Using MPI (MPICH) in C programs on our cluster

Kate Cowles

22S:295 High Performance Computing Seminar, Oct. 4, 2007

Outline

MPI

- “Message Passing Interface”
- not a language but a standard for libraries of functions to enable parallelization of code written in C, C++, or Fortran
- several implementations, including MPICH and LAM
- All parallelism is explicit: the programmer is responsible for correctly identifying parallelism and implementing parallel algorithms using MPI constructs.
- The number of tasks dedicated to run a parallel program is static. New tasks can not be dynamically spawned during run time. (MPI-2 addresses this issue).

MPICH

- compatible with parallel linear algebra library PlaPACK
- doesn't work with xmpi
- MPICH does not include new features in MPI-2 standard

MPICH on our cluster

- lines in my .cshrc file that make MPICH my default instead of LAM

```
setenv MPIRUN_HOME /opt/mpich/ch-p4/bin
set path = ( /opt/mpich/chp4/bin $PATH )
```

Compiling and running C programs for MPICH

- compiling

  ```
  mpicc <progname>.c -o <execname>
  ```

  - example

    ```
    mpicc greetingsm.c -o greetingsm
    ```

- running it

  ```
  mpirun -np <# processes> -machinefile
  <machinefilename> <execname>
  ```

  - example

    ```
    mpirun -np 12 -machinefile kc_machines greetingsm
    ```

Machine file

- specfies nodes you want to use
- default MPICH machine file in `/opt/mpich/shared` lists only `localhost`
- contents of example machine file for testing

```
node11
node12
node13
node14
node15
```

MPI naming conventions

- MPI identifiers begin with `MPI_`
- rest of *function* name is upper and lower: `MPI_Init`, `MPI_Comm_size`
- rest of *constant* name is all upper case: `MPI_COMM_WORLD`, `MPI_SUCCESS`

Communicators

- communicator: a group of processes that can send messages to each other
- `MPI_COMM_WORLD`: communicator predefined by MPI
  - consists of all the processes running when program execution begins (i.e. as many as requested with `-np` option on `mpirun`)
- *rank* or *process id*: integer identifier assigned by the system to each process within a communicator when the process initializes
  - consecutive and begin at zero
  - used by programmer to direct different processes to do different things in single-program, multiple-data approach

Note: discussion of `ping.c` goes here.

Communication

- point-to-point communication: one process sends message to one other process
- collective communication: one-to-many; many-to-one; many-to-many
- blocking versus non-blocking communication

Buffering

- system buffer space
  - not all MPI implementations use it
  - holds data in transit (e.g. if one process sends message and receiver isn't ready to receive it)
  - managed entirely by MPI
- applications buffer: program variables managed by user
  - user-managed send buffer to set up messages for sending

Note: discussion of `greetingsm.c` goes here.

Blocking and non-blocking communication

- blocking sends and receives
  - blocking send routine “returns” only when it is safe to modify the application buffer (your send data) for reuse.
  - blocking receive “returns” only after the data has arrived and is ready for use by the program.
- non-blocking sends and receives
  - Non-blocking send and receive routines return almost immediately; do not wait to verify that any communication events have completed.

MPI_Send and MPI_Recv: Blocking send and receive

Arguments

- Buffer
  Program (application) address space that references the data that is to be sent or received. In most cases, this is simply the variable name that is be sent/received. For C programs, this argument is passed by reference and usually must be prepended with an ampersand: &var1
- Data Count
  Indicates the number of data elements of a particular type to be sent.
- Data Type
  For reasons of portability, MPI predefines its elementary data types.

Arguments continued

- Destination
  An argument to send routines that identifies receiving process by rank.
- Source
  An argument to receive routines that identifies sending process by rank; may be set to the wild card MPI_ANY_SOURCE.
- Tag
  Arbitrary non-negative integer assigned by the programmer to uniquely identify a message. Send and receive operations should match message tags.

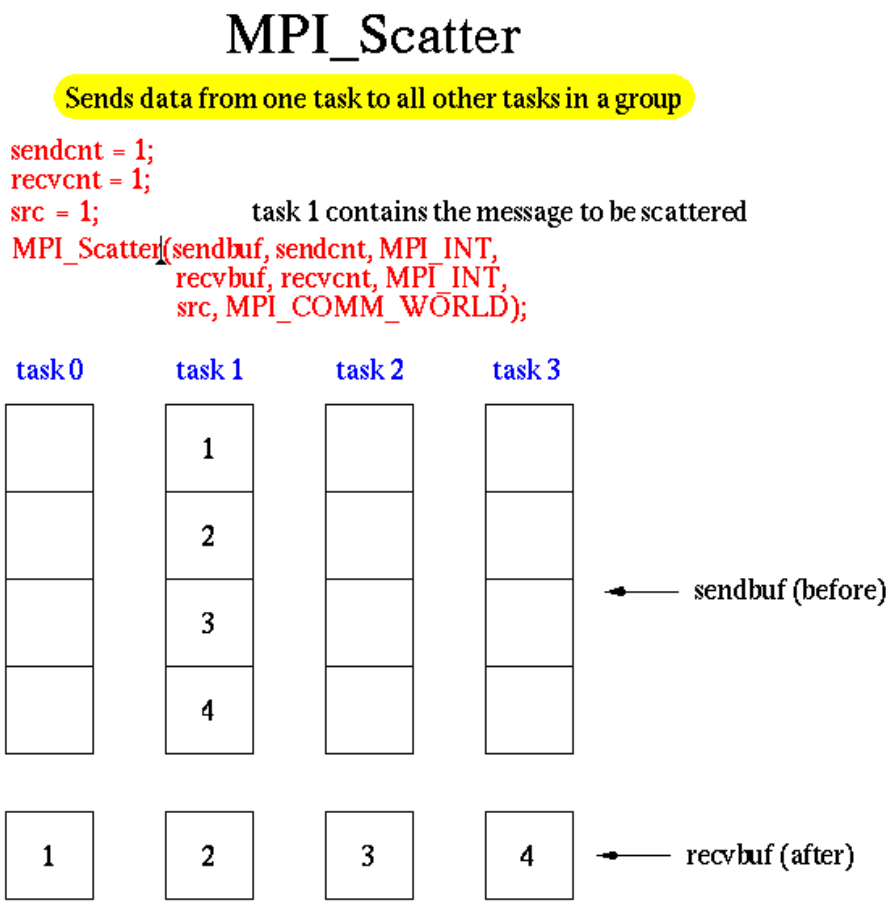
Arguments continued

- Communicator
  Indicates the communication context (set of processes for which the source or destination fields are valid); usually MPI_COMM_WORLD
- Status
  For a receive operation, indicates the source of the message and the tag of the message. In C, this argument is a pointer to a predefined structure MPI_Status. The actual number of bytes received is obtainable from Status via the MPI_Get_count routine.

Collective communcations

- must involve all processes in the scope of a communicator
- types
  - synchronization – processes wait until all members of the group have reached the synchronization point.
  - data Movement - broadcast, scatter/gather, all to all.
  - collective Computation (reductions) - one member of the group collects data from the other members and performs an operation (min, max, add, multiply, etc.) on that data.

Scatter



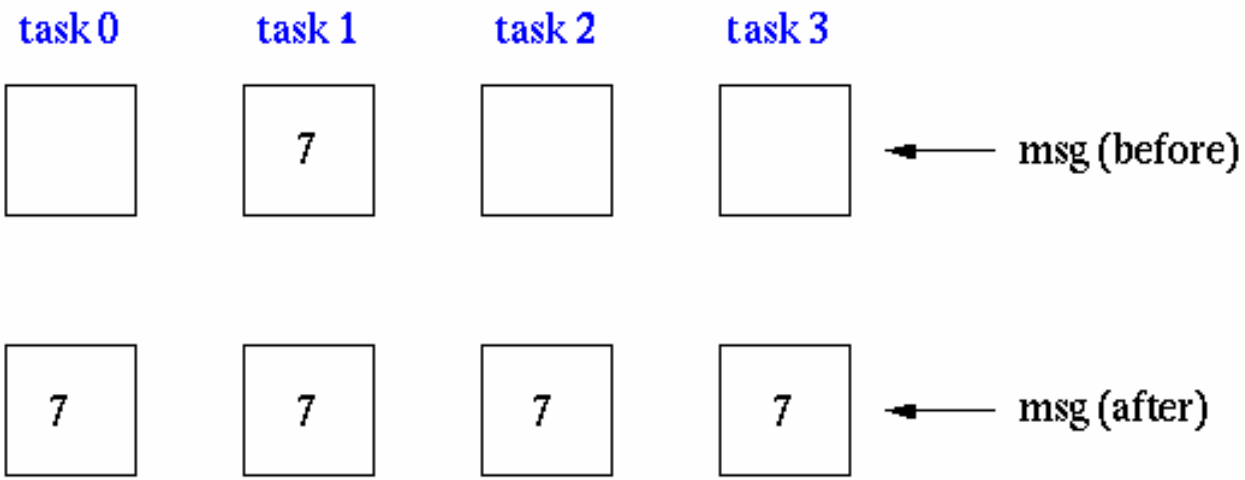
Note: discussion of `scatterrows.c` goes here.

Broadcast

MPI_Bcast

Broadcasts a message to all other processes of that group

```
count = 1;
source = 1;                    broadcast originates in task 1
MPI_Bcast(&msg, count, MPI_INT, source, MPI_COMM_WORLD);
```

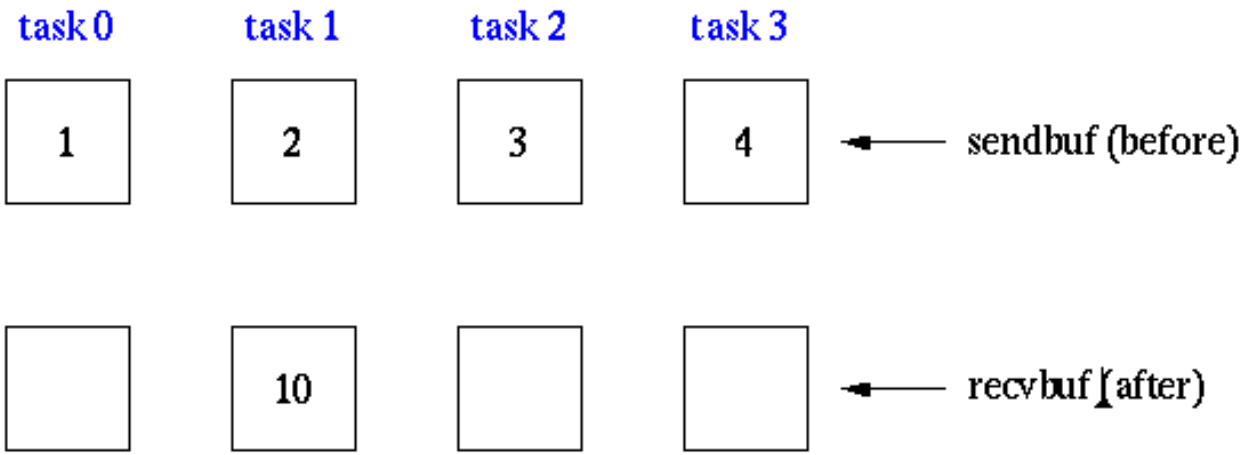


Reduce

MPI_Reduce

Perform and associate reduction operation across all tasks in the group and place the result in one task

```
count = 1;
dest = 1;                    result will be placed in task 1
MPI_Reduce(sendbuf, recvbuf, count, MPI_INT, MPI_SUM,
           dest, MPI_COMM_WORLD);
```



Note: discussion of `reduce.c` goes here.

Resources for further study

These are examples only.

- Pacheco, P.S. *Parallel Programming with MPI*, 1997. Morgan Kaufman.
  `http://www.cs.usfca.edu/mpi/`
- shorter version available online at
  `ftp://math.usfca.edu/pub/MPI/mpi.guide.ps`
- Lawrence Livermore National Laboratory MPI tutorial and examples
  `http://www.llnl.gov/computing/tutorials/mpi/`