REPORT

SOFTWARE AND HARDWARE PREFETCHING

ABSTRACT :

Prefetching methods for instruction caches are studied via trace-driven simulation. The two primary methods are "fall-through" prefetch (sometimes referred to as "one block lookahead") and "target" prefetch. Fall-through prefetches are for sequential line accesses, and a key parameter is the distance from the end of the current line where the prefetch for the next line is initiated. Target prefetches work also for nonsequential line accesses. A prediction table is used and a key aspect is the prediction algorithm implemented by the table. Fall-through prefetch and target prefetch each improve performance significantly. When combined in a hybrid algorithm, their performance improvement is nearly additive. An instruction cache using a combined target and fall-through method can provide the same performance as a two to four times larger cache that does not prefetch. A good prediction method must not only be accurate, but prefetches must be initiated early enough to allow time for the instructions to return from main memory.

To quantify this, we define a "prefetch efficiency" measure that reflects the amount of memory fetch delay that may be successfully hidden by prefetching. The better prefetch methods (in terms of miss rate) also have very high efficiencies, hiding approximately 90 percent of the miss delay for prefetched lines. Another performance measure of interest is memory traffic. Without prefetching, large line sizes give better hit rates; with prefetching, small line sizes tend to give better overall hit rates. Because smaller line sizes tend to reduce memory traffic, the top-performing prefetch caches produce less memory traffic than the top-performing nonprefetch caches of the same size.

INTRODUCTION :

Prefetching is a technique used by computer processors to boost execution performance by fetching instructions or data from their original storage in slower memory to a faster local memory before it is actually needed . Most modern computer processors have fast and local cache memory in which prefetched data is held until it is required. The source for the prefetch operation is usually main memory. Because of their design, accessing cache memories is typically much faster than accessing main memory, so prefetching data and then accessing it from caches is usually many orders of magnitude faster than accessing it directly from main memory.

PREFETCH TYPES :

There are two types of prefetches:

Hardware based prefetching is typically accomplished by having a dedicated hardware mechanism in the processor that watches the stream of instructions or data being requested by the executing program, recognizes the next few elements that the program might need based on this stream and prefetches into the processor's cache.

Software based prefetching is typically accomplished by having the compiler analyze the code and insert additional "prefetch" instructions in the program during compilation itself

EVALUATION METHODS

HARDWARE PREFETCHING

The hardware prefetching scheme is used to prefetch instructions via the use of additional hardware support, which increases the power consumption and the cost.

The scheme optimizes the instructions temporarily, i.e. It uses temporal locality.
This makes the particular scheme easiler to implement compared to the software prefetching scheme, which often requires the investigation of complex access patterns,

There are many hardware prefetchers available for instance : Markov Prefetchers, Stride Prefetchers, etc.

Our implementation analyzes Next Line prefetching (The easiest to implement), and Stride Prefetching.

The results show that around <to be written later> % decrease in the average execution time is observed

with hardware prefetching.

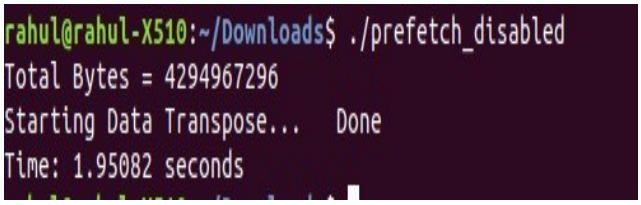
Evaluation :

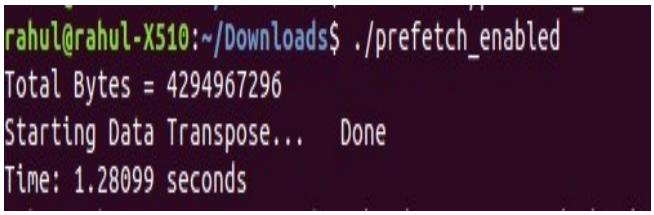
The evaluation of the prefetching scheme was done using 4 of the standard benchmarks/ traces :
(i) g++
(ii) grep
(iii) plamap
(iv) ls

The modules used for evaluation include :
1. CPU.C
2. Cache.C
3. MemQueue.C
4. Prefetcher.C
5. Auxilary modules

SOFTWARE PREFETCHING:

To evaluate the techniques intel's mmprefetch instruction is used which is supported by g++ compiler. It basically fetches the line of data from memory that contains the byte specified with the source operand to a location in the cache hierarchy specified by a locality hint.
We also used __m128d data type for memory blocks as it supports memory alignment.Compiler directed prefetching is used within loops with a large number of iterations. The compiler predicts future cache misses and inserts a prefetch instruction based on the miss penalty and execution time of the instructions. We calculate parameters with and without prefetch enabled.These prefetches are non-blocking memory operations, i.e. these memory accesses do not interfere with actual memory accesses. They do not change the state of the processor or cause page faults.So time taken is found to be significantly reduced by 20-25%.





CSN-221 Project

31-Oct-2019

https://github.com/techxpert99/project_prefetching_cs221

Project manager	
Project dates	04-Oct-2019 - 31-Oct-2019
Completion	100%
Tasks	6
Resources	6

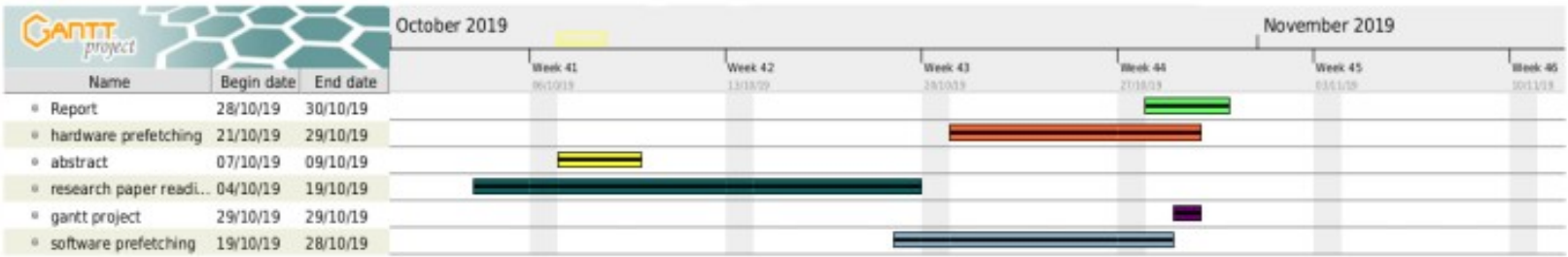
Tasks

Name	Begin date	End date
Report	28/10/19	30/10/19
hardware prefetching	21/10/19	29/10/19
abstract	07/10/19	09/10/19
research paper reading	04/10/19	19/10/19
gantt project	29/10/19	29/10/19
software prefetching	19/10/19	28/10/19

Resources

Name	Enrollment No
Prateek Sachan	18114056
Mehul Arya	18116048
Ritik Jain	18114068
Rahul Sahani	18114062
Ankiteshwar	18116011
Vikas Upadhyay	18116083

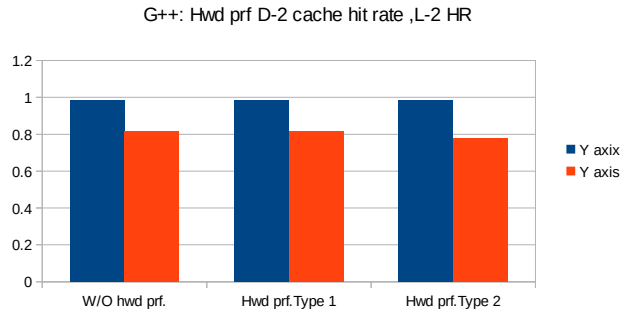
Gantt Chart

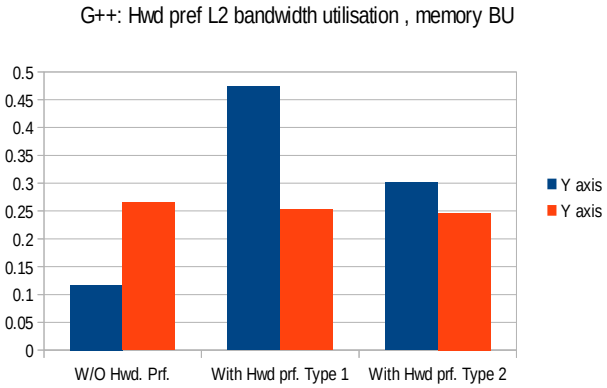


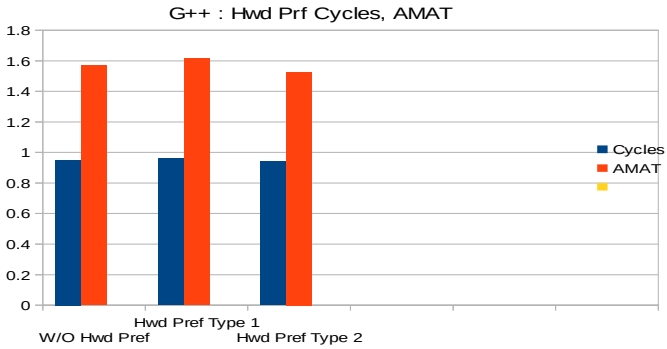
Resources Chart

HARDWARE PREFETCHING DATA

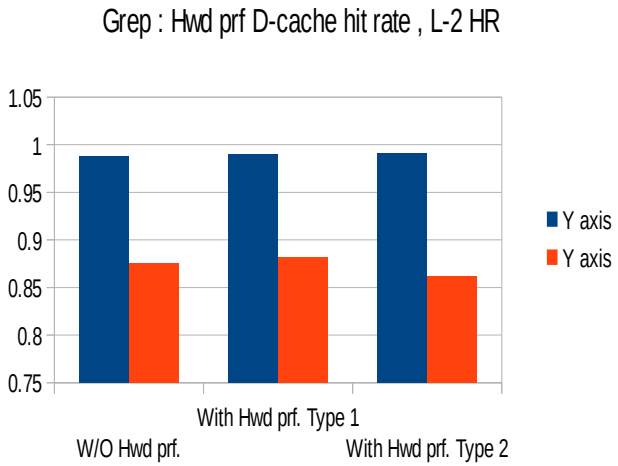
Benchmark/ Trace File : g++



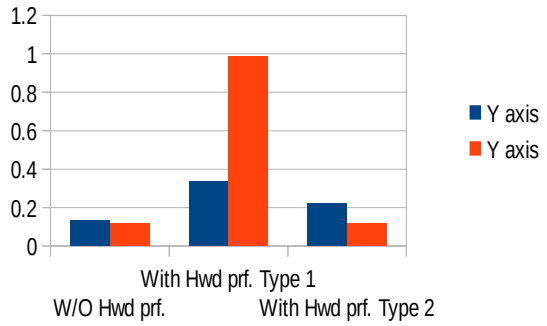




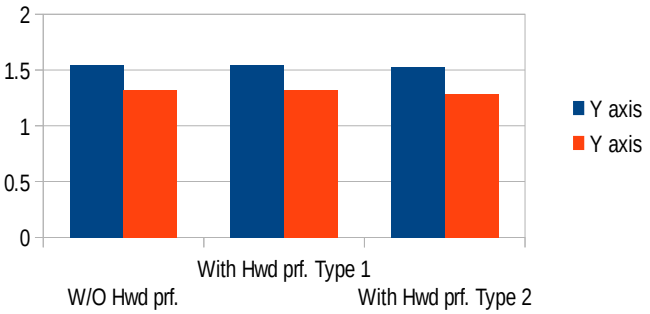
Benchmark/ Trace File : grep



Grep: Hwd prf L2 Bandwidth utilisation,Memory BU

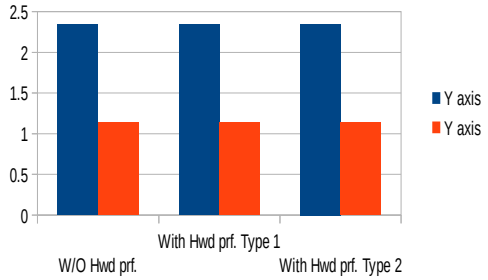


Grep: Hwd prf Cycles , AMAT

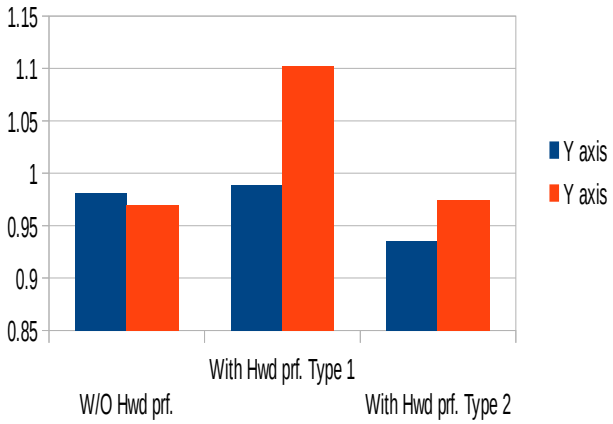


Benchmark/ Trace File : plamap

Plamap : Hwd prf Cycles, AMAT



Plamap: Hwd prf D-cache hit rate , L-2 Cache HR



Plamap: Hwd prf L2 bandwidth Utilisation, Memory BU