ECE 574 – Cluster Computing
Lecture 16

Vince Weaver

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

vincent.weaver@maine.edu

6 April 2017

Announcements

- HW#7 was finally posted. How is it going? Don't wait until last minute!
- PAPI paper submitted to SC'17 on time
- Project ideas were due
- ECE435 pre-requisite note

MPI Review

- 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.

- `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.

MPI and slurm

- HW `#SBATCH --tasks-per-node=4`
- -N = number of nodes
- -n = number of tasks, default is one task per node?
- N=4 tasks-per-node=4, 16
  N=4 tasks-per-node=4, sbatch -n 8, 16 (N=nodes, n=tasks)
  N=4 tasks-per-node=4, sbatch -N 8, 32

nothing, sbatch -N 8, 32
nothing, sbatch -n 8, (8, 2 nodes * 4 each)
nothing, sbatch -N 8 -n 8 (8, 8 nodes * 1 each)

SLURM update

Be careful with letting jobs run infinitely. I put a 5 minute timelimit on because some jobs were taking forever and locking people out of the queue. Will need to modify your slurm files.

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
  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

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
- Memory checksums
- 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.

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

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.

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.

Things you can do Software

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

Control Flow Checking

- Knows where code should be allowed to jump to
- If you jump somewhere impossible, checker stops things

Checking Data Structures

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

Memory Failures

- Memory Errors in Modern Systems
  ASPLOS 2015

- Battling Borked Bits
  IEEE Spectrum December 2015

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

Failure and Error Rates

- Cassini, flight recorders, each with 2.5GB RAM
  Single bit error rate of 280 errors/day
- Google SIGMETRICS 2009 paper
  25-70k errors per billion hours per megabit
  5 single bit errors in 8GB per hour
- ASCI White when came out, MTBF 5hrs, got it to 55hrs
- Sequoia MTBF around 1 day, Blue Waters: 2 per day,

Titan MTGF: less than a day

- 20% of computation is recovering from failures (big energy waste)

- Most of failures do not take down more than one node Jaguar/Titan 92% crashes single-node crashes

Things you can do Software

Byzantine Failure

- Byzantine General Problem, Lamport et al
  Generals surround a city. Want to all attack or all retreat; doing it part way will fail.
  Might be traitorous generals with complex things (split their vote, if 5R 4A, tell the 5A and 4R).
  Unreliable messengers.

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

Control Flow Checking

- Knows where code should be allowed to jump to
- If you jump somewhere impossible, checker stops things

Checking Data Structures

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

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.

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

- 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