ECE 574 – Cluster Computing
Lecture 12

Vince Weaver

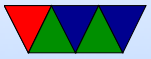
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

28 February 2019

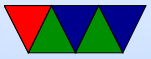
Announcements

- HW#4 Graded (sorry for delay)
- HW#6 will be posted
- Q for last time: yes, MPI will handle endianess on heterogeneous cluster



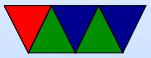
Project

- Project handout posted to the course website
- No action really needed until March 28th (topic selection)
- You can get a head start if you want



Midterm on 7 March 2019

- Be sure to know Speedup/Parallel Efficiency
- Know difference between shared mem and distributed system
- Know about pthreads and locking
- Know about OpenMP
- Know about MPI



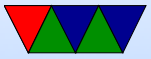
MPI continued

Some references

`https://computing.llnl.gov/tutorials/mpi/`

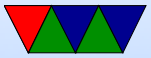
`http://moss.csc.ncsu.edu/~mueller/cluster/mpi.guide.pdf`

`https://cvw.cac.cornell.edu/MPIcc/default`



Efficient way of getting data to all processes

- master send to each individual, take a while
- some sort of tree, 0 to 1 and 2, 1 sends to 3 and 4, etc.
- use broadcast instead

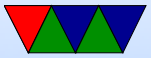


Collective Communication

- All must participate or there can be problems.
- Do not take tag arguments
- Can only operate on MPI defined data types, not custom
- Operations
  - Synchronization – all processes wait
  - Data Movement – broadcast, scatter-gather

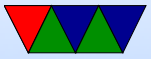
    scatter = take one structure and split among processes

    gather = take data from all processes and combine it
  - Reduction – one process combines results of all others



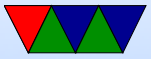
MPI_Barrier()

- All processes wait at this point.
- `MPI_Barrier (comm)`



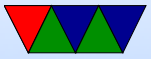
MPI_Bcast()

- `MPI_Bcast (&buffer,count,datatype,root,comm)`
- Sends data from the *root* process to each other process.
- Is blocking; when encountering a Bcast all nodes wait until they have received the data.
- There is no need to receive; the root sends the data and all other ranks will receive, just with the one command



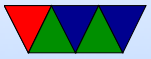
MPI_Scatter() / MPI_Gather()

- `MPI_Scatter (&sendbuf,sendcnt,sendtype,&recvbuf recvcnt,recvtype,root,comm)`
- Copies `sendcnt` sized chunks of sendbuf to each processes `recvbuf`
- `MPI_Gather (&sendbuf,sendcnt,sendtype,&recvbuf, recvcount,recvtype,root,comm)`
- Have to take care if area sending not a multiple of your number of ranks

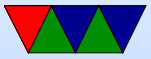


MPI_Reduce()

- `MPI_Reduce( void* send_data, void* recv_data, int count, MPI_Datatype datatype, MPI_Op op, in root, MPI_Comm communicator)`
- Operations
  - MPI_MAX,MPI_MIN – max, min
  - MPI_SUM – sum
  - MPI_PROD – product
  - MPI_LAND, MPI_BAND – logical/bitwise and
  - MPI_LOR,MPI_BOR – logical/bitwise OR



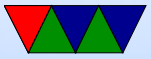
- MPI_LXOR,MPI_BXOR – logical/bitwise XOR
- MPI_MAXLOC,MPI_MINLOC – value and location
- Can also create custom



MPI_Allgather()

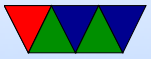
Gathers, to all.

Equivalent of gathering back to root, then rebroadcasting to all.

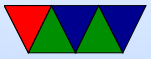


MPI_Allreduce()

- Like an MPI_Reduce followed by an MPI_Bcast
- `MPI_Allreduce( void* send_data, void* recv_data, int count, MPI_Datatype datatype, MPI_Op op, MPI communicator)`
- Once the reduction is done, broadcasts the results to all processes

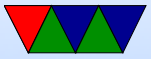


MPI_Reduce_scatter()



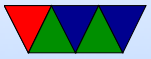
MPI_Alltoall()

Scatter data from all to all



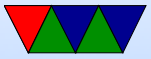
MPI_Scatterv()

Vector scatter. Send non-contiguous chunks. In addition to regular scatter parameters, a list of start offsets and lengths.



MPI_Scan()

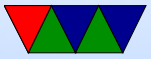
Lets you do partial reductions.



Custom Data Types

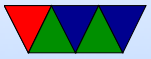
You can create custom data types that aren't the MPI default, sort of like structures.

Open question: can you just cast your data into integers and uncast on the other side?



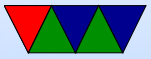
Groups vs Communicators

Can create custom groups if you don't want to broadcast to all.



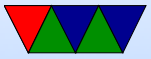
Virtual Topologies

- Map to a geometric shape (grid or graph)
- Doesn't have to match underlying hardware



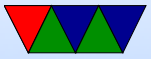
Examples

See the provided tar file with example code.



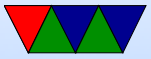
Running MPI code

- `mpiexec -np 4 ./mpi_test`
- You'll often see `mpirun` instead. Some implementations have that, but it's not the official standard way.

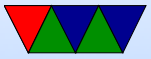


Send Example

- mpi_send.c
- Run with `mpirun -np 4 ./mpi_send`
- Sends 1 million integers (each with value of 1) to each node
- Each adds up 1/4th then sends only the sum (a single int) back
- Notice this is a lot like pthreads where we have to do a

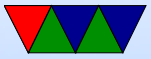


lot of work manually.



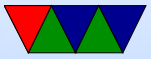
Blocking vs NonBlock Example?

TODO



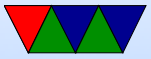
Wtime Example

- `mpi_wtime.c`
- Same as previous example. but with timing
- Unlike PAPI, the time is returned as a floating point value



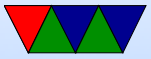
Barrier Example

- `mpi_barrier.c`
- Each machine sleeps some time based on rank
- All wait at barrier until last one arrives



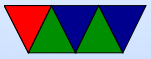
Bcast Example

- `mpi_bcast.c`
- Same buffer on each machine
- At the broadcast function, one sends its version of the buffer and the rest wait until they receive the value.
- In the end they all have the same value



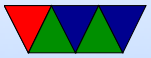
Scatter Example

- `mpi_scatter.c`
- Instead of sending all of A, breaks it into chunks and sends it to B in each rank.



Gather Example

- `mpi_gather.c`
- Each rank has its own copy of A which it sets to entirely its rank number
- Then a gather happens on rank0, of one int each. So what should B have in it? (0, 1, 2, 3, ...)



Reduce Example

- `mpi_reduce.c`
- Instead of waiting in a loop for tasks finishing and then adding up the results one by one, use a reduction instead.
- Many MPI routines are convenience things that could be done by a sequence of separate commands.