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/

Major Changes:

As discussed in the meeting immediately after the project proposal, the focus of our project changed majorly from what we initially written. Currently, our goals are:

- Use machine learning to predict the probability that our program goes down each path
- Predict where we can move loads back, as to hide their latency
- Combine information from our two steps to move loads back to where they are 'probably' anticipated

What You Have Accomplished So Far:

We've divided up our project into the two parts. We've been implementing the code motion in LLVM. And we've been trying to use Soot to profile Java programs to train the classifier that predicts branch.

As for the code motion part, we've already determined the algorithm we use. In order for a load at position A to be moved to earlier position B, we would require that:

- Position A dominates position B
- Position B is very likely to postdominate A (using the metric denoting "how likely" the branch is taken)
- The operand of the load instruction has not changed
- Position A and B are not too far (as to prevent register pressure)

Our code is not fully functional yet, but we've already been able to run passes that print out our 'very likely to postdominate' metric (when given branching information). We've already prepared some test code that could potentially improve from moving loads back.

As for profiling and training, we have automatically instrumented example Java classes to print out runtime branch-taken info. For example, an instrumented example Java program prints out "!Goto! AVeryRareCase == 0", meaning after evaluating the condition "AVeryRareCase == 0" the go-to branch was taken instead of the fall-through branch.

Revised Schedule/Resources Needed:

Wenlu is working on the profiling and training part. Her next step is to instrument and profile real-world java programs, and to train the classifier. If time allows, she is also going to explore more features like names of variables that the branch conditions depend on. This might involve a new data flow similar to copy propagation.

Jiyuan and Alejandro are focusing on the code motion part. Once we get it running on small example, we will test it on the benchmark.