# PLD Assignment 1

Anders Lietzen Holst, wlc376

March 6, 2021

# 1 A1.2) Rosetta 2 - resubmission

## 1.1 A.1.2.b) - resubmission

- *Discuss advantages and disadvantages of this approach over recompiling source code to run on ARM.*

In my feedback, I was asked about the overall running time as compared to simply executing ARM. The newly added answers are in red.

### Advantages

The main advantage is of course portability. With Rosetta 2, users can run programs written for x86 on their ARM machines. This means that users can purchase a new Mac machine without having to let go of their old software written for x86.

This is of course also a huge benefit for the developers, since they do not have to target two different architectures when they want their software to run on both new and older machines, but instead simply target x86 and then let users with newer machines run through Rosetta.

### Disadvantages

the JIT compiler is prone to making bad decisions. It might AOT-compile code which is not run sufficiently many times to amortize the cost of compilation, or it might neglect to AOT-compile parts of the code which *is* in fact run often, potentially producing a bottleneck where there shouldn't be one.

In addition, there is some memory overhead in the JIT-compiler, since to function it essentially needs to keep the interpreter running, whilst occasionally invoking the compiler. This overhead can be significant if the input program is small.

### Performance implications

Running a program through Rosetta obviously has an effect on the overall running time of that program. In the *very best* case, using Rosetta can improve running time - but this only happens when the most demanding parts of the code are AOT-compiled early and when the less used parts of the code is never AOT-compiled.

In the worst case, Rosetta will first execute the program through the interpreter a number of times before eventually having compiled the entire

program - in this case, the running time will be roughly as large as if the program had originally been compiled to ARM and run, **plus** whatever time was spent interpreting the program in the first place.

Compared to simply running ARM and ignoring the time it takes to AOT-compile to ARM, the running time is significantly larger.

---

## 2 A1.4) Exercise 5.13

- *Solve exercise 5.13 from the course compendium.*

In my feedback, I was asked to name more advantages and disadvantages to using between using a heap vs. stack. The newly added answers are in red.

A frame contains parameters and (space for) local variables to the given function, as well as a return address for the function. Most functions will not have very many parameters and local variables - perhaps 3-6 and 5-15, respectively - and therefore most frames are relatively small. This plays well with generational collection, since in most cases the entire list of frames in play can fit in the first generation (or the first couple of generations), and can reduce the time taken for frame deallocation.

However, if a program exhibits many consecutive function calls, such as in the loop below:

```
for (i = 0; i < n; i++) {
  acc = f(acc, input[i]);
}
```

then the lower generations of the heap may repeatedly be filled and consequently collected many times throughout the loop; in the worst case, the lower generations might be rendered entirely useless for the duration of the loop, since actual user data that is to stay alive throughout the loop is quickly being moved to higher generations to make room for frames in these lower generations. This may to some degree nullify the benefit from generational GC.

### Other considerations

There are of course other things to consider besides the effects on (generational) garbage collection. For one, one very big disadvantage in allocating frames in the heap is the problem of fragmentation, which can never be a problem with stack allocated frames.

On the other hand, allocating frames on the heap has the big advantage that whereas there is typically a static bound on the size of stack memory, the heap size is only bounded by the amount of available memory or by how much the operating system is willing to allocate for the program. This

allows for potentially much deeper recursion, which may or may not be a benefit in some programs.

---