Cache Performance Analysis

No Cache

Program	Cycles
prog_a	100030
prog_b	200040
prog_c	1580130
prog_d	1580130
prog_e	380030
prog_f	100010

64 Cache Lines & 16 Byte Block Size

Direct Mapped Cache

Program	Cycles
prog_a	90034
prog_b	40064
prog_c	465115
prog_d	494940
prog_e	86596
prog_f	228266

Fully Associative Cache

Program	Cycles
prog_a	90034
prog_b	40064
prog_c	447965
prog_d	474990
prog_e	78784
prog_f	228462

Two-Way Associative Cache

Program	Cycles
prog_a	90034
prog_b	40064
prog_c	447965
prog_d	474906
prog_e	78784
prog_f	228378

32 Cache Lines & 32 Byte Block Size

Direct Mapped Cache

Program	Cycles
prog_a	75028
prog_b	40052
prog_c	560119
prog_d	582965
prog_e	101840
prog_f	292032

Fully Associative Cache

Program	Cycles
prog_a	75028
prog_b	40052
prog_c	526965
prog_d	546929
prog_e	76892
prog_f	292450

Two-Way Associative Cache

Program	Cycles
prog_a	75028
prog_b	40052
prog_c	526569
prog_d	546621
prog_e	76892
prog_f	292252

Analysis

The data shows a significant performance improvement when using any cache configuration compared to having no cache at all.

- Program A gets almost no improvements when moving from cache to cache, however. This must be due to the fact that it is just doing the same calculation over and over, without any way to really optimize the process. There is a significant improvement when switching to a cache with a bigger block size. This must be trying to take advantage of more spatial locality
- Program B also did not see improvements from cache to cache, but it had a much larger improvement from no cache to having a cache. While it was also doing a repeated calculation, this one was a loop and so didn't need to switch out instructions, allowing the calculation to be done the fastest it could be and not needing to switch anything out of the caches.
- Program C showed improvement when moving from a direct mapped cache to one of the associative caches. This flexibility allowed it to keep memory address that were more temporally relevant, and only vacate cache lines that weren't being used.

- Program D had the most variance out of all of them. Since it was reading individual bytes instead of words, it was much more sensitive to how a cache handled both temporal and spatial locality. It did not get as much of a speed up from moving to caches as Program C did, but it still benefitted greatly.
- Program E was very much like C where it preferred the associative caches to the direct mapped cache. And also like B, it used a looping structure to optimize its locality, reducing the need to choose between temporal and spatial and being able to take greater advantage of both.
- Program F is interesting because it is not designed with any locality in mind. Across the board, all caches performed worse on this one. This is due to needing to vacate some cache line one basically every call. For a program like this, it is certainly better to not have a cache.

Overall, the programs benefit the most from caches with higher associativity, which reduce misses by offering more flexibility in block placement. However, gains begin to diminish between fully associative and two-way associative, suggesting diminishing returns for further complexity. In addition, changing the amount of cache lines and the block size didn't seem to have a consistent effect across the programs to be able to draw a certain conclusion on the matter. Some benefitted greatly, some it didn't matter at all, and some it made worse.