Prefetching in Functional Languages

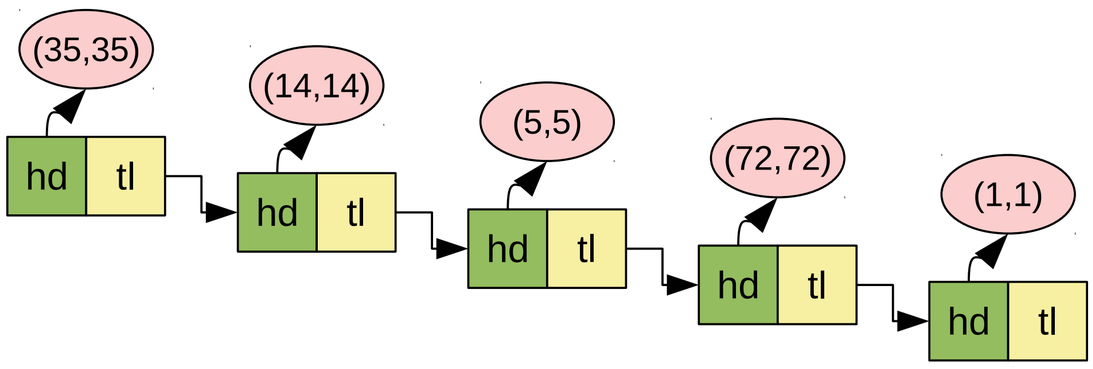
Sam Ainsworth and Timothy M. Jones
University of Cambridge

Summary

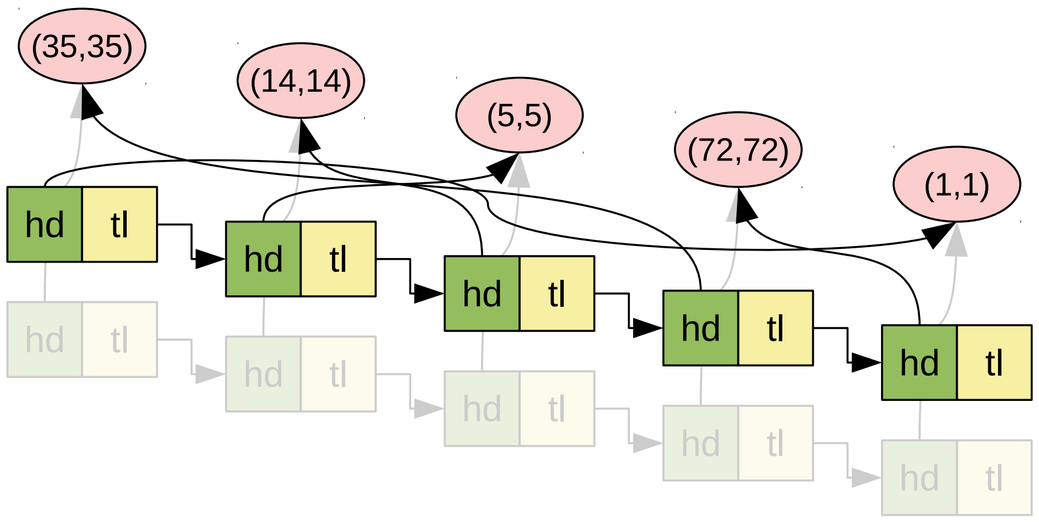
Functional programming languages contain a number of runtime and language features, such as garbage collection, indirect memory accesses, linked data structures and immutability, that interact with a processor's memory system. These conspire to cause a variety of unintuitive memory-performance effects, which can cause significant slowdown.

We mitigate by adding software prefetching primitives into OCaml, which can provide significant speedup. In addition, we observe just how important existing hardware prefetchers, in most modern devices, are to the performance of functional code: surprisingly, hardware prefetchers are even more effective for linked list code than they are for arrays.

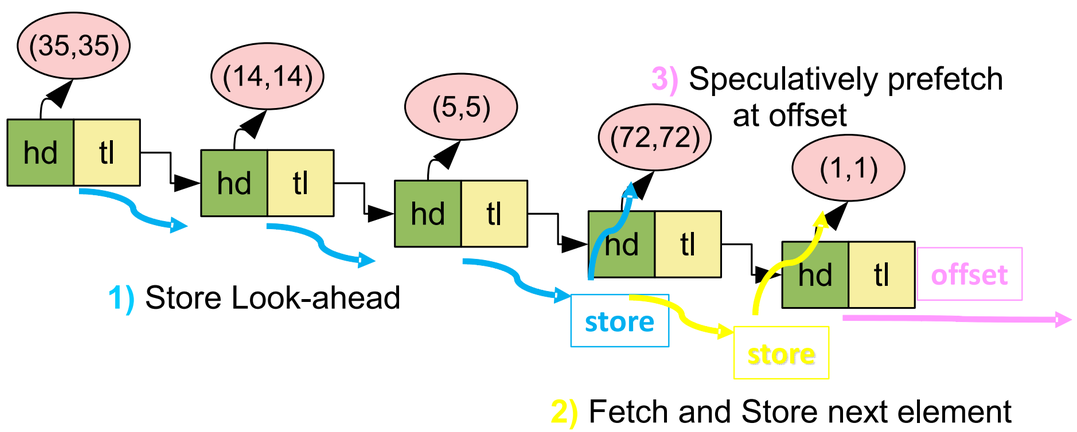
List Prefetching



When a list gets allocated, it is often stored approximately in the same order, meaning memory accesses are roughly contiguous.



However, when that data is sorted, the objects get reordered, and though the linked list is still roughly sequential, performance drops on the associated connections, which become unpredictable to the memory system.



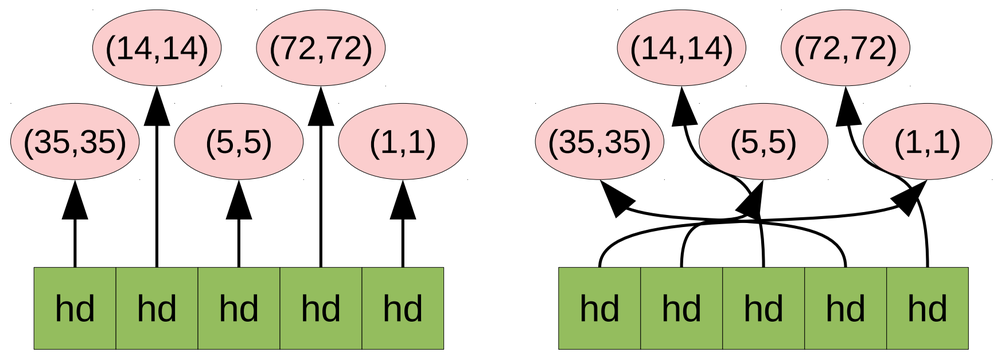
Still, we can mitigate through software prefetching, by 1) jumping ahead in a list and storing the location, 2) using this stored location, incrementing it, and issuing a prefetch on each walk through the list, and 3) speculatively prefetching off the tail, in the hope of discovering other list cells.

Acknowledgements

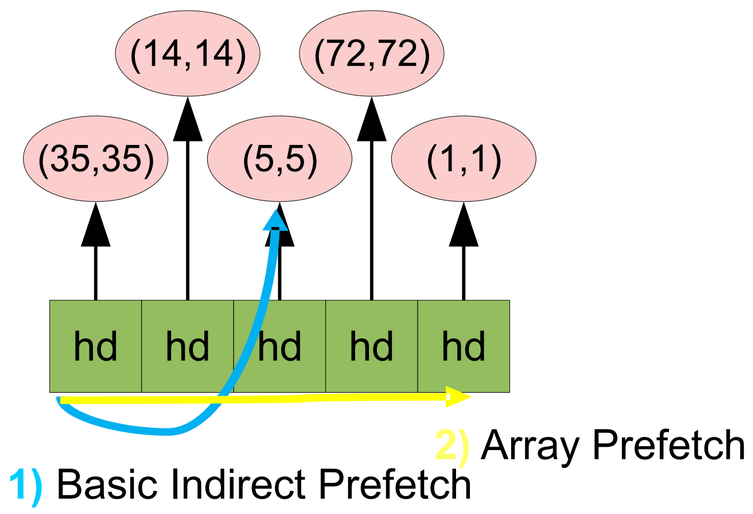
This work was supported by the Engineering and Physical Sciences Research Council (EPSRC), through grant references EP/K026399/1, EP/P020011/1 and EP/M506485/1, and ARM Ltd.

Array Prefetching

Unlike lists, array cells are guaranteed to be stored contiguously in memory, and thus sequential accesses are always easily predicted by the memory system and its prefetchers.

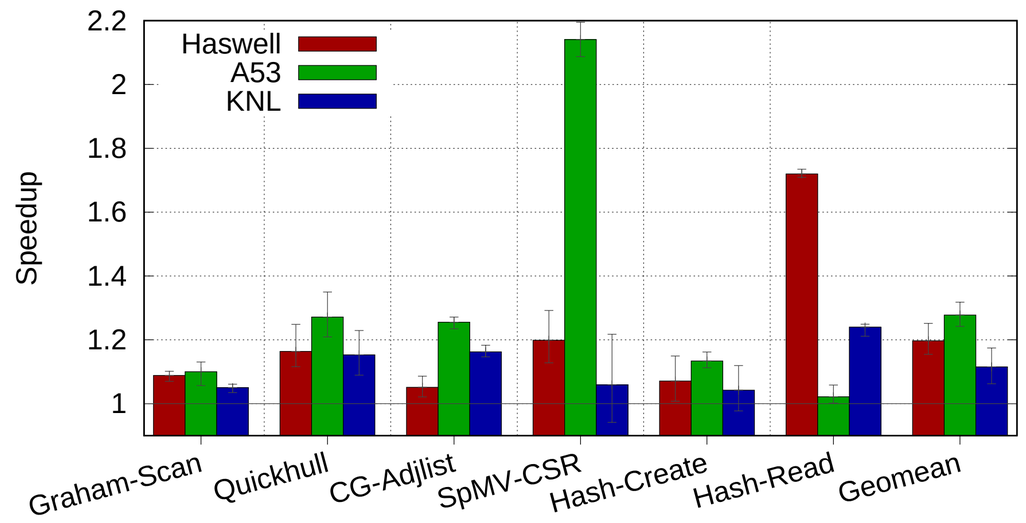


Still, under operations such as sorting, the pointers inside the array may become jumbled in memory, rather than being stored contiguously, significantly reducing performance as the hardware prefetcher can no longer predict them and bring them in early from main memory.



Here, we can mitigate by 1) looking ahead in the array, to prefetch the element we will access in a few iterations' time, and 2) prefetching ahead in the array itself, staggered from the first prefetch, to bring in the pointer address we will access for the first prefetch in a few further iterations.

Speedups



Speedups on a variety of list- and array-based code, ranging from sorting algorithms, sparse matrices, hash tables, library operations, and many others in the full paper - up to 2× on the out-of-order superscalar Intel Haswell, and up to 3× on the in-order Arm Cortex A53.

The Full Paper

See the ISMM talk, or `https://dl.acm.org/doi/10.1145/3381898.3397209` for more information!