Gavin Gresham

CSE 6140

Homework 2: Real-World Algorithm Performance

Technical Details

Processor: i7 920 3GHz

L1 Cache: 32K

L2 Cache: 256K

L3 Cache: 8M shared

Cache Line: 64 bytes

Timing methodology:

I used Java’s *currentTimeMillis()* function to get the current system time before, and after a given run. The only section timed is a loop that traverses the arrays. I averaged these times over 5 identical runs to diminish any anomalies from heavily impacting the results.

With the sequential reads you can see the resolution of the timing method is not precise enough measure fewer than arrays with 2^19 elements. With random access, the time is measurable after 2^16 elements. This number of elements is also the last run that can fit within the L1 and L2 cache of the processor.

As the algorithm to traverse the link list and arrays should both touch each element once, and only once, the complexity is O(N) for N elements. Because of this we could expect the access time of a new element to be the same as the element before it, assuming the RAM model holds true.

Analysis

Modern programming languages attempt to prefetch cache lines before they are needed for execution, so it makes sense that the sequential reads do not have an increase it read time as more elements are added. Random reads require one memory read per element, while one memory access for a sequential read could fetch 16 integers thanks to the 64 byte cache lines of the i7.

To test this I ran another experiment with a stride of 16. This did show a marked increase in time needed to access elements, compared to the strides of 2 and 4.

I believe this shows that the RAM model does not hold true. If it did, the access time for each element should be the same; One time unit per access. Instead, sequential reads seem to have similar access times, but any random reads increase rapidly per each additional element. This shows that memory accesses are not consistent, and reveal a performance gap between the arrays that can fit in the per-CPU L1 and L2 caches, and ones that must extend to the shared L3 cache and RAM.

Fig. 1: Time taken to access an element of an array of size 2^N.

Fig. 2: The same data as Fig. 1, but with the Random array removed for a better representation of the other access types.