Branch-prediction Effects in Quick Sort

Quick Sort with and Without Branching in the Partitioning Function

Max Boone (s2081318)

December 2, 2024

1 Introduction

This report briefly describes and benchmarks two implementations of quicksort: one implementation that uses a branching if-statement in its partitioning function, and one that uses an assigned conditional to avoid branch misses. First, the implementation of the sorting algorithm is discussed. Then, performance results of sorting are provided. Finally, the results are concluded.

2 Implementation

The quicksort algorithm can be divided into two steps. First, the sorting algorithm calls a partitioning function that separates the list into a partition with the smallest and a partition with the largest numbers. A pivot index shows where the partitions are split. Second, the sorting algorithm is called recursively on the left and right constituents and sorts them independently.

In the partitioning function, a conditional swapping is done based on whether the element is smaller than the chosen pivot. Due to the randomness of this operation, the branch predictor is likely to mispredict whether to jump or not to the swapping instruction. As an optimization, this is replaced by a conditional assignment and a temporary variable, to avoid having to jump to another instruction. The main difference between the functions is shown in Listing 1.

```
/// Branchless implementation (cargo build --features condition)
for j in 0..high {
    let c = list[j] <= list[high];
    let t = list[i];
    list[i] = list[j] * c as u32 + t * (1 - c as u32);
    list[j] = t * c as u32 + list[j] * (1 - c as u32);
    i += c as usize;
}

/// Branching implementation (cargo build --features branching)
for j in 0..high {
    if list[j] <= list[high] {
        list.swap(i, j);
        i += 1;
    }
}
```

Listing 1: Both partitioning implementations in Rust

In the final compiled instructions for the branchless implementation, the first two list accesses get a branching instruction to check for out-of-bounds access. However, as in-bounds access is certain, the branch predictor will not suffer here. Then, the conditional is calculated, and the swapping is implemented with conditional-move instructions.

The branching implementation is compiled in fewer instructions. It also does the out-of-bounds checks but jumps to the swapping and incrementing instructions if the condition evaluates to true. If the result of the condition is random, the branch predictor will have branch misses here.

3 Performance

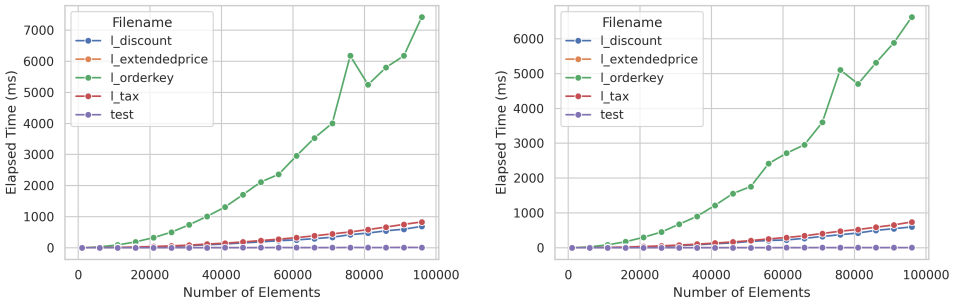
The system used for the benchmarks is a Surface Pro X SQ2 device with a Kryo-4XX-Gold ARMv8 SoC and 16 GiB of LPDDR4 4266 MHz RAM.

Each sorting run gets one warm-up run to get the file contents loaded and cached into memory. Even though only the sorting time is counted, a difference between a cold and warm run was observed during testing. Only warm runs are used in the results. The runtime is measured in the application by timings around the highest call of the sort function. A branch miss counter from the Linux `perf` API is used around the sort function.

Table 1: Speedup (Branching Time / Branchless Time)

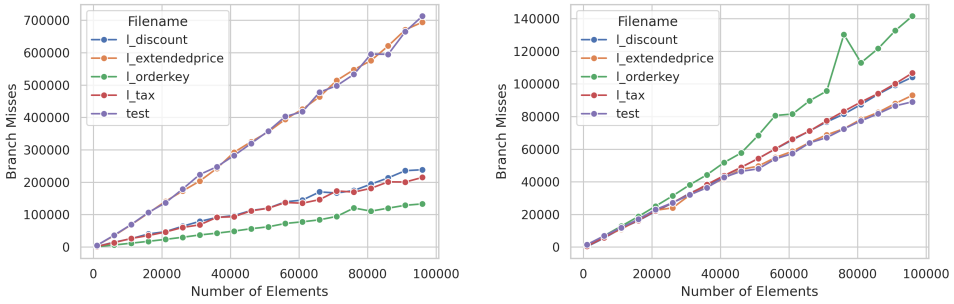
Mean	Std Dev	Min	25%	Median (50%)	75%	Max
1.429	0.544	0.819	1.097	1.128	1.736	3.876

In Figure 1, we see the runtimes of both implementations next to each other. The branchless implementation on the right is generally faster, with a speedup of 1.42 on average; further descriptives are given in Table 1. In Figure 2, we see the branch misses of both implementations. As expected, the branching implementation has many more branch misses than the branchless implementation. Furthermore, the branch misses occur more on datasets that are (more) randomly distributed.



(a) Branching (b) Branchless

Figure 1: Elapsed time to sort datasets.



(a) Branching (b) Branchless

Figure 2: Branch misses in sorting function.

4 Conclusion

The speedup of the branchless implementation is not as large as initially expected. This might be due to the architecture of the used processor or due to the compiler inserting branching instructions to check for out-of-bounds access. Nevertheless, the experiments confirm that the branch mispredictions are mitigated using the branchless implementation.