Computer Organization

Cache Simulation

Andrew Kee

Ryan Montoya

ECEN 4593

Introduction

The goal of the project was to simulate a memory system that implements a two level cache structure composed of L1 instruction, L1 data, and a unified L2 cache. In this system, misses in L1 are handled by L2, and misses in L2 are handled by main memory. The simulation is designed to evaluate the performance and cost of specific hardware configurations given real world traces.

Results

An initial way to view the performance of the traces with specific configurations is to look at the execution times for those Configurations. Fig. 1 shows each Trace and its execution time for each Configuration in relation to all the other traces.

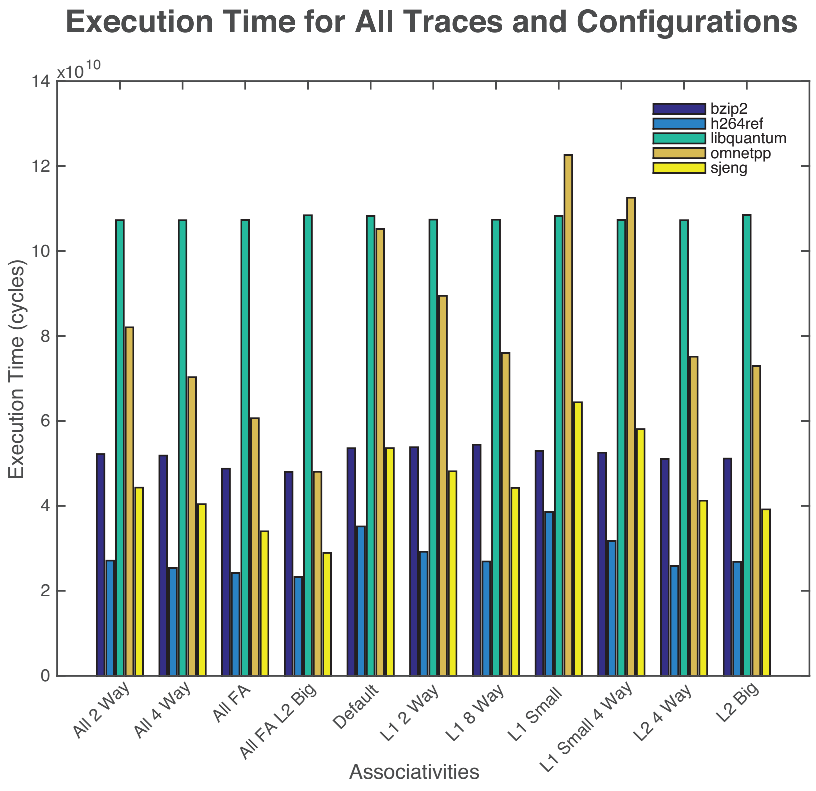


Figure 1: Execution times of all traces with correspondence to all configurations

The execution time for some traces varies very little with changes in cache configuration while other traces are largely effected by the cache configuration. As seen in Fig. 1, libquantum has a very steady execution time which does not seem to depend on cache configuration. A trace like omnetpp however, is very dependent on the cache configuration. The execution times range from 40 billion cycles to 120 billion cycles.

Another way to compare the performance of each trace is to look at the Cycles per Instruction (CPI). This is plotted in Fig. 2.

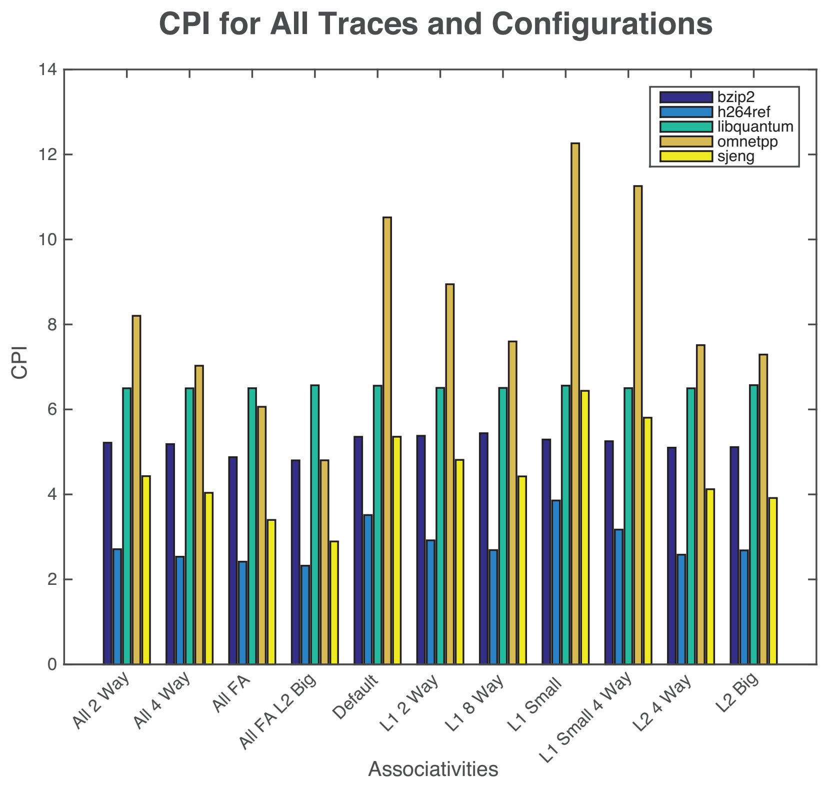


Figure 2: CPI of all traces with correspondence to all configurations

Similar to the execution times in Fig. 1, Fig. 2 shows that the configuration of the cache can dramatically effect the CPI.

Figures. 1 & 2 show that Fully Associative configurations give the fastest results, but at what cost

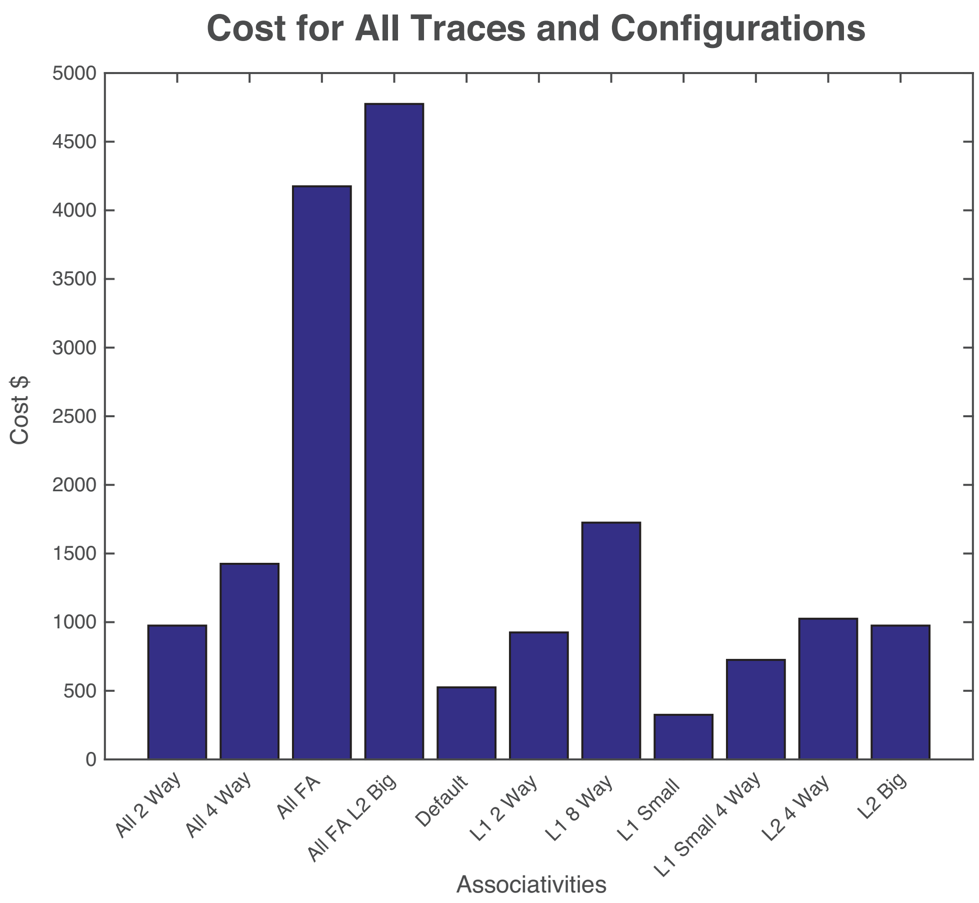


Figure 3: Cost of all the different configurations

Fig. 3 shows the total cost of having each configuration. The fully associative configurations are much more expensive than the other configurations. Viewing both Fig. 2 & 3 show that cost transfers quite proportionally to performance.

Discussion

Examining the execution times of each configuration for the separate traces is not a useful way to measure configuration performance. The execution time is dependent on the number of instructions in each trace file, and so varies both by configuration and trace size. To eliminate this variable, we divided the execution time or cycles by the number of references in each trace to find the CPI. This can seen in Fig. 2.

The most noteworthy feature of Fig. 2 is the variable dependence of trace files on configuration. Both traces libquantum and bzip demonstrate extremely low variance across configurations, while traces such as onmetpp and sjeng demonstrate high variance. This variance is due to function and structure of the code written for these trace files. For example, it is likely that libquantum and bzip frequently reference spatial dissimilar locations in memory, making the spatial locality benefit of caching obsolete. Additionally, if libquantum and bzip were written sequentially and do not frequently loop, the benefit of temporal locality in caching is lost. Traces onmetpp and sjeng however are examples of code that frequently references spatially and temporally similar locations of memory. Frequent looping and indexing of large, static data arrays are good examples of this.

When considering the best performing configuration, it is useful to consider the average instructions per cycle versus configuration. Fig. 4 demonstrates this relationship. An optimally designed multi-purpose architecture should maximize the IPC for all traces. The fully associative cases exhibit the highest instructions per cycle for all traces. This is because a fully associative cache utilizes a least recently used (LRU) buffer for a single set. In direct mapped and non-fully associative configurations multiple locations in memory or lower tier caches can reference the same index and cause the cache to miss. In the fully associative case, the LRU determines what block is overwritten based on temporal locality.

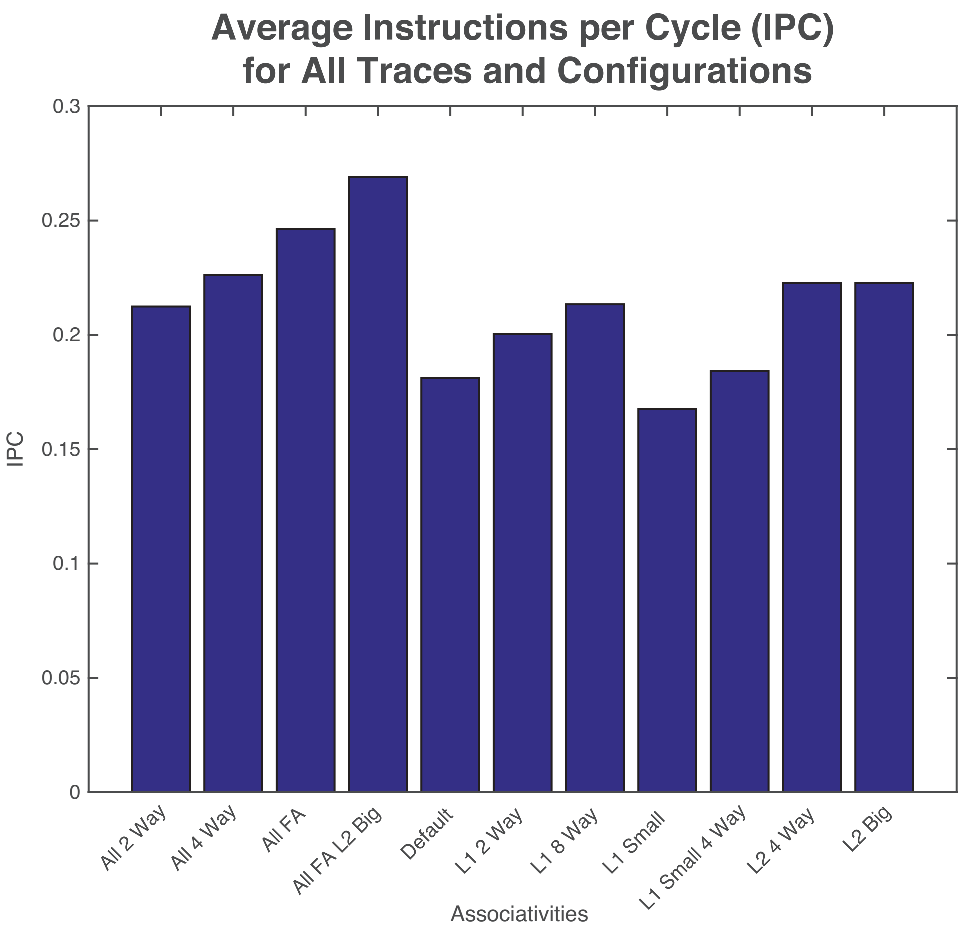


Figure 4: Average IPC for each configuration

Pure instructions per cycle performance however is not the only consideration to make when choosing the best machine. Cost is an important factor. While the FA configurations are the fastest on average, they are also the most expensive. FA caches are extremely expensive, and in the simulation according to specifications cost in the thousands of dollars. When considering what system to buy, it is therefore useful to look at IPC per dollar, as seen in Fig. **??**.

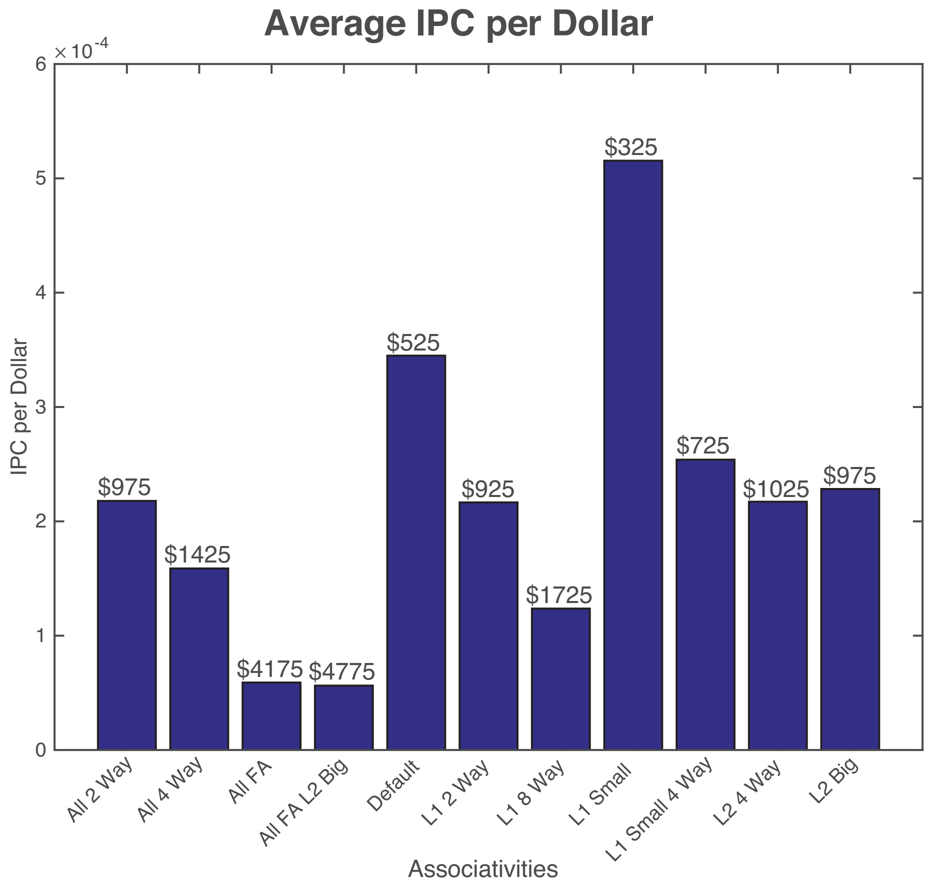


Figure 5: Average IPC per Dollar for each configuration

When choosing a system, Fig. **??** displays the best performing system for the price. It is however also useful to identify what configuration gives the maximum increase in performance from the cheapest configuration per dollar. Fig. **??** exhibits these results. L1 small is the cheapest configuration, and from the figure it is apparent that L2 big gives the maximum performance increase per dollar. Interestingly, the FA cases are yield the smallest performance increase per dollar due to their high cost.

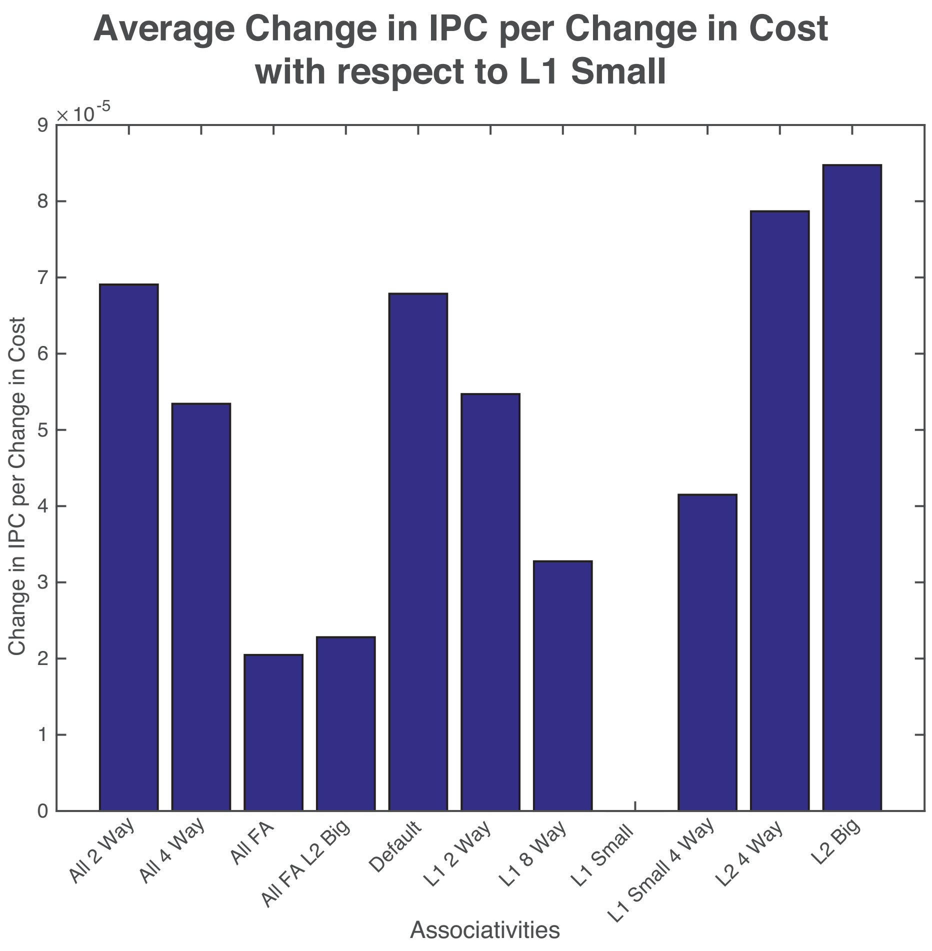
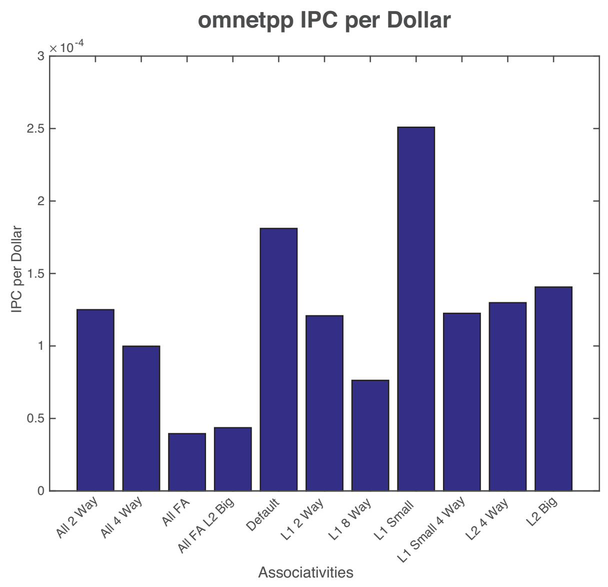
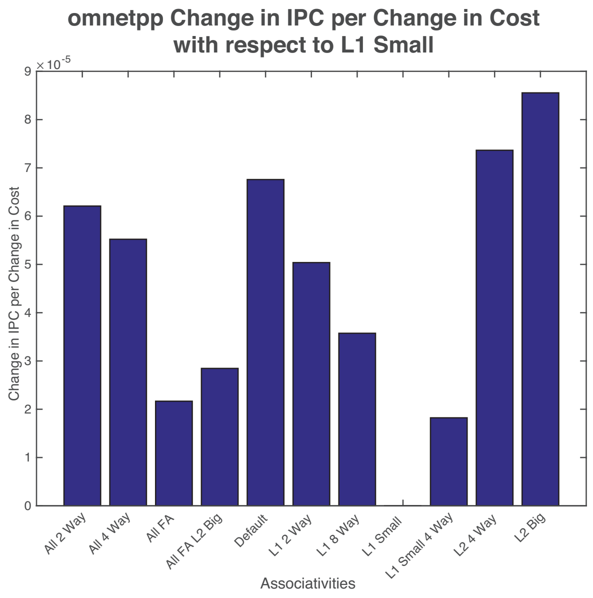


Figure 6: Change in average IPC per change in cost for each configuration

These results are useful only to selecting a general purpose system. Clearly, configurations such as libquantum and onmetpp vary considerably in their utilization of the cache structure. Due to the fact that libquantum does not vary in performance over the configurations, it is unnecessary to purchase any cache configuration other than the cheapest case, L1 small. The trace onmetpp has a high variance, and its respective IPC per dollar as well as change in IPC per dollar should be considered for each configuration. The resulting figures can be seen below in Fig. 7a and Fig. 7b.



(a) IPC per Dollar



(b) Change in IPC per change in cost

Figure 7: omnetpp IPC

From the figures above, it is evident that onmetpp follows a similar trend as the averaged case.

The chunksize of main memory is an important consideration when evaluating what configuration has the best performance given its cost. The chunksize is the width of the bus interface to memory. Access time to main memory is extremely slow, and so it is beneficial to performance to read as many bytes as possible when memory is accessed. The trade-off is cost, and so it is useful to observe the performance increase versus cost of various configurations of memory chunksize. The default cache configuration for chucksizes of 8, 16, 32, and 64 bytes are compared to their respective costs in Fig. 8 below.

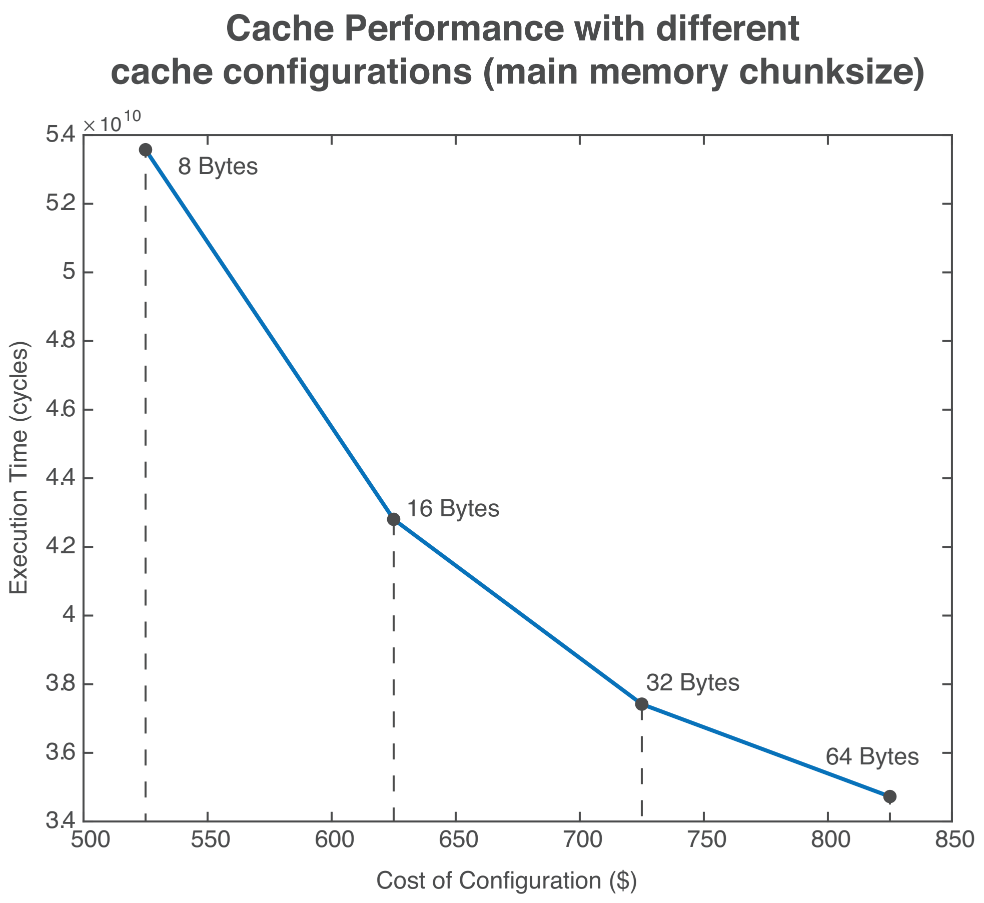


Figure 8

The largest performance increase occurs between 8 and 16 bytes. As chunksize is further increased, the change in performance begins to decrease. Extrapolating from the graph, it is unlikely that a chunksize greater than 64 bytes would yield a significantly greater performance increase. The performance increase from 8 to 16 bytes is approximately double that of the performance increase from 16 to 32 bytes for the same change in cost.

Conclusion

The simulation successfully evaluated input traces for various hardware configurations. Certain traces demonstrated more variance than others, depending on the structure and functionality of the code used to generate them. Traces that exhibited low variance and relatively high cycles per instruction were likely written sequentially, did not utilize large arrays, and had few loops.

The cost of the hardware is an important consideration when evaluating real world performance increases. While the FA configurations yielded the highest instructions per cycle, they were also the most expensive and therefore not the best performance increase per dollar. Additionally, some traces such as libquantum that had very low variance across configurations do not require more advanced hardware than the low cost configuration of L1-small. Traces such as onmetpp however have very high variance and benefit greatly from more expensive cache structures. When considering the larges performance increase per dollar from the lowest cost configuration, doubling the L2 cache size and changing cache associativity to 2-way (L2-Big) was the best option.

Finally, memory chunksize is an important variable in performance and cost. As chunksize increases, the relative gains in performance are diminished. After a chunksize of 64, performance change is negligible compared the increase in price. Initially doubling the chunksize from 8 to 16 however yielded significant performance gains.

Given more time, we would run more simulations to confirm and identify trends in our results. Many of the configurations tested changed more than one cache attribute such as size and associativity, making the results difficult to evaluate. Simulating all cache configurations would give greater insight into what configuration changes yield the largest performance increases per cost.