Parallel Computing with MPI

Lawrence Murray
Outline

- Motivation
- The Message Passing Interface (MPI)
- MPI implementations
- Computing resources
- Programming with MPI
Example: particle filtering

\[
\begin{align*}
\mathbf{x}(t_0) &\rightarrow f(\mathbf{x}, \mathbf{w}, \theta) &\rightarrow \mathbf{x}(t_1) &\rightarrow \mathbf{x}(t_2) &\rightarrow \cdots \\
&\downarrow \quad u(t_1) \\
&\downarrow \quad \mathbf{u}(t_2) \\
y(t_1) &\downarrow \quad \mathbf{y}(t_1) &\rightarrow \mathbf{y}(t_2) &\rightarrow \cdots \\
&\downarrow \quad g(\mathbf{x}, \mathbf{v}, \theta) \\
&\downarrow \quad \mathbf{y}(t_2)
\end{align*}
\]
Motivation

- Synchronization
- Load balancing
- Resource management
- ...in all cases communication between nodes is required.
Message Passing Interface (MPI)

- Synchronous communication
  - MPI_Send, MPI_Recv

- Asynchronous communication
  - MPI_Isend, MPI_Irecv

- Collective communication
  - MPI_Broadcast, MPI_Gather, MPI_Reduce
MPI implementations

- MPICH www-unix.mcs.anl.gov/mpi/mpich/
- LAM/MPI www.lam-mpi.org
- Open MPI www.open-mpi.org
- MVAPICH2 mvapich.cse.ohio-state.edu/overview/mvapich2/

- The MPI interface is standard, so programming in all is identical.
- Booting and terminating nodes, and running jobs on the system, differs slightly.
- MPICH and LAM/MPI are now in maintenance mode, having merged to form the Open MPI project.
- Recommend Open MPI.
## MPI implementation support

<table>
<thead>
<tr>
<th></th>
<th>DICE FC 5</th>
<th>DICE FC 6</th>
<th>Eddie</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPICH</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LAM/MPI</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Open MPI</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MVAPICH2</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
</tbody>
</table>

- ANC compute servers still running FC 5.
- LAM configuration changed from FC 5 to FC 6.
### Resources

<table>
<thead>
<tr>
<th></th>
<th>MPI</th>
<th>Dedicated</th>
<th>Hardware Homogeneity</th>
<th>Resource Reservation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condor pool</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>Compute servers</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Clusters</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
</tr>
<tr>
<td>Eddie</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

- Condor can run MPI jobs using the *parallel* universe, however, these can only be run on *dedicated* nodes. No nodes are configured as such.
Programming with MPI

- Single program, multiple data (SPMD)
  - Easy to program, easy to maintain.
  - Only as fast as slowest node.
- Multiple program, multiple data (MPMD), e.g. Master-worker setup
  - Master program packages and distributes batch jobs.
  - Worker programs process batch jobs.
  - Provides natural load balancing on heterogeneous system – faster nodes get more jobs than slower nodes.
  - More work to program, difficult to maintain, especially communication protocols.
int main(int argc, char **argv)
{
  char idstr[32], buff[128];
  int size, rank, i;

  MPI_Status stat;
  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD,&size);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);

  if (rank == 0) {
    printf("We have %d processors\n", size);
    for(i = 1; i < size; i++) {
      sprintf(buff, "Hello %d", i);
      MPI_Send(buff, 128, MPI_CHAR, i, 0, MPI_COMM_WORLD);
    }
    for(i = 1; i < size; i++) {
      MPI_Recv(buff, 128, MPI_CHAR, i, 0, MPI_COMM_WORLD, &stat);
      printf("%s\n", buff);
    }
  }
  else {
    MPI_Recv(buff, 128, MPI_CHAR, 0, 0, MPI_COMM_WORLD, &stat);
    sprintf(idstr, "Processor %d ", rank);
    strcat(buff, idstr);
    strcat(buff, "reporting");
    MPI_Send(buff, 128, MPI_CHAR, 0, 0, MPI_COMM_WORLD);
  }

  MPI_Finalize();
}
Programming in C++

- Can use the C interface, but a nice C++ wrapper now available in Boost [www.boost.org](http://www.boost.org)
Boost.MPI

- To be included for the first time in Boost v1.35, in the meantime can be obtained from CVS snapshot and installed locally.

- Advantage of Object Oriented interface...
Boost.Serialization

- ...but more importantly, supports object serialization with Boost.Serialization, which makes message passing much easier.
- Alleviates headaches of handling buffers and converting messages to primitive types.
- Can also make data storage and management a lot easier, serialize to files using the same interface.
Boost.uBLAS

- Supports serialization of vectors and matrices using Boost.Serialization, also in the CVS snapshot.
- Bindings to LAPACK etc.
int main(int argc, char **argv) {
    std::string str;
    std::stringstream buff;
    int size, rank, i;

    mpi::environment env(argc, argv);
    mpi::communicator world;
    size = world.size();
    rank = world.rank();

    if (rank == 0) {
        std::cout << "We have " << size << " processors" << std::endl;
        for(i = 1; i < size; i++) {
            buff << "Hello " << i;
            world.send(i, 0, buff.str());
        }
        for(i = 1; i < size; i++) {
            world.recv(mpi::any_source, mpi::any_tag, str);
            std::cout << str;
        }
    } else {
        world.recv(0, mpi::any_tag, stat);
        buff << "Processor " << rank << " reporting";
        world.send(0, 0, buff.str());
    }
}