MPI

Henrique Fingler - 11/18/2015
CS 377P
MPI

- Message Passing Interface
- MPI is a standard
- http://www.mpi-forum.org/
● Send and receive? Why not sockets?
MPI

- Send and receive? Why not sockets?
- Much more complex auto setting up and functions (spoiler: it does reductions)
MPI

- Defines tools for HPC programming
- Some abstractions
- Low and High level functions
MPI

- **SPMD**
- **Single Program**
- **Multiple Data**
- **Each process runs a copy**
Open MPI

- Open Source implementation of MPI
- Maintained by academia, research and industry
- You can do whatever you want with it*
Open MPI

- Originally developed for C and Fortran
- Now with C++ and Java
- There are wrappers, like mpi4py
Why use it?
Why use MPI

- relatively easy to use
- plenty of help
- mature (<1994)
- most clusters support it
- people and industry use it
Why use MPI

- but...
Going into MPI
Necessary concepts

- Communicators a.k.a. groups
- `MPI_COMM_WORLD`
- Rank a.k.a. id or uid
- blocking and non-blocking (there’s also synchronous and buffered)
● Following code was copied from
● https://computing.llnl.gov/tutorials/mpi/
#include "mpi.h"

... 

int numtasks, rank, dest, source, rc, count, tag=1;
char inmsg, outmsg='x';
MPI_Status Stat;

MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
if (rank == 0) {
    dest = 1;
    source = 1;
    rc = MPI_Send(&outmsg, 1, MPI_CHAR, dest, tag, MPI_COMM_WORLD);
    rc = MPI_Recv(&inmsg, 1, MPI_CHAR, source, tag, MPI_COMM_WORLD, &Stat);
}
else if (rank == 1) {
    dest = 0;
    source = 0;
    rc = MPI_Recv(&inmsg, 1, MPI_CHAR, source, tag, MPI_COMM_WORLD, &Stat);
    rc = MPI_Send(&outmsg, 1, MPI_CHAR, dest, tag, MPI_COMM_WORLD);
}

MPI_Finalize();
That’s it?
Using Open MPI

- Yep, except..
Using Open MPI

- Yep, except..
- You don’t know how to compile and run it
Compiling and running

- C: mpicc
- C++: mpiCC, mpicxx or mpic++
- mpirun -np 4 my_mpi_program
- mpirun --hostfile my_hosts -np 4 my_mpi_program
The hostfile

foo.example.com

bar.example.com slots=2

quad.example.com slots=4 max-slots=4
● You probably know or can easily guess at this point..
● But why does UTCS asks you to never shut down a machine?
Condor job scheduling / Mastodon

- “In addition to a large number of dedicated compute nodes, the cluster also makes use of idle desktop machines.”
- “Mastodon is available to the faculty, staff, and students of the department for research and educational purposes”
“... but Condor also supports parallel jobs using MPI”
Using Open MPI

- init
- rank
- send
- receive
More useful stuff

- Collective communication
- broadcast, scatter, gather, reduction
- synchronization (barriers)
- figures shamelessly copied from https://computing.llnl.gov/tutorials/mpi/#Collective_Communication_Routines
**MPI_Bcast**

Broadcasts a message from one task to all other tasks in communicator

```c
count = 1;
source = 1;
MPI_Bcast(&msg, count, MPI_INT, source, MPI_COMM_WORLD);
```

<table>
<thead>
<tr>
<th>task0</th>
<th>task1</th>
<th>task2</th>
<th>task3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

msg (before)

| 7     | 7     | 7     | 7     |

msg (after)
MPI_Scatter

Sends data from one task to all other tasks in communicator

sendcnt = 1;
recvcnt = 1;
src = 1;

MPI_Scatter(sendbuf, sendcnt, MPI_INT
recvbuf, recvcnt, MPI_INT
src, MPI_COMM_WORLD);

task1 contains the data to be scattered

sendbuf (before)  recvbuf (after)
**MPI_Gather**

Gathers data from all tasks in communicator to a single task

```
sendcnt = 1;
recvcnt = 1;
src = 1;
MPI_Gather(sndbuf, sendcnt, MPI_INT,
            rcvbuf, recvcnt, MPI_INT,
            src, MPI_COMM_WORLD);
```

<table>
<thead>
<tr>
<th>task0</th>
<th>task1</th>
<th>task2</th>
<th>task3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

- `sendbuf` (before)
- `recvbuf` (after)
MPI_Allgather

Gathers data from all tasks and then distributes to all tasks in communicator

```c
sendcnt = 1;
recvcnt = 1;
MPI_Allgather(sendbuf, sendcnt, MPI_INT
               recvbuf, recvcnt, MPI_INT
               MPI_COMM_WORLD);
```

<table>
<thead>
<tr>
<th>task0</th>
<th>task1</th>
<th>task2</th>
<th>task3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

sendbuf (before): 1 2 3 4
recvbuf (after):   1 1 2 2

1 1 3 3
2 2 3 3
4 4 4 4
MPI_Reduce

Perform reduction across all tasks in communicator and store result in 1 task

count = 1;
dest = 1;

MPI_Reduce(sendbuf, recvbuf, count, MPI_INT,
            MPI_SUM, dest, MPI_COMM_WORLD);

<table>
<thead>
<tr>
<th>task0</th>
<th>task1</th>
<th>task2</th>
<th>task3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

sendbuf (before)

<table>
<thead>
<tr>
<th>task0</th>
<th>task1</th>
<th>task2</th>
<th>task3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

recvbuf (after)
<table>
<thead>
<tr>
<th>MPI Reduction Operation</th>
<th>C Data Types</th>
<th>Fortran Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_MAX</td>
<td>maximum</td>
<td>integer, real, complex</td>
</tr>
<tr>
<td>MPI_MIN</td>
<td>minimum</td>
<td>integer, real, complex</td>
</tr>
<tr>
<td>MPI_SUM</td>
<td>sum</td>
<td>integer, real, complex</td>
</tr>
<tr>
<td>MPI_PROD</td>
<td>product</td>
<td>integer, real, complex</td>
</tr>
<tr>
<td>MPI_LAND</td>
<td>logical AND</td>
<td>integer</td>
</tr>
<tr>
<td>MPI_BAND</td>
<td>bit-wise AND</td>
<td>integer, MPI_BYTE</td>
</tr>
<tr>
<td>MPI_LOR</td>
<td>logical OR</td>
<td>integer</td>
</tr>
<tr>
<td>MPI_BOR</td>
<td>bit-wise OR</td>
<td>integer, MPI_BYTE</td>
</tr>
<tr>
<td>MPI_LXOR</td>
<td>logical XOR</td>
<td>integer</td>
</tr>
<tr>
<td>MPI_BXOR</td>
<td>bit-wise XOR</td>
<td>integer, MPI_BYTE</td>
</tr>
<tr>
<td>MPI_MAXLOC</td>
<td>max value and location</td>
<td>float, double and long double</td>
</tr>
<tr>
<td>MPI_MINLOC</td>
<td>min value and location</td>
<td>float, double and long double</td>
</tr>
</tbody>
</table>
Barrier
More useful stuff

- Allreduce
- Reduce_scatter
- Alltoall
- Scan (?)
MPI_Scan

Computes the scan (partial reductions) across all tasks in communicator

count = 1;
MPI_Scan(sendbuf, recvbuf, count, MPI_INT, MPI_SUM, MPI_COMM_WORLD);

<table>
<thead>
<tr>
<th>task0</th>
<th>task1</th>
<th>task2</th>
<th>task3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>sendbuf (before)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>task0</th>
<th>task1</th>
<th>task2</th>
<th>task3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>recvbuf (after)</td>
</tr>
</tbody>
</table>
#include "mpi.h"

#define SIZE 4

main(int argc, char *argv[]) {
    int numtasks, rank, sendcount, recvcount, source;
    float sendbuf[SIZE][SIZE] = {
        {1.0, 2.0, 3.0, 4.0},
        {5.0, 6.0, 7.0, 8.0},
        {9.0, 10.0, 11.0, 12.0},
        {13.0, 14.0, 15.0, 16.0}  
    };
    float recvbuf[SIZE];
MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);

if (numtasks == SIZE) {
    ...
}
else
    printf("Must specify %d processors. Terminating.\n",SIZE);
source = 1;
sendcount = SIZE;
recvcount = SIZE;

MPI_Scatter(sendbuf, sendcount, MPI_FLOAT, recvbuf, recvcount, MPI_FLOAT, source, MPI_COMM_WORLD);

printf("rank= %d Results: %f %f %f %f\n",rank,recvbuf[0], recvbuf[1],recvbuf[2],recvbuf[3]);
Even more useful stuff

- Numbering of async requests
- So you can check progress
- Wait, Waitall, Waitany, Waitsome
- Test, Testall, Testany, Testsome
Even more useful stuff

MPI_Irecv(&buf[0], 1, MPI_INT, prev, tag1, MPI_COMM_WORLD, &reqs[0]);

...

MPI_Waitall(4, reqs, stats);
Even more useful stuff

● You can also tag a message
● network may not be FIFO
Some other stuff

- Shared Memory operations (put,get)
- MPI-3: non-blocking collective
- MPI-3: MPI Tool Interface: exposes internal variables/counters for performance tools
Some other stuff

- Parallel IO
- File view: portion of a file visible to one process
- Can overlap, be disjoint, change overtime and have multiple views
Process spawning

- MPI_Comm_spawn
- OK!
Thread support

- Meh..
- MPI_Init_thread
Thread support

● Might support multiple threads calling MPI concurrently
  ● “Note that MPI_THREAD_MULTIPLE support is only lightly tested.”
  ● Note that Open MPI’s thread support is in a fairly early stage; the above devices are likely to work, but the latency is likely to be fairly high. Specifically, efforts so far have concentrated on correctness, not performance (yet).
So.. do people use MPI?
Applications

- Weather prediction
- Bioinformatics
- Cardiac simulation
- Finance
- Oil and gas exploration
Applications

- Molecular dynamics
- Physical Simulation
- Bio-tech, nano-tech
- Drug design
Chances are they use MPI
Applications

- [http://www.ansys.com/](http://www.ansys.com/)
- or google ansys mpi
The bad part
The C/Python analogy
Why not MPI

- Online top notch rant/critique:
- “HPC is dying, and MPI is killing it”
- Don’t take it as 100% the truth
Pictured: The HPC community bravely holds off the incoming tide of new technologies and applications. Via the BBC.
Why not MPI

- Large scale problems were not that common
- Not many people working on it
Why not MPI

- Internet-scale data
- Genomics data
Why not MPI

- A flood of people started researching and using HPC
- Suddenly: scalability, accuracy, large data storage, distributed matrix arithmetic
- Mainstream and urgent
Easy for the ol’ MPI people, right?
Why?
Other ideas

- Creating their own solution
- Old ideas
- Stuff that wasn’t that good
- Stubbornness: other people are doing it wrong, the “one true path”
Meanwhile, “they” are solving problems
Other ideas

- As in C vs. Python…
- Different level of abstraction
- How fast/low do you NEED
- No real fault-tolerance, although work is being done
1D Diffusion equation

<table>
<thead>
<tr>
<th>Framework</th>
<th>Lines</th>
<th>Lines of Boilerplate</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI+Python</td>
<td>52</td>
<td>20+</td>
</tr>
<tr>
<td>Spark+Python</td>
<td>28</td>
<td>2</td>
</tr>
<tr>
<td>Chapel</td>
<td>20</td>
<td>1</td>
</tr>
</tbody>
</table>
Other ideas

- As in C vs. Python...
- Different level of abstraction
- No real fault-tolerance, although work is being done
Back to the good MPI
IBM MPI

- Platform MPI
- Supports CUDA memory Unified Virtual Addressing
MVAPICH

- Ohio University
- Another MPI implementation
- Also has GPU support
GUESS WHAT
<table>
<thead>
<tr>
<th>Specification</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Site</td>
<td>Texas Advanced Computing Center/Univ. of Texas</td>
</tr>
<tr>
<td>System URL</td>
<td><a href="http://www.tacc.utexas.edu/stampede">http://www.tacc.utexas.edu/stampede</a></td>
</tr>
<tr>
<td>Manufacturer</td>
<td>Dell</td>
</tr>
<tr>
<td>Cores</td>
<td>462,462</td>
</tr>
<tr>
<td>Linpack Performance (Rmax)</td>
<td>5,168.11 TFlop/s</td>
</tr>
<tr>
<td>Theoretical Peak (Rpeak)</td>
<td>8,520.11 TFlop/s</td>
</tr>
<tr>
<td>Nmax</td>
<td>3,875,000</td>
</tr>
<tr>
<td>Power</td>
<td>4,510.00 kW</td>
</tr>
<tr>
<td>Memory</td>
<td>192,192 GB</td>
</tr>
<tr>
<td>Processor</td>
<td>Xeon E5-2680 8C 2.7GHz</td>
</tr>
<tr>
<td>Interconnect</td>
<td>Infiniband FDR</td>
</tr>
<tr>
<td>Operating System</td>
<td>Linux</td>
</tr>
<tr>
<td>Compiler</td>
<td>Intel</td>
</tr>
<tr>
<td>Math Library</td>
<td>MKL</td>
</tr>
<tr>
<td>MPI</td>
<td>MVAPICH2</td>
</tr>
</tbody>
</table>
Blue Gene also uses MPI (own implementation) 3rd on TOP500

So, even with the same hate, people still use MPI?

http://mvapich.cse.ohio-state.edu/users/
Still going

- For extreme performance, combine
- MPI/OpenMP
- Even CUDA/OpenCL