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
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

First Node	Last Node	# of Nodes	Node Type	/tmp	Queue	Nodesets			
hpc0249	hpc0496	256	Dual Hexcore Intel Xeon 2.66 GHz 24GB Memory	895GB	priya	sl160	Xeon	x86_64	hexcore
hpc1044	hpc1050	7	Dual Dodecacore AMD Opteron 2.3 GHz 48GB Memory	1TB	main large quick	dl165	Opteron	x86_64	dodecacore
hpc3030	hpc3264	236	Dual Octocore Intel Xeon 2.4 GHz 64GB Memory Dual k20 NVIDIA	1TB	main large quick	sl250s	Xeon	x86_64	octocore

How to Use USC HPC Cluster

Log in

```
> ssh anakano@hpc-login2.usc.edu
```

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

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

Add in .cshrc to use the MPI library

```
source /usr/usc/openmpi/1.6.4/setup.csh
```

Compile an MPI program

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

Execute an MPI program

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

```
[anakano@hpc-login2 ~]$ which mpicc
/usr/usc/openmpi/1.6.4/bin/mpicc
```

Submit a PBS Batch Job

Prepare a script file, mpi_simple.pbs

```
#!/bin/bash
#PBS -l nodes=1:ppn=2,arch=x86_64
#PBS -l walltime=00:00:59
#PBS -o mpi_simple.out
#PBS -j oe
#PBS -N mpi_simple
#PBS -A lc_an2
WORK_HOME=/home/rcf-proj2/an2/yourID
cd $WORK_HOME
np=$(cat $PBS_NODEFILE | wc -l)
mpirun -np $np -machinefile $PBS_NODEFILE ./mpi_simple
```

Submit a PBS job

```
hpc-login2: qsub mpi_simple.pbs
```

Check the status of a PBS job

```
hpc-login2: qstat -u anakano
                                                        Req'd  Req'd Elap
Job ID                  Username Queue    Jobname    SessID NDS   TSK Memory Time  S Time
-------------------- -------- -------- ---------- ------ ----- --- ------ ----- - -----
9475408.hpc-pbs.        anakano  quick    mpi_simple  --    1     2    --    00:00 Q  ---
```

Kill a PBS job

```
hpc-login2: qdel 9475408
```

Sample PBS Output File

```
hpc-login2: more mpi_simple.out
----------------------------------------
Begin PBS Prologue Thu Aug 28 13:09:39 PDT 2014
Job ID:            9475408.hpc-pbs.hpcc.usc.edu
Username:          anakano
Group:             m-csci
Name:              mpi_simple
...
Nodes:             hpc2333
TMPDIR:            /tmp/9475408.hpc-pbs.hpcc.usc.edu
End PBS Prologue Thu Aug 28 13:09:39 PDT 2014
----------------------------------------
n = 777
----------------------------------------
Begin PBS Epilogue Thu Aug 28 13:09:40 PDT 2014
...
Limits:            nednodes=1:ppn=2,nodes=1:ppn=2,walltime=00:00:59
Resources:         cput=00:00:00,mem=0kb,vmem=0kb,walltime=00:00:02
Queue:             quick
Shared Access:     no
Account:           lc_an2
End PBS Epilogue Thu Aug 28 13:09:40 PDT 2014
----------------------------------------------------
```

hpc2283	hpc2337	55	Dual Quadcore Intel Xeon 2.5 GHz 12GB Memory	60GB	main large quick	pe1950	Xeon	x86_64	quadcore

Interactive Job at HPC

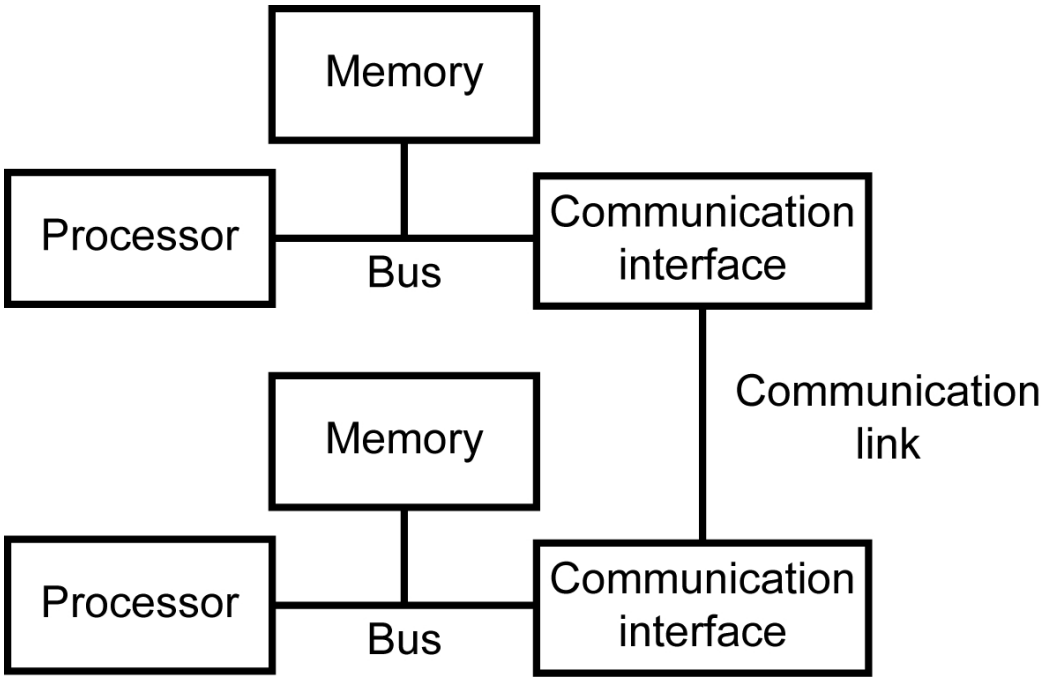
```
$ qsub -I -l nodes=2:ppn=4,arch=x86_64 -l walltime=00:04:59
qsub: waiting for job 9494228.hpc-pbs.hpcc.usc.edu to start
qsub: job 9494228.hpc-pbs.hpcc.usc.edu ready

----------------------------------------
Begin PBS Prologue Sun Aug 31 11:21:01 PDT 2014
Job ID:            9494228.hpc-pbs.hpcc.usc.edu
Username:          anakano
Group:             m-csci
Name:              STDIN
Queue:             quick
Shared Access:     no
All Cores:         no
Nodes:             hpc1975 hpc1976
PVFS:              /scratch (124G), /staging (328T)
TMPDIR:            /tmp/9494228.hpc-pbs.hpcc.usc.edu
End PBS Prologue Sun Aug 31 11:21:17 PDT 2014
----------------------------------------

[hpc1975]$ echo $PBS_NODEFILE
/var/spool/torque/aux//9494228.hpc-pbs.hpcc.usc.edu

[hpc1975]$ more $PBS_NODEFILE
hpc1975
hpc1975
hpc1975
hpc1975
hpc1976
hpc1976
hpc1976
hpc1976
```

Symbolic Link to Work Directory

symbolic link source alias

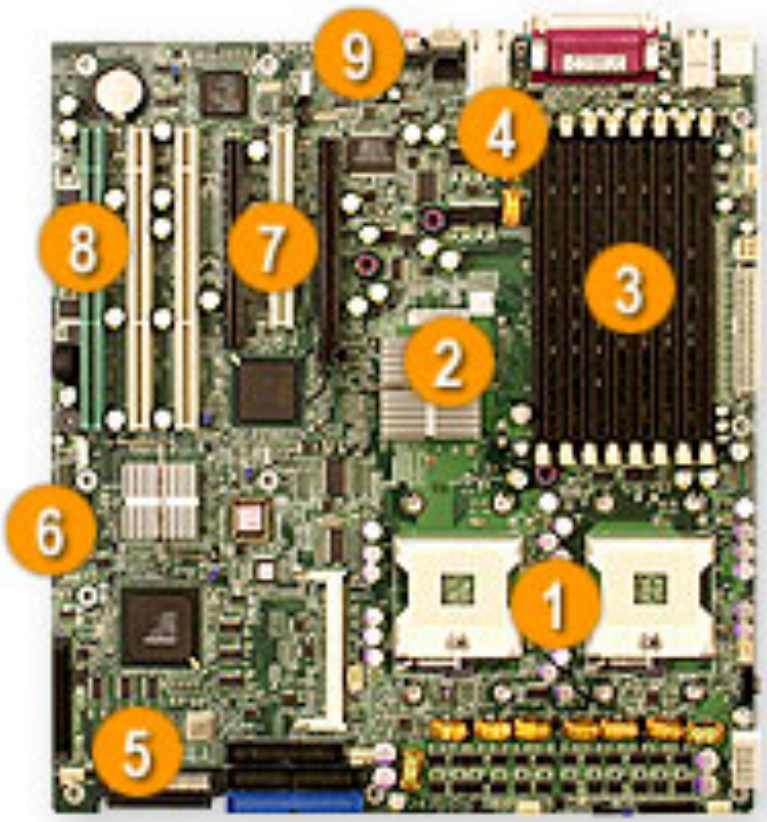
```
[anakano@hpc-login2 ~]$ ln -s /home/rcf-proj2/an2/anakano work596
[anakano@hpc-login2 ~]$ ls -l
total 2196
drwx------ 14 anakano m-csci    4096 May 11  2014 course/
drwx------  2 anakano m-csci    4096 Feb 28  2003 mail/
-rw-------  1 anakano m-csci   30684 Sep 10  2002 mbox
drwx------ 16 anakano m-csci    4096 Jun 13  2014 src/
...
lrwxrwxrwx  1 anakano m-csci      30 Aug 31 11:58 work596 -> /home/
rcf-proj2/an2/anakano/
[anakano@hpc-login2 ~]$ cd work596
[anakano@hpc-login2 ~/work596]$ pwd
/auto/rcf-proj2/an2/anakano
```

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



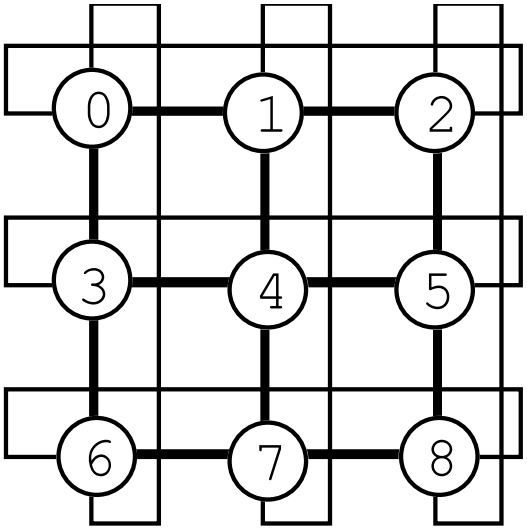
Key Features

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

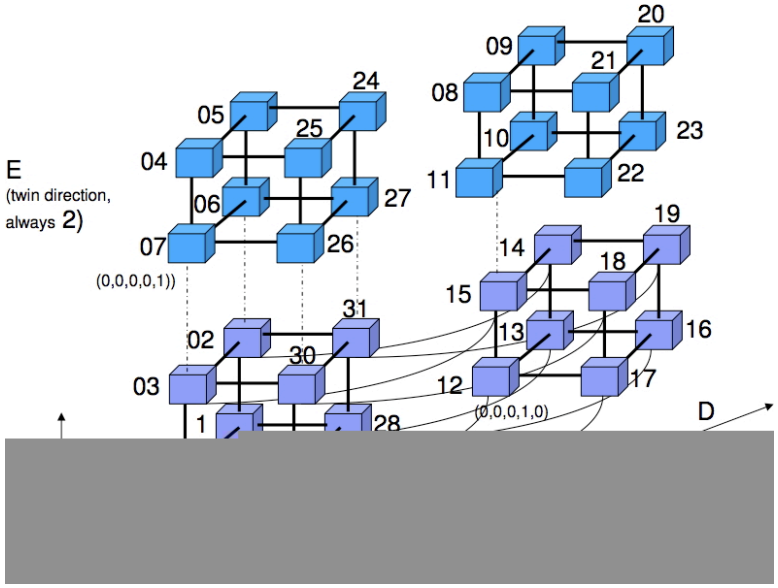
Communication Network

Mesh (torus)

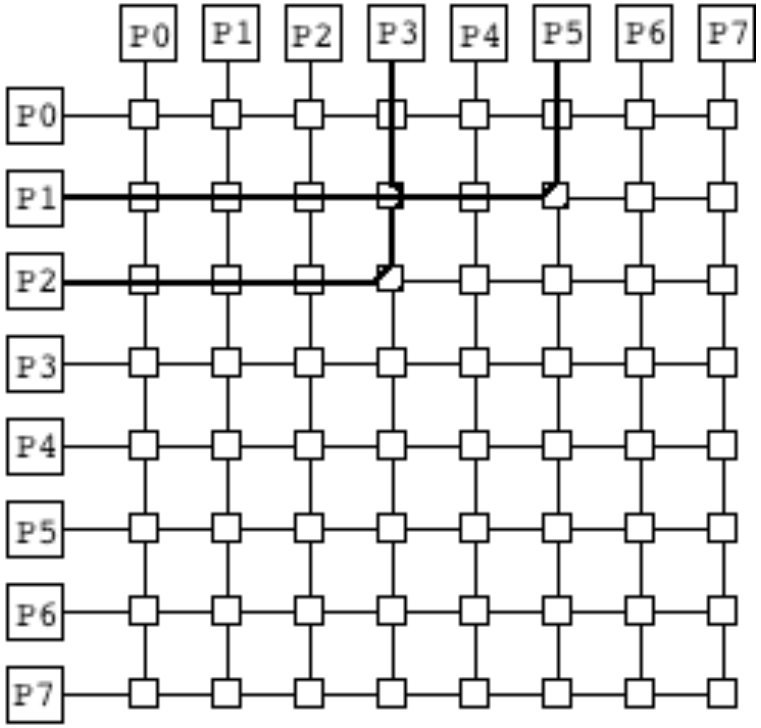


NEC Earth Simulator (640x640 crossbar)

IBM Blue Gene/Q (5D torus)



Crossbar switch



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

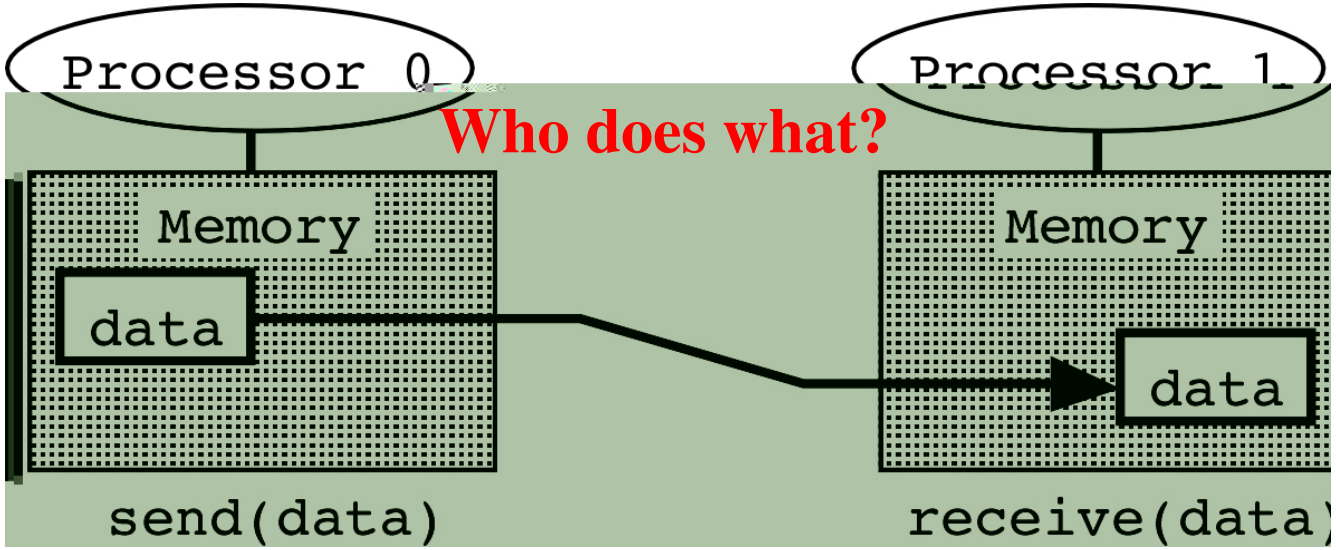
```
> mpirun -np 2 mpi_simple
```

MPI Programming

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

```
#include "mpi.h"
#include <stdio.h>
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();
}
```

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(...);
}
```

MPI Minimal Essentials

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

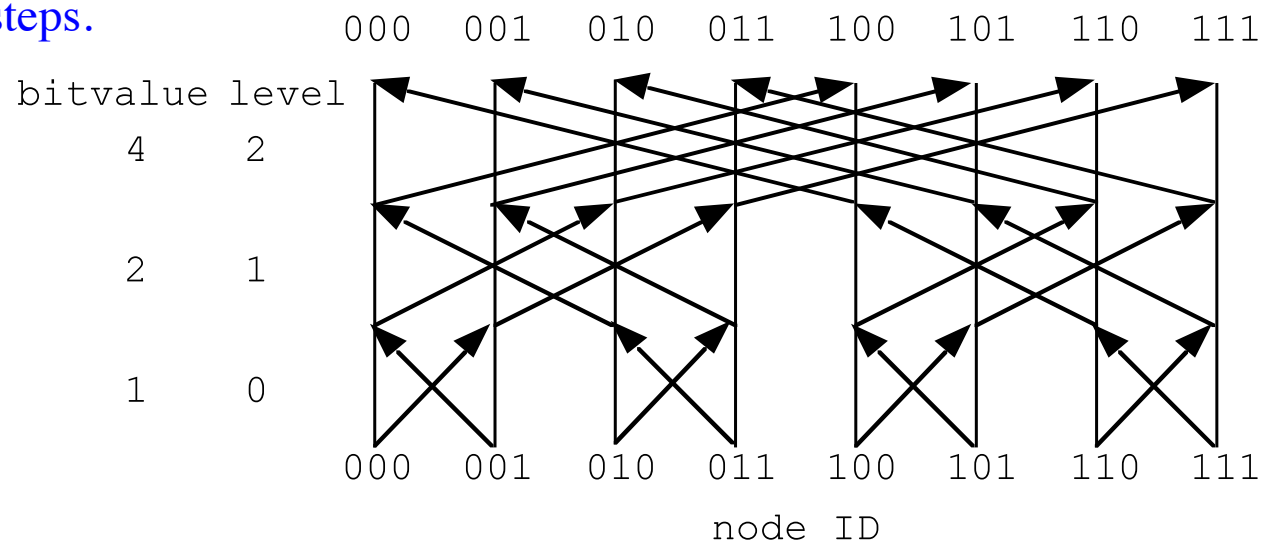
```
MPI_Send(&n, 1, MPI_INT, 1, 10, MPI_COMM_WORLD);
MPI_Recv(&n, 1, MPI_INT, 0, 10, MPI_COMM_WORLD, &status);
```

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)
```

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.

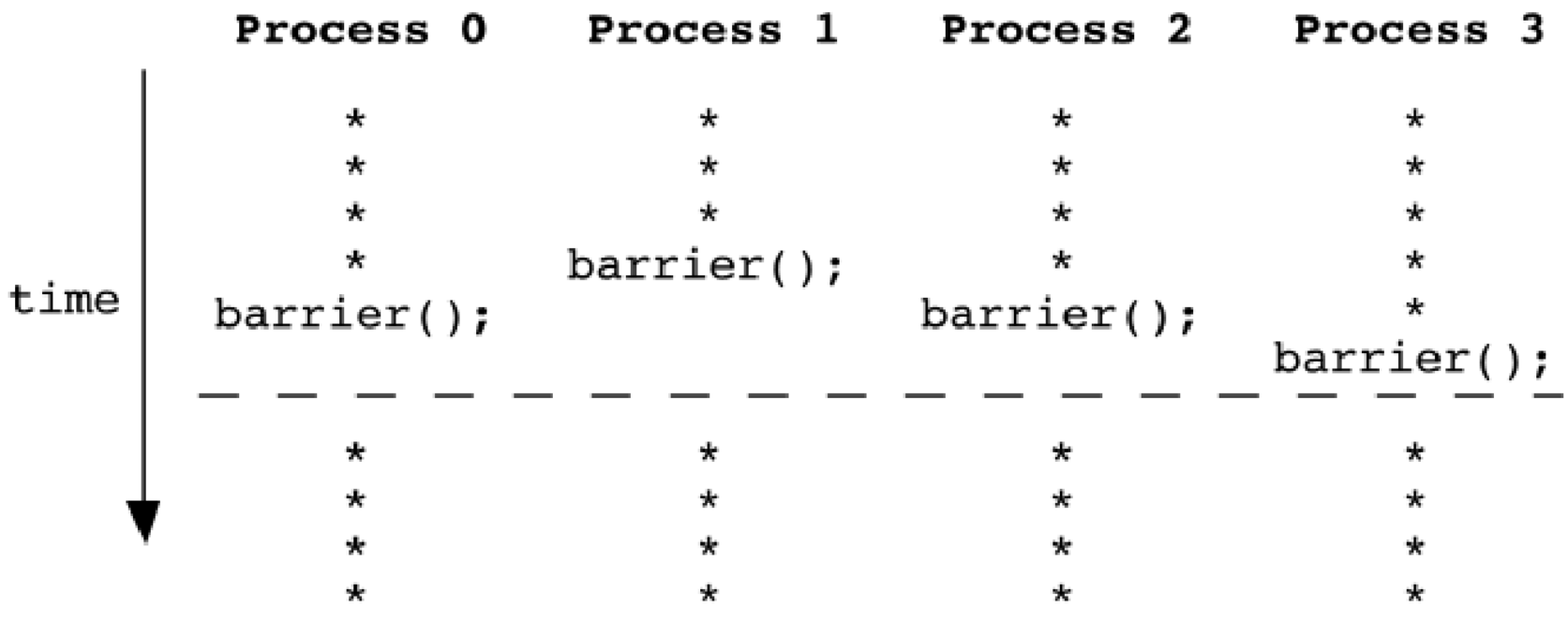


Butterfly network

a000 + a001 + a010 + a011 + a100 + a101 + a110 + a111
= ((a000 + a001) + (a010 + a011))
+ ((a100 + a101) + (a110 + a111))

Barrier

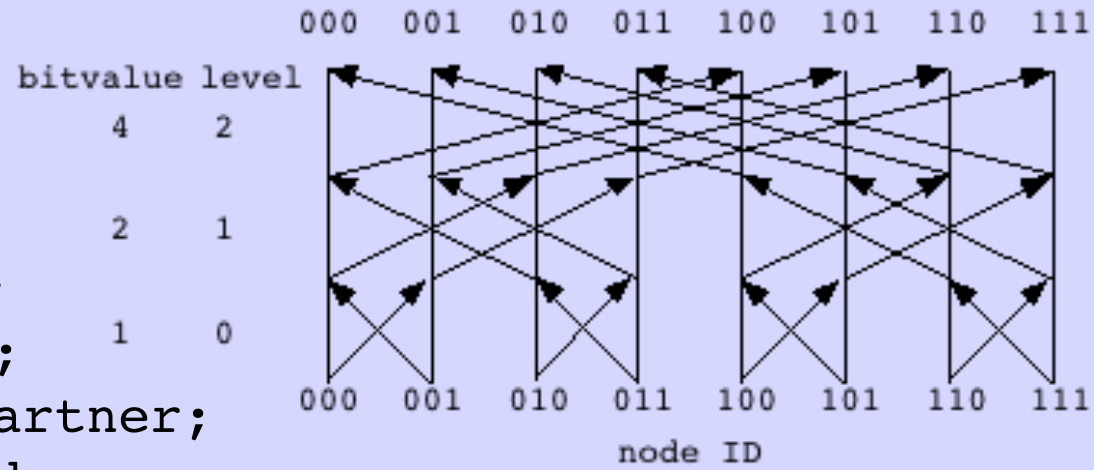
```
<A>;
barrier();
<B>;
```



MPI_Barrier(MPI_Comm communicator)

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
```



Exclusive OR

a	b	a XOR b
0	0	0
0	1	1
1	0	1
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 */
}

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("Node %d has %le\n", myid, partial);
  sum = global_sum(partial);
  if (myid == 0) {
        avg = sum/nprocs;
        printf("Global average = %d\n", avg);}
  MPI_Finalize();
}
```

Sample PBS Script

```
#!/bin/bash
#PBS -l nodes=2:ppn=4,arch=x86_64
#PBS -l walltime=00:00:59
#PBS -o global.out
#PBS -j oe
#PBS -N global
#PBS -A lc_an2
WORK_HOME=/home/rcf-proj2/an2/Your_ID
cd $WORK_HOME
np=$(cat $PBS_NODEFILE | wc -l)
mpirun -np 4 -machinefile $PBS_NODEFILE ./global
mpirun -np $np -machinefile $PBS_NODEFILE ./global
```

Output of `global.c`

- **4-processor job**

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

- **8-processor job**

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

Communicator

`mpi_comm.c`: Communicator = process group + context

```
#include "mpi.h"
#include <stdio.h>
#define N 64
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();
}
```

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

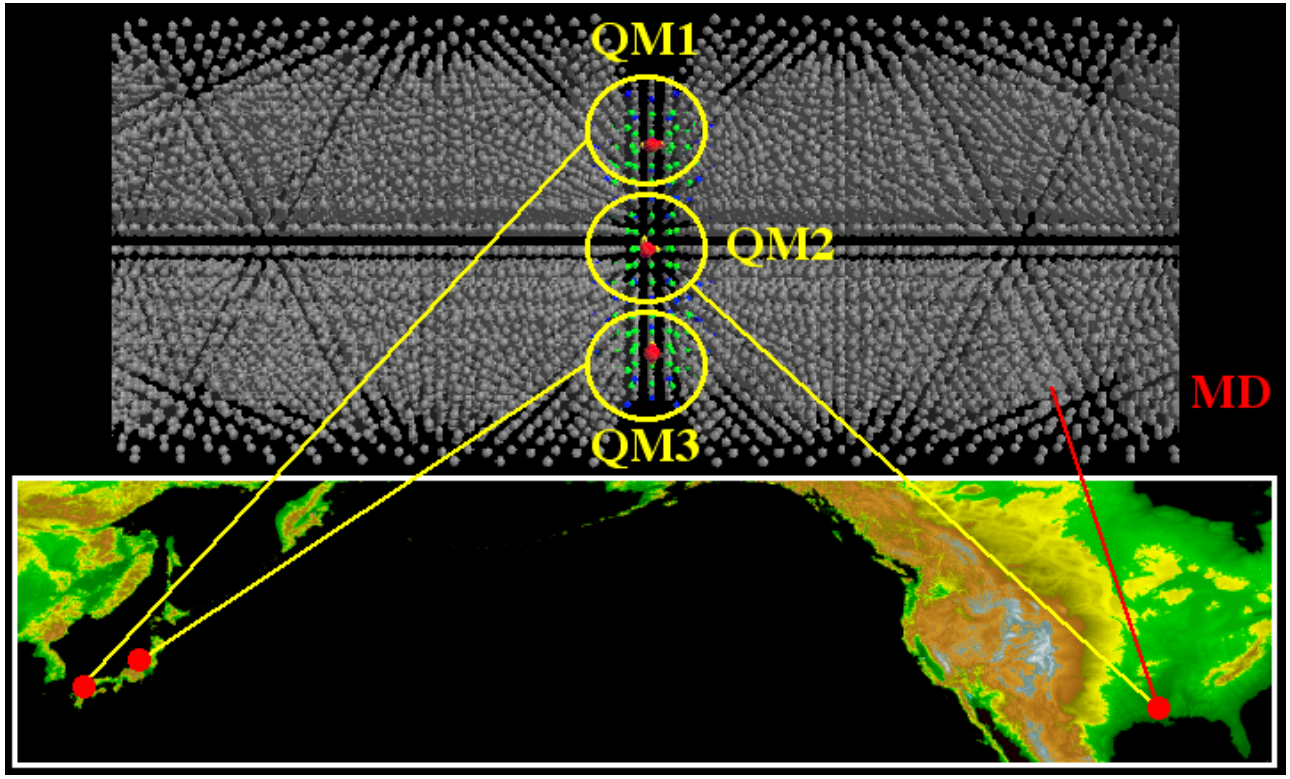
```
MPI_Comm_rank(world, &usc_world);
MPI_Comm_rank(us, &usc_national);
```

Output from mpi_comm.c

```
----------------------------------------
Begin PBS Prologue Sun Aug 31 12:33:38 PDT 2014
Job ID:            9494516.hpc-pbs.hpcc.usc.edu
Username:          anakano
Group:             m-csci
Name:              mpi_comm
Queue:             quick
...
Nodes:             hpc1953 hpc1954
PVFS:              /scratch (124G), /staging (328T)
TMPDIR:            /tmp/9494516.hpc-pbs.hpcc.usc.edu
End PBS Prologue Sun Aug 31 12:33:56 PDT 2014
----------------------------------------
...
process  2: n =      3
process  3: n =     -1
process  0: n =      3
process  1: n =      3
----------------------------------------
Begin PBS Epilogue Sun Aug 31 12:33:59 PDT 2014
...
Session:          16026
Limits:           nednodes=2:ppn=2,nodes=2:ppn=2,walltime=00:00:59
Resources:        cput=00:00:00,mem=3232kb,vmem=144064kb,walltime=00:00:01
...
End PBS Epilogue Sun Aug 31 12:34:05 PDT 2014
----------------------------------------
```

Grid Computing & Communicators

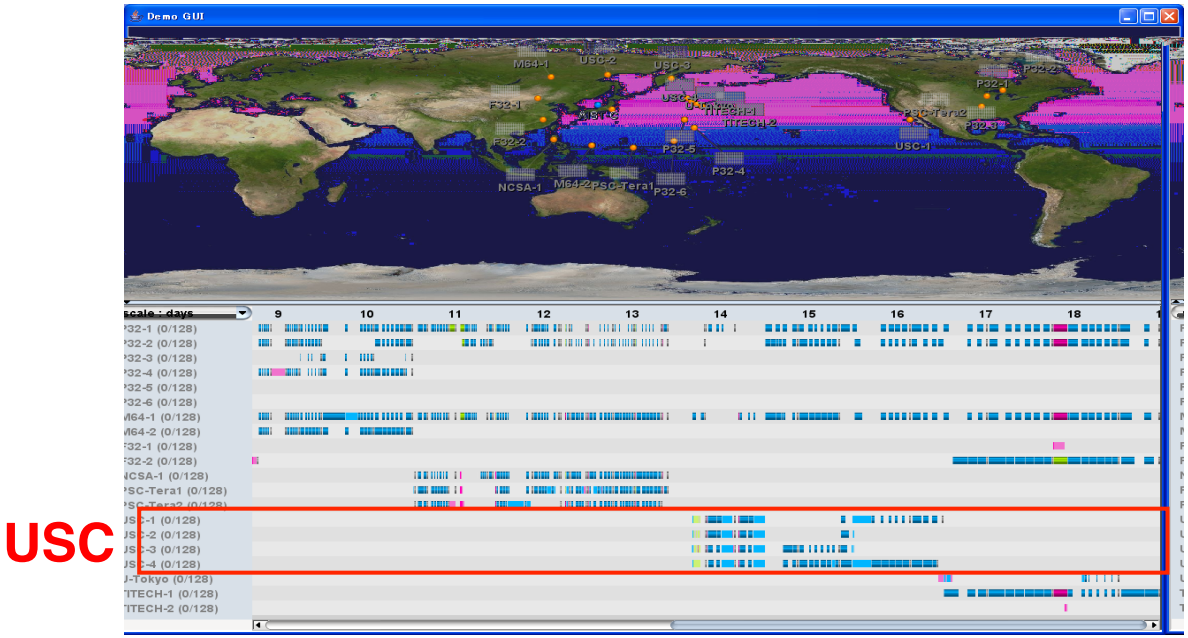
H. Kikuchi et al., "Collaborative simulation Grid: multiscale quantum-mechanical/classical atomistic simulations on distributed PC clusters in the US & Japan, *IEEE/ACM SC02*



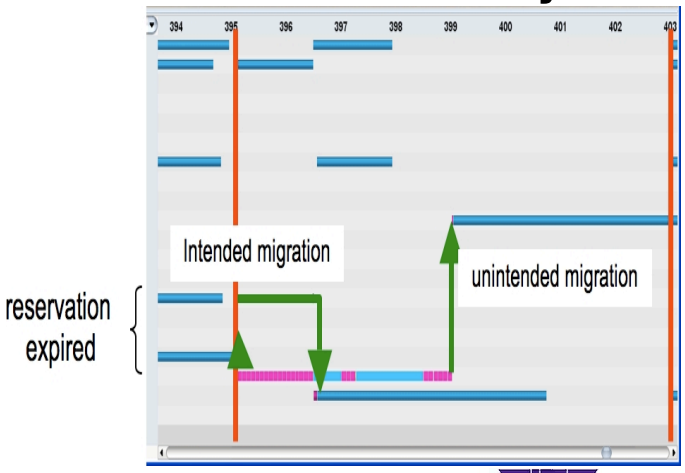
- **Single MPI program run with the Grid-enabled MPI implementation, MPICH-G2**
- **Processes are grouped into MD & QM groups by defining multiple MPI communicators as subsets of MPI_COMM_WORLD; a machine file assigns globally distributed processors to the MPI processes**

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)**



Automated resource migration & fault recovery

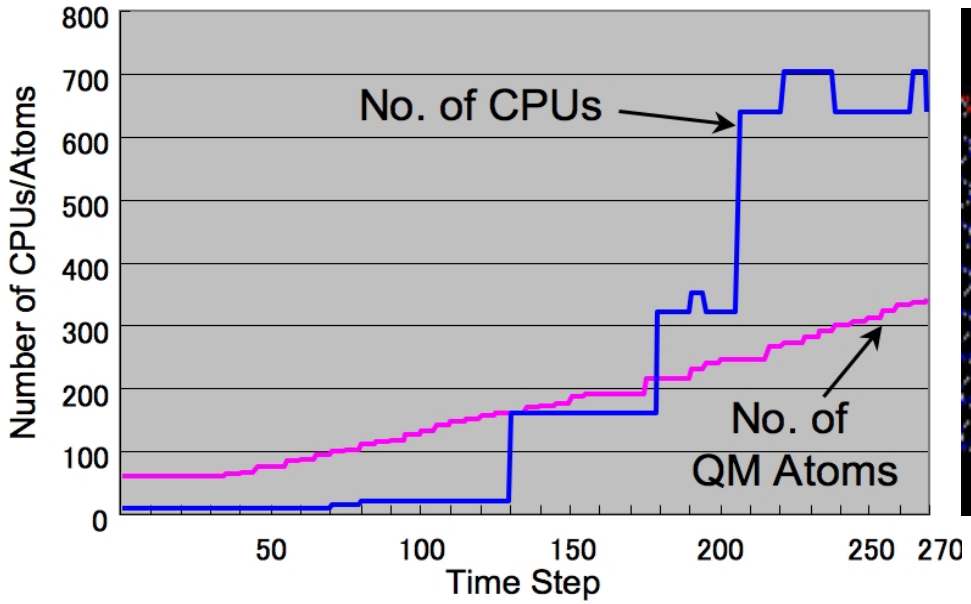


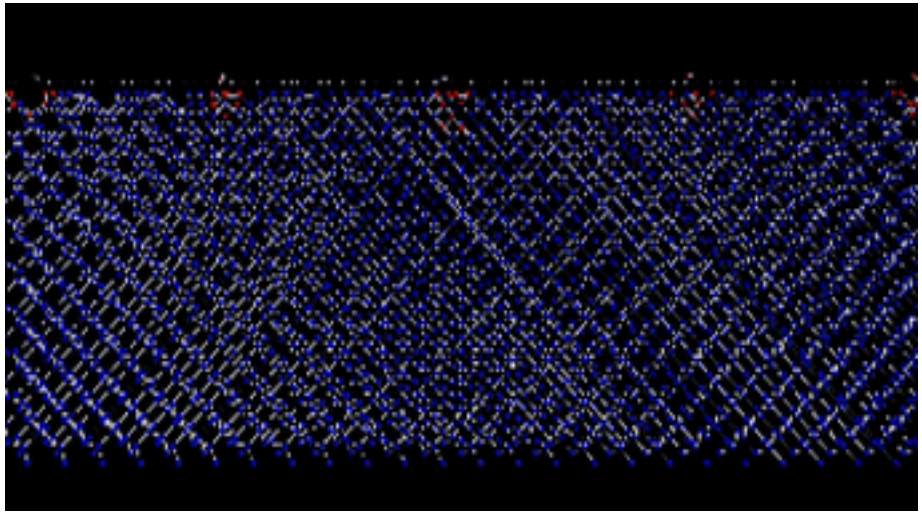
Takemiya et al., "Sustainable adaptive Grid supercomputing: multiscale simulation of semiconductor processing across the Pacific," *IEEE/ACM SC06*

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

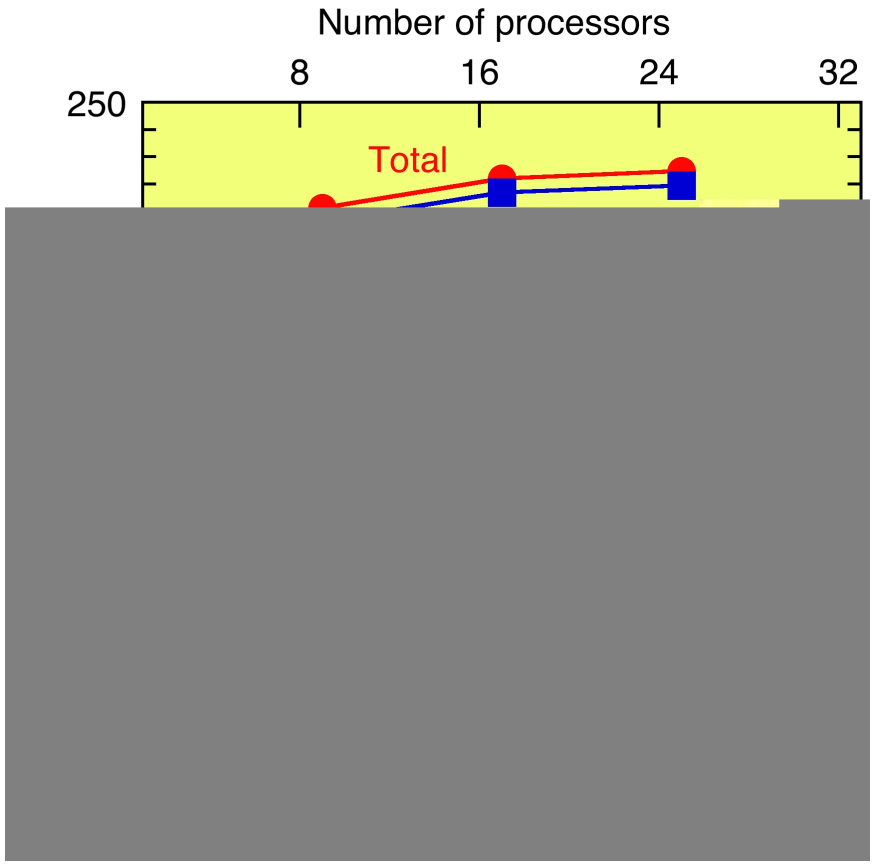




Multiscale QM/MD simulation of high-energy beam oxidation of Si

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**

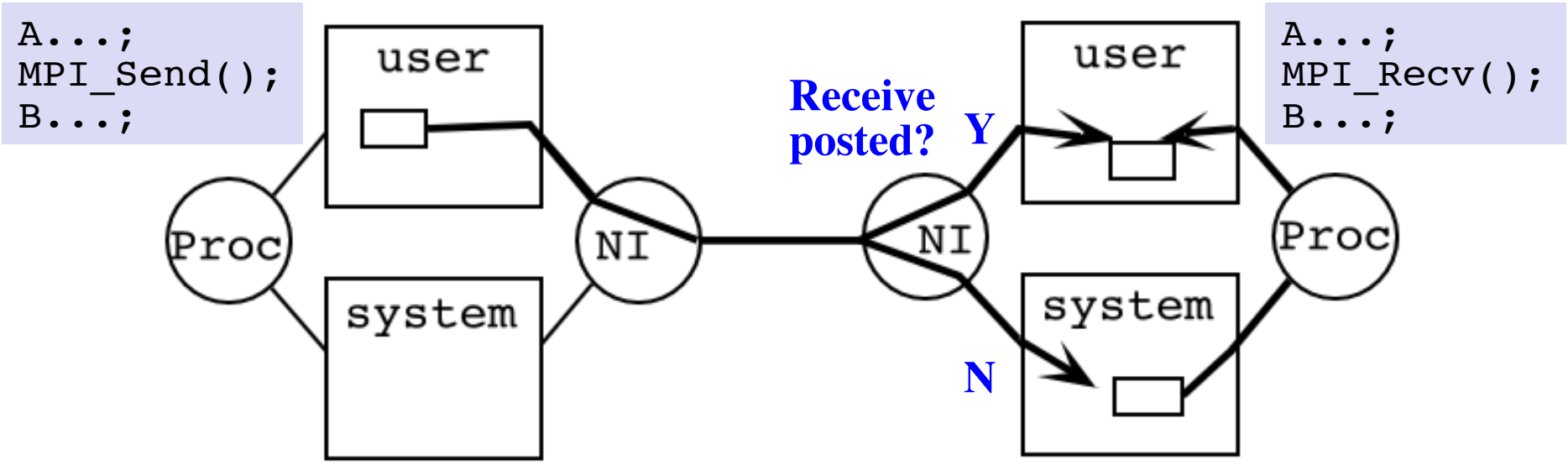
Synchronous Message Passing

`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
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();
}
```

Output from `irecv_mpi.c`

```
----------------------------------------
Begin PBS Prologue Sun Aug 31 13:34:33 PDT 2014
Job ID:            9495226.hpc-pbs.hpcc.usc.edu
Username:          anakano
Group:             m-csci
Name:              irecv_mpi
Queue:             quick
Shared Access:     no
All Cores:         no
Nodes:             hpc1979 hpc1980
PVFS:              /scratch (124G), /staging (328T)
TMPDIR:            /tmp/9495226.hpc-pbs.hpcc.usc.edu
End PBS Prologue Sun Aug 31 13:34:49 PDT 2014
----------------------------------------
...
Node 3: Send 3499500 Recv 499500
Node 2: Send 2499500 Recv 3499500
Node 0: Send 499500 Recv 1499500
Node 1: Send 1499500 Recv 2499500
----------------------------------------
Begin PBS Epilogue Sun Aug 31 13:34:51 PDT 2014
...
Session:          32692
Limits:           neednodes=2:ppn=2,nodes=2:ppn=2,walltime=00:59:00
Resources:        cput=00:00:00,mem=3232kb,vmem=144064kb,walltime=00:00:02
Queue:            quick
Shared Access:    no
Account:          lc_an2
End PBS Epilogue Sun Aug 31 13:34:58 PDT 2014
```

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 */
  ...
}
```