Detailed Design & Evaluation of Redundant Multithreading Alternatives

- **Redundant Multithreading (RMT):** running the same program on multiple (2) threads of a threaded processor and comparing the answers to catch transient faults. On mismatch: Hardware of Software error correcting sequence or replay initiated
- **This paper:**
  - Evaluates RMT in a detailed Alpha processor model. RMT causes 32% average slowdown.
  - Evaluates RMT on an SMT (2 leading, 2 trailing threads). Now causes 40% slowdown. This is called SRT (simultaneous redundant threading). Only 32% slowdown with per-thread store queues.
  - Evaluates RMT on a 2-way CMP, called CRT, with each processor having 2 threads. Run leading thread and trailing of opposite workloads on each processor. Checks for permanent hardware faults in addition to transient faults by running the threads of the same workload on different hardware.
  - CRT less efficient than SRT because each thread (trailing and leading) has to mis-speculate because harder for trailing thread to use information from the leading thread when running on separate processors.
- **Adding RMT to an SMT:**
  - Requires an input replicator to replicate the input to feed to both threads and an output comparator to compare outputs from both threads and detect faults
  - In their framework:
    - All values loaded from the L1 cache must be replicated and fed to both threads
    - All values stored from the processor must be compared and checked for errors
    - All values loaded from the cache by the leading thread are held in a FIFO until they can be compared to the values loaded from the trailing thread
    - Stores must be held In the store buffer until the trailing thread commits its store and the stored values can be compared. Once compared, they are sent to the cache as a single store. Any type of SRT makes the store buffer a critical resource for performance
    - Branch results kept from leading thread to avoid mis-speculation trailing thread
- **Deadlock Issue:**
  - Because store values have to be compared in the store buffer, deadlock could result if the threads are far out of sync and the store buffer fills before the trailing thread store commits -> no comparison can be made
- **Conclusion:**
  - With reliability being a major issue with future server processor designs, redundant multithreading can check for transient faults with minimal slowdown. If the leading and trailing threads run on different hardware, permanent faults can also be detected