1 Project Description

Fetching instructions from memory only as they are needed may cause unnecessary delays due to the cache/memory speed difference in modern memory hierarchies. Instruction prefetching mitigates this delay by preemptively requesting blocks of potential future instructions from memory, even if they may not be executed. One cannot simply prefetch all instructions, as the instruction cache is limited and doing so may result in more cache misses. Note that Instruction prefetching is complementary to data prefetching because modern architectures maintain separate instruction and data caches, meaning the two processes are unlikely to substantially interfere with one another.

A simple algorithm for instruction prefetching is next-$N$-line prefetching \cite{Smi78,Smi82}, in which the next $N$ lines of the program are prefetched at a given program point. One limitation of this approach is that it fails to account for possible non-sequential access patterns. More sophisticated algorithms have been developed to address this \cite{LM98}, but next-$N$-line prefetching has been shown to improve the performance of the code.

This project will combine machine learning techniques as well as the LLVM IR interface to answer the following questions for next-$N$-line prefetching:

1. Which features of the LLVM IR are most relevant to instruction prefetching?
2. On a given program, what value of $N$ is optimal?
3. Which future execution paths should be considered for prefetching?

Evaluation Metrics Our evaluation metric should be the wait-time of the program on the instruction cache. Modern architectures do not support instruction caching, so we will not empirically measure the runtime. We intend to build a simulator to capture the expected wait-time based on the instruction cache behavior.

Goals Our ideal scenario (125% goal) is to implement an instruction prefetching model that substantially improves performance of the cache miss ratio on the test benchmarks. Our realistic goal (100%) is to train a model that improves performance on the training set and to understand the reasons it performs poorly on the test set. Our pessimistic goal (75%) is to implement instruction prefetching and to gather training data that does not fit well to our learning model.

*srallen@cs.cmu.edu
†bamos@cs.cmu.edu
2 Schedule

2.1 Plan of Attack

Our approach is as follows. We will use the Intel PIN dynamic instrumentation tool [LCM+05] as an instruction cache simulator so that we can estimate wait-times on the instruction requests. One possibility is to modify a third-party PIN instruction cache library [Rab05]. Next we will implement two LLVM passes to perform instruction prefetching. One pass will be without machine learning in order to ensure that the prefetching has an effect on the training code. The second pass will implement the model that we intend to machine learn. After that, we will gather test data to determine the optimal values of $N$ on our test programs. Finally, we will implement regression to develop a model for predicting the optimal value of $N$ on the test benchmarks.

Our anticipated schedule is as follows:

- **March 26:**
  - Get PIN simulator up and running.
  - Obtain benchmarks to use as both training and test data
  - Run the cache simulator on the benchmarks and ensure the results are reasonable.

- **April 2:**
  - Implement next-$N$-line prefetching in LLVM without machine learning.
  - Test prefetching efficacy for various values of $N$ both to ensure that it has an effect and to observe the overall trend.
  - Obtain new benchmarks or find a different algorithm if necessary.

- **April 9:**
  - Implement the extraction of features on the control flow graph in LLVM.
  - Determine the optimal value of $N$ for data in the training set.
  - Run regression to output a model for determining the best value of $N$.
  - Ensure the model preforms well on the training set.

- **April 16:**
  - Find optimal values of $N$ for test benchmarks and determine the effectiveness of the model
  - Make changes as necessary depending on the results.

- **April 30:**
  - Interpret results
  - Write Report
  - Make poster

2.2 Milestone

We have decided to push a good deal of the work for this project before the milestone. This is because one of us will be traveling to conferences between April 17 and May 3. As a result, we want to finish the portions that are not easily accomplished remotely before this date. This is a little bit ambitious, but our goal is to have at least an initial version of the code written by April 16th, after which we can make improvements as necessary. We expect that a few iterations will be necessary.
3 Literature Search and Resources

This work stems from early literature in instruction prefetching [Smi78, Smi82]. While many articles on prefetching suggest the use of machine learning as a future direction, we were unable to find any implementation of such a system.

We will be able to obtain all of the necessary resources for this project. We intend to run LLVM and the instruction cache simulator on a desktop machine. Furthermore, both LLVM and the cache simulator are open-source and readily available. We will use an undetermined open-source implementation for the machine learning portion of the project. This will be decided once we have a better idea of what our classifier should be.

4 Webpage

The webpage for this project can be found at [http://www.cs.cmu.edu/~srallen/project.html](http://www.cs.cmu.edu/~srallen/project.html)

References


