SOFTWARE AND HARDWARE PREFETCHING

- By-
- Rahul Sahani-18114062
- Ritik Jain-18114068
- Prateek Sachan-18114062
- Vikas Upadhyay-18116083

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 :

Hardware based prefetching

Software based prefetching

HARDWARE BASED PREFETCHING :

- 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.
- 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.

SOFTWARE BASED PREFETCHING :

- Software based prefetching is typically accomplished by having the compiler analyze the code and insert additional "prefetch" instructions in the program during compilation itself.
- The success of software prefetching depends primarily on identifying and inserting prefetch instructions only for those accesses that are most likely to generate cache misses.

IDENTIFYING CACHE MISSES :

- The software scheme exploits three types of reuse: temporal, spatial, and group.
- Since reuses do not guarantee locality, these reuses are mapped to data locality by taking into account the loop iteration count and the cache size.
- To illustrate the concept of reuse, let us consider a loop.

(a) A loop example

```
for i = 0 to 255
        X[i] = Y[i+1] + Y[i+2] - Z
end
```

(b) Instrumented code

```
for i = 0 to 3 by 2
        prefetch(&X[i])
        prefetch(&Y[i+1])
end                                          } prologue
for i = 0 to 251 by 2
        prefetch(&X[i+4])
        prefetch(&Y[i+5])
        X[i] = Y[i+1] + Y[i+2] - Z
        X[i+1] = Y[i+2] + Y[i+3] - Z
end                                          } main loop
for i = 252 to 255 by 2
        X[i] = Y[i+1] + Y[i+2] - Z
        X[i+1] = Y[i+2] + Y[i+3] - Z
end                                          } epilogue
```

- The accesses to X[i] have spatial reuse since the same cache line is reused in consecutive iterations.
- Accesses to Y[i] and Y[i+1] share group reuse and the access Z has temporal reuse since it is referenced in different iterations.
- Once a potential cache miss has been identified, the software scheme inserts a prefetch instruction.
- If accesses have spatial or group locality in the same cache line, only the first access to the line will result in a cache miss and only one prefetch instruction should be inserted.
- However, testing for this condition, i.e., computing a prefetch predicate, can be very expensive mostly if it occurs in an inner loop.
- Instead, the compiler will generally perform loop splitting and loop unrolling (or loop peeling).

- We split the original loop in three sections: prologue, main, and epilogue.
- The prologue prefetches the initial data set for the first four iterations.
- The main loop consists of the largest portion of the loop execution where the loop is in a steady state, that is, the demand of data can be satisfied by those prefetches occurring several iterations ahead.
- Finally, the epilogue finishes the last four iterations without any prefetching.

HARDWARE VS SOFTWARE PREFETCH :

- The hardware scheme has no information that allows it to avoid unnecessary prefetches.
- There is no CPU-overhead associated with these extra prefetches as long as they are not on the critical path of the processor in hardware prefetching.
- Since the prefetches have no knowledge of potential reuse, the hardware scheme is more likely to bring data that are not useful.
- On the other hand, the hardware mechanism can prefetch data that have been replaced due to conflict misses.

EVALUATION :

The evaluation of the prefetching scheme was done using 3 of the standard benchmarks/ traces :

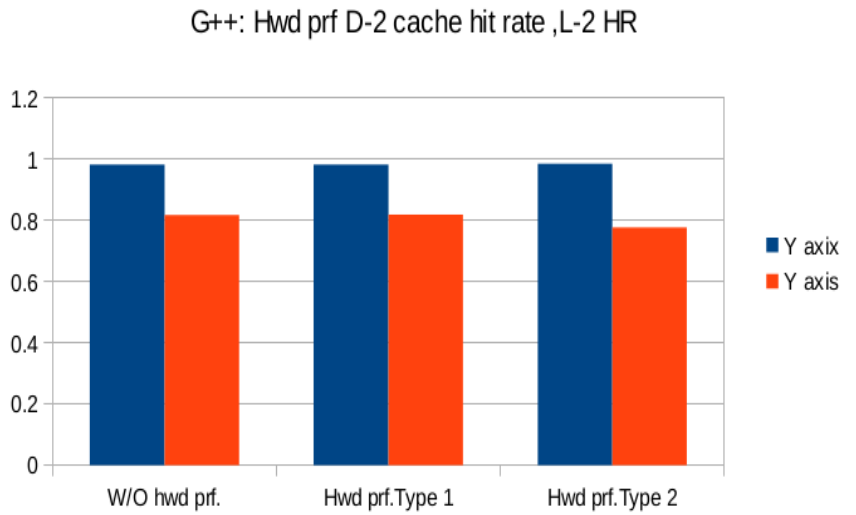
(i) g++

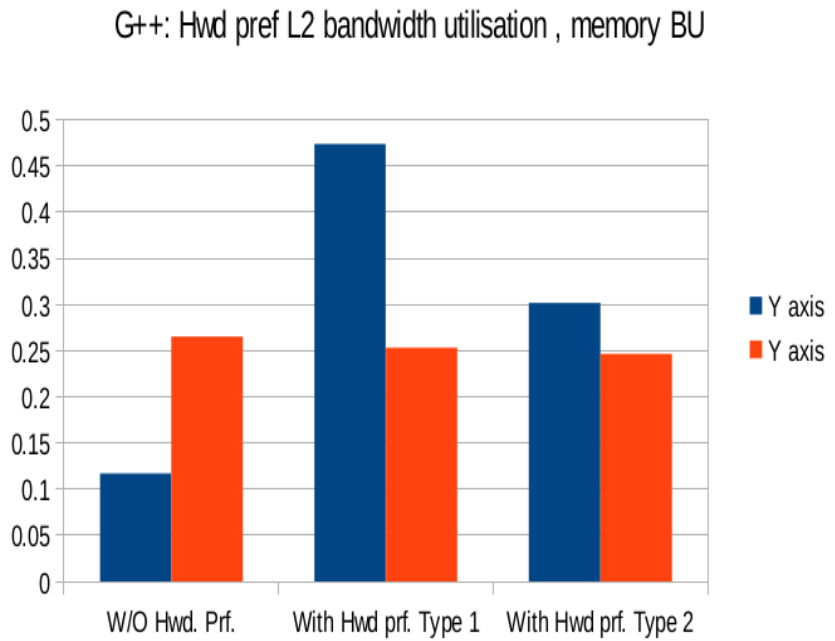
(ii) grep

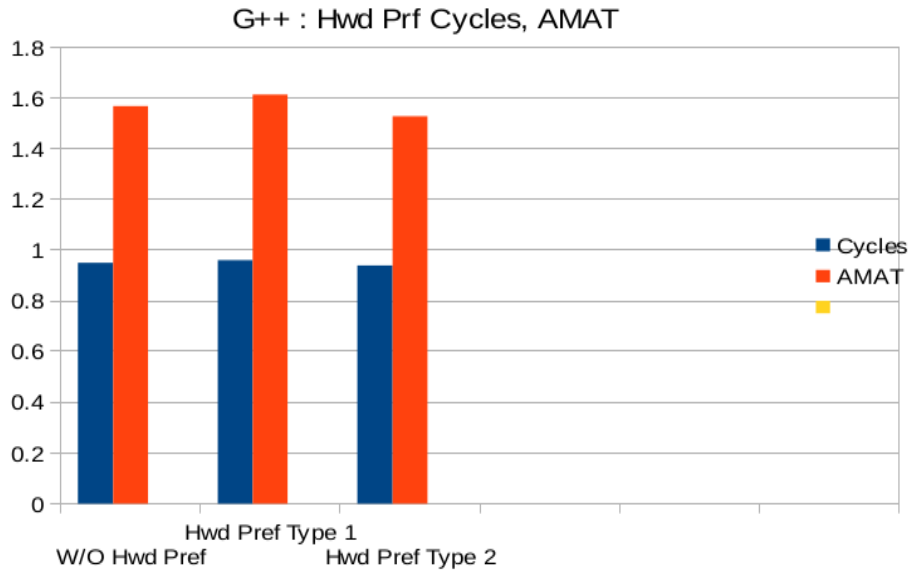
(iii) plamap

HARDWARE PREFETCHING DATA

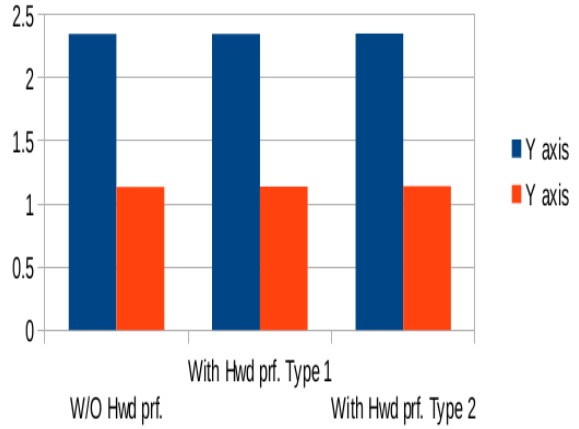
Benchmark/ Trace File : g++

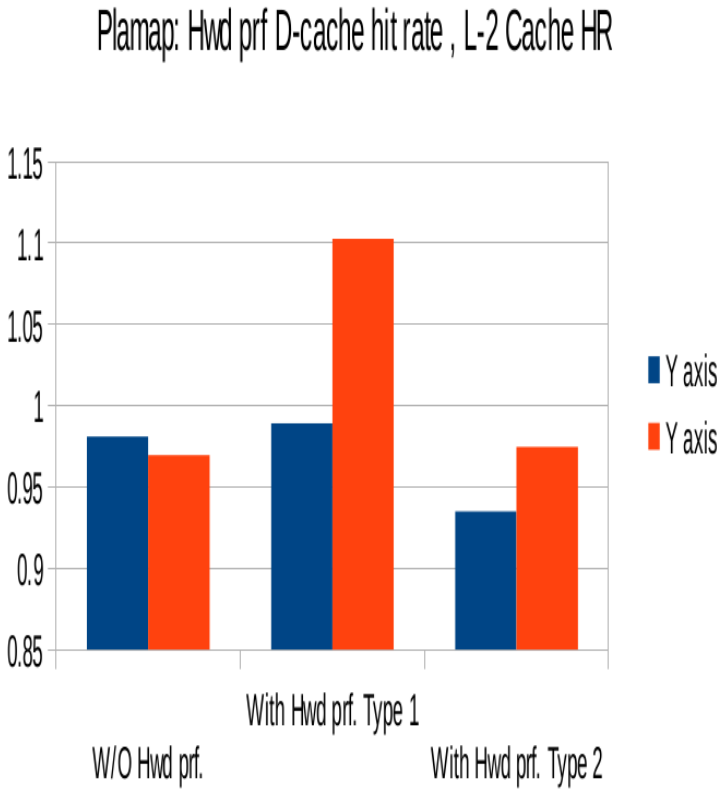


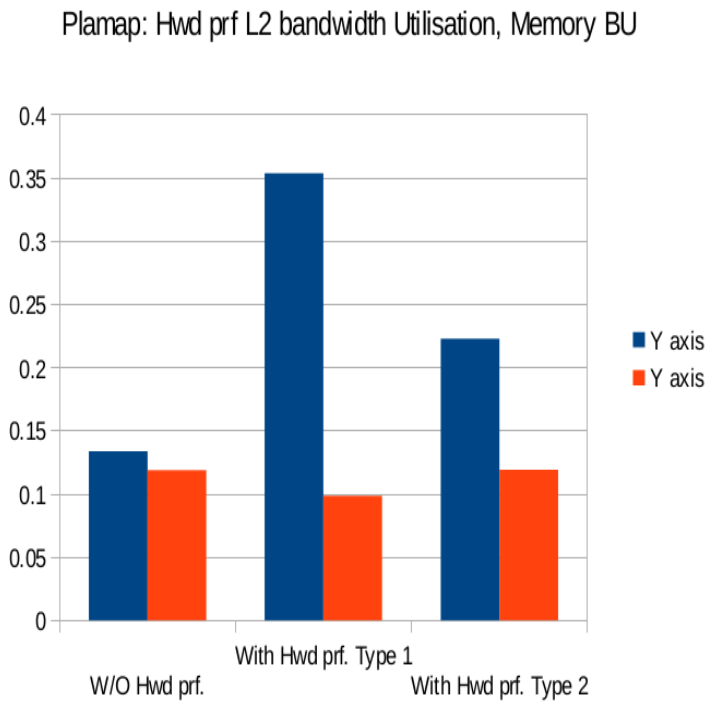




Benchmark/ Trace File : plamap

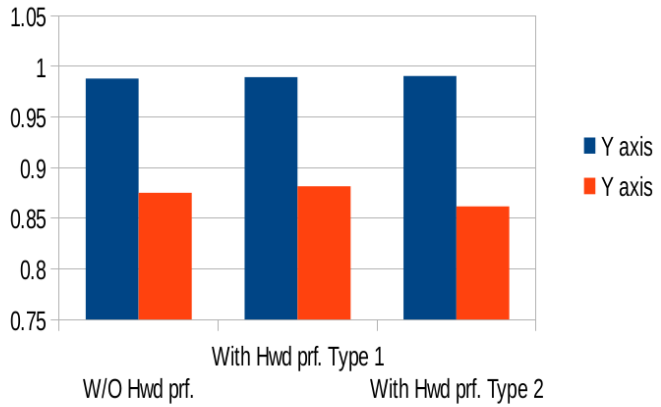




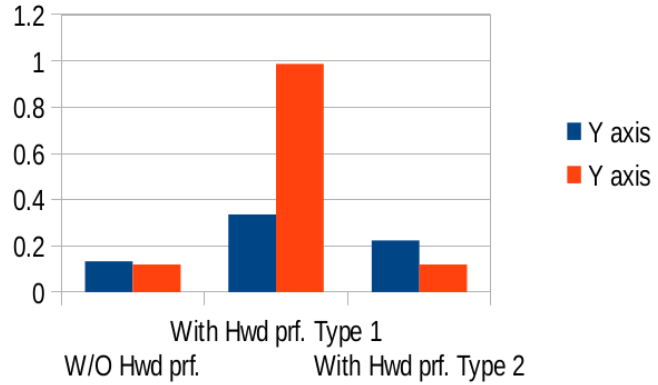


Benchmark/ Trace File : grep

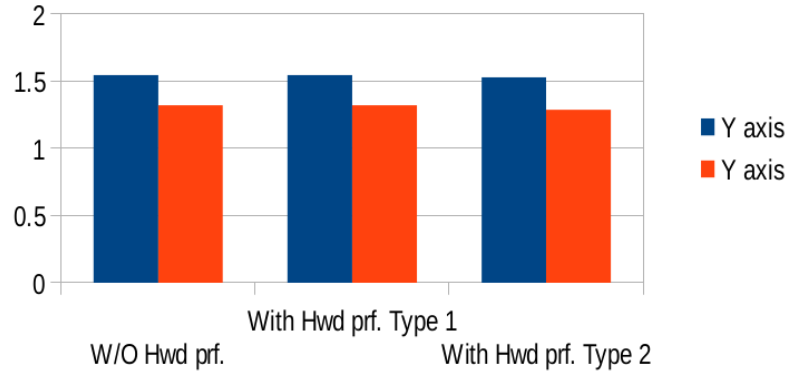
Grep : Hwd prf D-cache hit rate , L-2 HR



Grep: Hwd prf L2 Bandwidth utilisation,Memory BU



Grep: Hwd prf Cycles , AMAT



SOFTWARE PREFETCHING DATA

```
rahul@rahul-X510:~/Downloads$ ./prefetch_disabled
Total Bytes = 4294967296
Starting Data Transpose...   Done
Time: 1.95082 seconds
```

```
rahul@rahul-X510:~/Downloads$ ./prefetch_enabled
Total Bytes = 4294967296
Starting Data Transpose...   Done
Time: 1.28099 seconds
```

THANK YOU…