Project 2 Report

Jincheng Li (jl4569), Tianci Zhong (tz2278)

# Experiment Setup

1. Machines used: CLIC machines (mainly beijing, we ran a limited amount of experiments on nassau to make sense of a problem with the hardcoded 959 optimizations)
2. Machine characteristics (beijing)  
   clock frequency: 2.67 GHz  
   RAM: 5970MB  
   CPU cores: 3  
   L1 cache: 32K  
   L2 cache: 256K  
   L3 cache: 12288K
3. The machine was likely not idle since CLIC is a multi-user system and other people are probably running programs. However, we did run the experiments at different points in a day, and performance was similar.
4. We used gcc to compile the programs. The compiler flags used were:  
   -Wall –g –msse4.2 –msse4a –std=c99 –O3 –flto –march=native
5. We put in calls to the clock() function before and after the probes to measure the time taken for phase 2. Time taken to generate probes and print results are not taken into consideration.

# Results

Some additional details about our experiments: We are using 250 keys to construct the trees for all 3 cases, since we wanted to keep the number of keys constant across the different setups to make them comparable. We used several different values for the number of probes: 1e06, 2.5e06, 5e06, 1.25e07, 5e07, 1e08, to see how the number of probes affects performance (and as expected, time taken is linear with respect to number of probes).

## 17-17 Trees

For 17-17 trees, we can see from the chart that performance with SIMD optimizations is significantly better than the normal (part 1) approach. The speedup is roughly 5x the normal approach. Also, time taken is linear in the number of probes used, which is expected since each probe should take roughly the same amount of time. This result is not really surprising - SIMD instructions should be faster than the normal implementation of binary search within array elements. There are no conditional branches in the SIMD implementation, which saves the cost of branch misprediction. There are also no loops, so the number of instructions executed is likely to be less than binary search.

# 9-5-5-9 Trees

For the normal approach, 9-5-5-9 trees seem to take longer to perform a probe than 17-17 trees, possibly because the tree now has two additional levels. However, 9-5-5-9 trees take less time to perform a probe than 17-17 trees in the SIMD optimized case. Our explanation for this is that with smaller fanout factors, the SIMD operations for 9-5-5-9 trees operate on fewer keys at each level, and the total number of SIMD instructions goes down compared to searches on 17-17 trees. For a 17 fanout factor tree, a search on any level of the tree has to load 16 keys into CPU registers and perform 4 SIMD comparison operations. 3 of these instructions are actually somewhat “useless”, since the probe key only fits in one of the 17 ranges. For smaller fanouts, we perform fewer comparison operations, with the cost of traversing more levels of the tree.

Again, SIMD implementation outperforms the normal approach by a large margin. This time the performance improvement is nearly 7x the normal approach. Our understanding is that the reason for this is similar to what we mentioned in the above paragraph: with smaller fanout factors, SIMD can be more effective since fewer comparison operations are wasted.

## 9-5-9 Trees

The 9-5-9 tree case is much more interesting than the previous two. The comparison between the normal approach and SIMD is still similar to the previous two cases: the SIMD approach is much faster. However, this time there is also the hard-coded 959 tree optimizations (caching root node in registers + loop unrolling). As it turns out, on the beijing machine we were using, the SIMD implementation without further optimizations actually outperforms the additional 959 optimizations implementation. However, as we were trying to make sense of this, we tried compiling the program on a different machine, nassau, and running the program on both nassau and beijing again. Results show that what made the difference is which machine we used to compile the program. We believe that this difference is caused by the march=native flag, which enables CPU-specific compiler optimization options. If we compile the program on nassau, then the SIMD implementation runs slower than hard-coded 959 implementation. We don’t fully understand what kind of optimizations gcc uses, but this seems understandable, since gcc could be capable of figuring out whether to cache the root node in registers, and doing loop unrolling by itself.