Prefetching With Cluster Simulation

Håkon SVANE MELLBYE, Kristian RYGH JERNDAHL, Marton LEREN TEILGÅRD, Hilde Marie SCHADE

TDT 4260 - Computer Architecture
Department of Computer and Information Science
Norwegian University of Science and Technology

Manuscript delivered April 23, 2012.

Abstract: This paper presents a simple implementation of a Delta Correlating Prediction Table, DCPT, prefetcher algorithm. The algorithm is based upon the paper Storage Efficient Hardware Prefetching using Delta-Correlating Prediction Tables, written by M. Grannaes, M. Jahre and L. Natvig. The paper details the results produced from simulating the algorithm in a L2 cache using the SPEC CPU2000 benchmarks. We discuss the results, and evaluate the performance of the system, while illuminating a few basic concepts for measuring the performance of a prefetcher. As an addition, we also compare the DCPT prefetcher design with a Czone Delta Correlation design, and discuss our implementation of the two designs.

Index Terms: prefetching, CDC, cluster simulation, DCPT

1. Introduction

In our assignment we were supposed to find a prefetcher algorithm and implement it. With the still widening gap between main memory access time and processors cycle times, the need for fast on-chip caches is ever growing. However, the data requirements of typical programs is already exceeds the size of on-chip memory. This leads to high cache miss rates and subsequently worse performance. To make this difference less prominent, many architects have turned to prefetching. Prefetching is a data access latency hiding technique, where one tries to predict future data access, and store the needed data in cache before it is needed.

2. Related Work

2.1. ACDC - Adaptive Data Cache Prefetcher

From the article ACDC-Adaptive Data Cache Prefetcher written by Nesbit, Dhodapkar and Smith we learn about the prefetcher earlier mentioned, from now on known as ACDC. This is a prefetcher which uses concentration zones(CZones) that divide the memory into several blocks of same size. When it finds a access pattern within a CZone, it launches a prefetch requests. The upside of this method is that it does not need to use the program counter values for the load instructions that cause cache misses. Since the lower levels of memory often is far from the processor it is a time consuming deal to reach the

program counter. All the history from the CZone is stored in a so called Global History Buffer, a GHB. This is a FIFO structure that stores all the recent L2 cache misses in the order that they occur and the miss addresses within the same CZone is stored in a time-ordered linked list. The positive thing about this about FIFO queue is that it gives a natural priority to recent programs and eliminate "'stale"' history by evicting the oldest miss address history. With the GHB, longer the address streams are kept for the active miss streams, while the inactive ones age away.

This prefetcher uses delta correlations to prefetch miss addresses, with a fixed pattern between them. The delta is the distance from one address to another. For example if we have the addresses :

0 - 8 - 40 - 64 - 72 - 104 - 128

we will have the deltas

8 - 32 - 24 - 8 - 32 - 24.

If the program counter values of the load instructions were available, then tree constant stride patterns could be extracted from the address stream. But when the program counter is not available the strides would not appear.

The general opinion is that the prefetchers performance is sensitive to the size of the CZone and the optimal size depends on what kind of program it is. As a rule of thumb is that the CZone size should be about the same size as the data structures that is being accessed. For the best prefetcher performance the degree should also be adjusted.

2.2. DCPT - Delta-Correlating Prediction Tables

Another article that is related to this is the "Storage Efficient Hardware Prefecthing using Delta-Correlating Prediction Tables", written by Marius Grannaes, Magnus Jahre and Lasse Natvig. In this article they write about the Delta-Correlating Prediction Tables(DCPT). This prefetching algorithm is based on the Reference Prediction Tables(RPT) and the Program Counter/Delta Correlation Prefetching(PC/DC).

The RPT table is a table consisting of the program entries: a program-counter, last address, delta and state. When an address causes a miss, it will make an index to the table. The first time a load instruction triggers a miss, it will reserve a table entry, and if necessary evict one of the old ones. The miss address is stored in the last address-field, and the state is set to initial. The second time the instruction causes a miss, the last address will be subtracted from the new miss address, and a delta will be stored.

The PC/DC prefetcher is described by Nesbit and Smith, and is very similar to the ACDC. The difference between the two is that the PC/DC does not use CZones. In PC/DC the different entries in the GHB are tagged by the instruction that called it, and the pointer to the most recent miss by said instruction is stored in a index table. The AC/DC is indexed by CZones, zones of the address space, and does not utilize any relation to the instruction that called it.

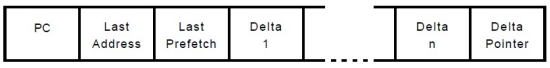

Figure 1. A DCPT Entry

These methods are the basis of the DCPT technique. The prediction table(PT)

of the RPT is changed to contain a delta buffer instead of a single delta, and the entry table is as seen in figure 1. The addresses are indexed by the program counter of the load instruction. The deltas can be stored in a circular buffer, with the last n deltas requested by the load instruction and the delta pointer pointing to the head of the buffer. Like the PC/DC prefetcher the deltas are found by comparing the last address and the current address, and the buffer is only updated if the deltas is found to be non-zero. Then the new delta is inserted into the delta buffer and the last address is updated. Then the delta correlaton compare state machine from the AC/DC method searches through the delta buffer looking for a recognizable pattern. If there is a match the prefetching addresses will be calculated from the last address and the stored deltas in the buffer. To find the best working prefetcher for a given system we have to find a table size and a prefetching degree that gives good results.

3. Prefetcher Description

The design used is a delta correlation prediction table(DCPT) prefetcher. DCPT is a prefetching algorithm based on a combination of RPT and PC/DC prefetching algorithms. Like reference prediction table(RPT), DCPT is based on a PC indexed table. The table also contains a field for the last address that caused a miss, and n delta fields that store the last n deltas between adresses accessed by the PC.

If a miss occurs, the algorithm will search the table for a corresponding program counter, if one is not found, a new table entry is created. If the PC already exists in the table, the distance between the current address and the last address is calculated, and the delta is stored in the delta buffer. The table, which is a FIFO queue, is then updated so that the recently accessed PC is at the back of the queue, with other words the most recently used prediction table will be the last one to be removed of the PT's currently in the queue. After storing the delta, if the delta buffer contains more than two deltas, the algorithm attempts delta correlation. The delta buffer is then traversed in reversed order, and searched for matches to the two most recently inserted values.

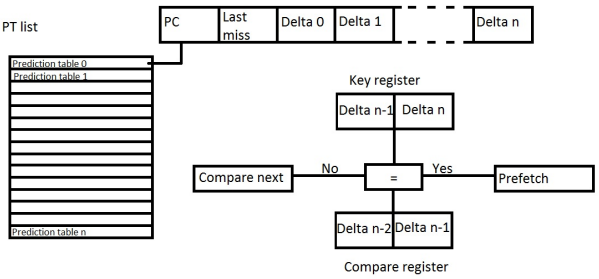

Figure 2. A visual description of the DCPT tables and entry layout, and the comparing state machine in its first compare cycle

Once a matching pair of delta values is located, prefetching will commence. Prefetches will be issued one by one, starting with the current miss address + next delta, starting from the one before the matching pair. This means if the matching hit is with delta[m], delta[m-1] the first address prefetched will be current miss address + delta[m-2]. The second address prefetched will be previous address(current miss address + delta[m-2) + delta[m-3]. The prefetcher will continue this way until we have prefetched a number of addresses given by the prefetching degree. If we should reach the beginning of the delta buffer before we reach the given prefetching degree we will start over from delta[m], e.g. the next address prefetched will be previous address + delta[m]. When we have an address ready for prefetching, we first check that the address is not in the cache, queue or not within the address range. only

if these checks pass, will the prefetch be issued.

4. Methodology

The prefetcher is designed to be a L2 cache prefetcher in a simulated computer architecture. The architecture is loosely based on a DEC Alpha Tsunami system. The main component in the system (Alpha 21264 microprocessor), is a superscalar, out of order CPU. The L2 cache has a size of 1MB and a block size of 64b.

The prefetch algorithm will serve as method for the cache controller to decide what to prefetch. All requests are added to a MSHR (miss status holding register) queue, and are actually performed by the cache controller. This allows the cache controller to identify and ignore duplicate prefetch requests.

The simulated computer system is tested trough a benchmark scheme known as SPEC CPU2000. This is a series of tests designed to test the performance of a computer system by measuring various parameters during the execution of the test algorithms. Most of the tests are advanced mathematical computations that stress the CPU. Judging by the results of the prefetcer on the different tests, the tests are designed to test different areas of the cpu.

To quantify the performance of the system, certain terms are used:

IPC (instructions per cycle) Using a superscalar architecture allows the possibility of IPC rates > 1.

Speedup Speedup is a simple formula that relates the system performance compared to that of a system without a prefetcher.

$$Speed = \frac{execution\_time_{(no\_prefetcher)}}{execution\_time_{(with\_prefetcher)}} \quad (1)$$

$$= \frac{IPC_{(with\_prefetcher)}}{IPC_{(no\_prefetcher)}} \quad (2)$$

Good/Bad prefetch Good/Bad prefetches are simply names for whether or not a prefetched block was referenced or not.

Accuracy Accuracy measures the ratio of good to bad prefetches.

$$Accuracy = \frac{Good\_prefetches}{Bad\_prefetches} \quad (3)$$

Coverage Coverage refers to how many potential candidates were actually identified by the prefetcher.

$$Coverage = \frac{good\_prefetches}{cache\_misses\_without\_prefetching} \quad (4)$$

Table I
DIFFERENT BENCHMARK TESTS

Amiable	Hostile	Indifferent
$applu$	$ammp$	$apsi$
$facerec$	mcf	art
$galgel$	$twolf$	$equake$
$lucas$	vpr	$fma3d$
$mgrid$		$mesa$
$swim$		$crafty$
$wupwise$		eon
$bzip2$		gcc
gap		$gzip$
$parser$		$perl$
		$vortex$

5. Results

We have tested two different prefetchers, a C/DC and a DCPT. As seen in figure 3(DCPTvsCDC), the difference in speedup between the two prefetching techniques is not too large. It's the ammp and applu tests where the difference is greatest, in the other test, except the wupwise, the speedup is centered around 1. The speedup is mostly achieved in the ammp and wupwise test and for the D/DC prefetcher one can see a clear slowdown on the applu and swim test. In figure 4, we can see the speedup of the DCPT prefetcher, with different sizes of prediction tables and prefetching degree. It's clear that the prefetching degree affects the result far more than the size of the prediction table, although we still see a speedup on almost

all DCPT prefetchers. Another observation is that with the prefetching degree set to 10 we have quite good results for the smaller prediction tables, but when the prediction table size exceeds 32 deltas, it suffers a setback in performance. While the prefetcher with the prefetching degree set to 10 suffers a setback when the table size exceeds 32, both higher and lower prefethcing degrees prefetchers experience a boost in performance. As seen in figure 6, high accuracy of the prefetcher don't clearly relate to a high speedup.

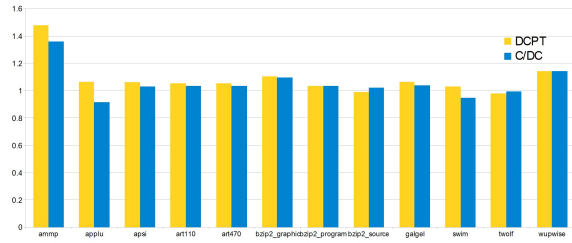

Figure 3. Difference between DCPT and CDC

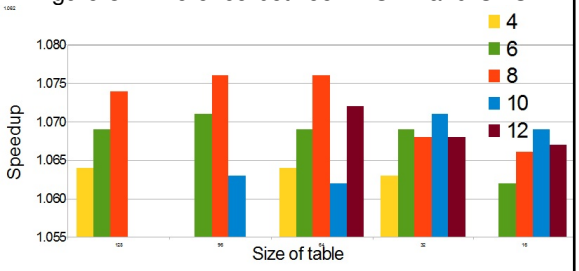

Figure 4. Table aggresivity with different prefetching degrees

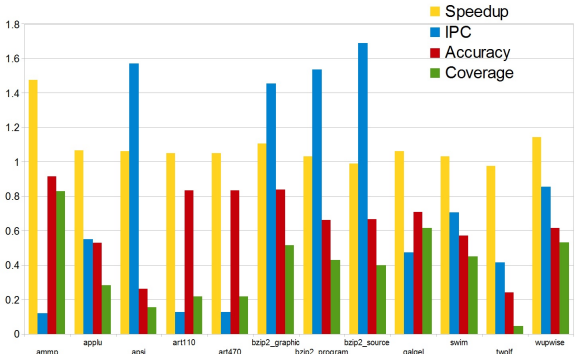

Figure 5. Results from dcp

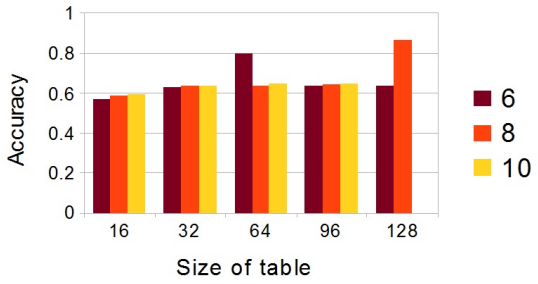

Figure 6. Accuracy with different prefetching degrees

Table II
CLUSTER SIMULATION RESULTS

	Speedup	*IPC*	*Accuracy*	*Coverage*	*Identified*	*Issued*
ammp	1.475	0.122	0.916	0.830	13938441	12620494
applu	1.066	0.550	0.529	0.283	9164909	1187347
appsi	1.062	1.570	0.261	0.153	328106	70548
art110	1.051	0.128	0.836	0.221	78128578	3636291
art470	1.051	0.128	0.836	0.221	78128578	3636291
bzip2_graphic	1.103	1.453	0.838	0.515	57907	57652
bzip2_program	1.034	1.534	0.663	0.429	36190	35756
bzip2_source	0.989	1.690	0.677	0.398	20041	19736
galgel	1.064	0.473	0.708	0.613	456229	283000
swim	1.030	0.705	0.572	0.450	3981911	1782333
twolf	0.977	0.415	0.241	0.049	212067	209581
wupwise	1.144	0.855	0.614	0.531	375412	375412

6. Discussion

The goal of the project was to implement a prefetching algorithm, and we found it best to implement an existing algorithm rather than creating our own from scratch. Initially, we settled on an implementation of an C/DC prefetcher, but eventually we changed to on a DCPT design that proved to be far more efficient.

During our testing, we tweaked our prefetcher by changing the size of the prediction table, and the prefetching degree. The best results were achieved by using a table size of 96, and prefetch degree of 8. According to figure 4, we see that by increasing the size of the PC indexed table, we increased the speed of the prefetcher. This is possibly because of an inherent issue with DPCT, a small table size leads to entries being evicted from the table before they are given time to calculate a decent amount of previous deltas. This will lead to low accuracy, as many possibly detected prefetches will be undiscovered.

Although we decided we got better results from DPCT prefetcher, the difference between the prefetchers in most tests were quite small, except for the Ammp test, where DPCT outclassed C/DC. We wanted to determine why the Ammp test was so drastically different, but after seeing the description of the test, we quickly realized this was outside the scope of this project.

Our total top scoring test, with a size 96 table, and a prefetch degree of 8, gave us a total speedup of 1.076. At the time of writing, this leaves us in 9th place. We've tested many variations in terms of table size and prefetch degree, and the 96_8 version does give the best results.

After reviewing the code, we discovered that there are no limitations to the amount of stored delta values. And that the amount of delta values in each entry will only be limited by the entry being removed from the table. However, after adding such limitations, we have been unable to find any setting that increase the speedup beyond 1.076. It also appears as if the prefetcher traverses the delta buffer in the wrong direction. However, after changing the loop, the new code was slower than the 96_6 version.

We realize that there are ways of creating better prefetchers than the one we currently use, however we spent a lot of time implementing the C/DC design. Unfortunately this design performed quite poorly on the Kongull cluster, and we decided to change implementation instead of introducing AC/DC to the C/DC algorithm.

7. Conclusions

We have during several weeks of work made a prefetcher that uses delta correlation prediction tables (DCPT) to predict addresses needed by the CPU. We started by creating an algorithm that utilized CZone delta correlation (C/DC) methods. We did get some speedup, but not a lot. We then tried prediction table(PT) and DCPT and got the best results from DCPT. The different methods also had several parameters that can be changed and adapted for different systems. For C/DC these parameters are number of zones, global history buffer(GHB) size and prefetching degree. We found that for this system the number of CZones had little to no impact. The GHB size had some impact in the extreme cases, very little or very big, but otherwise almost no impact. The big change here is the prefetching degree. We found the best combination at CZones = 16, GHB size = 256, prefetching degree 6. For the DCPT the parameters consist of list size and prefetching degree. The list size is the

number of prediction tables stored at any given time. In this case the list size had a comparable effect to the GHB size, and the prefetching degree was most important. The best results were found with list size = 96 and prefetching degree = 8.

The DCPT prefetcher rendered the best results of our many attempts with an average speedup of 1.076. We believe that this result could be improved if we were able to spend more time in perfecting the issues that arised during the last week of testing concerning delta buffer size and how the entries in the DCPT list are stored or thrown out.

References

[1] Marius Grannaes, Magnus Jahre and Lasse Natvig, *Storage Efficient Hardware Prefeching using Delta-Correlation Prediction Tables*. Department of Computer Science and Information Science, Norwegian University of Science and Technology, 2011.

[2] Kyle J. Nesbit, Ashutosh S. Dhodapkar and James E. Smith, *AC/DC: An Adaptive Data Cache Prefetcher*. Department of Electrical and Computer Engineering, University of Wisconsin.

[3] www.spec.org