ECE 574 – Cluster Computing
Lecture 12

Vince Weaver

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

vincent.weaver@maine.edu

9 March 2021

Announcements

- HW#4, HW#5: still grading
- HW#6 – will be posted, will be due the 19th

Midterm on 11 March 2021

- Online, open book and notes
- Take it during class time. Let me know if have any conflicts.
- Easiest if you take it remotely. You can show up and take it in person, but in that case have laptop and such, won't be handing out paper copies.
- I will monitor zoom for any questions
- 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#5 Review

- Have to put “parallel” either in separate directive, or in sections.
- Also time measurement outside parallel area (time in each section is the same with or without threads, the difference is they can happen simultaneously). i.e. be sure to measure wall clock, not user, time
- Don’t nest parallel! remove sections stuff for fine.
- Also, does it makes sense to parallelize the most inner loop of 3?

- Also what if you mark variables private that shouldn't be? scope!
- Also if have sum marked private in inner loop, need to make sure it somehow gets added on the outer (reduction).
- Be careful with bracket placement. Don't need one for a for, for example.
- Also, remember as soon as you do parallel everything in the brackets runs on X threads. So if you parallel, have loops, then a for... those outer loops are each running X times so you're calculating everything X times over.

This isn't a race condition because we don't modify the inputs so it doesn't matter how many times we calc each output.

- Does dynamic vs static vs chunksize affect our code? 9 muls and adds should take consistent size. When might it not? Cache!

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.x, image.y, 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

Reliability in HPC

Good reference is a class I took a long time ago, CS717 at Cornell:

`http://greg.bronevetsky.com/CS717FA2004/Lectures.html`

Sources of Failure

- Software Failure
  - Buggy Code
  - System misconfiguration
- Hardware Failure
  - Loose wires
  - Tin whiskers (lead-free solder)
  - Lightning strike
  - Radiation
  - Moving parts wear out

- Malicious Failure
  - Hacker attack

Types of fault

- Permanent Faults – same input will always result in same failure
- Transient Faults – go away, temporary, harder to figure out

What do we do on faults?

- Detect and recover?
- Just fail?
- Can we still get correct results?

Metrics

- MTBF – mean time before failure
- FIT (failure in Time)

  One failure in billion hours. 1000 years MTBF is 114FIT. Zero error rate is 0FIT but infinite MTBF Designers just FIT because additive.
- Nines. Five nines 99.999% uptime (5.25 minutes of downtime a year)

  Four nines, 52 minutes. Six nines 31 seconds.
- Bathtub curve

Architectural Vulnerability factor

- Some bit flips matter less
- (branch predictor) others more (caches) some even more (PC)
- Parts of memory that have dead code, unused values

Things you can do Hardware

Hardware Replication

- Lock step – Have multiple machines / threads running same code in lock-step Check to see if results match. If not match, problem. If replicated a lot, vote, and say most correct is right result.
- RAID – (redundant array of inexpensive disks)
- Memory checksums – caches, busses
- Power conditioning, surge protection, backup generators, UPS

- Hot-swappable redundant hardware

Lower Level

- Replicate units (ALU, etc)
- Replicate threads or important data wires
- CRCs and parity checks on all busses, caches, and memories

Lower-Level Problems

Soft errors/Radiation

- Chips so small, that radiation can flip bits. Thermal and Power supply noise too.
- Soft errors – excess charge from radiation. Usually not permanent.
- Sometime called SEU (single event upset)

Radiation

- Neutrons: from cosmic rays, can cause ”silicon recoil” Can cause Boron (doped silicon) to fission into Li and alpha.
- Alpha particles: from radioactive decay
- Cosmic rays – higher up you are, more faults Denver 3-5x neutron flux than sea level. Denver more than here. Airplanes. Satellites and space probes are radiation-hardened due to this.

- Smaller devices, more likely can flip bit.

Shielding

- Neutrons: 3 feet concrete reduce flux by 50%
- alpha: sheet of paper can block, but problem comes from radioactivity in chips themselves

Case Studies

- "May and Woods Incident" first widely reported problem. Intel 2107 16k DRAM chips, problem traced to ceramics packaging downstream of Uranium mine.
- "Hera Problem" IBM having problem. ^{210}Po contamination from bottle cleaning equipment.
- "Sun e-cache" Ultra-SPARC-II did not have ECC on cache for performance reasons. High failure rate.

Hardware Fixes

- Using doping less susceptible to Boron fission
- Use low-radiation solder
- Silicon-on-Insulator
- Double-gate devices (two gates per transistor)
- Larger transistor sizes
- Circuits that handle glitches better.
- Memory fixes
  - ECC code
  - spread bits out. Right now can flip adjacent bits, flip

too many can't correct.

- Memory scrubbing: going through and periodically reading all mem to find bit flips.

Extreme Testing

- Single event upset characterization of the Pentium MMX and Pentium II microprocessors using proton irradiation", IEEE Transactions on Nuclear Science, 1999.
- Pentium II, took off-shelf chip and irradiated it with proton. Only CPU, rest shielded with lead. Irradiate from bottom to avoid heatsink
- Various errors, freeze to blue screen. no power glitches or "latchup" 85% hangs, 14% cache errors no ALU or

FPU errors detected.