15-740 Project Proposal: Instruction Prefetching and Object-Oriented Programs

An-Cheng Huang
<pach@cs.cmu.edu> 
Leejay Wu
<lw2j@cs.cmu.edu>

October 12, 1999

1 Introduction

Object-oriented programming, or OOP, exhibits behavior that differs significantly from that of traditional imperative non-object-oriented languages [4]; these differences range from shorter function lengths, to greater use of indirect jumps.

It is therefore reasonable to expect that traditional architectures well-suited for traditional programming may be non-optimal for programs written with such a dissimilar style.

One factor is that instruction cache misses occur significantly more often with OO-style programs [4]. This, combined with more frequent function calls, suggests that hiding the latency may improve IPC more so on these programs than with more traditional C programs.

One approach is that of cooperative prefetching [7], an improvement over completely hardware-based methods for instruction prefetching. These older methods include next-N-line-prefetching, target-line prefetching, wrong-path prefetching, and Markov prefetching, ([14], [15], [9], [6], all cited respectively in [7])).

2 Objectives

- To assess the relative impact of these prefetching schemes on C and C++ programs.
- To consider possible optimizations for C++ programs, such as predicting virtual function call addresses and prefetching appropriately [11], as time allows.

http://www.cs.cmu.edu/~pach/740/proj740.html
3 Logistics

3.1 Methodology

Our tests will be run on the SimpleScalar [1, 2, 3] simulator. For benchmarking purposes, the OOCSB suite and possibly other publicly-available C and C++ programs should suffice, such as those comprising a recent SPEC suite.

The first priority was ensuring that C++ compilation can actually be done with this simulator. We found that SimpleScalar supports a superset of MIPS-IV, and with some minor bug-fixing and the addition of libg++ is capable of supporting C++ development and testing [3].

3.2 Schedule

The following is only an estimated schedule.

<table>
<thead>
<tr>
<th>Week Beginning</th>
<th>An-Cheng</th>
<th>Leejay</th>
</tr>
</thead>
<tbody>
<tr>
<td>October 10</td>
<td>Finish proposal. Write web page. Build OOCSB suite on SimpleScalar.</td>
<td></td>
</tr>
<tr>
<td>October 17</td>
<td>Run initial simulations.</td>
<td></td>
</tr>
<tr>
<td>October 24</td>
<td>Implement simple prefetching techniques.</td>
<td></td>
</tr>
<tr>
<td>October 31</td>
<td>Benchmark selected programs on modified system.</td>
<td></td>
</tr>
<tr>
<td>November 7</td>
<td>Analyze results. Attempt C++-targeted optimiza-</td>
<td></td>
</tr>
<tr>
<td>November 14</td>
<td>Poster presentation.</td>
<td></td>
</tr>
<tr>
<td>November 21</td>
<td>Finishing touches on poster.</td>
<td></td>
</tr>
</tbody>
</table>

The entire path is critical, since it’s very linear. For this reason, we have not divided the tasks.
3.2.1 Milestone

We intend to have results showing that, I-cache prefetching improves C++ program performance to a greater degree than C, due to the typically higher miss rates. We also intend to attempt prefetching based on virtual function call prediction, followed by benchmarking.

3.3 Literature Search

In addition to papers cited above, various other papers were used to gain a brief overview of some of the issues involved with object-oriented programming. Predicting the targets of indirect branches is one approach [5, 11]. Another approach towards avoiding unneeded saves and restores was that of “dead” values [8]. Prefetching methods have been examined for saving memory latency when dealing with linked data structures [10, 13]. A variety of methods was considered in [12].

3.4 Resources

We need, and have the usage of, a big-endian machine on which to build SimpleScalar, along with libg++ and the cross-compiler.

We have a cross-compiled binary implementation of the SPEC95 suite.

We may need to forego sleep for prolonged periods of time.

3.5 Stuff So Far

We have obtained SimpleScalar, and built our development environment on a Solaris/SPARC server.

References


