An Event-Triggered Programmable Prefetcher for Irregular Workloads

Sam Ainsworth, Timothy M. Jones
University of Cambridge

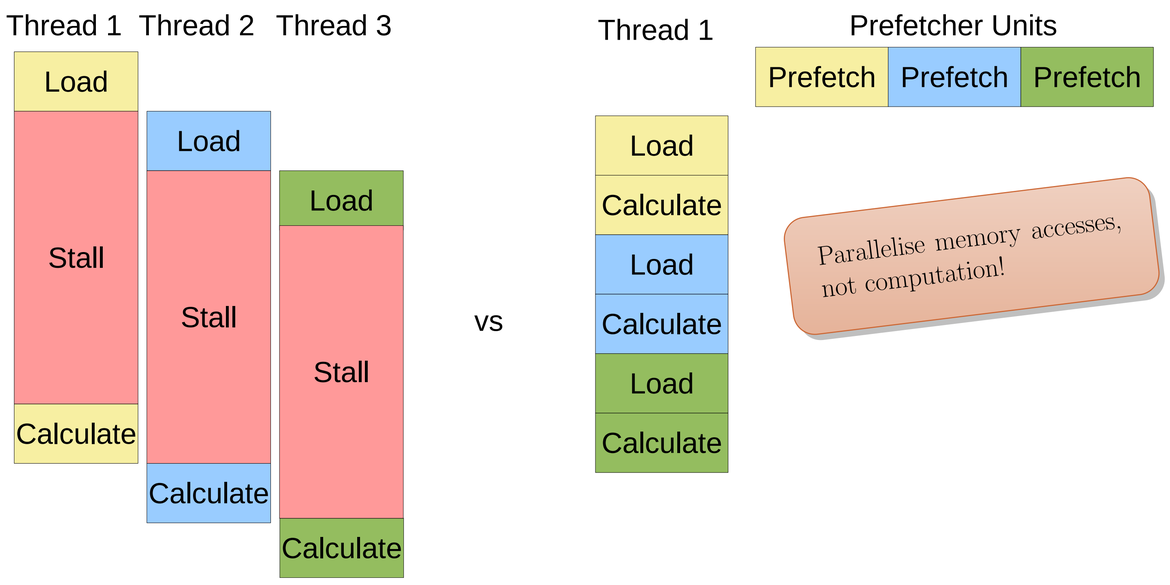
Overview

Current prefetching implementations in real systems are predominantly address-based: these work well for regular patterns, but irregular workloads are increasingly important in scientific and big data domains. Performance for these workloads is highly constrained by the number of processor stalls generated as a result. Prefetchers targeted at irregular workloads in the literature tend to match too few workloads and require too much state to be useful.

So how do we design a prefetcher that knows about the workloads it is processing, and can be programmed to adapt to a variety of complex access patterns?

We have developed an event-triggered prefetcher, which extracts memory-level parallelism using a large set of microcontroller-sized miniature programmable cores integrated into the main core, in a latency tolerant way, along with compiler-assistance techniques to ease programmer efforts in targeting code to run on these prefetchers.

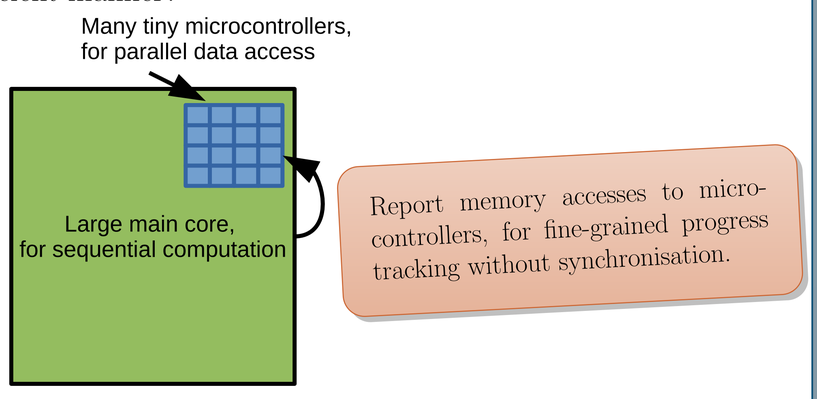
Memory-Level Parallelism



Instead of running computation on multiple threads, with complicated synchronisation and inefficient work splitting, parallelise the loads via prefetching, and run the computation code on a single core.

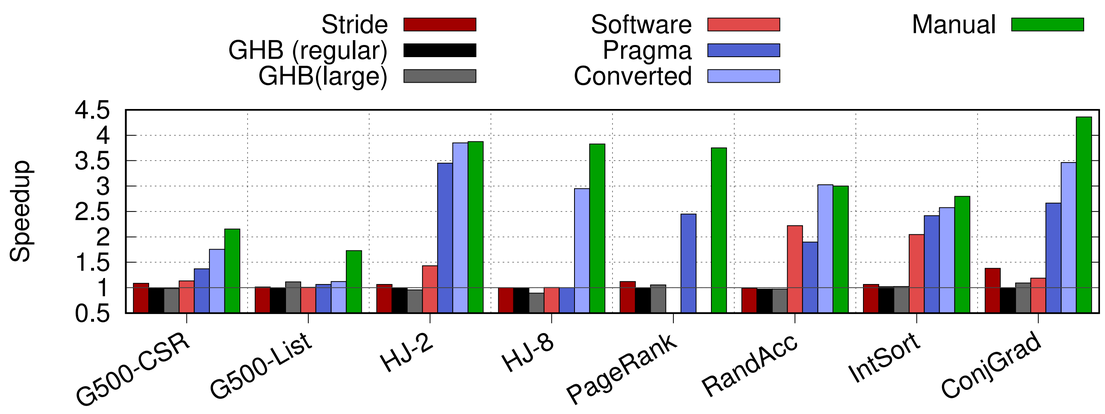
Parallel Units

Staying ahead of the main thread requires a lot of compute power. How can we provide this in a power and space-efficient manner?

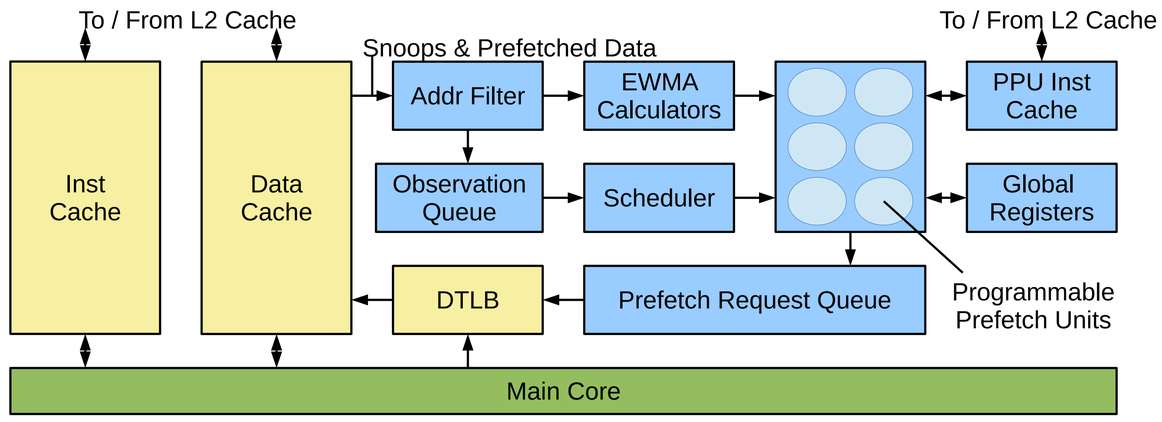


Solution: make use of parallelism in the data access rather than the computation, and run prefetches on multiple tiny microcontrollers embedded in the core!

Large Speedups on Irregular Workloads



Programmable Prefetcher Structure



- Send all loads and prefetches observed by the cache through to the prefetcher, to generate new prefetches.
- Filter out uninteresting data: only send load observations within specified address ranges through to the microcontrollers.
- Use moving average hardware to schedule appropriate prefetch lookahead.
- Use latency tolerant queues to allow for decoupling of processing and loads.
- Disallow loads and stores from the prefetcher units: no synchronisation required, so can be speculative, and no stalling on memory. Only allow latency tolerant prefetches, which may cause another prefetch to be triggered later.
- Reconfigurable registers to store important long-term values such as array bounds.

Events

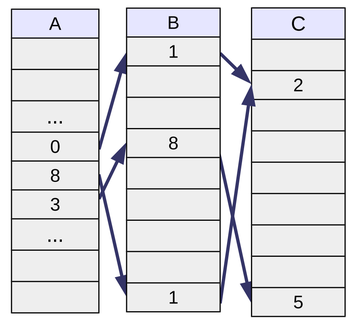
- To keep up with many parallel loads, make sure prefetcher units don't stall.
- Use events: trigger a new prefetch by reacting to old prefetches and loads from the main core, do some computation, issue a prefetch, then end the event - highly parallel!
- No loads: just prefetches, which may or may not return data for another event to process!

Original Code:

```
for(int x=0; x<N; x++) {
    C[B[A[x]]]++;
}
```

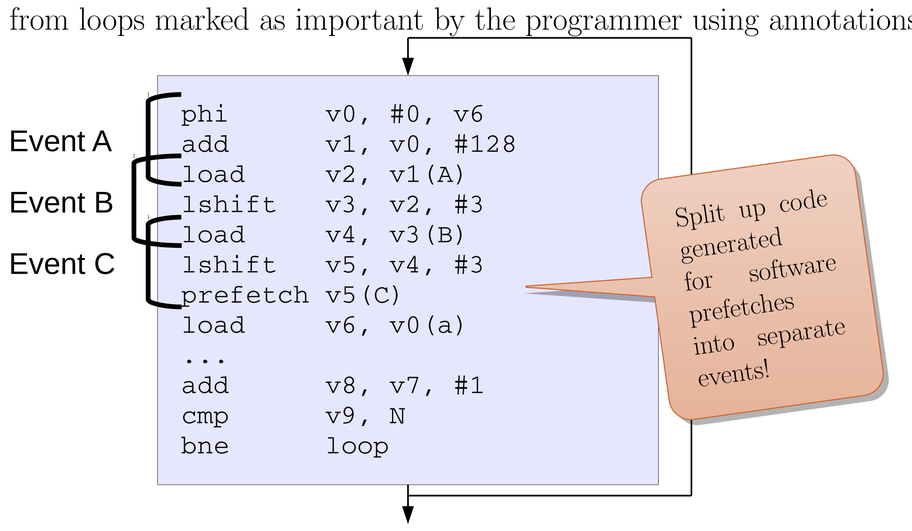
Programmable Prefetcher Events:

```
Observe Load to A[n]
Prefetch A[n+N]
Prefetch B[A[n+N]]
Prefetch C[B[A[n+N]]]
```



Compilation

Automate the generation of event code for the programmer, to ease manual effort. Convert software prefetches in code into non-blocking events, or try to automatically generate events from loops marked as important by the programmer using annotations.



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.