Message Passing Programming

Parallel Programming

- Abstractions on top of machines
- Message Passing Paradigm
  - p processes, each with own process space
  - communication via messages
- Works on clusters ...
  - requires explicit data partitioning
  - requires explicit parallel programming
- Message passing programs
  - asynchronous
  - loosely synchronous
- SPMD -- single program multiple data
- MPMD -- multiple program multiple data
- environment --
  - separate machines, no shared file systems
  - shared file systems, separate OS (nfs, lustre, hadoop, GFS, ...)
  - network OS and FS

Message Passing basics

send (void *sendbuf, int nelems, int dest)
receive (void *redbuf, int nelems, int *source)

- Questions
  - Blocking receive?
  - Blocking send?
  - Data transmission issues

```
a = 100;
send (&a, 1, 1);
a = 0;
```

- Does "a = 0;" cause a problem?
-

```
int a[1000];
// init a
send (&a, 1000, 1);
```

- Latency vs speed here

Sending mechanisms

- non buffered (blocking)
  - request to send, Ok to send ...
  - deadlocks? (both send...)
- buffered
  - blocking, non-blocking
  - deadlocks?
  - bounded buffer problems, performance issues
- Other issues
  - store and forward
  - data kinds
  - check status, data types, who from ...
- MPI & PVM ... libraries to send messages

Message Passing Interface ... MPI (Grama Ch 6, Pacheco Ch 3, Matloff Ch 8)

- A standard interface for message passing
  - Matloff: "MPI is the de facto standard for message-passing software."
- Used for Fortran, C, C++ (and others that can use C or C++)
- Many different implementations
  - some by vendors
  - others by independent parties
  - We will be using "openmpi" (have used mpi-ch in the past)
    - http://www.open-mpi.org
    - http://www-unix.mcs.anl.gov/mpi/mpich/
    - Free implementations
- Many different routines (up to 125?)
- supports blocking and non-blocking messages
- does not show connection structure
- support (primarily) SPMD programming
- There are standarts too:
  - MPI-3.1 standard (openmpi does this)
  - MPI-4.0 standard (openmpi has many .... but not all)

6 Basic Routines

- int MPI_Init (int *argc, char ***argv)
- int MPI_Finalize()
  - return values ... MPI_SUCCESS
- concept of the communication domain
  - default is all processors
  - can make subsets
  - MPI_Comm identifiers
  - MPI_COMM_WORLD
- int MPI_Comm_size (MPI_Comm comm, int *size)
- int MPI_Comm_rank (MPI_Comm comm, int *rank)
- int MPI_Send (void *buf, int count, MPI_Datatype dt, int dest, int tag, MPI_Comm comm)
- int MPI_Recv (void *buf, int count, MPI_Datatype dt, int src, int tag, MPI_Comm comm, MPI_Status *status)

Look at mpi-hello.c

Details ...

datatype

- MPI_CHAR, MPI_SHORT, ..., MPI_UNSIGNED_CHAR, ...
- MPI_FLOAT, MPI_DOUBLE, MPI_LONG_DOUBLE, MPI_BYTE,
- MPI_PACKED

tag -- types of messages ... user defined

- up to MPI_TAG_UB ... at least 32767

recv

- MPI_ANY_SOURCE
- MPI_ANY_TAG

```
typedef struct MPI_Status {
  int MPI_SOURCE;
  int MPI_TAG;
  int MPI_ERROR; };
```

Return data:

- MPI_SUCCESS
- MPI_.... (error numbers)

Deadlock issues:

- Send and recv
  - blocking or buffered -- implementation dependent
  - tag orders: s: T1, T2 r: T2, T1
  - All send, then all recv ...
  - rank % 2: 0 -> send, recv; 1 -> recv, send

MPI_Sendrec: 2 buffers specified, send buffer and receive buffer

- may send to one and receive from another

MPI_Sendrec_replace: 1 buffer specified, may have dest and source different

Example -- Odd-Even Sort (prog-6.1, on departmental Linux systems in /home/phil/public/csci515)

- Edd-Even Sort (Grama section 9.3. 6.3.5: "Recall from Section 9.3")
- n phases to get numbers sorted (a_1, a_2, ... a_n)
  - odd phase: exchange pairs (1,2), (3,4), ...
  - even phase: exchange pairs (2,3), (4,5), ...
- Parallelization ... exchange and compare n times.
  - n processes, n time. (cost n^2)
- Use of MPI_PROC_NULL -- can send messages to it!

Topologies / Embedding

- MPI -- Linear
- Algorithms -- mesh, hypercube, ...
- how to map rank <-> algorithm
  - row major
  - column major
  - hypercube
- Possible Problem?
  - proper mapping to hardware
  - possible better placements on clusters, makes use of special hardware
- Solution -- communication grids
  - Cartesian topologies
    - MPI_Cart_create(MPI_Comm old, ..., MPI_Comm *comm_cart)
      - all processes must call this
    - MPI_Cart_rank(MPI_Comm, int *coords, int *rank)
      - coords to rank -- for communication
    - MPI_Cart_coord(MPI_Comm, int rank, int maxdims, int *coords)
      - rank to coordinates (MPI_Comm_Rank gets rank)
    - MPI_Cart_shift
      - compute rank of source and/or destination processes
- Example -- Cannon's Matrix-Matrix multiply (prog6-2.c)

Overlapping Comm with Comp

MPI_send/recv -- blocking

many real machines have smart comm hardware

- MPI_Isend -- Start a send, don't block
- MPI_Irecv -- Start a receive, don't block
- MPI_Test -- Test if I/O done
- MPI_Wait -- Wait for I/O to complete

Other Comm operations:

- MPI_Barrier
- MPI_Bcast (all call, one is source ...)
- MPI_Reduce
  - max, min, sum, prod, land, band, lor, bor, lxor, bxor
  - maxloc, minloc (value, location) pairs
- MPI_Scan (similar op list)
- MPI_Gather, MPI_Scatter
- MPI_Alltoall

MPI I/O ??

- Node 0, all others send to 0 for output, 0 sends to others for input
- Common file system (NFS), open file, lseek ...
- Collection of MPI_File_* routines, primarily disk files
  - can be sequential
  - can be concurrent, set a "view" that is part of a file
- Large collection of MPI file routines (not in books)
- http://www.open-mpi.org/doc/V4.1
- http://www.mpich.org/documentation/guides/

Other

- MPI_Wtime
  - time since an arbitrary point
  - p1 = MPI_Wtime(); // returns double, in seconds.
  - computation_you_want_to_time();
  - time = MPI_Wtime()-p1;

Groups and Communicators

- May want sub-groups of MPI_COMM_WORLD

- MPI_Comm_split(comm, groupid, orderid, *newcomm)
  - groupid (aka color) -- groups with same groupid
  - orderid (aka key) -- ties broken by previous rank

- MPI_Cart_sub(comm, keep_dims[], *subcomm)
  - keep_dims[i] says to keep that dimension in subcomm

MPI debugging ...

- printf(3)s ...
- gdb with the attach command and infinite loops
  - int proceed = 0; while (!proceed);
- attach to process and "set var proceed=1"