Final Project Report

Phase Based Instruction Prefetcher

Submitted by
Ayaz Akram

ECE 5950

Department of Electrical and Computer Engineering
WESTERN MICHIGAN UNIVERSITY

Fall 2015

Abstract

This project proposes evaluation of an instruction pre-fetch technique that uses program phases to perform pre-fetching. Analysis of miss address traces of various SPEC CPU2006 benchmarks suggest that number of instruction cache misses and miss addresses correlate with the phase of program execution. Using this information, optimal prefetching decisions can be made. Due to low instruction cache misses in SPEC CPU2006 benchmarks, only few of them have been studied. A program phase based instruction prefetcher is also implemented in gem5 simulator. This implementation effort involved studying and understanding code of prefetching mechanism and memory heirarchy in gem5. Evaluation of this newly implemented prefetcher is performed by comparing it with an existing Tagged prefetching mechanism in gem5. Comparison for only three benchmarks could be performed. Phase based prefetcher works well for only one out of these three, while tagged prefetching gives lower number of cache misses for other two benchmarks. But, the number of prefetches generated by phase based prefetcher are always less than tagged prefetcher. However, different designs can be tested for this phase based prefetcher, which has the potential to give much better results.

Contents

List of Figures

1 Introduction

This project is an evaluation of an instruction pre-fetching mechanism that uses program execution phases to make prefetching requests to lower level caches or memory. Correct instruction pre-fetching can result in boost up of performance by hiding the instruction cache miss latency. Significance of instruction prefetching is increased in case of workloads (e.g. server workloads, cloud based software servicing) where process binaries are too large to be placed in caches. [1, 2] have previously proposed instruction prefetchers which use program context information like RAS status and committed instructions to detect future accesses.
This project involved two phases: (1) analysis of miss address traces of benchmarks and (2) implementation of a execution phase based prefetcher in gem5 [3] simulator. A subset of SPEC CPU2006 benchamrks were used for analysis and testing purposes due to availability of phase distribution information files for these benchmarks (work done by some other students). Although miss address traces for many benchmarks have been recorded, but only few of them are used for further analysis. For example, benchmarks like namd, gemsfd, mcf have extremely low number of instruction cache misses. Moreover, Omnetpp and H264ref though show quite a few instruction cache misses, required phase information files are not available for them. Next section talks about behaviour analyzed while processing the instruction cache miss traces.

2 Analysis of Miss Address Traces

First of all instruction cache miss address trace files were processed using the available phase distribution of benchmarks with the number of executed instructions. Figure 1, 2, 3 and 4 show the existing correlation between number of instruction cache misses and program phases. These misses are seen at L1 instruction cache. Bzip and Mcf have very low number of instruction cache misses, but still show some correlating/repeating behaviour.

Figure 1: Correlation of I$ misses with phases for gcc

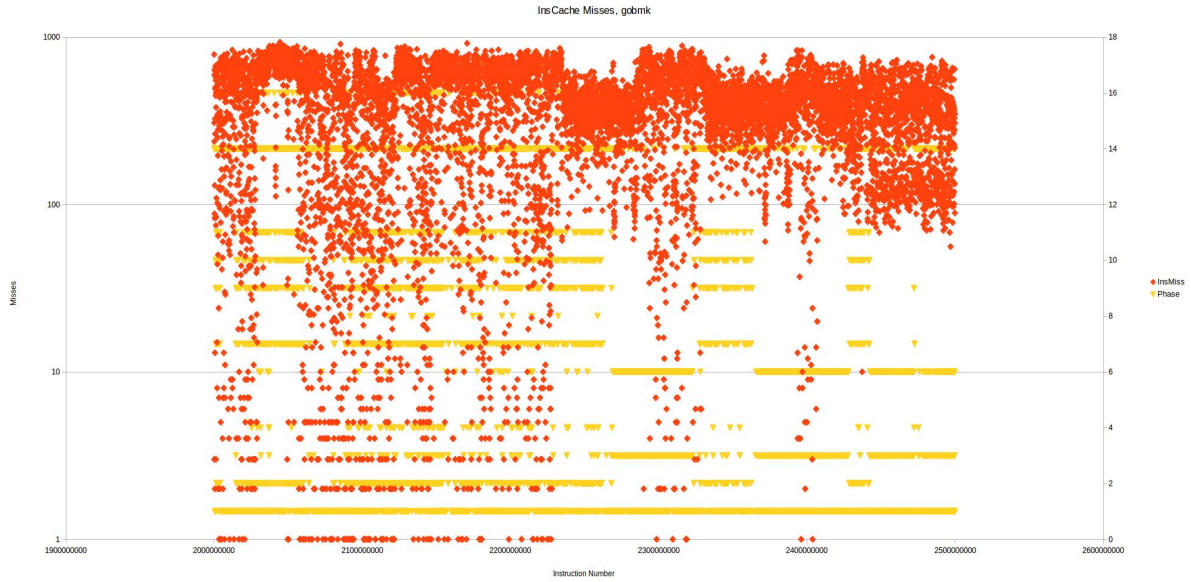

Figure 2: Correlation of I$ misses with phases for gobmk

Next, I analyzed to see if the addresses of instruction cache misses also show some sort of correlation with program phases. For this study, very large miss adress containers/buffers are assumed. As a particular phase executes/repeats any newly seen miss address is put in a buffer corresponding to that phase. But this address is counted as “not covered” miss address. In this way at the end, fraction of those miss addresses is counted which have been previously seen. This fraction is depicted in Figure 5, 6 and 7 respectively for three benchmarks. High fraction of previously seen addresses, point to existing correlation between miss addresses and program phase.

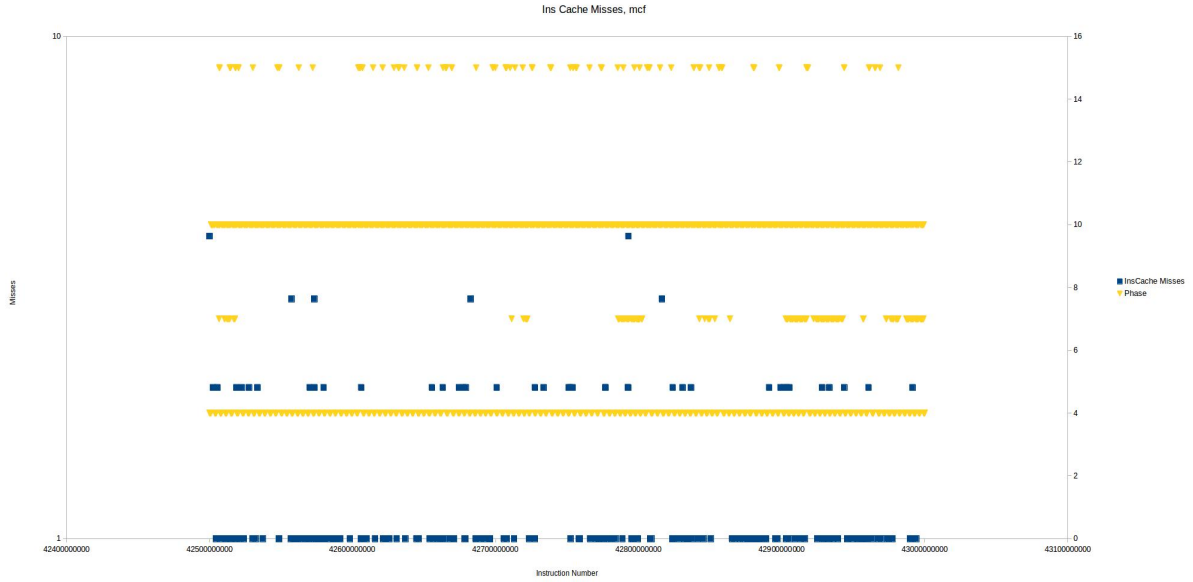

Figure 3: Correlation of I$ misses with phases for mcf

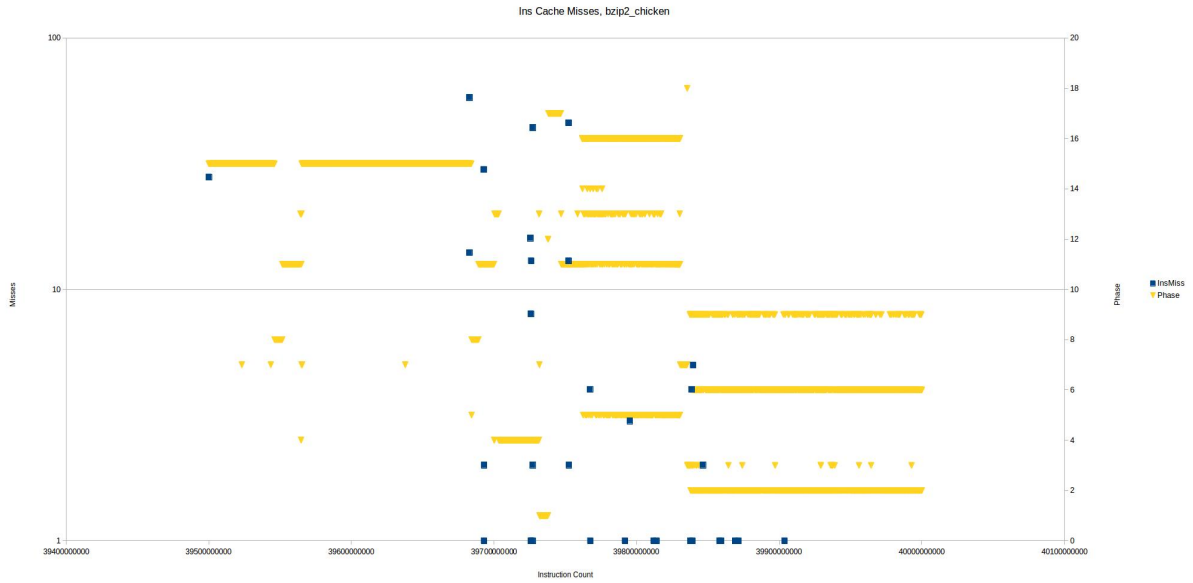

Figure 4: Correlation of I$ misses with phases for bzip

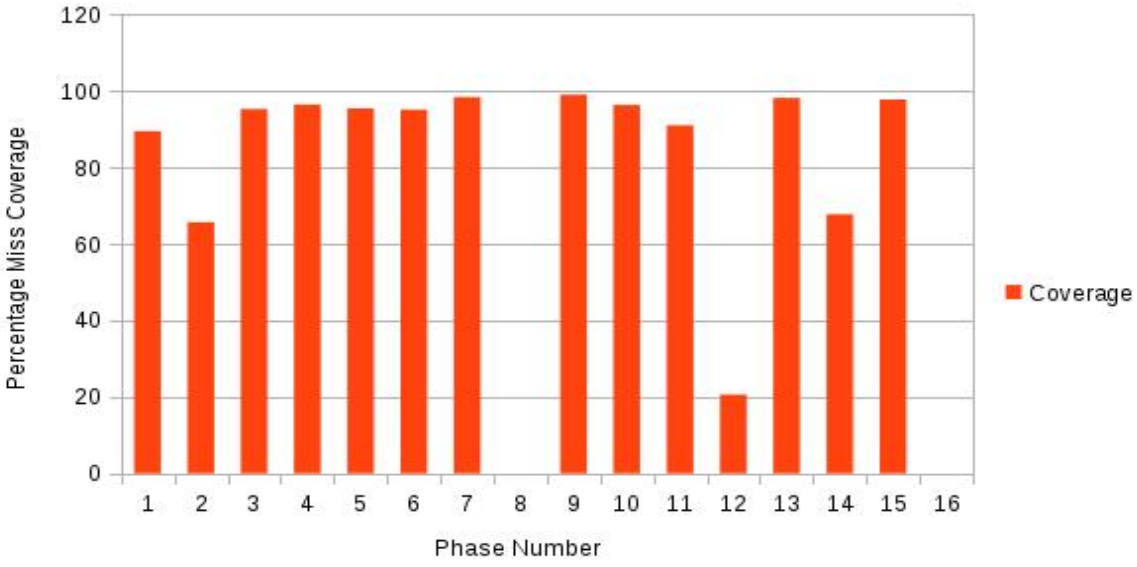

Figure 5: Fraction of seen I$ miss addresses for gcc

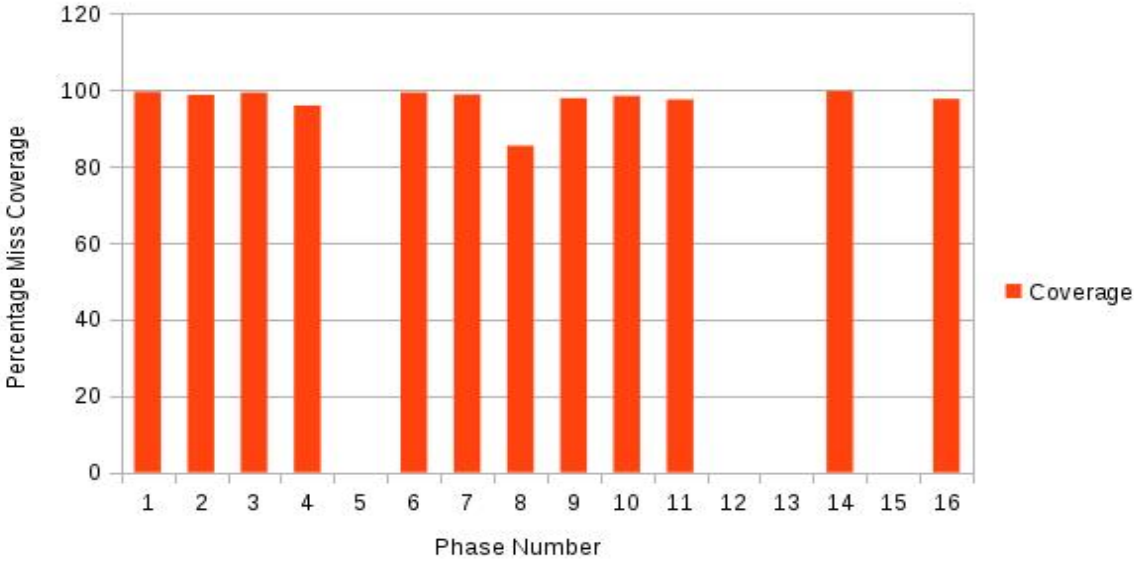

Figure 6: Fraction of seen I$ miss addresses for gobmk

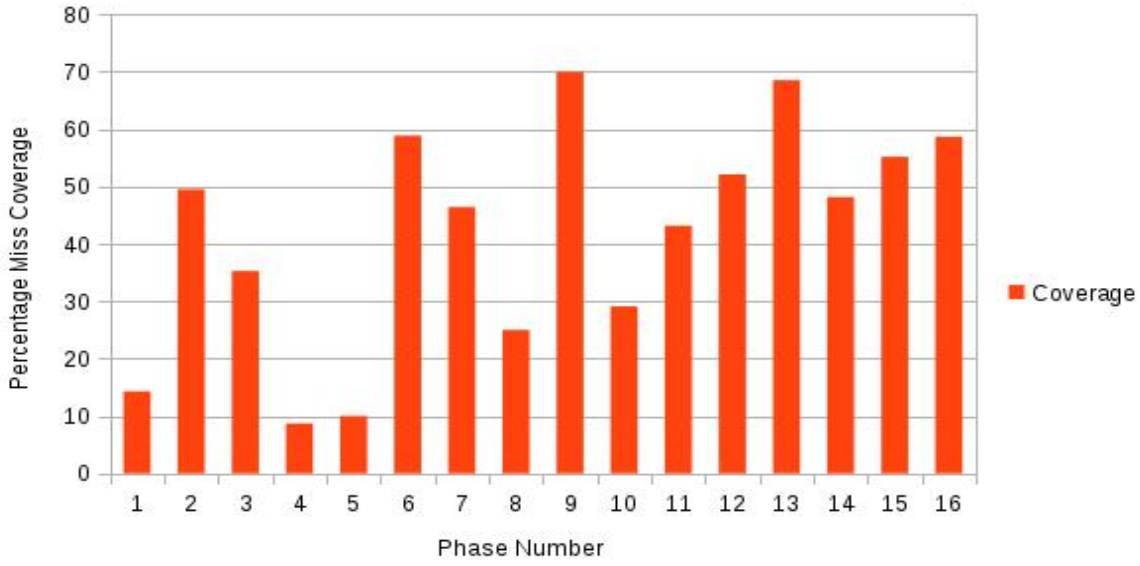

Figure 7: Fraction of seen I$ miss addresses for soplex

3 Implementation of Prefetcher in gem5

Next, a new prefetcher is added in gem5 which uses previously seen miss adresses for a particular phase and prefetches accordingly. Gem5 has recently got an improved version of supported prefetchers: tagged and stride prefetcher. Since, prefetch requests are serialized and then put into simulated MSHR (miss status holding register), new added phase prefetcher extends the class of already implemented queue prefetcher which requests the phase prefetcher to generate prefetch requests on notification event (instruction cache miss in our case). In order to implement this functionality a thorough study of code was done and finally some changes were put in different parts of code base. The prefetcher added to gem5, is notified on all instruction cache misses. On notification of a miss, prefetcher looks into history buffer corresponding to active phase of execution. If address is found, next d (where d is equal to configured degree of prefetcher) addresses from the buffer are forwarded as prefetch requests. If address is not found, no prefetch request is forwarded. This prefetcher can also work as a hybrid of next line and phase buffer prefetcher.

4 Results of few benchmarks

Three benchmarks were run, to test the working of the simulator and compare its performance with one of the existing prefetchers of gem5 namely tagged prefetching. I ran numerous simulations with varying parameters associated with prefetchers to get idea about effect of these parameters on performance of prefetcher. The final results shown here are run with configured prefetcher degree value of 6 and prefetch queue size of 16 entries.

Figure 8 shows the percentage reduction in instruction cache misses for both tested prefetchers. This is achieved reduction over the base case of no prefetcher. Soplex has very low number of instruction cache misses so, probably that is not a very good comparison point, but other benchmarks results show that tagged prefetcher does better than current implementation of phase based prefetcher. FIgure 9 shows the generated prefetches by both the prefetchers. Tagged preftcher always issue more prefetches, which means it should be consuming more energy for those prefetch request issues.

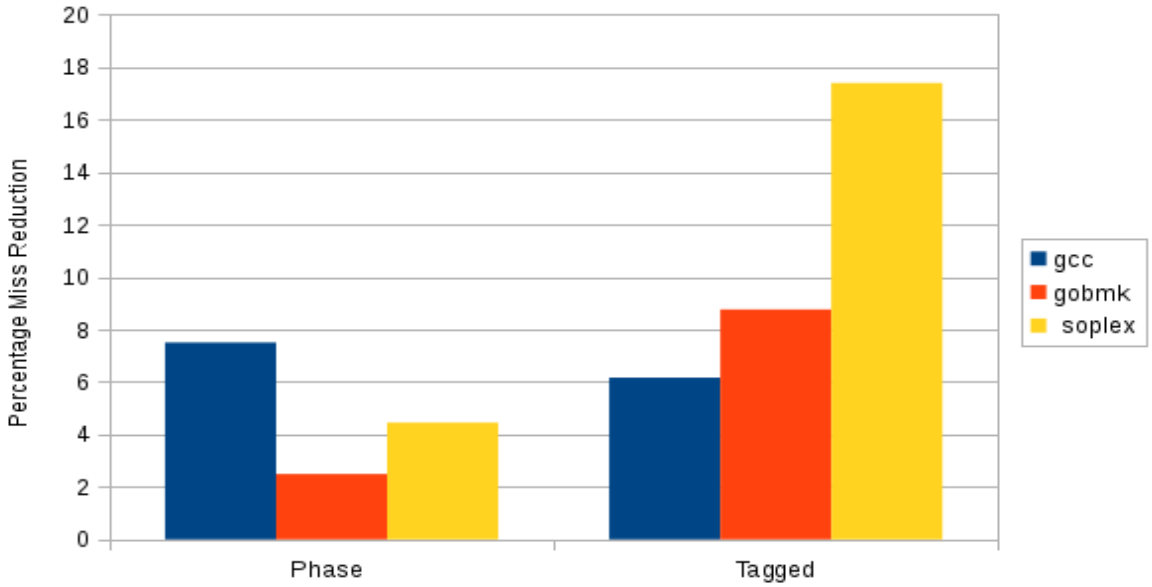

Figure 8: Percentage Reduction in total I$ cache misses

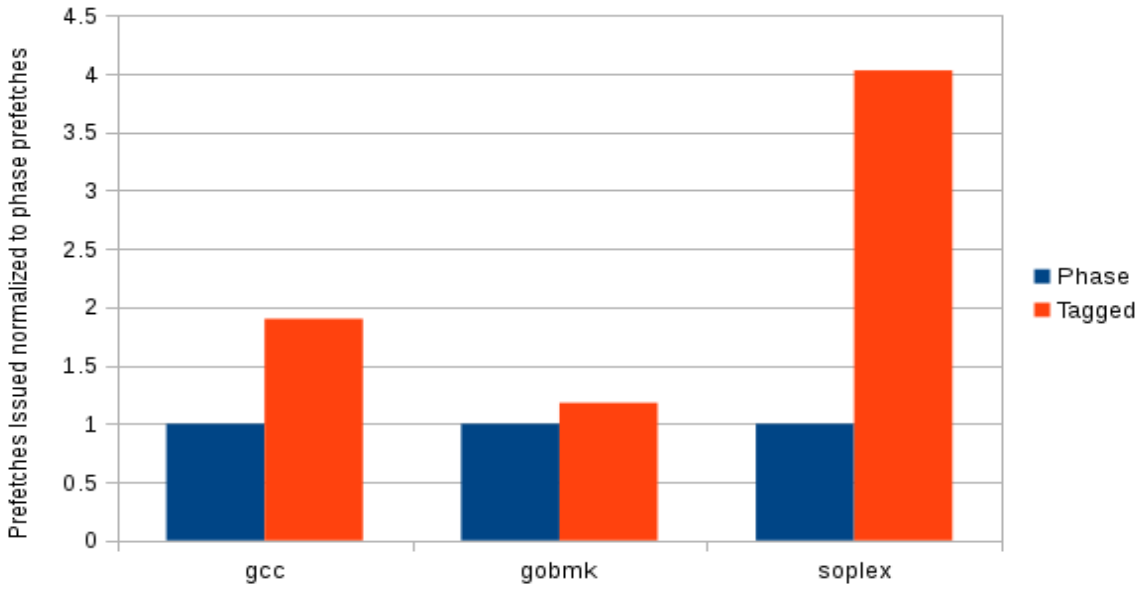

Figure 9: Number of prefetches issued normalized to prefetches issued by phase preftcher

5 Conclusion and Future Work

This project demonstrates the potential of program phase based prefether design. A base prefetcher is setup in gem5. Various optimizations can be applied and tested there to get a better performing final prefetcher. Secondly, some server type workloads need to be tested so that comparison with other recent published work can also be made.

References

[1] A. Kolli, A. Saidi, and T. F. Wenisch, "Rdip: return-address-stack directed instruction prefetching," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 260–271, 2013.

[2] M. Ferdman, C. Kaynak, and B. Falsafi, "Proactive instruction fetch," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 152–162, ACM, 2011.

[3] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, *et al.*, "The gem5 Simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, pp. 1–7, May 2011.