ASSIGNMENT 2: Bit Manipulation, Cache Effects

Task 1: Bit Arrays

Cahces:

Development enviroment: L1: 768KB, L2: 4MB ja L3: 16MB

Turso enviroment: L1: 32KB, L2: 256KB ja L3: 30MB

3.
During my test if fount that Set() operation is about 22-26% slower compared to get()
Most likely, ”only” 20% performance decrease is not related to memory hierarchy but is caused by data write to the array index compared to get() operations read.

4.
Result between first and second get() where inconclusive. Most of the time second get was slightly slower but when it was waster it could be double the speed of first get.

My theory is that running set first, cached most of the array to L1-cache, and because get uses the same random value vector as set-operation. It accessed the same cahcelines in the same order. Second get’s new vector causes same cache misses as some of the old data is already written over. Valgrind reported 0.2% cache misses for data.

Task 2: Bit Arrays

The source code currently contains implementation for sum() operation but I did not have the time to fix it validity. The calculating the index for uint64_t arrays is clearly broken. The current benchmark with the broken system show Sum-times increasing approximately linearly even though the precalculations aren’t part of the benchmark.
The sum times should decrease when block_size is smaller in contrary when block size larger. All operations are made with bit manipulation except __builtin_popcountl is used for calculating bits that aren’t on the precalculates sum_array. Precalculations increase linearly with block_size by design.