Dynamic Optimization Framework - CDOF

1 Aims and Objectives

I am proposing to implement a dynamic optimization framework for C. Specifically, I am proposing a subset implementation of an existing system called *ADORE*, as described by [1] in their paper 'The Performance of Runtime Data Cache Prefetching in a Dynamic Optimization System'. Design decisions for my implementation are also inspired by the runtime framework *Dynamo*, as described in [2].

2 Proposed Implementation Features

My implementation, titled CDOF, will work to dynamically optimize C code during runtime. CDOF is a shared library for Linux that is dynamically linked to the application at startup. CDOF will contain the following implementations:

1. Identify delinquent loads using Intel's profiling API, *PCM* [3].
2. Component for Data Reference pattern detection:
    - Indirect pointer identification
    - Direct reference patern identification
3. Component for managing the optimized trace buffer
    - e.g Allocating and generating the optimized traces
4. Component for linking original program binary to optimized code segments in trace buffer.

3 Deliverable Methodology

The final deliverable to be submitted includes the standalone code repository along with a written report. This report will provide an overview of CDOF and present performance results through graphs.

I will use three benchmarks to evaluate the performance. The benchmarks will be executed in four different scenarios: using CDOF, using native hardware prefetching, using GCC's static prefetching, and with native hardware prefetching disabled. The three benchmarks will consist of:

1. A slow pointer-chasing benchmark

2. QuickSort implementation in C
3. Sparse Matrix-Vector Multiplication implementation in C

Deliverable Grading Criteria

- C:
  - Working CDOF implementation as described above.
  - Presentation of experimental results in graphical form.
  - Explanation of the code base and experimental results.
- B:
  - CDOF performs better on all benchmarks compared to executing with hardware prefetching disabled
  - CDOF yields better performance compared to GCC static software prefetch insertions on at least one of the benchmarks.
- A:
  - CDOF performs as well as, or better than, hardware prefetching on at least one benchmark.

4 Relevance to Computer Architecture

Throughout our study of computer architecture, we have learned a variety of optimization techniques that work to increase the overall performance of the machine. Such techniques include: cache memory that reduce access latency and their related cache optimizations (e.g., multilevel caching, replacement, and prefetch policies) that further reduce average data access latencies. Moreover, we learned about dynamic scheduling in the context of out-of-order processors that are able to better exploit instruction level parallelism in cases that otherwise prove difficult for static compilers.

The implementation of CDOF is related to both of these in that it is a type of dynamic scheduling but in software, and also works to reduce average access latency with software prefetch insertions rather than relying on hardware.

5 Learning Objectives

Throughout the course of the project, I aim to learn and strengthen the following skills:

- Develop a deeper understanding of the effect that indirect and pointer-based references have on parallelization, both at the hardware and software level.
- Learn how to use profiling tools to identify performance bottlenecks in software.
- Develop a basic understanding of software runtime optimizations, like those in the Java Virtual Machine.
- Develop large project management skills.

6 References

[1] Lu, Jiwei & Chen, Howard & Yew, Pen-chung & Hsu, Wei-Chung. (2004). Design and Implementation of a Lightweight Dynamic Optimization System. Journal of Instruction-Level Parallelism. 6.

[2] Bala, Vasanth & Duesterwald, Evelyn & Banerjia, Sanjeev. (2003). Dynamo: A Transparent Dynamic Optimization System. Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI). 46. 10.1145/349299.349303.

[3] Intel® Performance Counter Monitor (Intel® PCM): https://github.com/intel/pcm