Progress Update: Array Prefetching for Recursive Methods in Java

Hayden Hudgins and Christopher Siu
CSC 515, Fall 2018
{hrhudgin,cesiu}@calpoly.edu

I. Background Research

We began by investigating the general concept of prefetching in Java. We found that, at one point, Oracle did experiment with adding prefetch statements: the reference Hotspot JVM supported the `Unsafe.prefetchRead` and `Unsafe.prefetchWrite` intrinsic methods. However, the developers decided that these experimental instructions did not provide a worthwhile performance improvement, so they were removed in Java 9. We theorize that this is because almost everything in a Java program is an object, and the memory layout of these objects can vary between JVM implementations. It appears that this will be managed by the garbage collector, which will group objects in memory generationally. Thus, in the typical use case, it does not make sense to prefetch data — the typical Java program involves references to objects scattered throughout memory, having been organized based not on when they are used, but on when they are likely to be deallocated.

II. Proof of Concept

We wrote a simple recursive method that operated on an array of 512k integers. The runtimes for this operation varied between 20–40ms. We found that, during the i^{th} recursive call, accessing the $i + 1^{\text{st}}$ value in the array was sufficient to consistently improve runtime to 20ms. This supports the idea that the JVM does not practice any sophisticated explicit prefetching, so there is opportunity to give it a *hint*, so to speak, in cases such as this.

III. Beginning Implementation

The OpenJDK implementation is full of legacy code and lacking in documentation. Fortunately, we discovered that, beginning with Java 8, the Java compiler provides an API for adding plugins that can traverse and alter the AST. We've begun exploring the node definitions of that tree, and we've written an extremely simple plugin to ensure that it's possible to hook into the lifecycle of the compiler.

IV. Next Steps

We must extend our plugin to:

- Be able to identify recursive methods that process arrays.
- Be able to determine "induction variables" (e.g., "`i`") within those methods.
- Be able to insert statements to fake prefetching before recursive calls (e.g., "`int y = arr[i + 1]`" — we found that, even when assigning these values to variables that were never used, the Java compiler did not appear to discard these statements).

Once this is done, we could optionally:

- Compare performance to the equivalent (manually) inlined methods — we expect that this will produce even better performance. Not only will array elements be accessed an iteration early, but they will then be processed in the same stack frame with fewer instructions.