Cache Simulation Project Report

ECEN 4593 Computer Organization

Edward Zhu
Maxwell Russek

1 INTRODUCTION

Caching in computer systems has proven to be vital for application performance. As the speed of processors rapidly increased in the past several decades, while the speed of off-chip dynamic RAM has remained relatively constant, processors have had to rely more and more on efficient caching in order to avoid pipeline stalls and reach their full performance potential. Thus, the cache performance for any processor system is critical in determining its overall performance.

2 OUTLINE

For this report, several sample program traces were processed using a basic cache simulation implemented in C on several configurations. Various statistics for each program trace using each given memory configuration were gathered, and the data were analyzed to determine which cache configuration yielded the optimal performance relative to the predicted cache cost.

3 SIMULATION

These simulations were ran on both Fedora Linux and Mac OS X. Each of the 5 traces were simulated on 12 different cache configurations, as well as three extra sjeng simulations with different memory bandwidth configurations. In addition to the default 10 configurations, 2 additional configurations were written, custom-L2-Big-All4way and custom-L1-Big, for additional investigation. The ‘custom-L2-Big-All4way’ configuration doubles the default L2 cache

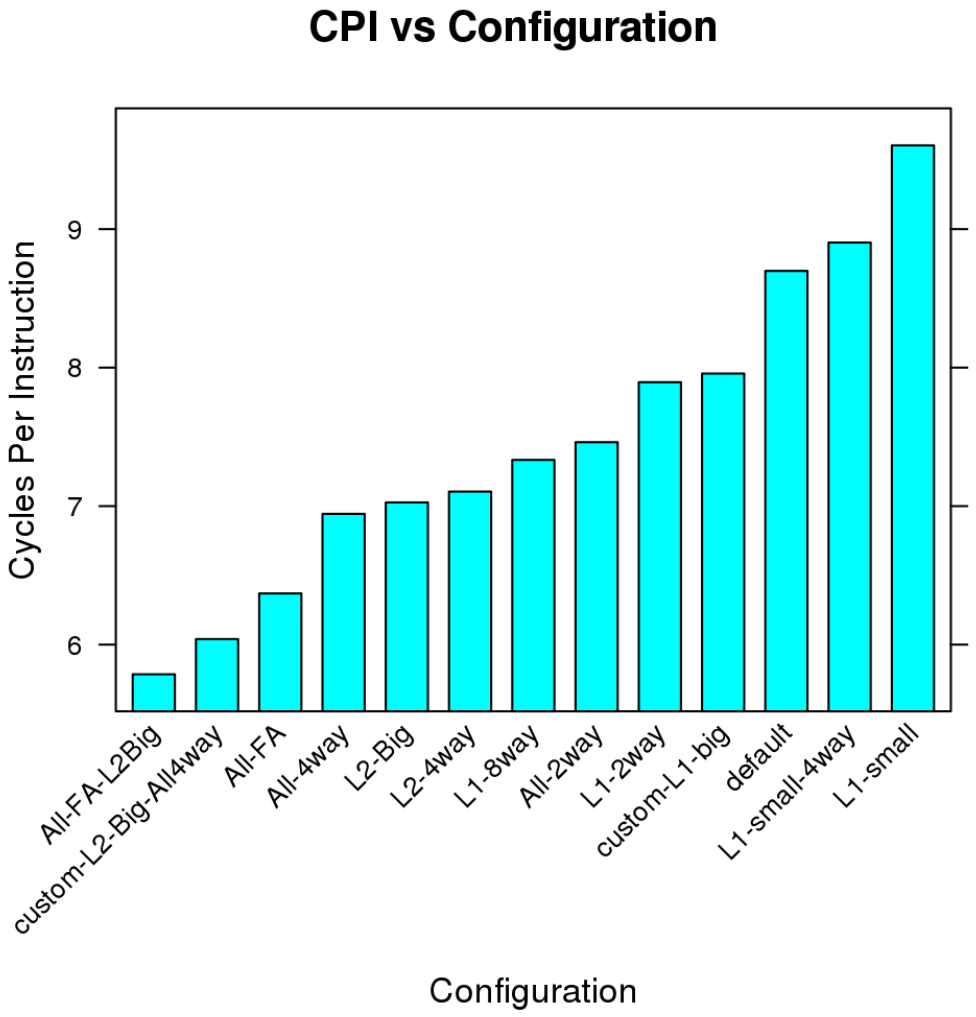
configuration and makes all caches 4 way set-associative. The ‘L1-Big’ configuration doubles the size of the L1 cache.

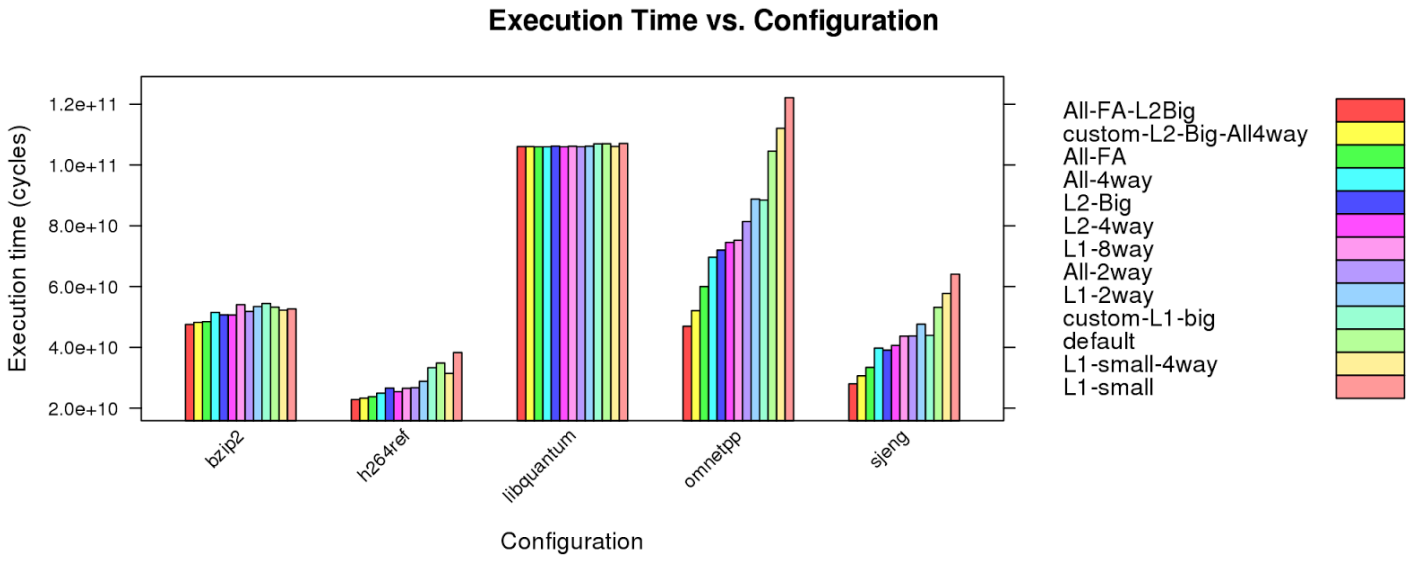
All of the simulations ran in roughly 25-30 hours total; some configurations, such as those with direct mapping and smaller associativity, ran up to twice as fast as the fully associative configurations, which is to be expected due to the expensive nature of the LRU implementation.

4 ANALYSIS

4.1 Execution Time and CPI

We first analyzed our data to determine which memory configuration resulted in the lowest execution time for each trace. A plot of execution time versus cache configuration, grouped by the trace, and a plot of average CPI per configuration appear in the figure on the following page.


CPI vs Configuration
Cycles Per Instruction
6
7
8
9
All-FA-L2Big
custom-L2-Big-All4way
All-FA
All-4way
L2-Big
L2-4way
L1-8way
All-2way
L1-2way
custom-L1-big
default
L1-small-4way
L1-small
Configuration


Execution Time vs. Configuration
Execution time (cycles)
2.0e+10
4.0e+10
6.0e+10
8.0e+10
1.0e+11
1.2e+11
bzip2
h264ref
libquantum
omnetpp
sjeng
Configuration
All-FA-L2Big
custom-L2-Big-All4way
All-FA
All-4way
L2-Big
L2-4way
L1-8way
All-2way
L1-2way
custom-L1-big
default
L1-small-4way
L1-small

As seen from the plot above, for most traces, the fully associative cache with the 64KB size L2 level cache outperforms all other configurations. After the fully associative, large L2 configuration, the next best performing given configuration is the fully associative cache with the default cache sizes. Both of these results are expected because the fully associative cache essentially only holds the most recently used blocks, and always evicts the least recently used cache block, thereby reducing the number of conflict misses due to addresses mapping to the same block. However, the custom-L2-Big-All4way configuration is the second best configuration. This is also expected since having a large, set associative L2 cache in addition to associative L1 caches will reduce the total number of misses in both caches.

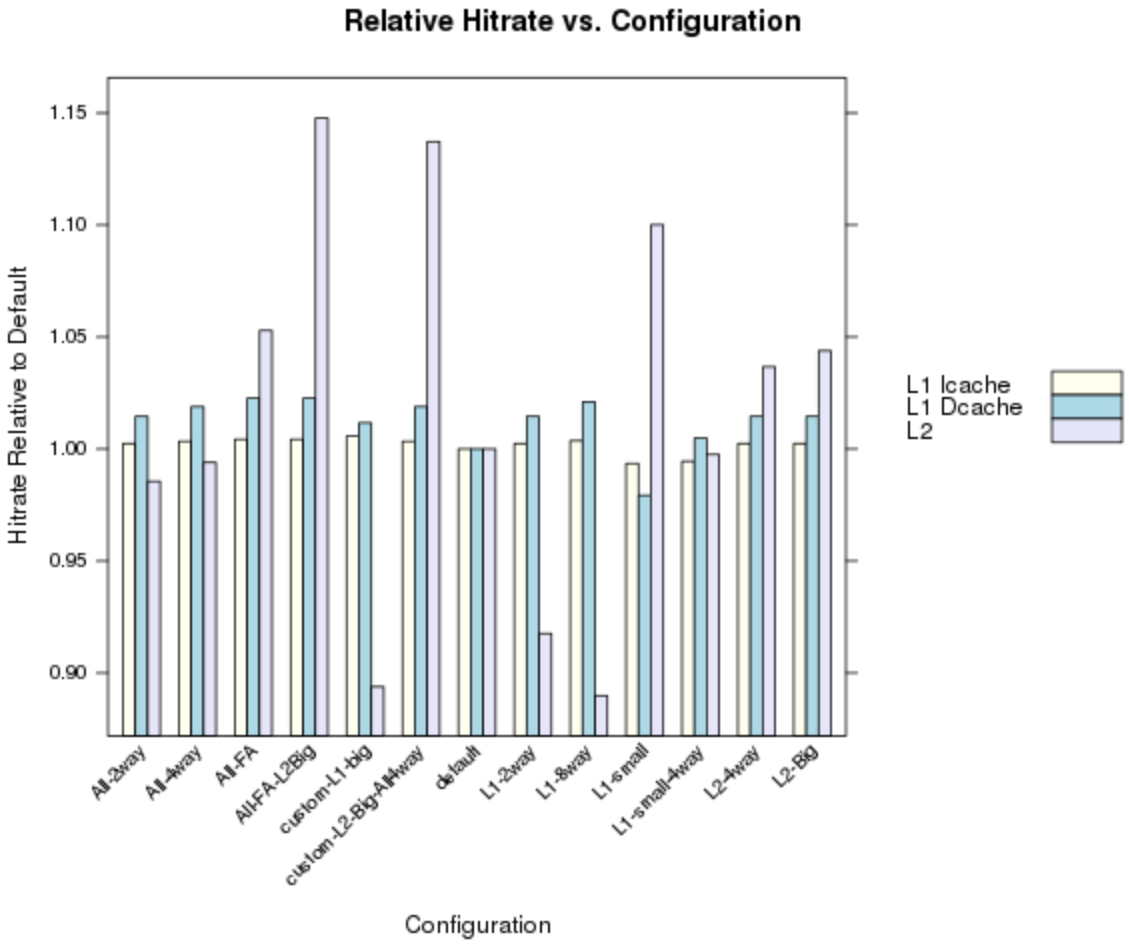
We also see that the configuration for the small L1 (L1-small) cache consistently performs the worst for all traces. This is understandable since the only change for this cache configuration is that the size of the L1 is halved, meaning that it has a higher chance of conflict misses, and also a larger number of capacity misses.

A further observation is that while increasing the size of the default L1 cache did improve performance, it did not improve performance as much as making the default size L1 cache associative. Notably, the performance of the doubled L1 cache was roughly comparable to the 2-way set associative L1 cache. This is expected since both techniques decrease the rate of conflict misses by roughly the same factor. However, higher associativity L1 caches perform better than an L1 cache of twice the size because of their increased ability to handle references that would otherwise be conflict misses.

Another thing to note from the execution time is that the libquantum trace appears to perform roughly the same on all cache configurations. When investigating this unexpected result, it was found that the libquantum trace exceeded all other traces in the total number of references, particularly in instruction references. On further analysis, it was found that the average instruction cache hit rate for libquantum was also much higher than the other traces. This explains the lack of variance in the data for libquantum. It is likely that libquantum has a high locality of reference, which means that it has a high hit rate for any cache configuration, regardless of capacity or associativity, and that most cache misses are compulsory.

4.2 HIT RATES

The next metric used to measure the performance of each configuration is the hit percentage for each cache. A figure plotting the average relative hit percentage for each configuration is shown in the figure on the next page.



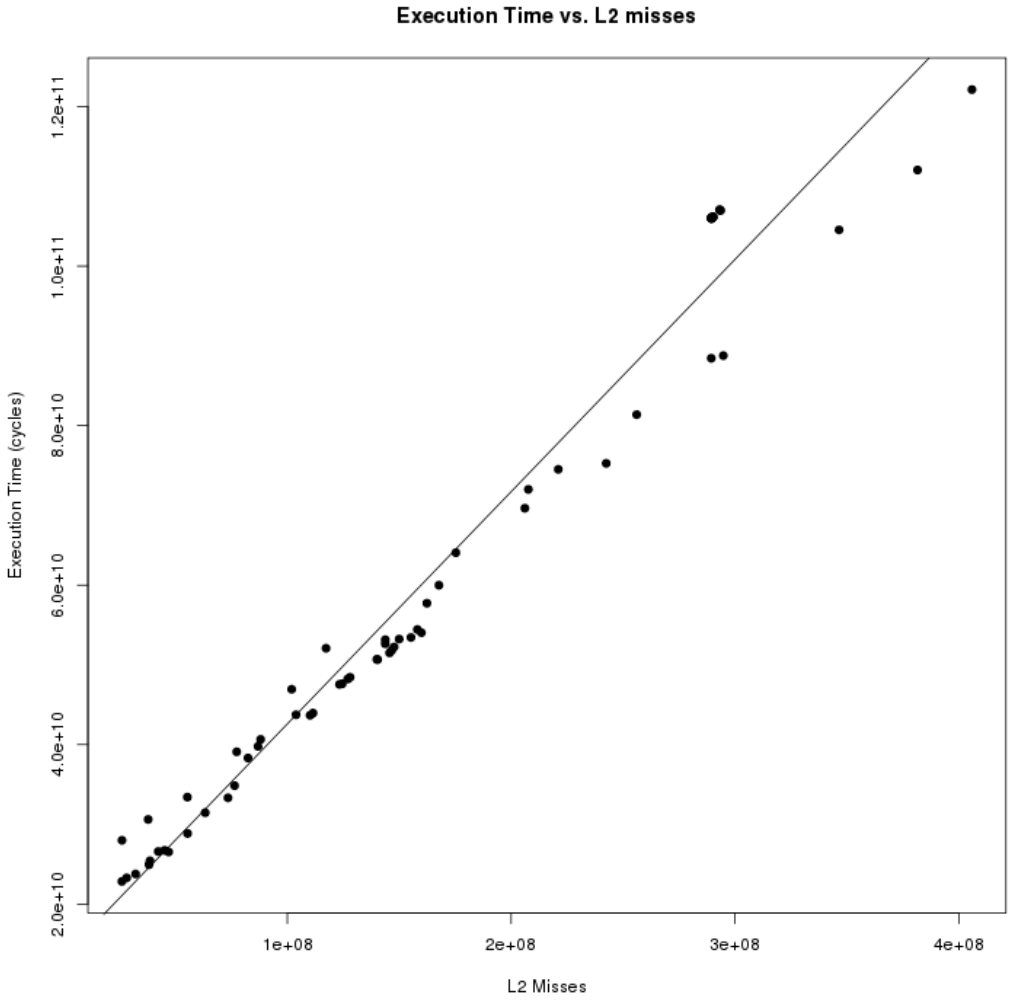
The plot above shows that the L2 cache hit percentage appears to vary the most among the different configurations. In particular, it appears that while the fully-associative large L2 cache has the best L2 hit rate, the 4-way associative large L2 cache appears to be comparable in hit rate.

Another observation is that better performing L1 cache configurations appear to do the worst for L2 hit rate. This is interesting since the L1 hit rates between all configurations are mostly the same. This observation can be explained by noting that a better performing L1 cache has a higher percentage of compulsory misses

which are less likely to sent to the L2 again at a later time, resulting in more L2 misses.

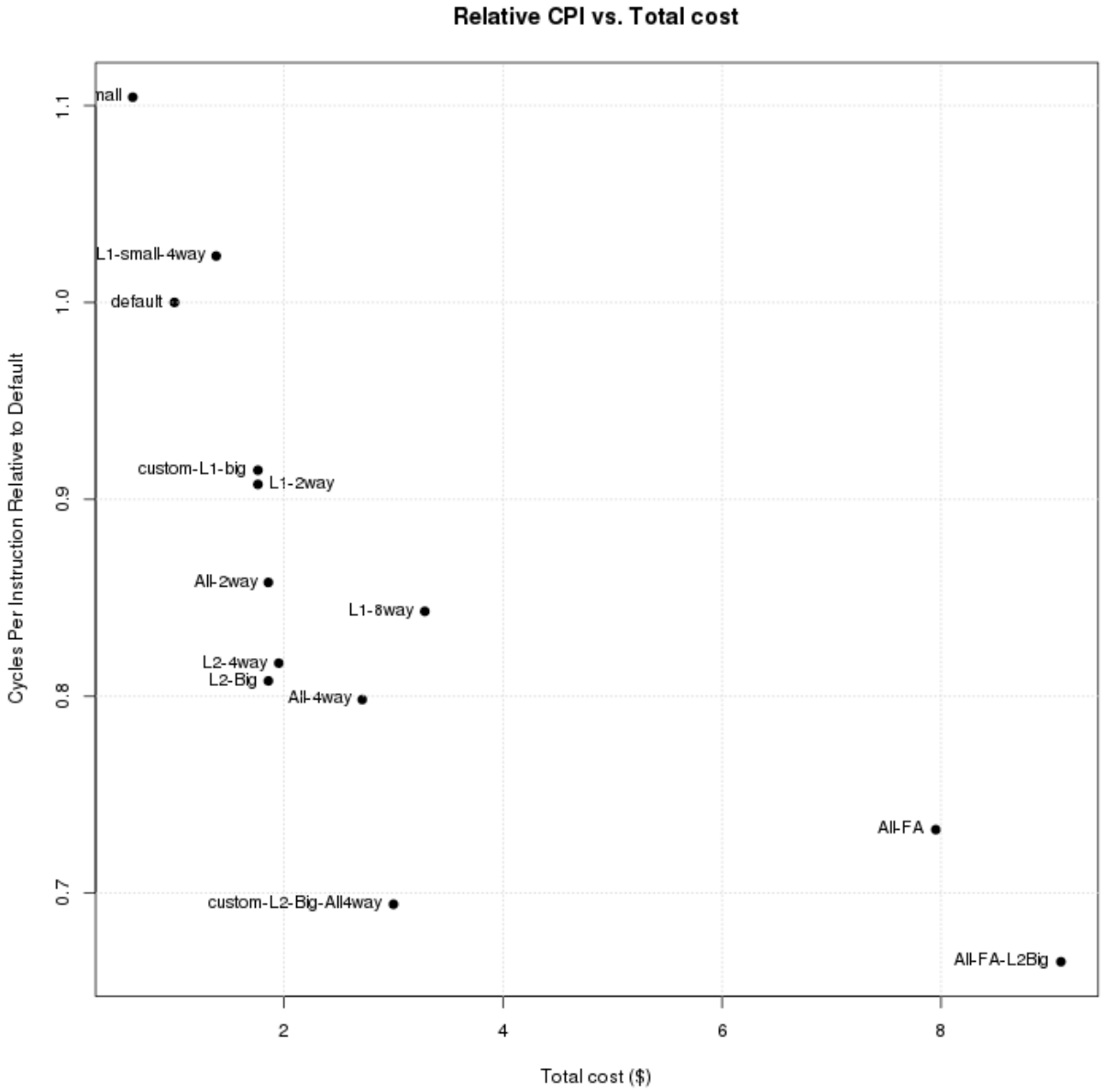
It is also noticeable that the small L1 cache has a surprisingly good L2 hit rate. This can be explained since a smaller L1 will spill more capacity misses to the L2 cache, and these are more likely to be requested again later in the program.

Another interesting statistic to compute is the correlation between the different types of misses versus the overall execution time. In particular, the data show an extremely strong correlation between L2 misses and execution time, and another, weaker correlation between L1 data misses and execution time. These are obvious results, but what is interesting is that the same correlation does not exist for L1 instructions misses. It appears that L1 instruction misses, and thus the L1 instruction hit rate, is not as relevant a factor for the execution time. The plot of execution time versus L2 misses is shown on the next page.



4.3 COST

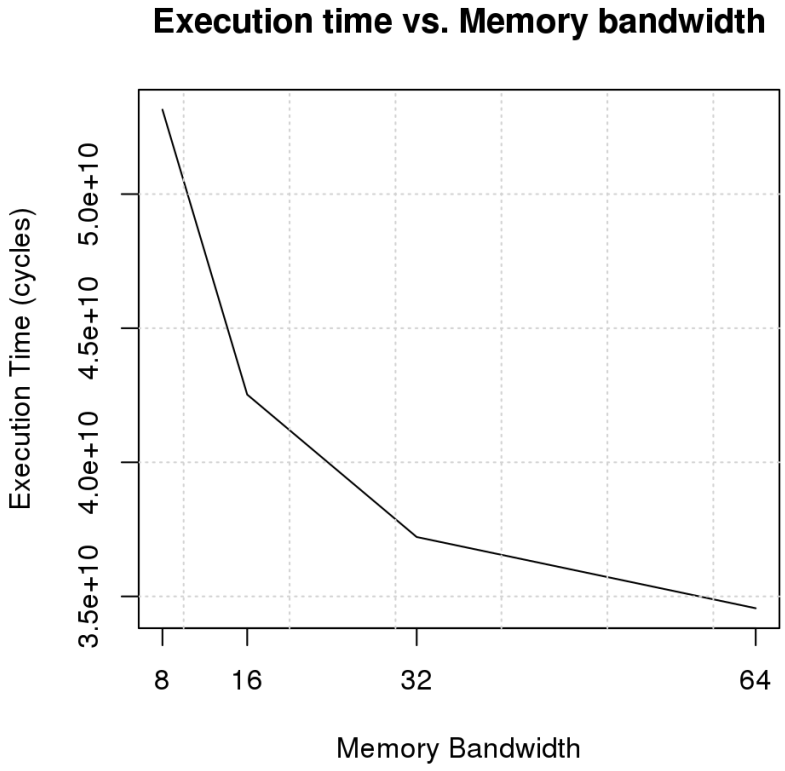
CPI relative to the default configuration, as a function of cost, is displayed in the plot on the next page. This plot shows the intuitive trend that CPI should decrease as the cost of the cache configuration increases. In particular, we can see the best given configurations, All-FA and All-FA-L2Big, far to the lower left.



However, it is worth noting the overall trend line appears non-linear, implying that the benefit of more expensive cache configurations diminish with their cost. Overall, it appears that the best improvements over the default configuration appear to be increasing the associativity of the caches as in L2-4Way, L2-Big, All-4way, and L2-Big-All4way. In particular, it appears that the custom L2-Big-All4way configuration has the best cost relative to its performance.

4.4 MEMORY CONFIG

Sjeng ran three extra memory configurations with the default cache setup; differing the bandwidth to main memory (memory chunk size) by increasing powers of two. Along with the default memory chunk size of 8, configurations with a memory chunk size of 16, 32, and 64 bytes were simulated and recorded as well. The next plot displays the correlation between the execution time and the change in the memory bandwidth.



This plot depicts a compelling relationship between execution time and the memory bandwidth size. The initial increase of memory chunk size from 8 to 16 bytes shows a massive performance improvement in execution time, however this trend does not persist. The next increase from 16 to 32 bytes shows a smaller increase in execution time, roughly half the performance increase of 8 to 16

bytes. Inferring from this general trend, as chunk sizes grow beyond 64 bytes, the relative increase in performance diminishes and is nearly negligible. Another thing to note is that any increase in memory bandwidth past 64 bytes is unlikely to be useful unless the block size of the L2 cache increases accordingly.

5 CONCLUSION

These simulations present an accurate overview on how different hardware configurations impact the performance of a basic cache implementation. From the results, it is clear that the fully associative configuration with double the L2 size has the best overall performance, but only when cost is not factored in. Cost of hardware plays a very important factor in determining the most effective cache. Although configurations such as All-FA and All-FA-L2Big did display incredible performance, the cost to implement the hardware for those configurations greatly outweighs their performance increases relative to more inexpensive options such as the custom L2-Big-All4way configuration. The results of the simulations running differing memory chunk sizes produced an interesting diminishing performance increase. As the memory chunk size increased, the execution times did speed up, but the relative gains started to recede very quickly; eventually becoming near trivial with memory chunk sizes of 64 bytes or greater. Overall, this simulation proved to be very enlightening, and effectively demonstrated the relative performance of different cache configurations.