Branch-prediction effects in quick sort

Quick sort with and without branching in the partitioning function

Max Boone (s2081318)

December 1, 2024

1 Introduction

This report briefly describes and benchmarks two implementations of quicksort, one implementation that uses an 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.

2 Implementation

The quicksort algorithm works in two main steps. First, a partition function divides the list into two parts by swapping elements to ensure those smaller than a chosen pivot are on the left and larger ones are on the right. Second, quicksort is applied recursively to each part until the whole list is sorted.

In the partitioning function, a conditional swapping is done based on if 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
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
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 compiled assembler for the branchless implementation it shows that 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 increasing 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

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.

4 Conclusion