Software Prefetching for Indirect Memory Accesses

Sam Ainsworth, Timothy M. Jones
University of Cambridge

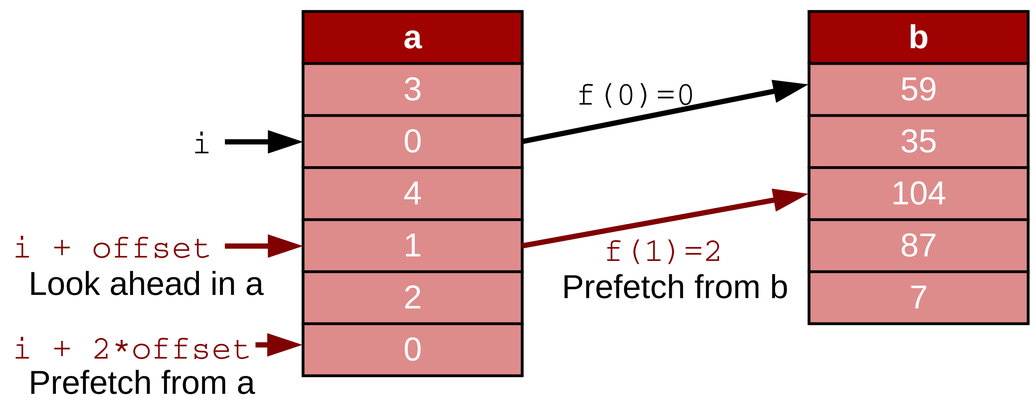
Overview

Many modern data processing and HPC workloads are heavily memory-latency bound. A tempting proposition to solve this is software prefetching, where special non-blocking loads are used to bring data into the cache hierarchy just before being required. However, these are difficult to insert to effectively improve performance, and techniques for automatic insertion are currently limited.

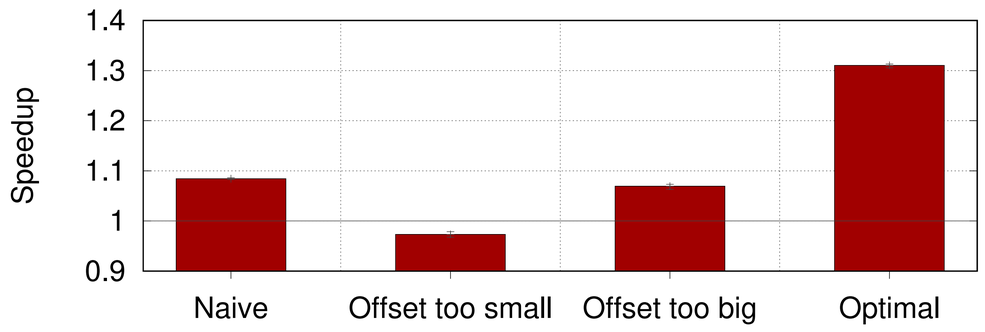
We have developed a novel compiler pass to automatically generate software prefetches for indirect memory accesses, a special class of irregular memory accesses often seen in high-performance workloads. Across a set of memory-bound benchmarks, our automated pass achieves average speedups of 1.3× and 1.1× for an Intel Haswell processor and an ARM Cortex-A57, both out-of-order cores, and performance improvements of 2.1× and 2.7× for the in-order ARM Cortex-A53 and Intel Xeon Phi.

Software Prefetching

```
for (i=0; i<a_size; i++) {
    SWPF(b[f(a[i + offset])]);
    SWPF(a[i + offset*2]);
    b[f(a[i])]++;
}
```



Good Prefetches are Challenging



- Need to stagger prefetches to each data structure, even those covered by the stride prefetcher!
- Need to set a good look-ahead offset, that brings in the data neither too late, nor too early.
- But the behaviour is surprisingly resilient across microarchitectures and workloads, in terms of both strategy and look-ahead distance!

Acknowledgements

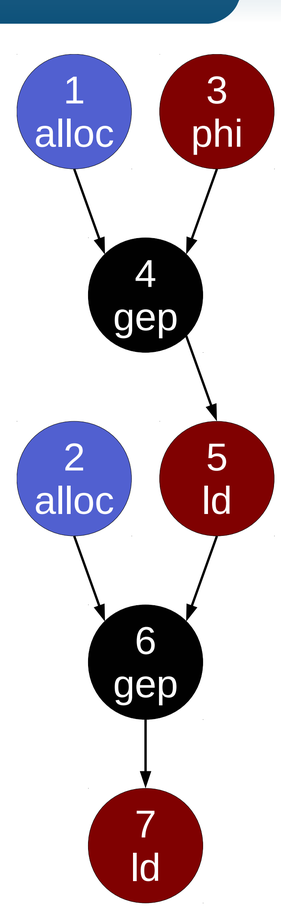
This work was supported by ARM Ltd and the Engineering and Physical Sciences Research Council (EPSRC) through grant references EP/K026399/1 and EP/M506485/1.

Algorithm

We use a dataflow analysis in LLVM IR to automatically insert prefetches:

- **Identification**: Trace back through loads, to find both an induction variable and a set of dependent loads based off it.
- **Safety Analysis**: Look for array bounds information, to ensure no real loads, used for prefetch address generation, cause faults.
- **Scheduling**: Use the load pattern to statically set look-ahead offsets for prefetches.

```
start:  alloc a, a_size
        alloc b, b_size
loop:   phi i, [#0, i.1]
        gep t1, a, i
i+64    ld t2, t1
        gep t3, b, t2
i+32    ld t4, t3
        ...
```



Large Speedups on Real Cores