SAND2014-18204PE

Memory Reliability and Performance Degradation

Hunting rabbits with an elephant gun?

Benjamin Allan

HPCMASPA Mini-talk 9/26/2014

Cluster 2014, Madrid, Spain

Outline

Why?

Quantifying silent memory errors in the wild is really hard

- – Bulk RAM is a necessary target
- – How HPC makes it even harder
- – Current approaches and limitations

Possible new approach?

Discussion

Sandia National Laboratories

Why RAM silent error rates?

- Fault tolerant numerical algorithms do not account for errors in pointers.
  - Quantify the hardware risk as built
  - Mitigate if necessary and possible
  - Identify and remove marginal RAM (ECC recovery delays computations) to improve performance
- Silent fault rate may be as much as 10% of corrected fault rate
- 5-10% of CPU logic is not protectable

Sandia National Laboratories

Why HPC is harder

Odds of a corrupt but plausible pointer:

- 24% chance a 3-bit error in a pointer is confined to significant bits $(40/64)^3$
  - Therefore untrapped.
  - Compare this to embedded computing with tiny address spaces.
  - Certain VMM approaches may increase this chance.
- Many applications fill RAM with similar object instances (bad pointer to a good object)
- 5-15% of application RAM is pointers
- Unknown odds of a 3-bit error happening

Sandia National Laboratories

Why not other subsystems?

- Lack of access to measure or change
  - Buy built-in reliability if we can.
- If we can trust everything in the CPU-to-RAM path, we can use software to work around less reliable components.

Detection methods

For shame, Doc! Hunting rabbits with an elephant gun!

- B. Bunny

A. Burn-in testing
   – Does not account for lifetime effects
B. User-level mem-check application
   – No time available
C. Persistent mem-check daemon
   – Interference with job memory placement
   – Difficulty scheduling checks on caches
D. Kernel thread
   – Unlikely uptake by a latency sensitive kernel community.
   – Insufficient kernel data to co-schedule idle CPUs and buses?

Do and I'll give you such a pinch!

Sandia National Laboratories

An Unusual Co-scheduling

Have the kernel scrub idle RAM

- Fill RAM, then idle almost until allocated.
- Predict CPU load, cache and RAM bandwidths to avoid interference.
- Create application hooks allowing users to hint about short idle periods or RAM usage planned to avoid interference.
- See what can be learned from kernel’s page zero-on-allocation code.

Sandia National Laboratories

Co-scheduling difficulties

- Must be controlled by user opt-in
- Choose data values carefully for memory testing
- NUMA locality issues
- Down-clocking awareness

Discussion

- Other continual benchmarks of interest if we have an idle-component scavenging framework?
- Other examples of large-memory, low-cpu, long-term task co-scheduling?
  - GPU characterization?
  - Burst buffer drain?
- Other approaches to quantifying silent errors?

Sandia National Laboratories

Partial audience responses

- Cray: interested in independent measurement of silent errors: included in contracts, but no metrics.
- Use queue drain times/idle times.
- Most GPU idle states are generally expected to preserve memory: opportunity?
- Modified kernel for experiments (not production: overheads) could chksum RO pages like zfs does disk

Sandia National Laboratories