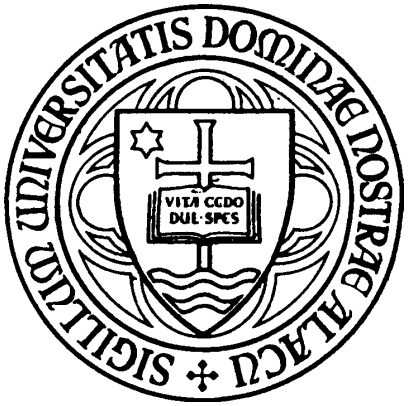
University of Notre Dame

MPI Tutorial

Part 3

Advanced Topics



Laboratory for Scientific Computing

Fall 1998

http://www.lam-mpi.org/tutorials/nd/

lam@lam-mpi.org

Section IX

SGI Origin 2000

SGI Origin 2000

- Two parallel SGI machines on campus:
  - `perseus.spi.nd.edu`: front end / interactive.

    4 196MHz CPUs. 512 MB real memory.
  - `medusa.spi.nd.edu`: batch jobs

    10 196MHz CPUs. 6 GB real memory.
- General SGI support page for Notre Dame:

  `http://perseus.spi.nd.edu/sgi/`

Interactive

- Compiling:
  - `cc [-n32 | -64] prog.c -o prog -lmpi`
  - `CC [-n32 | -64] prog.cc -o prog -lmpi`
  - `f77 [-n32 | -64] prog.f -o prog -lmpi`
- Executing:
  - `mpirun -np nprocs executable [arguments]`

Batch

- Complete information: `http://perseus.spi.nd.edu/sgi/` – See the link for “Creating and Submitting Origin2000 Batch Jobs”
- Basic Steps:
  - Create a file named `.nqshosts` in your `Public` directory. Add entries for `perseus` and `medusa`. This uses the same format as the `.rhosts` file. Make a symbolic link from your home directory.
  - Put working files in your `/usr1/people/afs_id` directory. This directory is NFS mounted between `perseus` and `medusa`. This directory is not accessable outside of `perseus` and `medusa`. All files used by your job must be in this directory.
  - Create a script that runs your program with either `csh` or `sh`. This script can be as simple as a single `mpirun` command.

Batch - continued

- Basic Steps - continued.
  - Submit the job using the `cqsub` command, for example: `cqsub scriptfile`.
  - Monitor jobs with the `qstat` command.
  - Delete jobs with `cqdel`.
  - Use `nqe` for a GUI interface for managing jobs.

Lab – Origin 2000

- Re-compile and re-run the following labs on `perseus`:
  - The ring program (lab from Part 1)
  - The manager/worker non-blocking average calculation (from Part 2)
- Remember that you are *not* using LAM on the Origin!
  - Do *not* `lamboot`, `lamrun`, `lamclean`, or `wipe`!
  - You must recompile your programs for IRIX and SGI's MPI
  - The command line syntax of `mpirun` is different; see the manual page for `mpirun(1)`: do `man mpirun` to see it

Section X

Communicators

Defining Groups

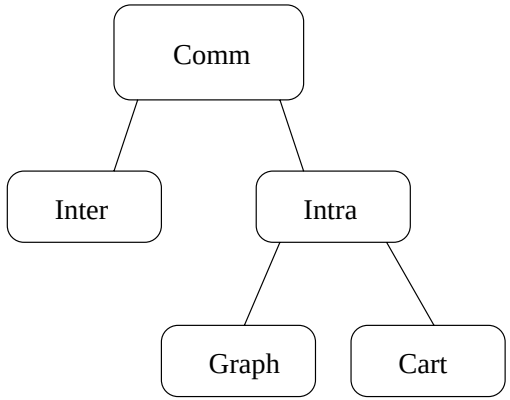
- All MPI communication is based on a *communicator* which contains a *context* and a *group*.
- Contexts define a safe communication space for message-passing.
- Contexts can be viewed as system-managed tags.
- Contexts allow different libraries to co-exist.
- The group is just a set of processes.
- Processes are always referred to by rank in group.

Pre-Defined Communicators

- `MPI_COMM_WORLD`: Contains all processes available at the time the program was started.
- `MPI_COMM_NULL`: An invalid communicator. Cannot be used as input to any operations that expect a communicator.
- `MPI_COMM_SELF`: Contains only the local process. Not used to communicate
- `MPI_COMM_EMPTY`: There is no such thing as `MPI_COMM_EMPTY`. (Why not?)

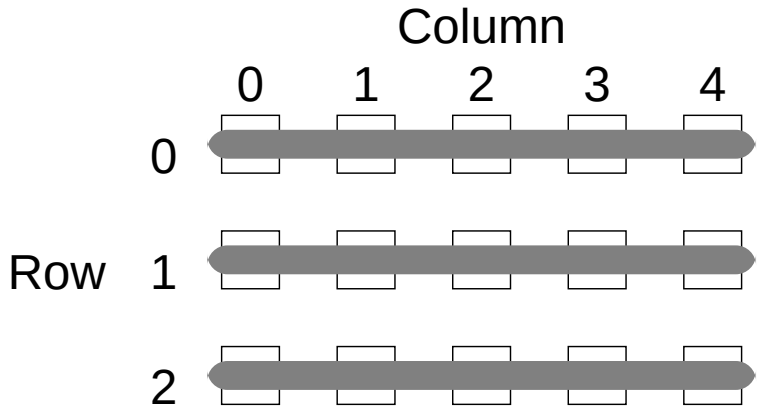
Communicator Types

- There are 4 types of communicators
  - Intercommunicators
  - Intracommunicators (further divided into two sub types)
    - Cartesian communicators
    - Graph communicators
- We will only discuss intracommunicators; inter-, Cartesian, and Graph communications will not be covered



Subdividing a Communicator

- The easiest way to create communicators with new non-overlapping groups is with `MPI_COMM_SPLIT`.
- For example, to form groups of rows of processes



use

```
MPI_COMM_SPLIT(oldcomm, row, 0, newcomm);
```

MPI_COMM_SPLIT

- Partitions the group associated with `comm` into disjoint subgroups.
- Each subgroup contains all processes having the same color.
- Within each subgroup, processes are ranked in the order defined by the value of the argument key, with ties broken according to their rank in the old group.

```
int MPI_Comm_split(MPI_Comm comm, int color,
                   int key, MPI_Comm *newcomm)

Intracomm Intracomm::Split(int color, int key) const

MPI_COMM_SPLIT(COMM, COLOR, KEY, NEWCOMM, IERR)
  INTEGER COMM, COLOR, KEY, NEWCOMM, IERR
```

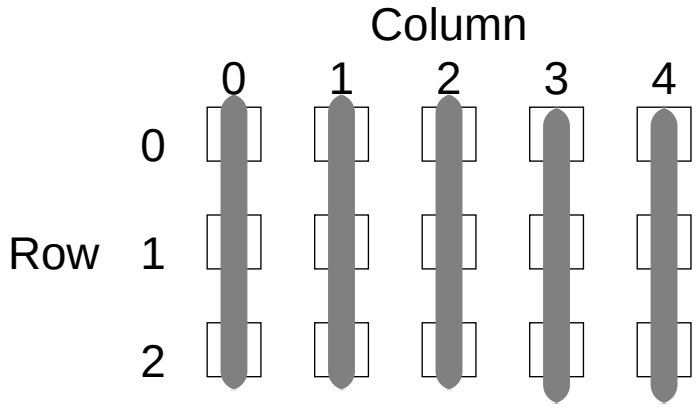
Subdividing a Communicator (cont.)

- To maintain the order by rank, use

```
MPI_COMM_RANK(oldcomm, rank);
MPI_COMM_SPLIT(oldcomm, row, rank, newcomm);
```

Subdividing (cont.)

- Similarly, to form groups of columns,



use

```
MPI_COMM_SPLIT(oldcomm, column, 0, newcomm2);
```

Subdividing (cont.)

- To maintain the order by rank, use

```
MPI_COMM_RANK(oldcomm, rank);
MPI_COMM_SPLIT(oldcomm, column, rank, newcomm2);
```

MPI_COMM_DUP

- It is a collective operation. All processes in the original communicator must call this function.
- Duplicates the communicator group, allocates a new context, and selectively duplicates cached attributes.
- Why isn't there a MPI_GROUP_DUP?

MPI_COMM_DUP

- The resulting communicator is not an exact duplicate. It is a whole new separate communication universe with similar structure.

```
int MPI_Comm_dup(MPI_Comm comm, MPI_Comm *newcomm)

Intracomm Intracomm::Dup() const
Intercomm Intercomm::Dup() const
Cartcomm Cartcomm::Dup() const
Graphcomm Graphcomm::Dup() const

MPI_COMM_DUP(COMM, NEWCOMM, IERR)
  INTEGER COMM, NEWCOMM, IERR
```

Simple C Example

```
#include <stdio.h>
#include <mpi.h>

main(int argc, char **argv)
{
  MPI_Comm new_comm;
  int myrank, size, even, value;

  MPI_Init (&argc, &argv);

  MPI_Comm_rank (MPI_COMM_WORLD, &myrank);
  MPI_Comm_size (MPI_COMM_WORLD, &size);

  /* Are we odd or even? */
  even = ((myrank % 2) == 0);

  /* Split comm into row and column comms */
  MPI_Comm_split(MPI_COMM_WORLD, even, myrank, &new_comm);

  value = myrank;
  MPI_Bcast(&value, 1, MPI_INT, 0, new_comm);
  printf("Rank %d: Got broadcast value of %d\n", myrank, value);

  MPI_Finalize();
  return 0;
}
```

Lab - Hierarchical Communications

- Objective: Use communicators to effect a hierarchy of communications.
- Prompt the user for X and Y (the size of the matrix)
- Create three communicators:
  - `COMM_MATRIX`: A duplicate of `MPI_COMM_WORLD`
  - `COMM_ROW`: Will consist of processes in the same row ordered by column index.
  - `COMM_COL`: Will consist of processes in the same column ordered by row index.

Lab - Hierarchical Communications (cont.)

- The row and column that a process belongs to can be determined using the following formulae:

```
row = rank / Y
col = rank % Y
```

- Each processor should print the sum of rank numbers (relative to the matrix communicator) in its row and column

Section XI

IBM SP

IBMs POWER Architectures

- Multiple chip, CMOS technology processor:
- Up to 4 instructions may be executed per clock cycle
- Combined floating point multiply-add (FMA) instruction which allows a peak MFLOPS rate equal to two times the MHz rate
- Zero-cycle branches - instruction path determined in advance by the instruction cache unit
- Simultaneous running of fixed- and floating-point operations.
- 32 general purpose registers (32 bit)
- 32 floating point registers (64 bit - double precision)
- Clusters: first RS6000 based parallel computing

IBMs Scalable POWERparallel Architectures

- Improved processor architecture
- High performance inter-processor communications
- System software for managing multiple machines
- Rack configuration - multiple machines in a frame

High Performance Switch

- Packet-switched network (versus circuit-switched)
- Bi-directional, any-to-any internode connection - allows all processors to send messages simultaneously
- Support for multi-user environment - multiple jobs may run simultaneously over the switch (one user does not monopolize switch)
- Multistage network: on larger systems, additional intermediate switches are added as the system is scaled upward.
- Path redundancy - multiple routings between any two nodes. Permits routes to be generated even when there are faulty components in the system.
- Error detection
- Architected for expansion to 1000s of ports

Protocols

- IP (Internet Protocol) - default; permits shared usage of HPS-2 adapter by multiple processes.
- US CSS (User Space Communication Subsystem) - intended for parallel applications that require maximum communications performance. Only one process per node may use US communications.

Performance

- 40 MB/sec peak bi-directional bandwidth between any two nodes. Yields 2.5 GB/sec bi-sectional bandwidth for a 128 node system.
- Hardware latency: 500 ns up to 80 nodes, 875 ns for systems with up to 512 nodes
- IBM High Performance Switch performance measurements

Protocol	Latency	Pt to Pt Bandwidth
MPI IP	$<$300 usec	$>$12 MB/Sec.
MPI user space	$<$45 usec	$>$35 MB/Sec.

Control Workstation

- Serves as the single point of control for System Support Programs used by System Administrators for system monitoring, maintenance and control.
- Separate machine - not part of the SP frame
- Must be a RISC System/6000
- Connects to each frame with
  - RS-232 control line
  - external ethernet LAN
- Acts as boot/install server for other servers in the SP system
- May also act as a file server

SP Software

- AIX Operating System
- System Administration
- Parallel Environment
- LoadLeveler
- Application Software

Parallel Operating Environment (POE)

- Parallel Compiler Scripts
- Environment Variables
- Visualization Tool
- Program Marker Array
- System Status Array
- Parallel Debugging Facility
- Parallel Profiling Capability
- **Environment Variables**

IBM SPs on Campus

- Two parallel SP resources on campus:

Machine names	Resource type
`spin13.spi.nd.edu` thru `spin16.spi.nd.edu`	Interactive development and debugging
`spin17.spi.nd.edu` thru `spin30.spi.nd.edu`	Batch submission; production runs

- Web page information and support of HPCC SP's:

  `http://www.nd.edu/~hpcc/`

- Interactive nodes: RS6000 370, 256MB RAM, 512MB swap, 2GB scratch space

- Batch nodes: RS6000 390, 128MB RAM, 512MB swap, 2GB scratch space

Interactive Use

- Compiling:
  - `mpcc prog.c -o prog`
  - `mpCC prog.cc -o prog`
  - `mpxlf prog.f -o prog`
- Executing (don't need to use `mpirun`):
  - `source IP_css0_hosts`
  - `./prog`

Setting up for POE

- Download and source the appropriate environment file from the tutorial web page
  - `IP_en0_hosts`: Sets up for IP over ethernet
  - `IP_css0_hosts`: Sets up for IP over the switch
  - `US_css0_hosts`: Sets up for user space over the switch
  - All files setup for 4 nodes, and look for a file named "`hosts`" in your current directory
- Feel free to edit/modify the files; they are commented.

Running MPI Programs on the SP

- The directory where your program resides *must* have the AFS permissions set such that it is `system:anyuser` readable
  - All parent directories must be at least `system:anyuser` listable.
  - Your program will *not* run with AFS authentication on the remote nodes.
- `mpirun` is available, but not necessary. Since all the setup is in environment variables, one can just invoke the program directly.
- For example:

```
spin13% mpcc foo.c -o foo
spin13% fs sa . system:anyuser read
spin13% ./foo
```

Batch Use

- Notre Dame uses IBM's LoadLeveler package for batch submissions.
- There are three pools to which parallel jobs can be submitted: two 4 node pools, and one 6 node pool.
- Exact information and examples can be found on:

`http://www.nd.edu/~hpcc/faqs.html`

LoadLeveler can also be used to submit serial jobs to the RS6000 farm in the HPCC.

Lab – IBM SP

- Re-compile and re-run the following labs on any of the interactive nodes of the SP (`spin13` through `spin16`):
  - The ring program (lab from Part 1)
  - The manager/worker non-blocking average calculation (from Part 2)
- Remember that you are *not* using LAM on the SP!
  - Do *not* `lamboot`, `lamrun`, `lamclean`, or `wipe`!
  - You must recompile your programs for AIX and IBM's MPI
  - You must source one of the environment files to set all the variables
  - The use of the `mpirun` command is not necessary; you can just execute the program directly