Nick Muenchen
Professor Srimani
CPSC 3220
25 November 2018

Project 4 Write Up

Trial Number	Time Taken Sequential (s)	Cache Hit Ratio Sequential	Time Taken Random (s)	Cache Hit Ratio Random
1	0.231808	0.0225	0.249485	0.013125
2	0.231034	0.0825	0.246835	0.016875
3	0.235487	0.04625	0.266222	0.01625
4	0.254313	0.031875	0.270807	0.01375
5	0.234333	0.021875	0.279804	0.0125
6	0.236949	0.04375	0.254923	0.014375
7	0.211617	0.043125	0.255365	0.01625
8	0.230625	0.03875	0.242194	0.016875
9	0.247618	0.026875	0.275284	0.01375
10	0.238639	0.098125	0.266925	0.014375
Averages:	0.2352423	0.0455625	0.2607844	0.0148125

As these results show, when the threads access sequential blocks, the cache hit ratio more than doubles. This makes sense as when threads which access data sequential access the same block, it is likely that they will actually share more than just the one block, as they will access the same next block if that wasn't the final block one of them accessed. This leads to a higher cache hit ratio, and because of that a faster runtime. However, the sequential accesses also showcases very high variability, much higher than random, because in the event that the threads do not begin near one another, they will not have any crossover leading them to have few cache hits. This is not as large a problem with random accesses because every block access has the same chance of being chosen at any given moment. These attributes of the two test scenarios also affects how the LRU policy works as the complete random scenario means that LRU has no real advantage over other policies because every block has an equal chance to be the next assessed rather than more recently used blocks being more likely to be accessed. On the other end of the spectrum, the sequential scenario results in LRU having large advantages because helps maximize the amount of shared blocks still present in the cache when a thread begins to reach blocks that have previously been used. When a thread reaches a single shared block, it is likely to continue having cache hits because the LRU policy had not yet removed the least recently used of the shared blocks, shown by the new thread having a cache hit on it. The downside of this, and also a potential cause of the variation in cache hit ratios of sequential tests, is that if a thread would share many blocks with another thread, but starts just slightly too late, all of the shared blocks in the cache may be overwritten just before the new thread has a chance to access them and have a cache hit. Essentially, LRU ensures that most of the blocks accessed by a given thread will be thrown from the cache in a short window, which can either be very helpful in generating cache hits as they will almost all be present, or be terrible if they are all pushed out right before they are needed.