Software Prefetching in LLVM

Nathan Snyder (npsnyder@andrew.cmu.edu)

Q Hong (qhong@cs.cmu.edu)

http://www.cs.cmu.edu/~qhong/course/compiler/

Project Description:

Our project will explore a variety of uses for software prefetching of data into caches. The successful application of software prefetching can greatly reduce the number of cache misses that occur while a program is running, which is important because cache misses are one of the major causes of inefficiency in modern architectures. Despite its benefits, software prefetching cannot be used rashly because if the prefetched memory is already in cache or turns out not to have been needed then the prefetch instruction accomplishes nothing and wastes a small amount of time. Potentially more significant is the danger of evicting useful data using a prefetch. Therefore, the one of the main focuses of our work will be to identify when prefetching is beneficial and when it isn't so that we can apply this tool appropriately.

We will work within the context of the LLVM compiler, which supports prefetching through an intrinsic function in the IR that is translated into architecture specific prefetching instructions in the back end. We will implement some fairly safe applications of prefetching in which it is highly likely that the data we are requesting is going to be used soon and also currently absent from the cache, as well as some more aggressive techniques that we will have to analyze empirically to determine whether they are actually helpful. The exact list of prefetching techniques we employ will change based on our experience during the project, but there are three areas we have currently highlighted.

1. Automatic prefetching of the next nodes during traversals of recursive data structures. This will involve automatically identifying what data structures fit this model, where they are being traversed, and where the best place to insert the prefetch instruction is.

2. Prefetching data in loops that iterate over arrays. It is very common for programs to have a loop that iterates over the elements of one or more large arrays performing some computation. In these cases the memory that is about to be used is very predictable, so we can insert prefetches for the data that *upcoming* iterations of the loop will need.

3. There are cases in which some memory address will only be need in one branch of a program. Depending on the details of the situation it may be beneficial to prefetch that data prior to making a branching decision, with safety due to prefetch instructions not raising exceptions.

The most interesting and clearly beneficial of these ideas is the prefetching of recursive data structures, so we will make sure to complete this aspect of the project even if work proceeds more slowly than expected. If we accomplish everything set out here early we can further explore aggressive prefetching of possibly unnecessary data.

Plan of Attack and Schedule

Week 1 (March 17 – 24)

Both: Read papers and create or find test cases which contain realistic opportunities for our optimizations.

Week 2 (March 25 – 31)

Both: Write code which identifies traversals of recursive data structures

Week 3 (April 1 – April 7)

Nathan: Write code which places prefetch instructions for recursive data structures.

Q: Implement the identification of array traversal loops.

Week 4 (April 8 – April 14)

Nathan: Try different prefetch placement policies for recursive data structure to find the best one.

Q: Write prediction of what memory loops will access in future iterations and prefetch insertion.

Week 5 (April 15 – April 21)

Q: Experiment with prefetch placement policies for array traversals.

Nathan: Insert prefetches for conditionally used memory locations.

Week 6 (April 22 – April 27)

Both: Collect detailed performance data and organize results into a written report

Milestone

By April 14th we plan to have implemented prefetching in traversals of recursive data structures.

Literature Search

Professor Mowry has directed us to the following papers related to what we intend to do

C.-K. Luk and T. C. Mowry. "Compiler-Based Prefetching for Recursive Data Structures." In Proceedings of ASPLOS-VII, Oct. 1996, pp. 222-233

C.-K. Luk. "Tolerating Memory Latency through Software-Controlled Pre-Execution in Simultaneous Multithreading Processors", In Proceedings of ISCA, May 2001, pp. 40-51

In addition, the following papers look like potentially useful sources of information

M. H. Lipasti, W. J. Schmidt, S. R. Kinkel and R. R. Roediger, SPAID: Software Prefetching in Pointer- and Call-intensive Environments. In Proceedings of the 28th Annual ACM/IEEE International Symposium on Microarchitecture, 1995, pp 231-23

S. Choi, N. Kohout, S. Pamnani, D. Kim, D. Yeung. "A General Framework for Prefetch Scheduling in Linked Data Structures and Its Application to Multi–Chain Prefetching", ACM Transactions on Computer Systems, Vol. 22, No. 2, May 2004, Pages 214–280.

Resources Needed

We will be using the LLVM compiler and our own personal x86 machines for this project. We should be able to begin immediately.

Getting Started

Thus far we have identified several papers on what we are trying to do, as well as what features of the LLVM system will be of use to us.