CDA-5106 Project Phase 2

Parsec benchmark to run: Freqmine

The Freqmine benchmark employs a data mining algorithm Frequent Itemset Mining which is relevant for areas such as protein sequencing and market data [1]. It employs an array based version of the FP-growth problem [2]. The problem is relevant in the current context of the industry because of the increasing demand for data mining techniques and subsequent optimizations. Freqmine solves the problem primarily by constructing an FP-tree and which can be very large. Freqmine then recursively traverses the tree to construct new FP-trees which makes the computation memory intensive. Hence, Freqmine presents a good benchmark to test possible optimizations, changes and extensions to the memory model of the CPU.

Components to investigate:

1. Size of DTLB:
   Due to the inner workings of Freqmine over large transactional data, the size of the data translation look aside buffer (DTLB), affects the overall computation. If the DTLB is large enough, then more data can be cached in it leveraging the temporal spatiality available here. This reduces the latency suffered by the processor since it does not need to repeatedly translate the addresses of data blocks thus reducing the time it takes to retrieve data from memory.

   In order to reduce the number of cache misses incurred by the DTLB, data should span a number of pages that is less than the number of DTLB entries. In that case, all of the data will be cached and no misses will be incurred. In practice, this is not feasible since the data can be much larger and it is not possible to cache all the data pages in the DTLB. Hence, it makes sense that a larger DTLB will inevitably reduce the number of misses as more entries can be cached.

   To study this hypothesis, the size of the DTLB should be varied from 128kB to 512kB with a step size of 128kB. The input to run in order to study the hypothesis is simmedium which includes 500,000 click streams. In order to minimize variance, the benchmark will be run 3 times for each DTLB size and the average taken of the time it takes to complete the computation.

2. Size of MSHR

   The MSHR is used to prevent the processor from stalling due to cache misses. The idea is, in the case of a cache miss, instructions which do not depend on the instruction causing the cache miss should not be delayed. They can be executed in the time it takes for the CPU to resolve the cache miss. Hardware support is required to provide support to keep track of the cache misses in order to provide data to the correct instruction when it is retrieved.

   Naively, it would seem that increasing the size of MSHR would allow for greater parallelization by reducing the number of instructions that need to be stalled in case of cache misses.

   However, this might not be accurate since the degree of parallelization depends greatly on the nature of the application being run and whether it is innately parallelizable. Secondly, a

greater number of MSHR registers would also increase that the possibility of two (or more) cache misses being resolved in the same cycle. In this case, at least of the instructions will have to be stalled.

To study this further, the size of MSHR should be varied from 1-8 in increments of 2.

To minimize variance, each configuration will be run 3 times as in the case of DTLB.

Statistical Processing:

Let t_i be the time taken by run *i.* Total number of runs is *n*.

Then, $\text{Mean (M)} = \frac{\sum_{i=0}^{n} t_i}{n}$

Standard deviation (SD) = $\frac{\sum_{i=0}^{n} (M - t_i)^2}{n}$

Confidence interval (CI) = M ± $Z * \frac{SD}{\sqrt{n}}$

Typical two sided intervals are [3]:

C (%)	Z
99	2.576
98	2.326
95	1.96
90	1.645

Table 1

We can use the values in Table 1 to calculate the error for confidence intervals.

References

[1] Christian Bienia, Sanjeev Kumar Jaswinder Pal Singh and Kai Li. The PARSEC Benchmark Suite: Characterization and Architectural Implications. *http://parsec.cs.princeton.edu/doc/parsec-report.pdf*. January 2008.

[2] G. Grahne and J. Zhu. Efficiently Using Prefix-trees in Mining Frequent Itemsets. November 2003.

[3] "Checking Out Statistical Confidence Interval Critical Values – For Dummies". *www.dummies.com*. Retrieved 2016-02-11.