ECE 574 – Cluster Computing
Lecture 15

Vince Weaver

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

`vincent.weaver@maine.edu`

11am, Barrows 133

7 March 2024

Announcements

- HW#5 was graded
- Don't forget HW#6. Extended to Monday
- HW#7 is going to follow immediately after break (basically, fix HW#6 and then run on Pi cluster)
  can't extend that too much as we are going to move on to CUDA/GPU
- Be careful deleting files on Haswell-EP, it's not backed up

Remember Project Topics Due

- Send e-mail with topic and group members by March 21st (Thurs)
- Can work alone or in groups of 2 to 3
- Do something interesting parallel computing related
- Can use any operating system and written in any language (asm, C, python, C++, Java, etc.)
- Coding, benchmarking
- Past projects: SIMD, parallelizing code, comparison of C vs Python, parallelizing matlab code, calculating physical

constants in parallel, building own cluster, raytracing, GPU shader coding

- Will be a final writeup, and then a 10 minute presentation and demo in front of the class during last week of classes.

Not Quite done Grading Midterms Yet

Feedback on HW#5 – Results

- Results are fairly straightforward so won't put them here Lots of weird corner cases, ask why it doesn't linear scale? Dunno? Can you track it down? Sure, perf, look at source code, etc, but it's not easy
- Also should be running on an idle system and this isn't
- Why did I have you print load/store time? Amdahl's law. Reduces overall potential speedup. Why does it vary so much? Not sure

Feedback on HW#5

- Be sure your code compiles, and that it doesn't crash
- OpenMP is supposed to be about taking existing correct linear code and dropping some pragmas in to make it parallel. You shouldn't have to majorly re-write your code.
    - ◦ If you are checking your thread-id and doing things based on it, it's probably not doing things properly
    - ◦ Will low-level messing about always work? (If you have 8 threads and you ask for 2, are you guaranteed

they are 0 and 1?)

- If the homework says use sections, use sections. Don't use serial/tasks, or open-code your own sections implementation.
- Don't use nowait unless you know what you are doing
  - Great place for a code comment
  - "Because it crashes otherwise" is not a good reason
- Use of private vars.
  - Possibly loop indices are always treated as private?
  - Not sure how some of the solutions worked without declaring sum to be private.

  - Not sure if it's compiler optimizations, or just luck
- Where you put your for (before d 0..2 or x 1..xsize), how it interacts with static vs dynamic
- Putting parallelization in inner loops instead of outer? Complex how this works? Implementation dependent? Probably not recommended but seems to work. Maxes out at total number of threads. Possibly some overhead for starting parallel over and over
- Static vs Dynamic speed. Setting up dynamic has a lot of overhead, and since our runs are quick and roughly same size didn't make a difference

- Reduction: you can use a reduction, but only if you are summing up results from a loop. So if you're using a loop to do the sums
- Don't mix pthread and OpenMP code. It might work, but it's not necessary
- Let OpenMP set thread number for you, don't try to parse OMP_NUM_THREADS yourself

Failure and Error Rate – Examples

Cassini Saturn Probe

- Gary M. Swift and Steven M. Guertin. “In-Flight Observations of Multiple-Bit Upset in DRAMs””. Jet Propulsion Laboratory
- Cassini, flight recorders, each with 2.5GB RAM
- Single bit error rate of 280 errors/day

Google Datacenter

- Google SIGMETRICS 2009 paper
- Schroeder, Bianca; Pinheiro, Eduardo; Weber, Wolf-Dietrich (2009). "DRAM Errors in the Wild: A Large-Scale Field Study". SIGMETRICS/Performance (ACM).
- 25-70k errors per billion hours per megabit
- 5 single bit errors in 8GB per hour

Various Supercomputer

- `http://www.computerworld.com/article/2493336/computer-hardware/supercomputers-face-growing-resili` `html` ASCI White when came out, MTBF 5hrs, got it to 55hrs
- "Analysis of the Tradeoffs between Energy and Run Time for Multilevel Checkpointing" PMBS 2014
  - Sequoia MTBF around 1 day
  - Blue Waters: 2 per day,
  - Titan MTBF: less than a day
  - 20% of computation is recovering from failures (big

energy waste)

- Scalable In-memory Checkpoint with Automatic Restart on Failures. Xiang Ni, Esteban Meneses, Laxmikant V. Kal

  Most of failures do not take down more than one node

  Jaguar/Titan 92% crashes single-node crashes

SSMD/ORNL

- https://csmd.ornl.gov/highlight/failures-large-scale-systems-long-term-measurement-analysis-and-i
- 2015
- 1.2 billion hours Jaguar/Titan/EOS
- MTBF can change over time, optimal checkpoint might depend on this
- Temporal and Spatial Locality
  - Temporal, fail at same time. Things like voltage surge, OS panic, filesystem error
  - Spatial, failure in same location. Cooling/overheat

issues

Frontier Supercomputer

- 10 oct 2022
- `https://www.datacenterdynamics.com/en/news/frontier-supercomputer-suffering-daily-hardware-failur`
- This also mentioned in the SC top500 video
- Problems with MPI Cray fabric
- Possibly also GPU issue under load
- MTBF hours, not days. “one day MTBF would be amazing”

GPU Lifetimes

- "GPU Lifetimes on Titan Supercomputer: Survival Analysis and Reliability" by Ostrouchov et al
- `https://www.christian-engelmann.info/publications/ostrouchov20gpu.pdf`
- 18,688 GPUs do most of computing
- There have been three re-works
- Two to fix chassis issue
- One to fix resistors failing due to silver sulfide corrosion
- DBE (double bit errors) and OTB (off the bus errors, i.e. GPU stop responding)

- MTBF for first batch around 3 years, but some fail more quickly
- Survival Analysis methods (similar to those used in medical field)

What can Software do to avoid HW Problems

- Note that a lot of reliability bugs are very similar to security bugs
- Programs crash due to out of bounds, memory overflows, stack smashing
- Hardware is starting to add protections against these types of things (Ryzen3 shadow stack)

Byzantine Failure

- Byzantine General Problem, Lamport et al
  - Generals surround a city.
  - Want to all attack or all retreat; doing part way will fail
  - Might be traitorous generals with complex motives
  - (split their vote, if 5R 4A, tell the 5A and 4R).
  - Unreliable messengers
- Roundabout way to say, can we write code that gives correct results even if we have hardware that we can't

trust to give the right answers

N-version software

- Implement same code many different ways
- Vote on result.
- Need a tight spec to make sure results will all match.

Algorithm Based

- Parity checks, CRC
- Spread out work so that if one gives wrong result it can be checked. Overlap work.
- Add some extra values to calculation that can be checked, can tell if something went wrong

Data Structure Based

- Extra state in data structure or checksum so can tell if it gets corrupted.

Control Flow Checking

- Knows where code should be allowed to jump to
- If you jump somewhere impossible, checker stops things
- Hardware these days can help with this
- For example: compiler knows all callers of function. Return from function should always return back to one of these locations.

Application Level Checkpointing

- Checkpoint your program state periodically.
- If a failure takes down a program or hardware node, you can restore to last checkpoint rather than starting from scratch.
- Two kinds
  - manual — (you save out your state manually and have to write code to restart from arbitrary point)
  - Automatic – kernel stores everything possible about your state and can restart a program from a snapshot.

Checkpointing Difficulty

- Must save all program state, network connections, RAM contents, disk state, open files, etc.
- Hard to do (I've written one). Some support in Linux kernel, need lots of patches as some syscalls are write-only.

Checkpointing Overhead

- Checkpoints have high overhead. Have to stop while taking them? Write GB to disk?
- Multilevel checkpoint – big checkpoint occasionally and smaller subcheckpoints

Crash Only Software

- Crash-only software – crashing and restarting can take less time than clean reboot.
- So why write code to cleanly shutdown? Instead write your code so it can handle crashes cleanly. That way your cleanup code is tested every exit, rather than rarely on a crash.

Approximate Computing

- Approximate Computing – some algorithms don't necessarily need the "right" value
- Video rendering, voice recognition, web search, robotics, GPS, image processing

BOINC

- Someone asked how distributed computing worked for things like Folding at Home
- Use Berkeley Open Infrastructure for Network Computing
- Sort of like grid computing
- As of 2020 worldwide added up to 41 PFLOPS (would be #5 on top500)
- Researchers upload binaries and datasets, the servers then distribute them to volunteers by client they run

- Server is just really old-fashioned PHP/MySQL LAMP server
- Server also validates results, also hands out workloads multiple times to be sure the answers match
- My friend runs the Machine Learning Comprehension at home project.