Finding optimal cache replacement policy

Problem

A computer cache is an essential piece of hardware which is responsible for limiting slow DRAM(memory) accesses and using the principles of temporal and special memory locality to bring *likely* important information on chip for fast access by the processor. Since a cache has very small storage capacity compared to DRAM, there are carefully picked *replacement policies* which caches use to determine which memory blocks are beneficial to keep on chip and which blocks can be overwritten for new information.

It is very difficult to decide what memory should be kept on cache and what can be evicted. The optimal solution is actually a greedy algorithm which discards information not needed for the longest amount of time. However, this is impossible to achieve in practice because it requires knowing future memory accesses. There are many policies existing already, most common one being LRU (least recently used).

Data

Since it is easy to generate data (memory trace files), and the ground truth greedy solution (*Bélády's algorithm*) is also computable, the applicability of using deep learning to assist replacement policies seems like a suitable fit. However, there are many hurdles, such as generalizing a replacement policy which will perform decent on all diverse kinds of programs. Another hurdle will be training a model with billons of lines of memory access information just for the sake of a single program and applying it on many others. The primary source of data for this project will be a memory trace. Each memory trace line will compose of (1) operation (store or load), (2) the address of the said memory block, and (3) the number of instructions that have passed from previous operation. This will make it easy to evaluate the expected number of cycles it takes to finish the specified set of instructions, making assumptions about the overhead of memory access.

Approach

[1] Suggests using a feed-forward network to assign probabilities of being used to each memory chunk and replaces in an LRU manner from those probabilities. On top of assigning probabilities to the memory locations, it may be beneficial to also model the exact cache set (row) and way(column) eviction/replacement trends. Finally, it may be helpful to develop a model which can pick the best existing replacement policies, an application which could tolerate the enormous overhead of computations required to get a model's prediction.

Since going over all SPEC test traces would sum up to contain over a trillion lines of code, it will require careful subsampling and careful inference of cache states in order to test performance over a random point in a given test.

Implementation of such a model would not be worth its overhead in any current cache hierarchy, unless the model performs significantly better than existing policies and would require that a cache miss would amortize the overhead of this computation.

Evaluation

The baselines of the performance of the LSTM will be the LRU policy, and the traces used will be directly using SPEC 2017 tests, which are the de facto tests for CPU simulations in academia. I will be using a trace based simulator called *ChampSim*[2] to evaluate the results of these baseline tests, and use the developed model to improve performance. While ChampSim can handle existing policies, I will have to write a manual script to evaluate the performance of the model on the given data.

References

[1] **Feedforward Neural Networks for Caching: Enough or Too Much?,** Fedchenko, Neglia , Ribeiro
http://www-sop.inria.fr/members/Giovanni.Neglia/fnn_caching.pdf

[2] **ChampSim**
https://github.com/ChampSim/ChampSim