Who's Julia?

Simon Knudsen, Sonni Jensen, Asbjørn Jensen

Department of Mathematics and Computer Science,
University of Southern Denmark

1. Introduction

Java, C++ and Python are on the top ten of the most used programming languages. There are hundreds of languages, but not minding the hard competition, new languages are still created with the thought of doing better. An example of this is the relatively new programming language Julia, which has been developed with the idea to combine the best features of other languages.

The purpose is to find out how well Julia perform compared to some of the standard languages as of 2016. Julia will be compared to Python, Java and C++. Julia draws a lot of inspiration from Python, including syntax and the dynamic type system. Julia shares similarities with Java, mostly behind the scene mechanics, such as garbage collection and compiler optimizations. C++ was chosen since the developers of Julia claim that Julia is as fast as C.

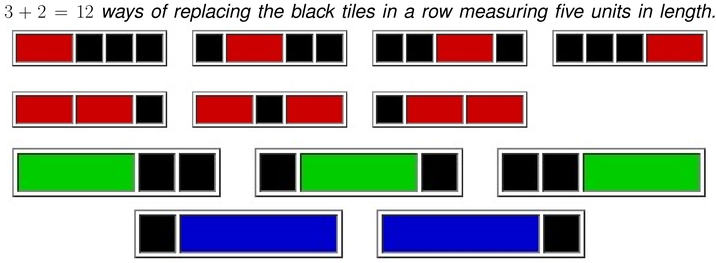
2. Julia

Julia is an object-oriented programming language, which has been under development since 2009. First released in 2012 and the newest stable version of Julia is version 0.4.5. The language has been created because the developers wanted a language with all the features they like from other languages.

The developers wanted the language to be open source, which means that everybody can read and modify the language. One of the ideas was to make the language as simple, readable and easy to learn as possible. The language is made for high performance and scientific computations while still supporting general purpose programming.

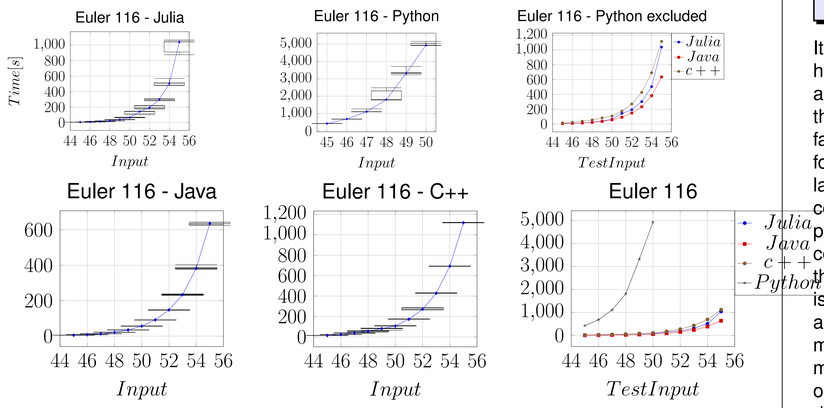
3. Project Euler problem 116

A row of five black square tiles is to have a number of its tiles replaced with coloured tiles from red(length two), green(length three), or blue(length four). If red tiles are chosen there are seven ways. If green tiles are chosen there are three ways. And if blue tiles are chosen there are two ways. Assuming that colours cannot be mixed there are $7 + 3 + 2 = 12$ *ways of replacing the black tiles in a row measuring five units in length.*



```julia
function solve(tileSize, squareSize) #m=color block size n = black box size
    if tileSize > squareSize
        return 0
    end
    solutions = 0

    for i = tileSize : squareSize
        solutions += 1
        solutions += solve(tileSize, squareSize-i)
    end

    return solutions
end

function calc(size)
    result = solve(2, size) #Red tiles
    result += solve(3, size) #Green tiles
    result += solve(4, size) #Blue tiles

    return result
end
```

Figur 2: *Julia implementation*



3.1 Results of Project Euler 116

It is clear that the graphs are exponential increasing. Another thing to notice is that the input is only increasing by one but is still making a huge difference in the run time. One of the reasons is that the problem is solved with recursion, for every extra one bit of space added to the black box a lot more recursive calls will have to be made.

The difference in time between the four languages are as expected. Python does not support any form of tail recursion optimization and the result is really slow. Java and Julia on the other hand does a good job at optimizing the recursive calls and is actually faster than C++ - keep in mind that no compiler flags were used in C++, so the default optimization level is used.

Julia and Java is close but Java is a bit faster, which is expected because of the fact that Java has been in development much longer than Julia and has a lot more optimization than Julia.

4. Pros and Cons of Julia

Advantages:

- Easy to learn and write
- Compiler optimizations
- General good performance
- No memory issues

Disadvantages:

- Compiler optimization breaks in some cases
- Not very flexible
- Missing features and bugs / weird behaviors

5. Conclusion

It can be difficult to make any conclusions by the benchmarking but it may give an idea of how Julia performs, compared to Java, Python and C++. One thing to keep in mind is that as of writing this report, Julia is in version 0.4.5 - it has yet to reach version 1.0.0 while the other languages are much older than Julia. This could mean that Julia might get even faster in the future and by the benchmarking it does look like that Julia is already performing well. The idea of the developers is clear, they are trying to make a programming language that is as easy to learn and write as Python but with a compiler that optimizes code like Java and both of these goals have more or less been reached by the developers. Julia was definitely easy to learn and by the benchmarking it does look like that the compiler is doing well with the optimization when it is possible. It does also feels like that the Julia developers are playing with the idea of giving the user some control over what is happening behind the scene. 'Playing' because this does not feel close to done. There are examples where they are giving the user an option to allocate memory for variables manually, but as soon as a new value is assign to that variable then Julia will change the memory size. Also it is not possible to free a declared variables memory entirely and the only operation the user can do with the garbage collector is force garbage collection and disable the garbage collector. Similarly, it seems that the option to implement code from other languages is kind of buggy and not complete.

Litteratur

Project Euler
www.projecteuler.net

Julia's offical homepage
www.julialang.org