Array Prefetching for Recursive Methods in Java

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

I. Background

Software prefetching is the compiler technique of adding instructions to load data before it is needed, mitigating memory latencies by allowing other work to be completed while data is brought into cache [1]. This can be done by introducing dedicated prefetching instructions; it can also be done simply by rearranging program logic (whether high-level statements or low-level instructions) such that the compiler can make more intelligent instruction selections. As with many other high level languages, the memory model used by Java allows compilers to perform such rearrangements, provided that, from the perspective of the programmer, sequential consistency appears to be maintained [2].

II. Proposal

Cahoon and McKinley [4] have previously described an algorithm for identifying loop induction variables and recognizing patterns of accesses in order to prefetch arrays processed by loops in Java. We propose exploring prefetching arrays processed by simple recursive methods in Java. In order to keep this project feasible, we will initially limit our work to structurally (i.e., not generatively), directly (i.e., not mutually) recursive methods in single-threaded programs.

This will not be as simple as loading data early. Since the methods we will attempt to optimize are recursive, the next iteration which requires that data will be a method call. The method will need to be inlined at least once, else the prefetched data would be in the wrong stack frame or could be pushed out of cache by the overhead of the call. Thus, there are two phases to this project: firstly, we will need to determine which methods are candidates for inlining — this alone may allow the compiler to improve its instruction selection and scheduling. Secondly, we may attempt to reorder the inlined statements or the generated bytecode itself.

III. Rationale

We selected Java because it provides an open source reference implementation that compiles a widely-used language into pseudo-assembly instructions with which we have had little experience. We already have experience with the IA32, ARM Thumb, and MIPS instruction sets, so we'd like to work with something different for this project. We recognize that this optimization of simple, recursive methods is, perhaps, less practical in Java, since those methods are less common, especially given the language's lack of support for tail-call optimization.

We limit ourselves to prefetching arrays because, while it is possible to perform data flow analysis to prefetch more complex structures [3], we feel that doing so is less of a architecture problem and more of a compilers problem.

IV. Resources

Primarily, we'll require time to understand the OpenJDK implementation.

V. Evaluation

We hope to see some speedup in recursive methods involving arrays, such as those that compute the sum or average of an array of integers or those that insert an integer into a sorted array. Based on our restrictions, we expect no speedup in recursive methods such as those that compute Fibonacci numbers or traverse linked lists. Non-recursive functions should similarly exhibit no changes in runtime nor in instruction selection.

References

[1] D. Callahan, K. Kennedy, A. Porterfield. "Software Prefetching", in *Proceedings of the Fourth Internation Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IV)*, 1991, pp. 40–52. Available: http://doi.acm.org/10.1145/106972.106979 [October 13, 2018].

[2] J. Manson, W. Pugh, and S. V. Adve. "The Java Memory Model", in *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '05)*, 2005, pp. 378–391. Available: http://doi.acm.org/10.1145/1040305.1040336 [October 13, 2018].

[3] B. Cahoon and K. S. McKinley. "Data Flow Analysis for Software Prefetching Linked Data Structures in Java", in *Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques*, 2001, pp. 280–291. Available: https://ieeexplore.ieee.org/document/953309 [October 13, 2018].

[4] B. Cahoon and K. S. McKinley. "Simple and Effective Array Prefetching in Java", in *Proceedings of the 2002 Joint ACM-ISCOPE Conference on Java Grande (JGI '02)*, 2002, pp.86–95. Available: http://doi.acm.org/10.1145/583810.583820 [October 13, 2018].