ECE 574 – Cluster Computing
Lecture 13

Vince Weaver

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

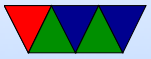
vincent.weaver@maine.edu

11am Barrows 133

27 February 2024

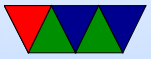
Announcements

- HW#4, HW#5: still grading
- HW#6 will be posted, will be due March 8th
- Bring projector

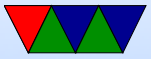


HW#4 Review

- Difficult part was picking bounds
- Other difficult part was avoiding race condition when passing parameters to the threads
  - Usual way is to just allocate a parameter struct for each thread
  - Can do complex locking, though that's not optimal
  - Please don't try to use usleep() to try to avoid the race condition by playing with the timings. This is extremely fragile and not a good idea.

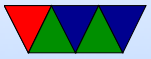


- The example code problem is an example where deadlock can occur

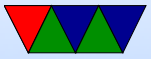


Midterm on 29 February 2024

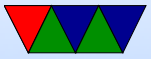
- Can bring one page (8.5" by 11" one sided) of notes. Otherwise closed notes, computers, cell-phones, Beowulf cluster, etc.
- There is speedup question which requires some math Should be able to do it w/o a calculator, but you can have one if you must.
- Performance
  - Speedup, Parallel efficiency
  - Strong and Weak scaling



- Definition of Distributed vs Shared Memory
- Know why changing order of loops can make things faster
- Pthread Programming
  - Know about race condition, deadlock
  - Know roughly the layout of a pthreads program. (define pthread_t thread structures, pthread_create, pthread_join)
  - Know why you'd use a mutex.
- OpenMP Programming
  - parallel directive

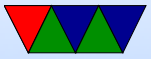


  - scope
  - section
  - for directive
- Know about MPI



HW#6 Guide – First Broadcast Image Data

- Get rank and num_ranks
- Load the jpeg, only in Rank0.
  Could you load it in all ranks? Why or why not?
- Need to tell other ranks the image data. `image.x`, `image.y`, `image.depth`. Why? So we can allocate proper sized image structures in each rank
- How can we 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 is probably the best way.

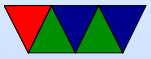


- Allocate space for the output images

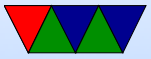
```
new_image.pixels=calloc(image.x*image.y*image.depth,sizeof(char));
sobel_x.pixels=calloc(image.x*image.y*image.depth,sizeof(char));
sobel_y.pixels=calloc(image.x*image.y*image.depth,sizeof(char));
```

- 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);
```

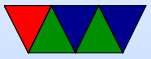


- Be sure to broadcast to `image.pixels`, not `image`, otherwise you'll overwrite the struct data
- It can be tricky getting this all working. I've added some simple checksum code so you can verify the right data is sent to all ranks before moving on to the next part



HW#6 Guide – Run the Convolution

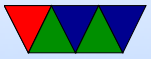
- You'll need to split up the work like we did with the pthread code
- Splitting it up at the row level is probably best
- You know your rank and total, so if 4 and you are #2, then you should calculate for X/4, so 0..(X/4-1), (x/4)..(x/4*2-1), etc.
- For now assume the result will be a multiple of the rank



HW#6 Guide – Gathering Back the Results

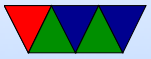
- Once your sobel is done, need to gather back to root
- Note: it's probably easiest to gather the results into a new image (so gather from sobelx.image into sobelx_new.image). Gathering from sobelx.image into sobelx.image is tricky
- MPI_Gather();

```
MPI_Gather(sobel_x.pixels,                  /* source buffer */
           sobel_x.depth*sobel_x.x*(sobel_x.y/num_ranks),   /* count */
           MPI_CHAR,                        /* type */
           sobel_x_new.pixels,              /* receive buffer */
           sobel_x.depth*sobel_x.x*(sobel_x.y/num_ranks),   /* count */
           MPI_CHAR,                        /* type */
           0,                               /* root source */
```

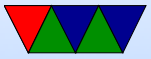


```
                MPI_COMM_WORLD);
```

- Note: gathers by default will gather from the start of an array, whereas your convolve code probably puts things at an offset (TODO: plot)
- There are three ways to do this once your convolve is done
  - Either do a `memcpy()` to move the results to the start of the array,
  - Alternately adjust your convolve routine to place the output at the start of the array (by subtracting y_start from your y_value),

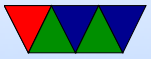


- Another way is you can change the source buffer to point to the proper offset in the data array sobel_x.pixels[rank*total_size/num-ranks]



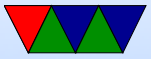
HW#6 Guide – Finishing Up

- After sobel_x, also do sobel_y
- For this homework just do combine step 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 this on all ranks?)



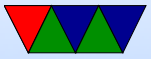
HW#6 Guide – Handling Non-Multiple Image Data

- Your best bet is using Gatherv() even though you have to go through the extra step of allocating the counts/offsets.



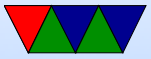
Additional notes on MPI

- 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



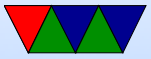
weird stuff where it runs 4 single-threaded copies of program rather than one 4-threaded

- Many of the commands are a bit non-intuitive

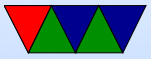


MPI Debugging (HW#6) notes

- 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