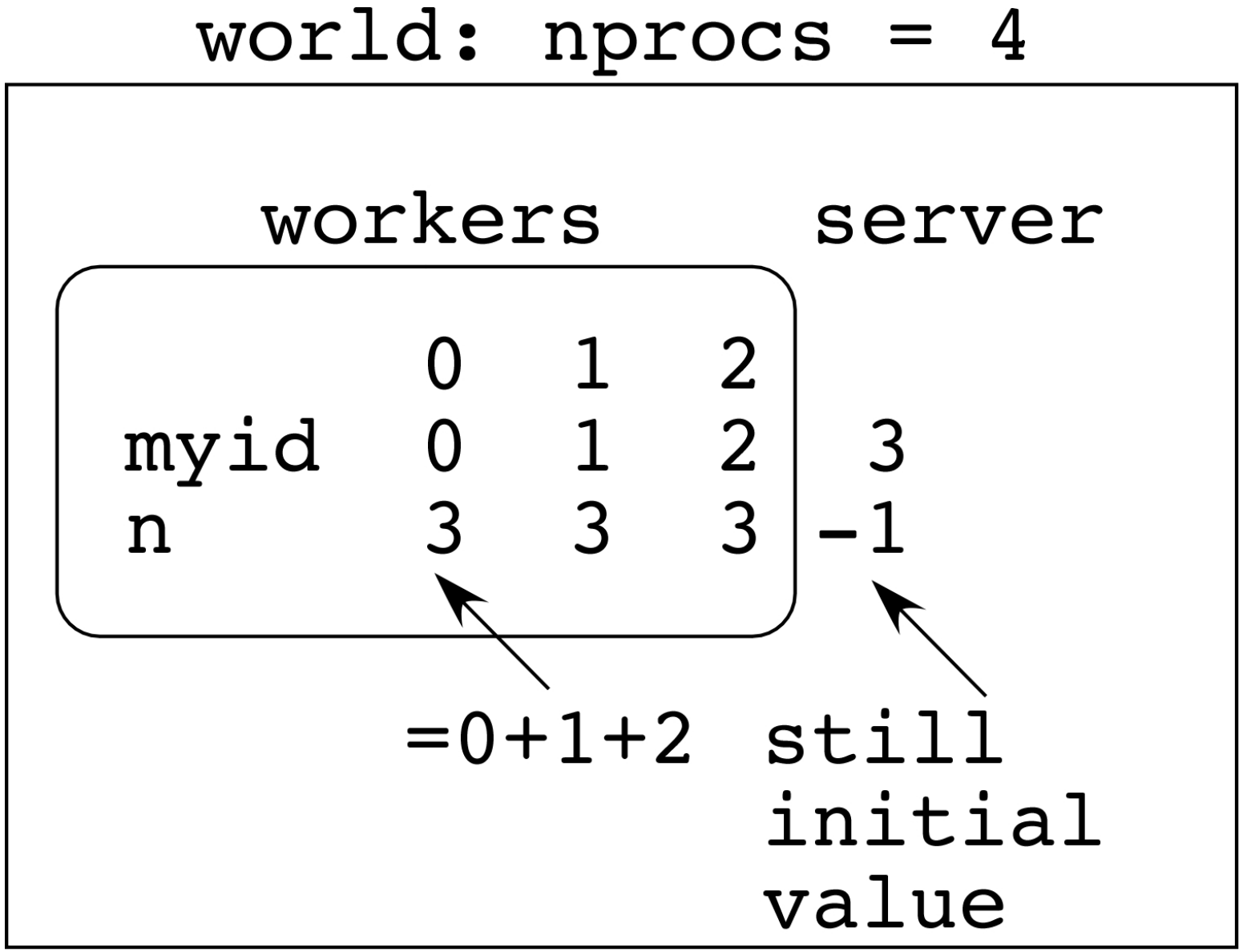
Output from mpi_comm.c

Slurm script

```
#SBATCH --ntasks-per-node=2
#SBATCH --nodes=2
...
srun -n $SLURM_NTASKS --mpi=pmi2 ./mpi_comm
```

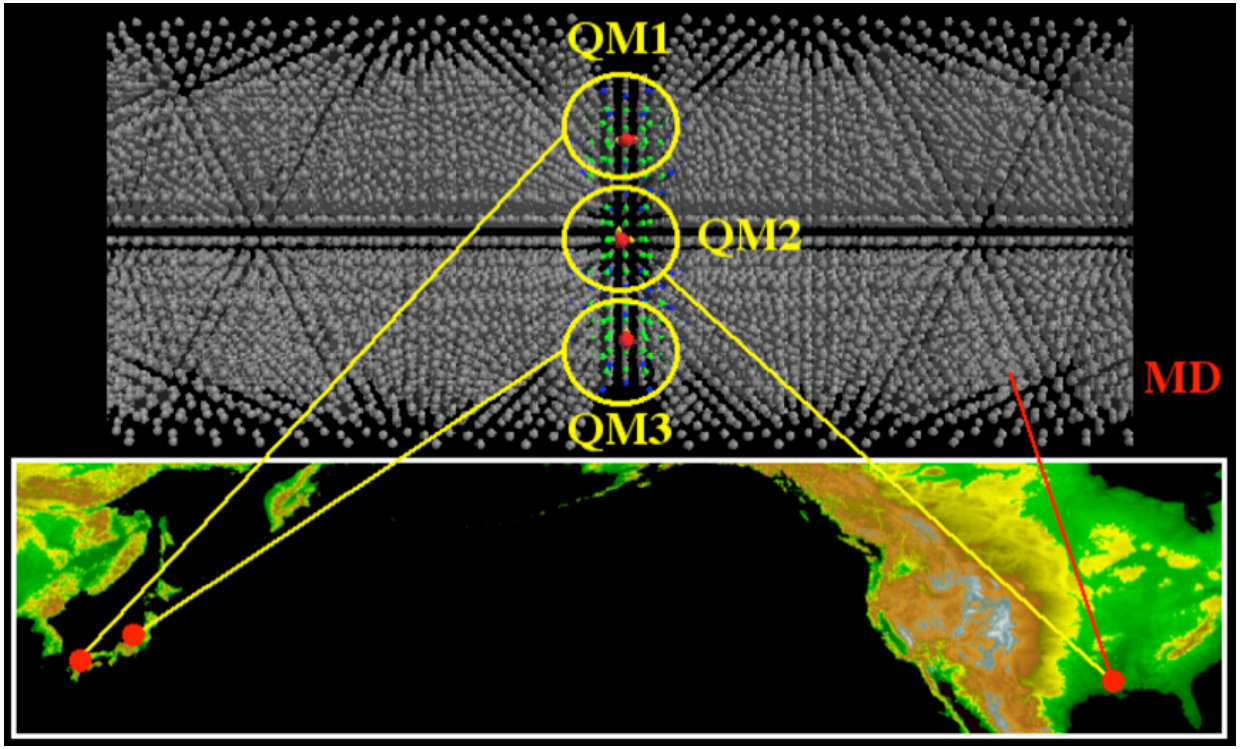
```
----------------------------------------
Begin SLURM Prolog Wed 29 Aug 2018 09:11:42 AM PDT
Job ID:         1429109
Username:       anakano
Accountname:    lc_an1
Name:           mpi_comm.sl
Partition:      quick
Nodes:          hpc[1120-1121]
TasksPerNode:   2(x2)
CPUSPerTask:    Default[1]
TMPDIR:         /tmp/1429109.quick
SCRATCHDIR:     /staging/scratch/1429109
Cluster:        uschpc
HSDA Account:   false
End SLURM Prolog
----------------------------------------
process  3: n =     -1
process  0: n =      3
process  1: n =      3
process  2: n =      3
```

What Has Happened?



Grid Computing & Communicators

H. Kikuchi *et al*., "Collaborative simulation Grid: multiscale quantum-mechanical/classical atomistic simulations on distributed PC clusters in the US & Japan, *IEEE/ACM SC02*

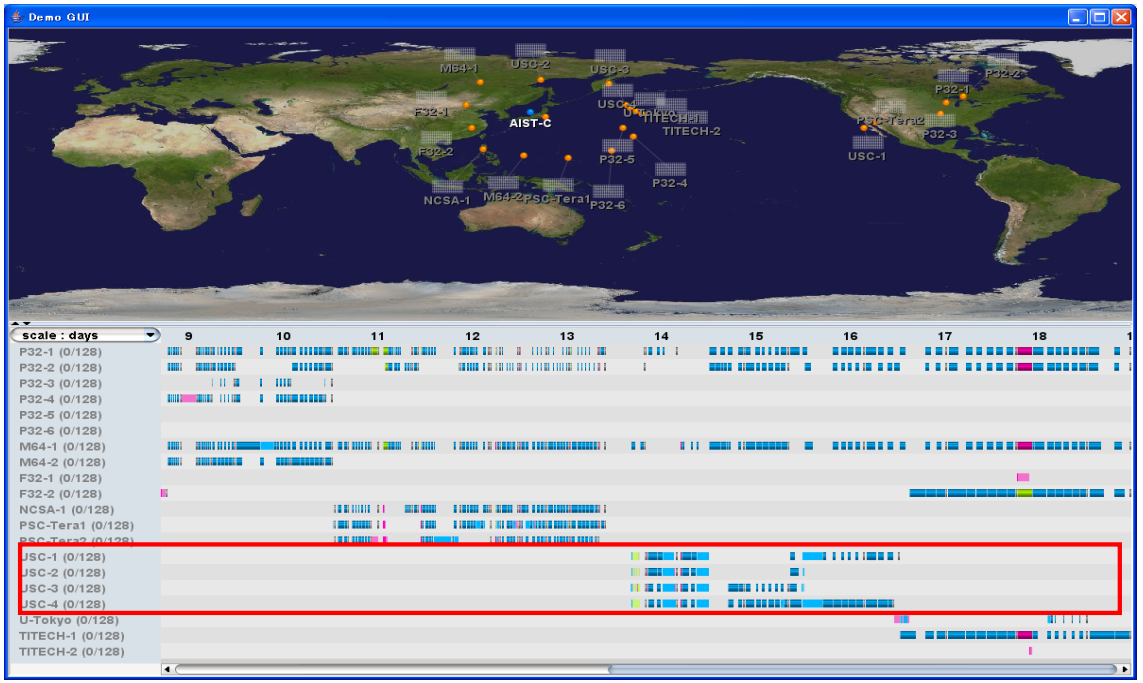


Communicator = a nice migration path to distributed computing

- **Single MPI program run with the Grid-enabled MPI implementation, MPICH-G2**
- **Processes are grouped into MD & QM groups by defining multiple MPI communicators as subsets of MPI_COMM_WORLD; a machine file assigns globally distributed processors to the MPI processes**

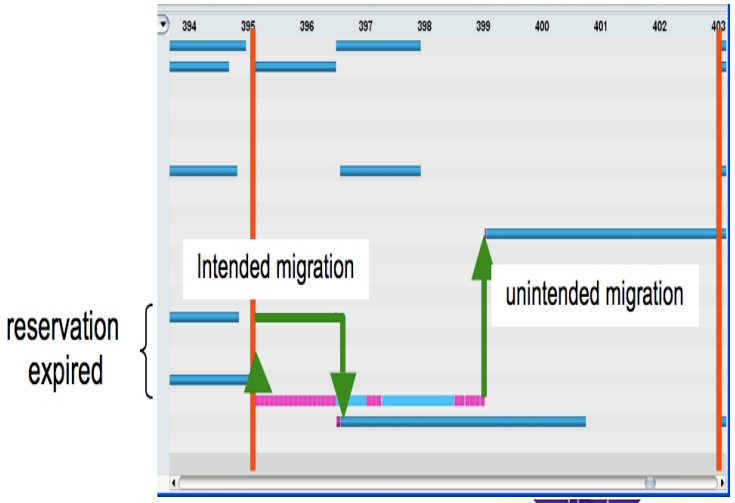
Global Grid QM/MD

- *One of the largest (153,600 cpu-hrs) sustained Grid supercomputing* **at 6 sites in the US (USC, Pittsburgh, Illinois) & Japan (AIST, U Tokyo, Tokyo IT)**



USC

Automated resource migration & fault recovery

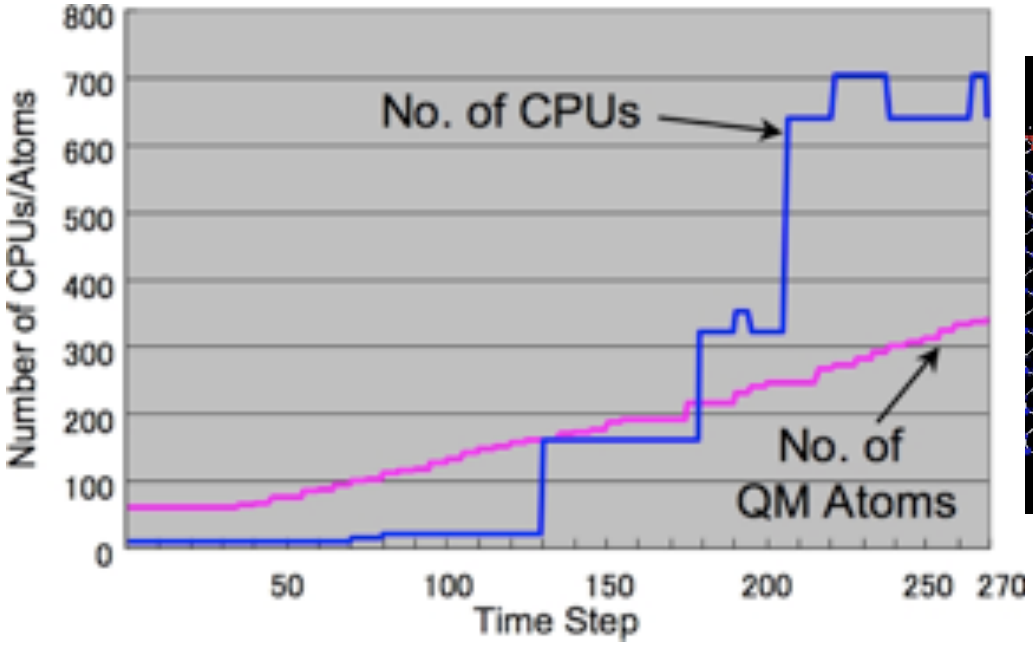


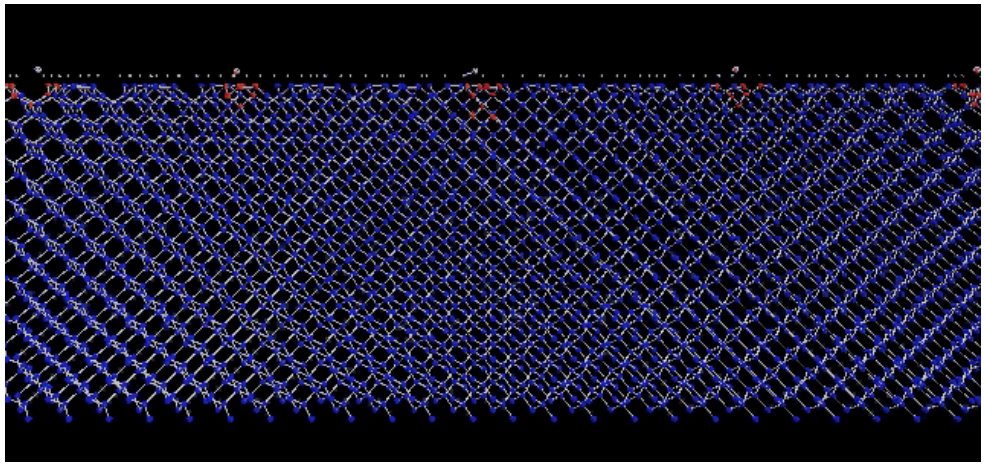
Takemiya *et al.*, "Sustainable adaptive Grid supercomputing: multiscale simulation of semiconductor processing across the Pacific," *IEEE/ACM SC06*

Sustainable Grid Supercomputing

- Sustained (> months) supercomputing (> 10^3 CPUs) on a Grid of geographically distributed supercomputers
- Hybrid Grid remote procedure call (GridRPC) + message passing (MPI) programming
- Dynamic allocation of computing resources on demand & automated migration due to reservation schedule & faults

Ninf-G GridRPC: ninf.apgrid.org; MPICH: www.mcs.anl.gov/mpi

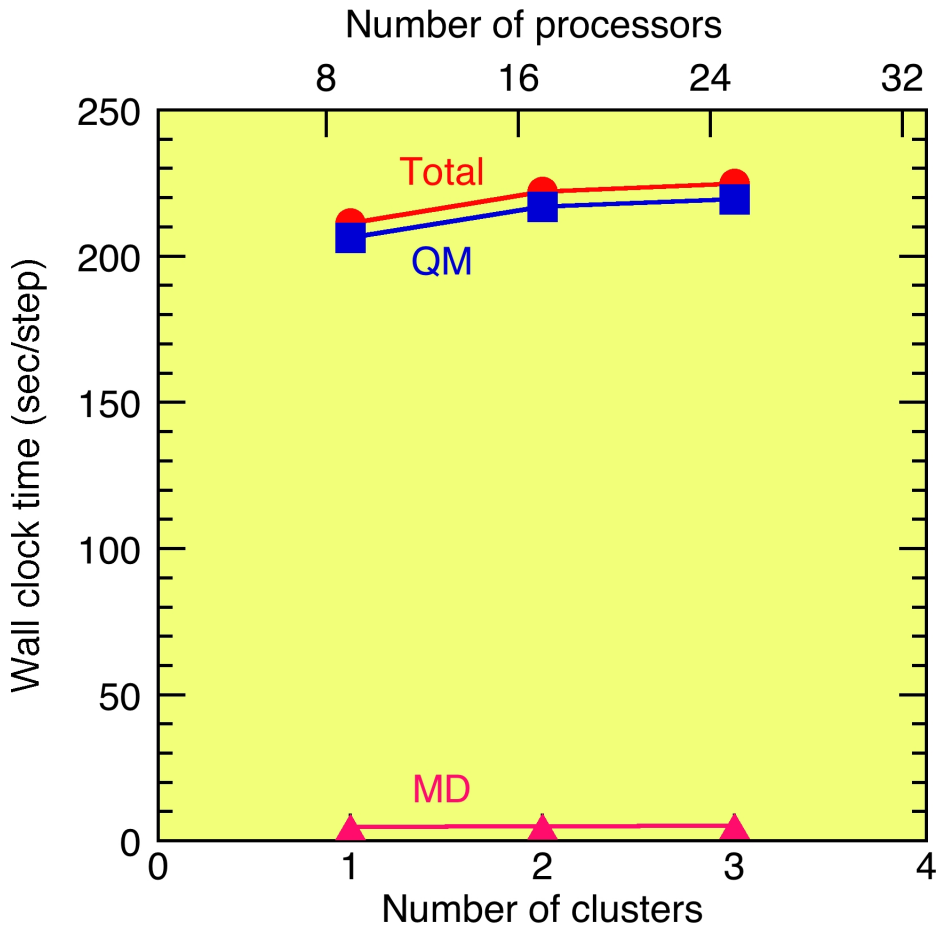




Multiscale QM/MD simulation of high-energy beam oxidation of Si

Computation-Communication Overlap

H. Kikuchi *et al*., "Collaborative simulation Grid: multiscale quantum-mechanical/classical atomistic simulations on distributed PC clusters in the US & Japan, *IEEE/ACM SC02*



Parallel efficiency = 0.94

- **How to overcome 200 ms latency & 1 Mbps bandwidth?**
- **Computation-communication overlap: To hide the latency, the communications between the MD & QM processors have been overlapped with the computations using asynchronous messages**

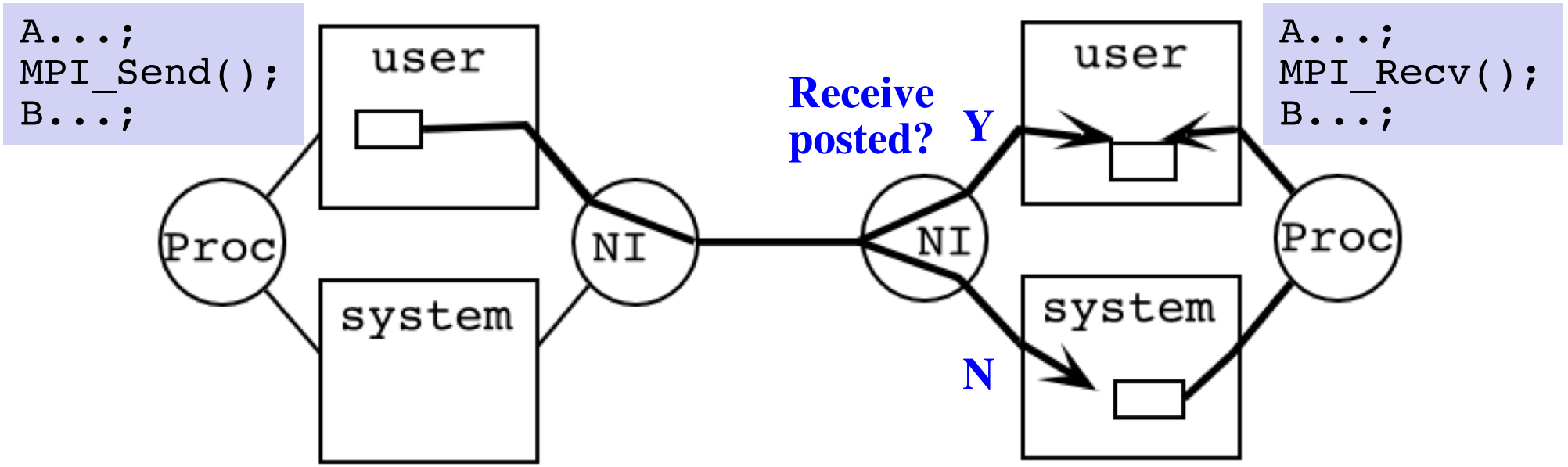
Synchronous Message Passing

`MPI_Send()`: (blocking), synchronous

- Safe to modify original data immediately on return
- Depending on implementation, it may return whether or not a matching receive has been posted, or it may block (especially if no buffer space available)

`MPI_Recv()`: blocking, synchronous

- Blocks for message to arrive
- Safe to use data on return



Asynchronous Message Passing

Allows computation-communication overlap

`MPI_Isend()`: non-blocking, asynchronous

- Returns whether or not a matching receive has been posted
- Not safe to modify original data immediately (use `MPI_Wait()` system call)

`MPI_Irecv()`: non-blocking, asynchronous

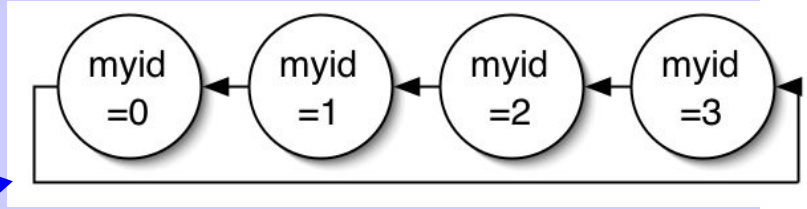
- Does not block for message to arrive
- Cannot use data before checking for completion with `MPI_Wait()`

```
A...;
MPI_Isend();
B...;
MPI_Wait();
C...; // Reuse the send buffer
```

```
A...;
MPI_Irecv();
B...;
MPI_Wait();
C...; // Use the received message
```

Program irecv_mpi.c

```
#include "mpi.h"
#include <stdio.h>
#define N 1000
int main(int argc, char *argv[]) {
  MPI_Status status;
  MPI_Request request;
  int send_buf[N], recv_buf[N];
  int send_sum = 0, recv_sum = 0;
  long myid, left, Nnode, msg_id, i;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &myid);
  MPI_Comm_size(MPI_COMM_WORLD, &Nnode);
  left  = (myid + Nnode - 1) % Nnode;
  for (i=0; i<N; i++) send_buf[i] = myid*N + i;
  MPI_Irecv(recv_buf, N, MPI_INT, MPI_ANY_SOURCE, 777, MPI_COMM_WORLD,
            &request); /* Post a receive */
  /* Perform tasks that don't use recv_buf */
  MPI_Send(send_buf, N, MPI_INT, left, 777, MPI_COMM_WORLD);
  for (i=0; i<N; i++) send_sum += send_buf[i];
  MPI_Wait(&request, &status); /* Complete the receive */
  /* Now it's safe to use recv_buf */
  for (i=0; i<N; i++) recv_sum += recv_buf[i];
  printf("Node %d: Send %d Recv %d\n", myid, send_sum, recv_sum);
  MPI_Finalize();
  return 0;
}
```



Output from `irecv_mpi.c`

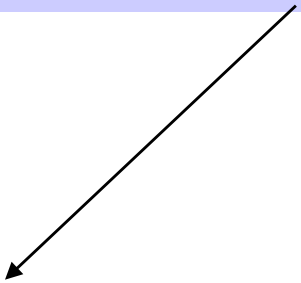
```
----------------------------------------
Begin SLURM Prolog Wed 29 Aug 2018 09:18:06 AM PDT
Job ID:         1429116
Username:       anakano
Accountname:    lc_an1
Name:           irecv_mpi.sl
Partition:      quick
Nodes:          hpc[1120-1121]
TasksPerNode:   2(x2)
CPUSPerTask:    Default[1]
TMPDIR:         /tmp/1429116.quick
SCRATCHDIR:     /staging/scratch/1429116
Cluster:        uschpc
HSDA Account:   false
End SLURM Prolog
----------------------------------------
Node 1: Send 1499500 Recv 2499500
Node 3: Send 3499500 Recv 499500
Node 0: Send 499500 Recv 1499500
Node 2: Send 2499500 Recv 3499500
```

Multiple Asynchronous Messages

```
MPI_Request requests[N_message];
MPI_Status status;
int index;

/* Wait for all messages to complete */
MPI_Waitall(N_message, requests, &status);

/* Wait for any specified messages to complete */
MPI_Waitany(N_message, requests, &index, &status);
```



returns the index (∈ [0,N_message-1]) of the message that completed

Polling MPI_Irecv

```
int flag;

/* Post an asynchronous receive */
MPI_Irecv(recv_buf, N, MPI_INT, MPI_ANY_SOURCE, 777,
          MPI_COMM_WORLD, &request);

/* Perform tasks that don't use recv_buf */
...

/* Polling */
MPI_Test(&request, &flag, &status); /* Check completion */
if (flag) { /* True if message received */
  /* Now it's safe to use recv_buf */
  ...
}
```