Using Natural Language Hints to Improve Scheduling and Prefetching in JAVA

Alejandro Carbonara (alejandro.carbonara@gmail.com)
Wenlu Hu (wenlu@cmu.edu)
Jiyuan Zhang (jiyuanz@andrew.cmu.edu)
Project web page: http://www.cs.cmu.edu/~wenluh/15745/

Research Description

Object-oriented languages like JAVA and C# have different patterns than normal C programs. This offers some interesting opportunities in compiler optimization. Variable names in JAVA and C# programs usually consist of very informative words written in natural language. Variable naming may provide hints to facilitate optimizations. Additionally, Java programs feature many object definitions and references. These features may require different analysis when applying optimization techniques.

We would like to investigate how these natural language hints can be used to improve compiler optimizations. In particular, we want to improve instruction scheduling and data prefetching.

1. Utilize JAVA naming customs for scheduling. The way to do this is by predicting how likely a LOAD instruction is going to experience a cache miss by their naming customs. Then we schedule them accordingly, either by overlapping several cache misses in time, or by tailoring loop scheduling to each loop according to its own cache access pattern similar to [7]

2. Explore prefetching in objected-oriented language. Java features use of traversing linked list. We will explore how to identify linked-list traversing in Java and insert prefetch instruction.

3. Explore prefetch-helper thread. Trivial prefetching in pointer based structure may not work well. The subsequent elements address is unknown until its predecessor is fetched; thus the prefetch instruction can only insert one iteration ahead. But techniques such as prefetching helper thread[6] might be advantageous in this scenario. We would like to implement the helper thread if feasible to explore its impact on performance.

Metrics

Prediction accuracy can be used as a basic metric to verify that the hints are useful. To directly evaluate success, we need to evaluate the performance of the new version of optimization in terms of **runtime speedup**. We would also make sure **compilation time** is reasonable.

75% goal

Explore prediction utilizing JAVA naming: what hits can the JAVA provide

100% goal

Design, implement, and evaluate prefetching and scheduling.

125% goal

Actually achieve runtime speedup. The difficulty can come from getting the performance on real machine, because the current machine provides some hardware prefetch that can complicate the measure and the performance gain may not be substantial.

Logistics

Week of	**Tasks**
March 12	Define the problem and write proposal
Proposal Due	
March 19	Get familiar with a JAVA compiler; Learn the intermediate representation for object-based language.
March 26	Learn how to identify linked structure traversing with objects; how to insert prefetch
April 2	Select some open-source JAVA programs and extract their naming, figuring out 1) if the naming in those JAVA programs can provide us with useful information; 2). what information is useful for compiler optimization
April 9	Accomplish mile 1 to utilize naming to do compiler optimization;
Milestone	
April 16	
April 23	Write the final report and make a poster
Final Poster	

Literature Search

Fu et al [1] used natural language hints in Microsoft's C# programs to predict whether a runtime exception should be logged. By simply splitting variable names into natural words and apply a simple decision-tree technique, they get a significant improvement in prediction accuracy. Their work focuses on software engineering while our work focus on compiler optimization. In spite of the difference, their shows that variable names in C# are informative.

We can expect those in Java to be similarly informative and potentially useful too, as the two languages are very similar.

Abraham's work [4] has shown that in major benchmarks, a small number of load/store instructions are responsible for the majority of cache misses. If we are able to identify which instructions are at high risk for cache misses and which are low risk, we can selectively choose to prefetch the high risk instructions. Lipaski [5] has shown that it is very possible to create such metrics and use them to have a significant performance boost.

Resources Needed

Software: We are going to use LLVM. According to [2], LLVM supports Java front-end. If this does not work, we could switch to other Java compilers like Jikes and Soot.
Hardware: We are going to work on x86_64 machines. When a cluster of machine is needed, we will use VMs in WolfStack, a small private data center.
Dataset: To measure the performance of our optimization, we can use benchmarks like SPEC JVM (https://www.spec.org/jvm2008/). We can also just use popular open-source Java projects, like Hadoop[3].

Getting Started

We have researched literature to find open-source optimization codes to build our work on. We have not found an appropriate one yet, which prevents us from getting started. We may need to implement existing work if we can not find their code.

Reference

[1] Fu, Qiang, Jieming Zhu, Wenlu Hu, Jian-Guang Lou, Rui Ding, Qingwei Lin, Dongmei Zhang, and Tao Xie. "Where do developers log? an empirical study on logging practices in industry." In Companion Proceedings of the 36th International Conference on Software Engineering, pp. 24-33. ACM, 2014.
[2] Wikipedia LLVM http://en.wikipedia.org/wiki/LLVM
[3] Apache Hadoop http://hadoop.apache.org/
[4] Santosh G. Abraham, Rabin A. Sugumar, Daniel Windheiser, B. R. Rau, and Rajiv Gupta. 1993. Predictability of load/store instruction latencies. In Proceedings of the 26th annual international symposium on Microarchitecture (MICRO 26). IEEE Computer Society Press, Los Alamitos, CA, USA, 139-152.
[5] Lipasti, Mikko H., Christopher B. Wilkerson, and John Paul Shen. "Value locality and load value prediction." ACM SIGOPS Operating Systems Review 30.5 (1996): 138-147.
[6] Yonghong Song; Kalogeropulos, S.; Tirumalai, P., "Design and implementation of a compiler framework for helper threading on multi-core processors," Parallel Architectures and Compilation Techniques, 2005. PACT 2005. 14th International Conference on , vol., no., pp.99,109, 17-21 Sept. 2005
[7] Winkel, Sebastian, Rakesh Krishnaiyer, and Robyn Sampson. "Latency-tolerant software pipelining in a production compiler." In Proceedings of the 6th annual IEEE/ACM international symposium on Code generation and optimization, pp. 104-113. ACM, 2008.