Parallel Error Detection Using Heterogeneous Cores

Sam Ainsworth, Timothy M. Jones

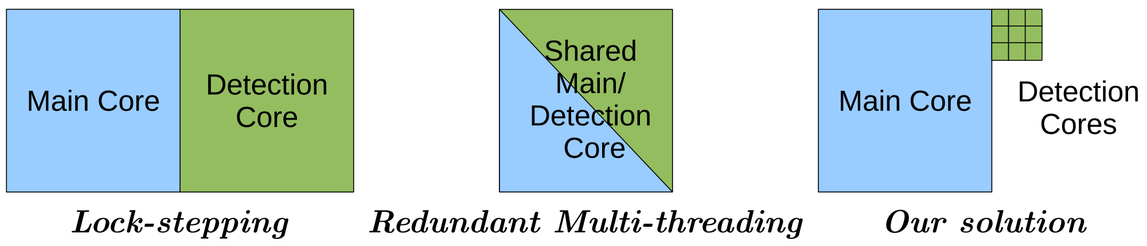
Computer Architecture Group
University of Cambridge Computer Laboratory

Overview

Hard and soft faults in silicon processors are becoming an increasing issue for reliability in modern systems: temporary or permanent errors from physical properties can cause executed computation to be incorrect. Traditional schemes to detect errors within the CPU involve repeating the computation on either an identical core, or on the same core at a later time, and have typically doubled power and silicon area requirements, or reduced performance significantly.

However, we have developed a scheme which significantly reduces the overheads of all three by exploiting new parallelism in the checking of computation which may not exist in the original run. This allows execution of the checks on highly efficient micro-controller sized cores attached to the CPU.

Comparison

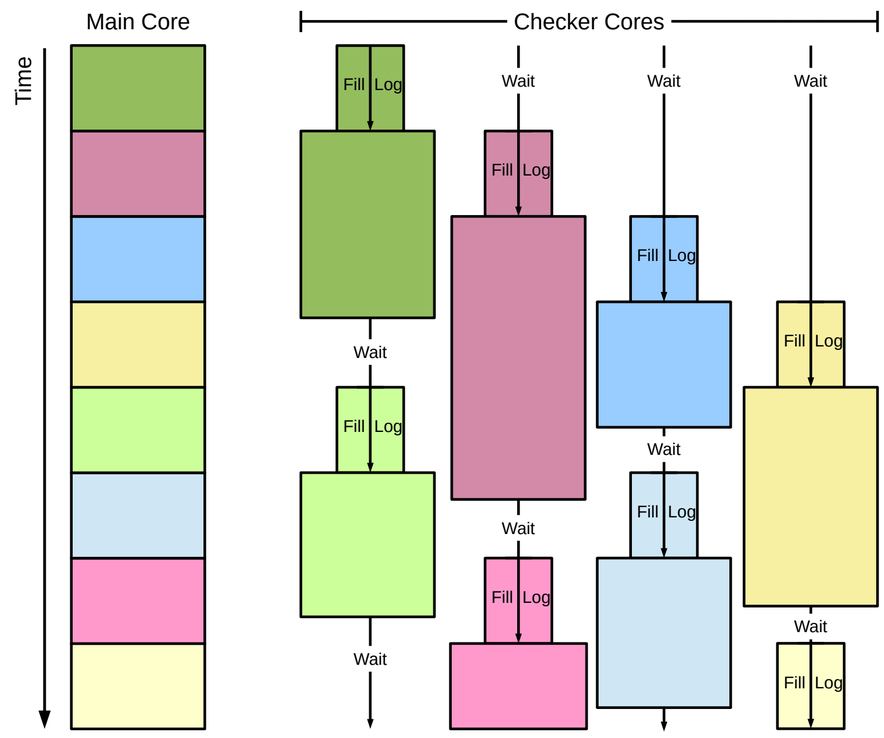


Overhead	Lockstep	RMT	Our Solution
Area	Large	Small	Small
Energy	Large	Large	Small
Performance	Negligible	Large	Negligible

Duplicating the core and comparing the results between each, as in **traditional lockstepping schemes** [1], is too expensive in both energy usage and silicon area. **Redundant multi-threading** [2], where the same code is run twice on the same core, reduces performance significantly, uses significant energy as instructions are executed twice, and can't cover hard faults within a particular core.

We solve these issues by exploiting **heterogeneity between the initial run and detection**, allowing parallelism of the detection stream, and thus the use of many tiny, low power checker cores to repeat the computation.

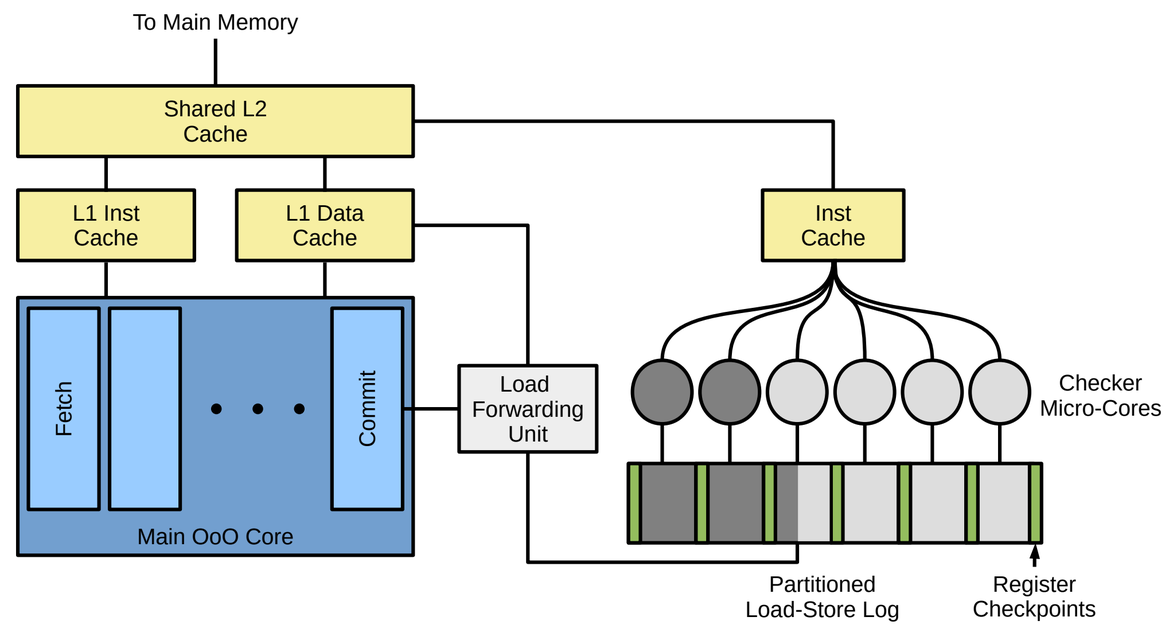
Parallelism



Our solution attains parallelism to allow running the duplicate execution for error detection on multiple small cores by:

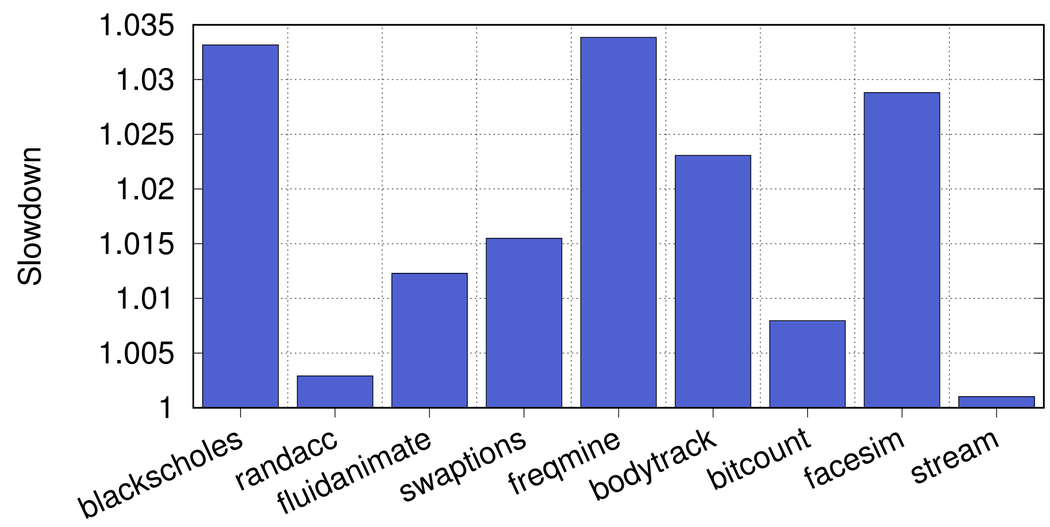
- Splitting the main computation's loads and stores into a **partitioned log** for forwarding onto checker cores.
- **Checking each log partition independently** on a different checker core, allowing different checks to overlap, then combining using strong induction to prove whole program correctness.
- Taking **register checkpoints** between each segment on the main core, for independent checker start points, making sure the start of one check is a valid end of the previous one.

Error Detection Structure



- Our main **out-of-order core** forwards results of all loads and stores at commit time, so that they are in order.
- These go into the **partitioned load-store log**. When a segment is filled, a **checkpoint** of architectural registers is initiated on the main core.
- These checkpoints and log segments are used to start the corresponding micro-controller sized **checker core**, which executes until an incorrect store is found or it reaches the end of a segment.
- The main core can continue execution until every checker core is busy, at which point it must wait until a new checker core becomes free.

Results



- Across a diverse set of workloads (Parsec, MiBench, HPCC) twelve micro-controller sized cores each running at 1GHz is sufficient to achieve an **average slowdown of only 2%** compared to unchecked code.
- Similarly, we estimate only around **24% area overhead** compared with the original processor without shared caches, and only **16% energy consumption** overhead.

Acknowledgements

This work was supported by ARM Ltd and the Engineering and Physical Sciences Research Council (EPSRC) through grant references EP/K026399/1 and EP/M506485/1.

References

[1] X. Iturbe, B. Venu, E. Ozer, and S. Das. A triple core lock-step (TCLS) ARM®Cortex®-R5 processor for safety-critical and ultra-reliable applications. In *DSN-W*, 2016.

[2] S. K. Reinhardt and S. S. Mukherjee. Transient fault detection via simultaneous multithreading. In *ISCA*, 2000.