ECE 574 – Cluster Computing
Lecture 12

Vince Weaver

`https://web.eece.maine.edu/~vweaver`

`vincent.weaver@maine.edu`

23 February 2023

Announcements

- Midterm Next Thursday (March 2nd)
  More details/review in class Tuesday
- HW#4 Grades will be out soon
- HW#6 will be posted
- Q from last time, what happens if sent data corrupted? MPI has no extra support for that, it relies on underlying network to catch that. TCP/IP and ethernet have checksums/CRC. It's possibly more likely your system RAM lacks ECC and data will get corrupted there rather

than on the network.

HW#4 Review

- Low-level C is a pain. Things like passing pointers to double-indexed arrays, and `(void *)` casting.
  I'd like to say you'll never see this, but if you ever get a job doing Linux kernel or similar low level work there's a lot of this that goes on.
- Hopefully you'll find OpenMP is a lot simpler.
- Some results on a 10848x10824 NASA image I found:

bench	Load	convolve	combine	store
before	945,172	20,972,969	1,740,545	865,404
coarse(2)	952,647	10,752,946	1,785,945	882,353
fine 1	960,527	10,582,954	12,303,506	921,339
fine 2		5,418,575	6,255,203	
fine 8	935,998	1,491,921	3,574,811	928,533
fine 16		729,125	2,097,431	
fine 32		627,906	714,431	

- Should see some speedup, even if not perfect.
  Be sure your joins are *after* both threads started.
- Max speedup? Below, significant time in load/store

combine so even if perfect convolution...

```
Load time: 98257
Convolve time: 871411
Combine time: 266956
Store time: 107583
```

- Question: was an example of deadlock.

MPI continued

Some references

`https://hpc-tutorials.llnl.gov/mpi/`

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

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

How to send data efficiently to all ranks?

- Rank 0 could send to each individual, take a while
- Some sort of tree, 0 to 1 and 2, 1 sends to 3 and 4, etc.
- Can we 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

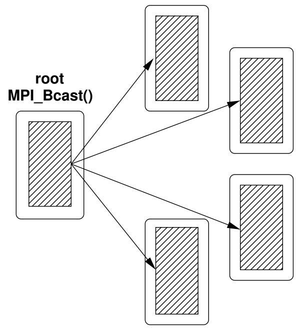
    scatter = take one structure and split among processes

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

MPI_Barrier()

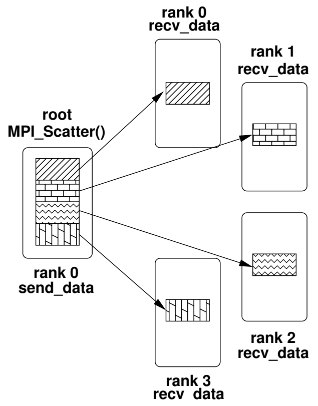
- All processes wait at this point.
- `MPI_Barrier (comm)`

MPI_Bcast()



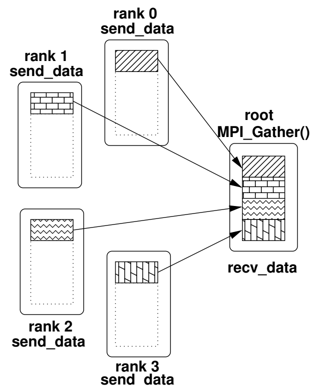
- `MPI_Bcast (&buffer,count,datatype,root,comm);`
- Sends data from the *root* rank to each other rank.
- Is blocking; when encountering a Bcast all nodes wait until they have received the data.
- There is no need to receive; the root sends the data and all other ranks will receive, just with the one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

MPI_Scatter()



- `MPI_Scatter (&send_data,sendcnt,sendtype,&recv_data, recvcnt,recvtype,root,comm);`
- Copies `sendcnt` sized chunks of `sendbuf` to each rank's `recvbuf`
- root also gets a share of data (just a local copy)

MPI_Gather()



- `MPI_Gather (&send_data,sendcnt,sendtype,&recv_data, recvcount,recvtype,root,comm);`
- Copies `sendcnt` sized chunks of sendbuf from each rank to `recvbuf` in root, offset by recvcount for full result
- **NOTE** values start at beginning of each rank's sendbuf

Scatter/Gather Boundary issues

- *NOTE* If the size of the data you are sending is not an even multiple of the number of ranks you'll have to manually handle the extra
- How?
  - Have the root manually handle the extra at end?
  - Pad your data to be a multiple of number of ranks and ignore the extra?
  - `MPI_Scatterv()` and `MPI_Gatherv()` routines let you send vectors (chunks of varying length) but complex to use

MPI_Reduce()

-
  ```
  MPI_Reduce(void* send_data, void* recv\_data,
        int count, MPI_Datatype datatype, MPI_Op op,
        int 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()

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

MPI_Reduce_scatter()

- Does a reduction, then scatters the results

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? This is not recommended and might have issues on a heterogeneous cluster

Groups vs Communicators

- Can create custom groups if you don't want to broadcast to all.
- Use groups to create Communicators, then can use instead of WORLD

Virtual Topologies

- Your workload might map to a geometric shape (grid or graph)
- In a mesh type problem you might only want to talk to the 4 surrounding ranks and none of the others, so might be handy if can be placed in hardware to take advantage of that
- Doesn't have to match underlying hardware

Examples

See the provided tar file with example code.

Running MPI code

- `mpiexec -np 4 ./mpi_test`
  Runs on 4 ranks
  note the space between np and 4 is important and things won't work if you leave it out

- 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 `mpiexec -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.
- Things to note:

- `MPI_Init()` at start
  passes command line args, on most implementations this will essentially broadcast the command line args across all ranks so
- `MPI_Comm_size()` to get number of ranks
- `MPI_Comm_rank()` to get our rank
- `MPI_Send()` in this case only from rank 0
- `MPI_Recv()` can use status value to get size, source, and tag

Blocking vs NonBlock Example?

- `mpi_nonblock.c`

Wtime (Wallclock Time) 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
- Note: seeing all printfs because in this case all ranks on same machine. This might not happen when running on a real cluster

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.
- Note that while the program runs ordered as expected, the printfs might not reflect this

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.

HW#6 Preview

- Suggested coarse implementation
  - Get rank and size
  - Load the jpeg. Only in Rank0. Could you load it in all? Why or why not?
  - Need to tell other processes the size of our images. image.xsize, image.ysize, image.depth. Why? So can allocate proper sized structures on each.
  - How can do this? Just send 3 integers. Could set up custom struct but not worth it. How send this array of

3 vars? Set up array. Bcast it? Send/receive to each, one at a time? Which is most efficient?

- Allocate space for the output images

```
new_image.pixels=malloc(image.x*image.y*
                        image.depth*sizeof(char));
sobel_x.pixels
sobel_y.pixels
```

- Use MPI_Bcast to broadcast image data from rank0 to other ranks. Note that Bcast acts as a send from the root source (usually root 0) but as a receive on all other ranks (there's no need to separately have the other ranks receive)

```
result = MPI_Bcast(image.pixels,        /* buffer */
          image.x*image.y*image.depth,/* count   */
          MPI_CHAR,                   /* type    */
          0,                       /* root source  */
          MPI_COMM_WORLD);
```

- Split up the work, you know your rank and total, so if 4 and you are #2, then you should calculate for X/4, so 0..(X/4-1), (x/4)..(x/4*2-1), etc. How to handle non-even multiple? Last rank should calc extra
- Once it is done, send back. How? MPI_Gather();

```
MPI_Gather(new_image.pixels,     /* source buffer */
    sobel_x.depth*sobel_x.x*
                (sobel_x.y/numtasks),    /* count */
    MPI_CHAR,                            /* type */
    sobel_x.pixels,            /* receive buffer */
    sobel_x.depth*sobel_x.x*
                (sobel_x.y/numtasks),   /* count */
    MPI_CHAR,                            /* type */
    0,                            /* root source */
    MPI_COMM_WORLD);
```

Note, it gathers from the beginning of the buffer, but put it in the right place on the root. Also, how to handle the leftover bit?

- Suggest you just do combine in rank#0, will in next HW do more fine grained
- Write out result. Remember to only write out on rank#0 (what happens if do so on all?)

Additional notes on MPI

- Hard to think about. Running on different machine, so setting variables *does not* get set on all, like it does with OpenMP or pthreads

- Tricky: before you can send to rest, they have to know how big of an area to allocate to store it in. How will they know this?

- MPI does not give good error messages. OpenMPI worse than MPICH. Will often get segfault, hang forever, or weird stuff where it runs 4 single-threaded copies of program rather than one 4-threaded

- Many of the commands are a bit non-intuitive

MPI Debugging (HW#6) notes

- MPI is *not* shared memory
- Picture having 4 nodes, each running a copy of your program *without* MPI.

  Also picture the various MPI routines as a network socket (or web browser query).

  Things initialized the same in all will have same values, no need to initialize.

  Things initialized in only one node will need to be somehow broadcast for the values to be the same in all.

- Problems debugging memory issues.
  Valgrind should work, but Debian compiles MPI with checkpoint support which breaks Valgrind :(
  Mpirun supposed to have -gdb option, doesn't seem to work.
- What does work is `mpiexec -n num xterm -e gdb ./your_app` but this depends on you running X11 plus logging into Haswell-EP with X forwarding (-Y) enabled
- The bug most people hit is improper bounds, leading to segfault. You can debug that with printfs of your bounds
- MPI does give useful error messages sometimes

- Some of the problem is malloc/calloc

Other MPI Notes

- `MPI_Gather( sendarray, 100, MPI_INT, rbuf, 100, MPI_INT, root, comm);`
  rbuf ignored on all but root
- All collective ops are blocking by default, so you don't need an implicit barrier
- MPI_Gather(), same as if each process did an MPI_Send() and the root note did in a loop MPI_Receive() incrementing the offset.

- `MPI_Gather()` aliasing
  cannot gather into same pointer, will get an aliasing error
  Can use `MPI_IN_PLACE` instead of the send buffer on rank0.
  Why is this an error? Partly because you cannot alias in Fortran. Just avoids potential memory copying errors.
  What happens if your gathers overlap?
- Can you handle non-even buffer sizes with MPI_Gather?
  No. Two options.
  - One, just handle in one of other threads (either master

or send/receive from other)

- Two, use `MPI_Gatherv()` where you specify the displacement and sizes of what you want to gather