EE 156/CS 140: Lab 2b

Hazel Leonard

April 10, 2022

1 Introduction

When designing caches, computer architects can choose to modify a variety of parameters, from replacement policy to cache size to cache associativity. In these experiments, I investigated the effects of different combinations of cache associativity and prefetchers on an L3 cache.

Cache associativity describes the range of locations where a memory address can be stored and ranges from direct-mapped to fully associative. In a direct-mapped cache, each memory address can only be stored in one cache location. If new data needs to be loaded into the cache and an address is already occupying that location, the old address must be replaced with the new data, resulting in a conflict miss when the previous address needs to be accessed again. On the other hand, in a 4-way set associative cache, each memory address can be stored in four different locations in the cache. This means that data does not need to be evicted if at least one of the four locations is unoccupied, but it also means that all four locations must be searched to determine whether an address is present in the cache. In a fully associative cache, any address can be stored in any line of the cache, lowering the amount of conflict misses but also making it necessary for the entire cache to be searched to determine whether or not it contains a specific memory address.

It is often said that increasing cache associativity also increases performance. To determine the strength of this trend, I chose to analyze the relationship between these two aspects. To what extent does associativity actually increase performance? What trade-offs exist between increased cache associativity and power consumption? Further, how do different benchmarks influence the effect of associativity on performance and power consumption?

Along with analyzing cache associativity and performance, I also introduced prefetching to my experiments. Prefetching refers to loading data into a cache before it is actually needed in anticipation of its use in the near future. Different prefetchers apply different algorithms to decide which memory addresses to load into the cache and when to prefetch this data.

Based on prior research, prefetching can decrease cache misses by multiple percentage points, but I was not fully convinced that implementing complex algorithms for a small performance increase was worth it. Are prefetching's benefits reflected in Sniper experiments? Which prefetchers work best for different benchmarks? Additionally, does prefetching increase power consumption more than it decreases miss rate? I investigated all of the above questions in my experiments.

2 Experimental Setup

Before running my experiments, I hypothesized that increasing last-level cache associativity would raise instructions per cycle (IPC), especially for workloads prone to conflict misses. I also predicted it would increase power and decrease miss rates. In terms of cache prefetching, I hypothesized that prefetchers with more complex algorithms would increase power, decrease miss rates, and raise IPC. I did not expect prefetchers to have different effects on workloads prone to conflict misses.

Based on my predictions, I chose to collect IPC, L3 miss rate, and L3 Peak Dynamic Power from my experiments. I used IPC and miss rate to reflect performance and Peak Dynamic Power to represent energy consumption. I decided to run the experiments using the Splash2 workloads `lu.cont, radix, raytrace, FMM,` and `cholesky`. `radix, FMM,` and `raytrace` appear prone to conflict misses as they access data in repeated patterns, which could result in information being evicted from the cache only to be used again shortly if set-associativity is not high enough. For example, `radix`'s "algorithm is iterative, performing one iteration for each radix r digit of the keys" [1]. `FMM` traverses a tree "in a single upward and downward pass (per timestep)," [1] and `raytrace` "generates multiple rays, and the recursion results in a ray tree per pixel" [1] with recursion potentially resulting in conflict misses.

On the other hand, `cholesky` and `lu.cont` did not appear to be especially prone to conflict misses. The `lu.cont` benchmark "factors a dense matrix into the product of a lower triangular and an upper triangular matrix" [1] and it attempts to exploit temporal locality. Cholesky is similar to lu.cont except for the fact that it "operates on sparse matrices, which have a larger communication to computation ratio for comparable problem sizes, and (ii) it is not globally synchronized between steps" [1].

To analyze performance on these benchmarks, I sweeped 1-way (direct mapped), 2-way, 4-way, 8-way, 16-way, 32-way, 64-way, 128-way, and 256-way (fully associative for a 16384 MB cache) last level cache associativities. In these experiments, I designated a 16384 MB L3 cache as the last-level cache. The below topology was used to sweep these associativites.

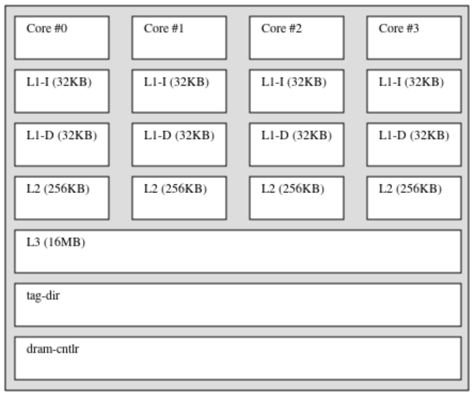

Figure 1: Topology used for experiments.

I also sweeped all available prefetcher configurations in Sniper: no prefetcher, simple prefetcher, and the Global History Buffer (GHB) prefetcher. The `simple` prefetecher is similar to a strided prefetcher [2], which adds data to the cache depending on the offset distance or stride between data accesses. This means the algorithm must store and calculate the stride with every cache miss or hit. The `GHB` prefetcher keeps track of recent miss addresses in FIFO order [3] using multiple linked lists to connect addresses that share properties. This structure can be applied to existing prefetching algorithms, and the researchers in the original paper applied it to a stride prefetcher to find increased stride prefetching performance of 6%. Therefore, I predicted that `GHB` would have better performance than the two other prefetchers tested, but also hypothesized that it would increase energy consumption compared to the other prefetchers because it has to maintain multiple lists of recent data.

I ran the above configurations on an L3 cache of 16384 MB, simulating every combination of prefetcher and set associativity.

3 Results and Analysis

3.1 McPAT and CPI Diagrams

Below are McPAT diagrams for a workload that appeared prone to conflict misses, `raytrace`, and a workload that did not, `cholesky`. These diagrams represent the power used by different operations during execution. I chose to display the difference in direct-mapped and fully associative caches using every prefetcher setting in these graphs to compare a conflict miss-prone benchmark to one that is not.

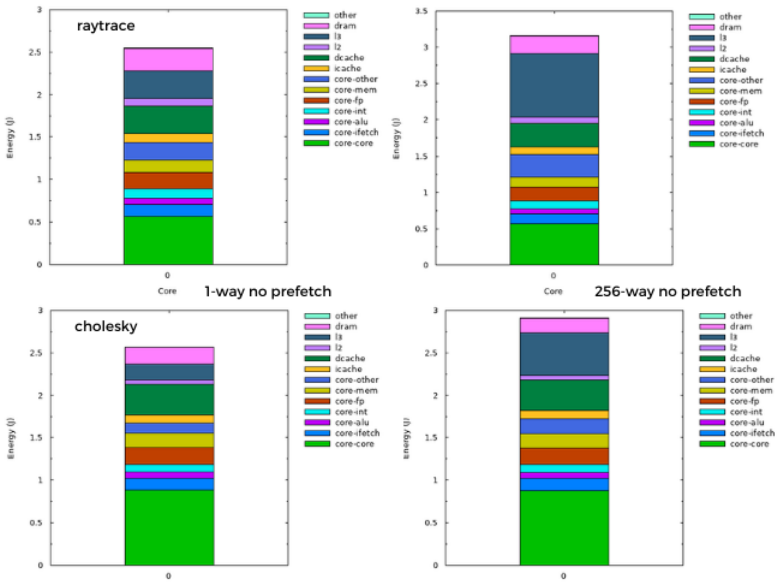

Figure 2: No prefetcher power diagram for `cholesky` and `raytrace` in direct mapped and fully associative caches.

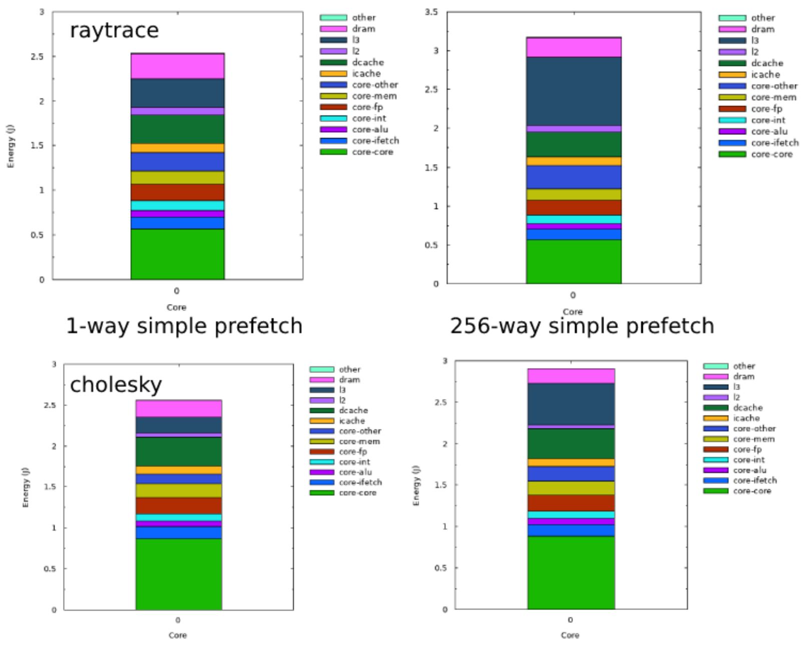

Figure 3: `Simple` prefetcher power diagram for `cholesky` and `raytrace` in direct mapped and fully associative caches.

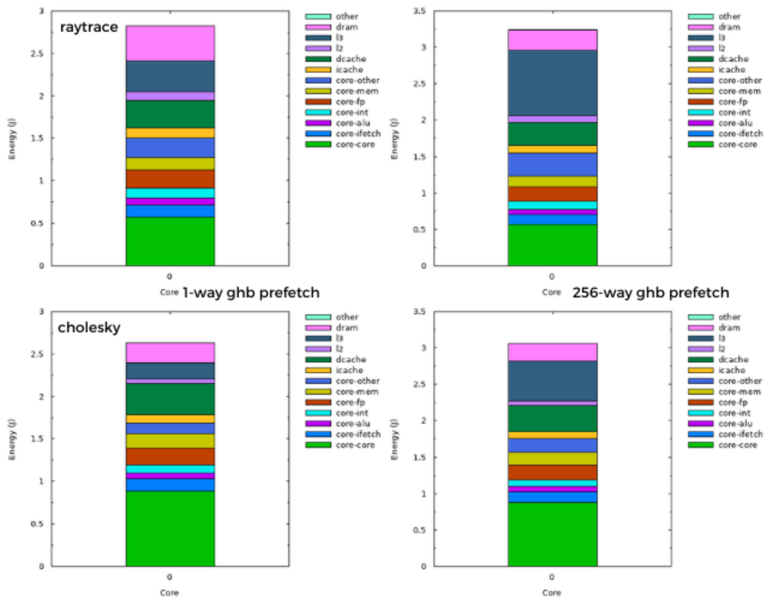

Figure 4: GHB prefetcher power diagram for cholesky and raytrace in direct mapped and fully associative caches.

As a whole, these diagrams show that the number of joules spent on the L3 cache increases with associativity. Interestingly, the type of prefetcher used does not seem to have much of an effect on the size of the L3 portion of the stack, though the size of the stack as a whole appears slightly greater for experiments run on the GHB prefetcher. The other components of the stack also do not seem to change much between associativity or between benchmarks, which was unexpected. Because raytrace and cholesky do not appear to be performing similar operations, I had predicted that the composition of their power diagrams would vary to a larger extent.

To further explore the relationship between benchmark, prefetcher, and associativity, I generated the below CPI diagrams.

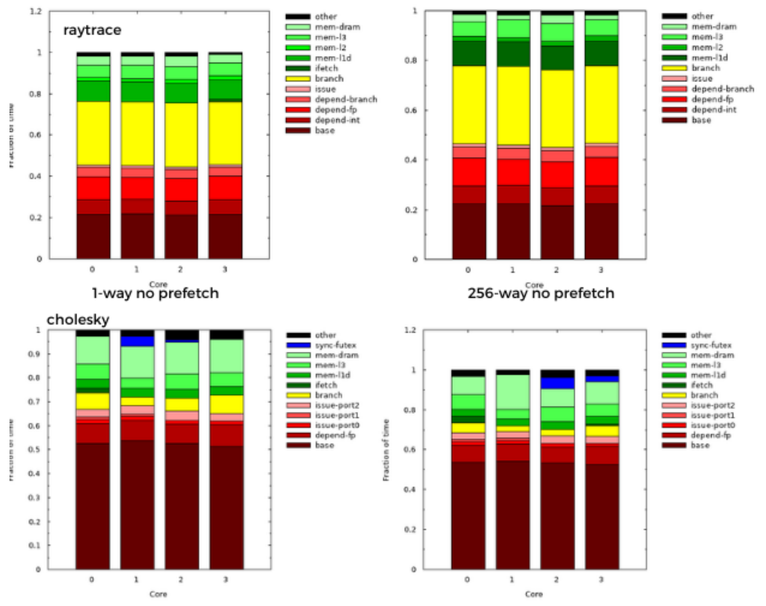

Figure 5: No prefetcher CPI diagram for `cholesky` and `raytrace` in direct mapped and fully associative caches.

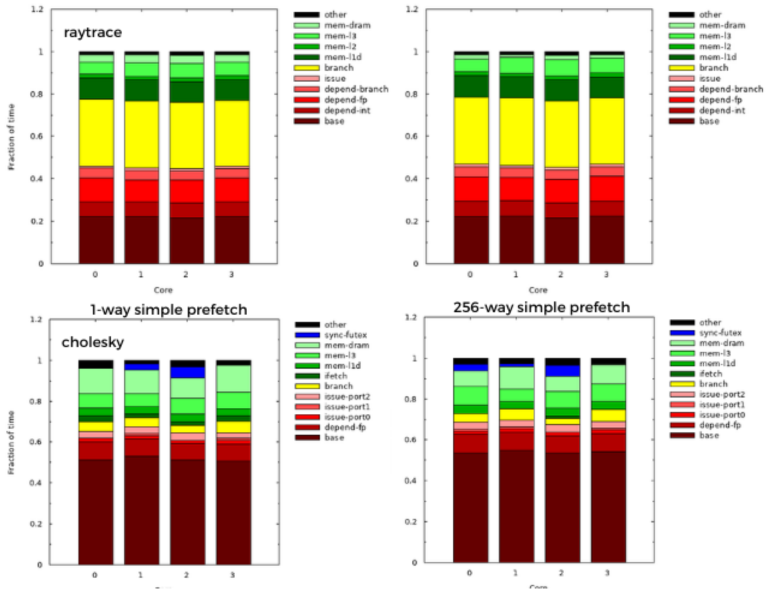

Figure 6: `Simple` prefetcher CPI diagram for `cholesky` and `raytrace` in direct mapped and fully associative caches.

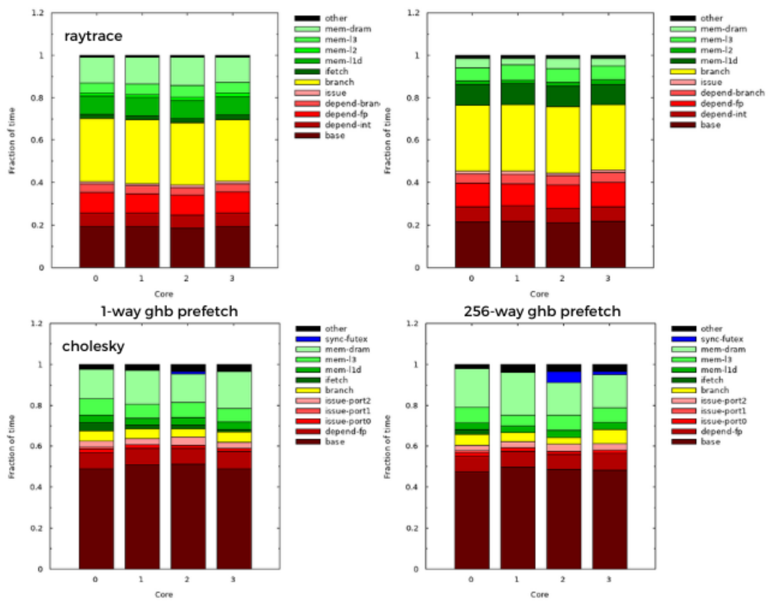

Figure 7: `GHB` prefetcher CPI diagram for `cholesky` and `raytrace` in direct mapped and fully associative caches.

These CPI stacks show that each benchmark's CPI stack remains roughly consistent no matter which associativity or prefetcher is applied. Even though the fully associative CPI stacks from the no prefetcher category appear quite different from the stacks representing direct mapped caches, the numbers on the vertical axis reveal that each benchmark's CPI stack remained about the same height.

Despite each benchmark's stack remaining about the same height and spending about the same fraction of time on each instruction category across different associativities and prefetchers, there are large disparities in the stacks between benchmarks. The most obvious difference is the large amount of time `raytrace` spends in 'branch' compared to the large amount of time `cholesky` spends in 'base'. This disparity is present when comparing the benchmarks' charts regardless of prefetcher or associativity. The 'base' section is made up of cycles doing actual work, not stalling, which is in contrast to the other sections on the stack that represent waiting. It can be inferred that the 'branch' section refers to cycles spent stalling on each branch prediction instruction.

Given what 'base' and 'branch' represent, the disparity between these benchmarks indicates that a a larger fraction of cycles per instruction were spent doing actual calculations in the `cholesky` benchmark than the `raytrace` benchmark, and that `raytrace` experienced a high amount of stalling in branch prediction. This may have resulted from the fact that correct branch predictions were easier to make in the `cholesky` benchmark as it works on matrices. `Raytrace` renders a three-dimensional scene which likely has many more branch options than `cholesky`.

3.2 Performance

I analyzed IPC, miss rate, and Peak Dynamic Power for various combinations of prefetchers and associativites.

3.2.1 Miss Rate

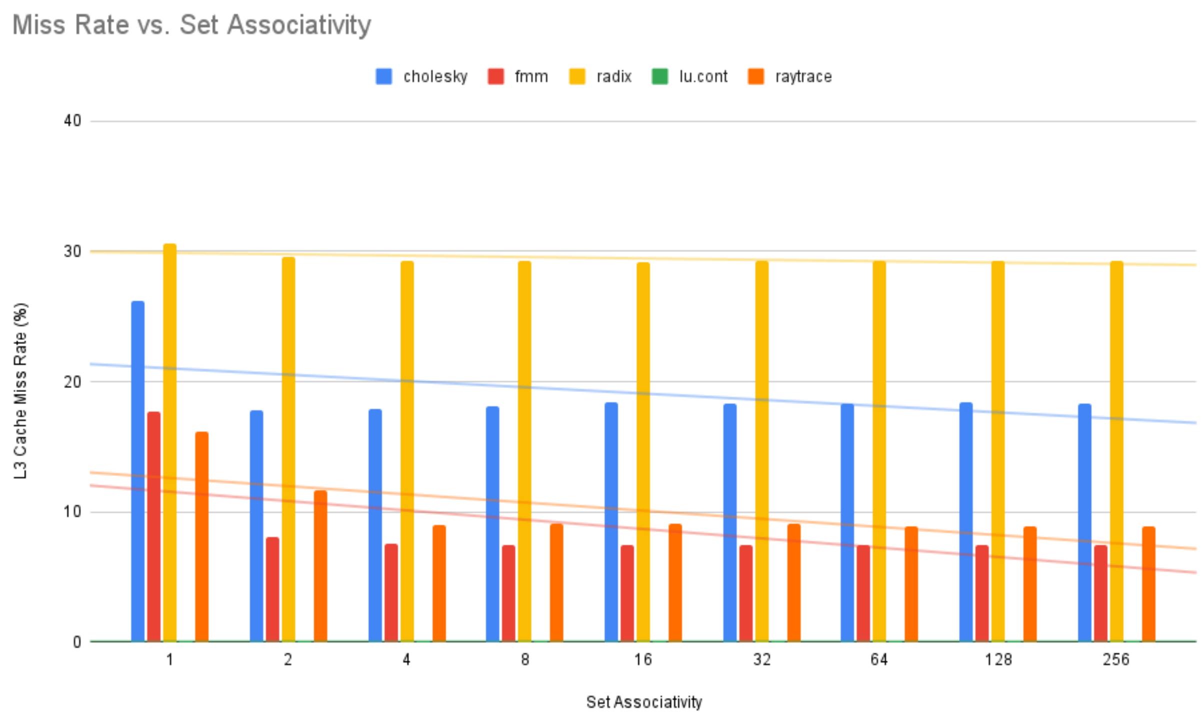

Figure 8: Miss rate compared to L3 cache associativity (no prefetcher used).

This graph shows that miss rate declines, although sometimes very gradually, across the benchmarks as L3 cache associativity increases. For all the benchmarks, miss rate declines the most as associativity increases from direct mapped to 2-way and 4-way set associative. Overall, the graph corresponds with my hypothesis that increasing associativity decreases miss rate. The steepest negative trend lines appear in `cholesky, raytrace`, and `fmm` as associativity increases, while I had predicted `radix, FMM`, and `raytrace`'s miss rates to decrease the most. It is interesting that `cholesky` saw such a decrease in miss rate compared to `radix`, whose miss rate did not change much. Learning more about the benchmarks or analyzing their code could help explain this result.

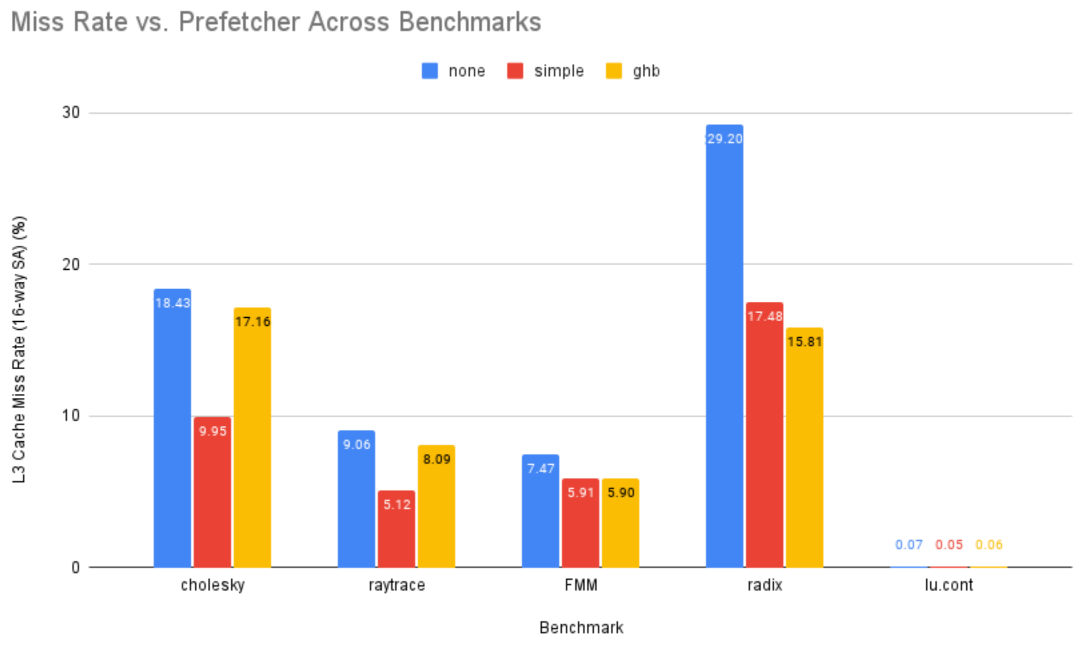

Figure 9: 16-way set associative L3 cache miss rate compared to prefetcher configuration.

I had expected miss rate to be highest for no prefetcher, intermediate for the simple prefetcher, and lowest for the GHB prefetcher. These results show that a prefetcher's effect on miss rate depends highly on the benchmark it is used on. However, it is clear that using the appropriate prefetcher for a benchmark does decrease its miss rate. It is interesting that `cholesky` and `raytrace` appear to have similar ratios of miss rates between the three prefetcher settings, with no prefetcher resulting in the highest miss rate, the GHB prefetcher resulting in the second lowest miss rate, and the simple prefetcher resulting in the lowest miss rate. I had not expected these results, as the work done in each benchmark does not appear similar. It would be interesting to explore the reasons for these benchmarks' similar performance under different prefetchers. On the other hand, `FMM` and `radix` saw the highest miss rate with no prefetcher, the second lowest miss rate with the simple prefetcher, and the lowest miss rate with the GHB prefetcher. This could correspond with the fact that both benchmarks appear prone to conflict misses. Finally, `lu.cont` experienced such a low miss rate overall that it is difficult to ascertain the effects of different prefetchers.

3.2.2 IPC

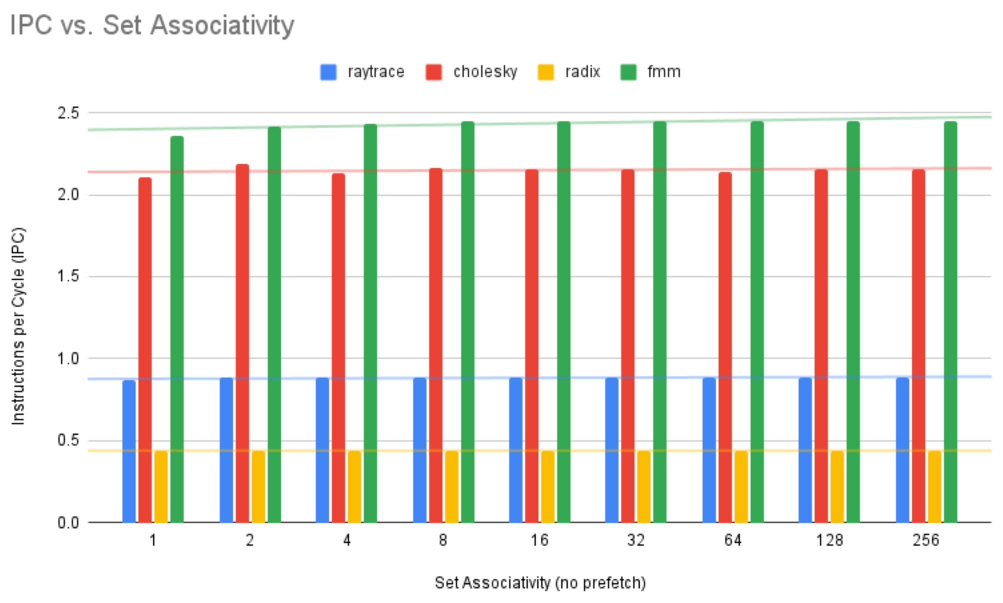

Figure 10: IPC compared to L3 cache associativity (no prefetcher used).

As cache associativity rises, IPC appears to rise very slightly or remain the same. This partly corresponds with my hypothesis that IPC would increase with cache associativity. IPC may not have changed much as cache associativity increased because miss rate did not experience very large changes as associativity increased, but this would still not fully explain why IPC remained largely the same as miss rate did decrease more noticeably than IPC increased. Perhaps the benchmarks were using instructions that took a relatively long time to complete regardless of external factors. In analyzing the difference in IPC between different benchmarks, `cholesky` appears to execute a high number of instructions per cycle compared to the other benchmarks, which is unexpected given `cholesky`'s relatively high miss rate. This indicates that miss rate is likely not the largest contributor to IPC.

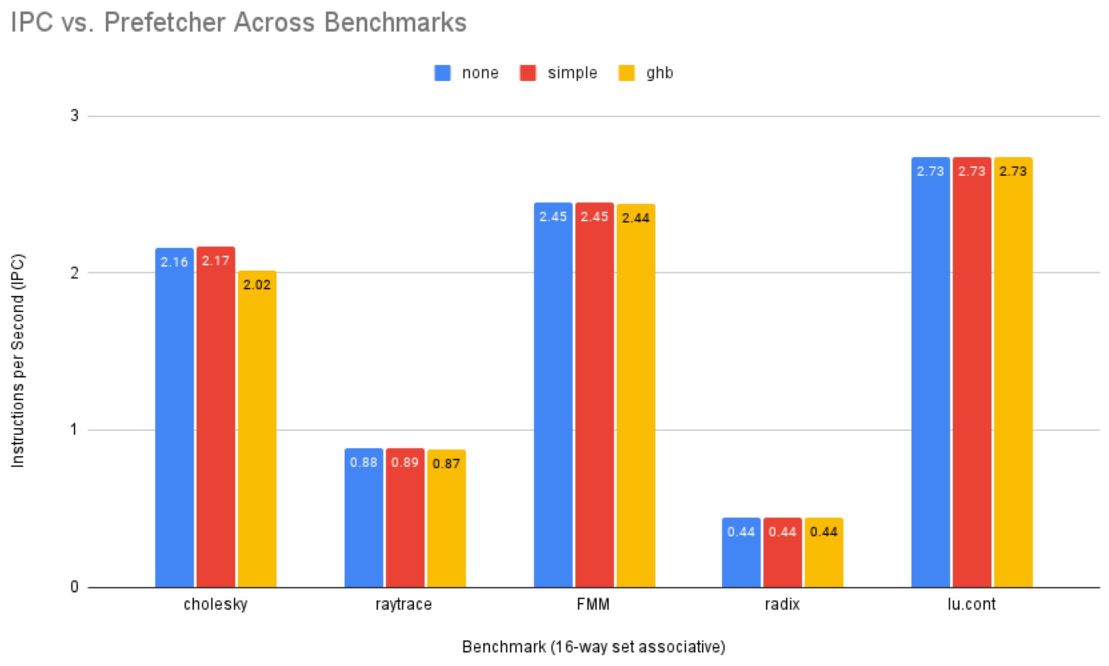

Figure 11: 16-way set associative L3 cache IPC compared to prefetcher configuration.

Similar to the previous graph, this chart reveals that the prefetcher selected does not have much of an effect on IPC across different benchmarks. Each benchmark's IPC stayed almost exactly constant regardless of whether no prefetcher, the simple prefetcher, or the GHB prefetcher was applied. This disproves my hypothesis that IPC would increase as more complex prefetchers were applied. The constant IPCs in this chart could have been produced by the benchmarks' instructions being too long to be affected by small changes in performance resulting from prefetchers. Given that cache associativity and prefetcher have little effect on IPC, what can influence this performance measurement? This would be an interesting topic for future research.

3.2.3 Peak Dynamic Power

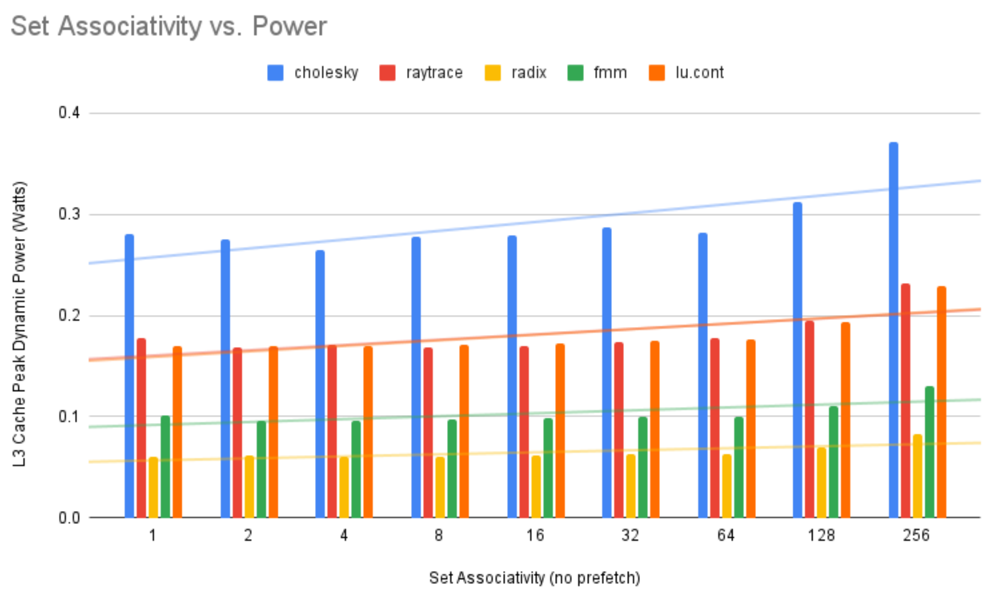

Figure 12: Peak Dynamic Power compared to L3 cache associativity (no prefetcher used).

This graph indicates that Peak Dynamic power of the L3 cache increases as set associativity increases over all of the benchmarks, which corresponds with my hypothesis. I expected this to happen because a larger set associativity means more time and energy must be spent searching through a larger range of potential locations of data stored in the cache. However, though the trend lines in this diagram appear steep, it can be seen from the numbers on the vertical axis that Peak Dynamic Power does not actually vary much between direct-mapped to fully associative caches. This is a positive finding because it shows that the processor in this configuration may not overheat as a result of increased set associativity in the L3 cache.

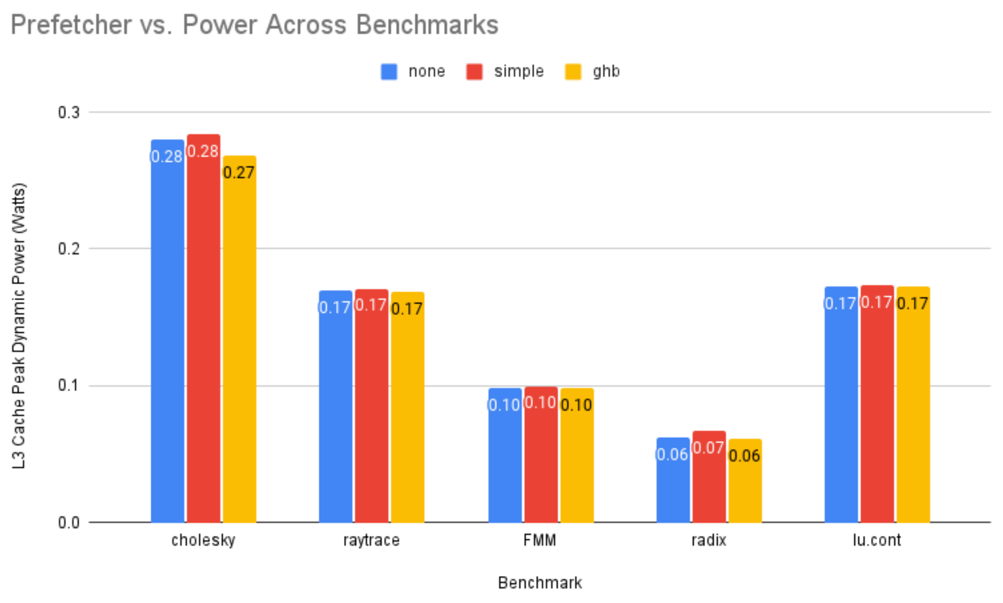

Figure 13: 16-way set associative L3 cache Peak Dynamic Power compared to prefetcher configuration.

I had predicted that Peak Dynamic Power would increase as pretechers became more complex, which this graph disproves. It appears that each prefetcher generates about the same Peak Dynamic Power as the others. This could have occurred because though the GHB prefetcher may have a more complex algorithm for determining which data to prefetch than the simple prefetcher, for example, this would not be reflected in the Peak Dynamic Power of the cache; it would likely be reflected in instruction execution time or other similar measure. However, it would be interesting to research whether prefetchers modify the cache more often than non-prefetching cache replacement policies do. If prefetchers do modify the cache more frequently, I would expect the Peak Dynamic Power of the cache to increase because data would be accessed and written to it more frequently.

4 Conclusion

Overall, this experiment revealed that miss rate declines as cache associativity increases and Peak Dynamic Power increases as cache associativity increases, both of which correspond with my hypotheses. However, my findings also revealed that prefetcher does not affect Peak Dynamic Power or IPC; additionally, I found that miss rate per prefetcher is highly variable across different benchmarks and that that cache associativity does not greatly affect IPC. These last four findings do not correspond with my hypotheses. The results of the simulations were conclusive, and they brought up further questions about why certain prefetchers work better for some benchmarks, which computer architecture factors affect IPC, and why the prefetchers tested did not affect Peak Dynamic Power. These questions would be meaningful to answer in future research.

References

[1] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh and A. Gupta, "The SPLASH-2 programs: characterization and methodological considerations," *Proceedings 22nd Annual International Symposium on Computer Architecture*, 1995, pp. 24-36, doi: 10.1109/ISCA.1995.524546.

[2] Girelli, Valeria Soldera. "A Study of the Prefetcher Impact on High-Performance Computing Applications." UFRGS Digital Repository, Universidade Federal Do Rio Do Sul Instituto De Informatico Curso De Ciencia Da Computacao, June 2021, https://www.lume.ufrgs.br/handle/10183/223683.

[3] K. J. Nesbit and J. E. Smith, "Data Cache Prefetching Using a Global History Buffer," 10th International Symposium on High Performance Computer Architecture (HPCA'04), 2004, pp. 96-96, doi: 10.1109/HPCA.2004.10030.