UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA

RELATÓRIO FINAL:
“Energy Efficiency across Programming Languages”

Construção de Compiladores I

Prof. Dr. Clauirton Albuquerque

Bianca Karla Amorim de Sousa Melo - 11408143
Ednaldo Martins da Silva - 11427668
Glauber Ferreira Angelo - 20160144357

RELATÓRIO

CONSTRUÇÃO DE COMPILADORES

1. Introdução

Este trabalho é referente ao projeto final da disciplina de Construção de Compiladores I. Nesse documento, será feita uma análise comparativa do desempenho de três linguagens de programação em termos de consumo de memória, tempo de processamento e energia utilizada em implementações de cinco programas gerados por compiladores diferentes. O objetivo principal é compreender a relação entre o uso de processamento, energia e memória entre programas compilados de maneiras diferentes e responder a dois questionamentos fundamentais: "A linguagem mais rápida é sempre a mais eficiente em termos de energia?" e "Como o uso da memória está relacionado ao consumo de energia?"

2. Metodologia

Para a realização deste projeto, foram escolhidas as linguagens de programação *Kotlin*, muito utilizada para o desenvolvimento de aplicativos *server-side*, permitindo a escrita de códigos expressivos e concisos e mantendo total compatibilidade tecnologias baseadas em *Java* [1]; além dela, também foi escolhida a linguagem *Java*, extremamente popular, em que o seu código é compilado em *bytecode* e interpretado pela *Java Virtual Machine* [2]; por último, foi escolhido também a linguagem de programação *Python*, que está se tornando cada vez mais popular, e é uma linguagem interpretada, em que seus códigos fontes transformados em uma linguagem intermediária, que será interpretada pela máquina virtual da linguagem quando o programa for executado [3].

Os testes de medição foram realizados em uma máquina com o sistema operacional Ubuntu 18.04.2 (LTS), que possuía um processador Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz com 8GB de RAM e ITB de armazenamento. Além disso, para a medição, foi utilizado o software indicado no artigo *"Energy Efficiency across Programming Languages",* o *Intel RAPL (Running Average Power Limit) Tool.*

2.1 Medição do Consumo de Memória

Para calcular o consumo de memória para os testes feitos em Java e em Kotlin, foi utilizado o Objeto **Runtime** e o método estático **getRuntime()**, que retorna o próprio **Runtime**. Para entender o que cada método retorna, vejamos abaixo uma breve descrição de cada um deles (as informações foram obtidas do site da Oracle, disponível em *https://docs.oracle.com/javase/7/docs/api/java/lang/Runtime.html*):

freeMemory(): Retorna a quantidade de memória livre na Java Virtual Machine.

totalMemory(): Retorna a quantidade total de memória na máquina virtual Java. O valor retornado por

esse método pode variar com o tempo, dependendo do ambiente do host.

maxMemory(): Retorna a limite máxima de memória que a máquina virtual Java tentará usar. Se não houver um limite inerente, será retornado o valor Long.MAX_VALUE.

Para retornar o resultado da memória em uso, foi feito o cálculo de totalMemory()-freeMemory(). Em *Python*, ao importar o módulo OS, com uma simples linha de código utilizando *memory_info()*, é possível obter o valor de memória naquele momento da execução. Sendo assim, o cálculo pode ser feito da mesma forma que explicitado anteriormente.

2.2 Medição do Tempo

Para calcular o tempo necessário para executar as tarefas, em Java e Kotlin foi utilizado ***System.currentTimeMillis()***, que retorna o tempo de execução em milisegundos. Para obter o resultado, guarda-se o valor do tempo no início da execução em uma variável do tipo long/Long e ao final da execução da tarefas subtraímos o tempo atual pelo tempo inicial. Já em *Python*, foi utilizado ***millis = int(round(time.time() * 1000))***, importado do módulo *Time*, da própria linguagem.

2.3 Medição do Consumo de Energia

Para a medição do consumo de energia, foi utilizado o *software Intel RAPL (Running Average Power Limit) Tool*. Este programa só funciona em distribuições Linux, e sua instalação e funcionamento estão documentados no repositório [6] da *Green Software Lab*. A estrutura básica de funcionamento desse programa consiste em; cada linguagem de programação é separada em um diretório diferente e os algoritmos implementados em cada uma são lá colocados. Cada sub-pasta do benchmark, incluindo a das linguagens, contém um arquivo *Makefile*, em que está contido os comandos paras as regras: (1) compile; (2) run; (3) measure; (4) mem;

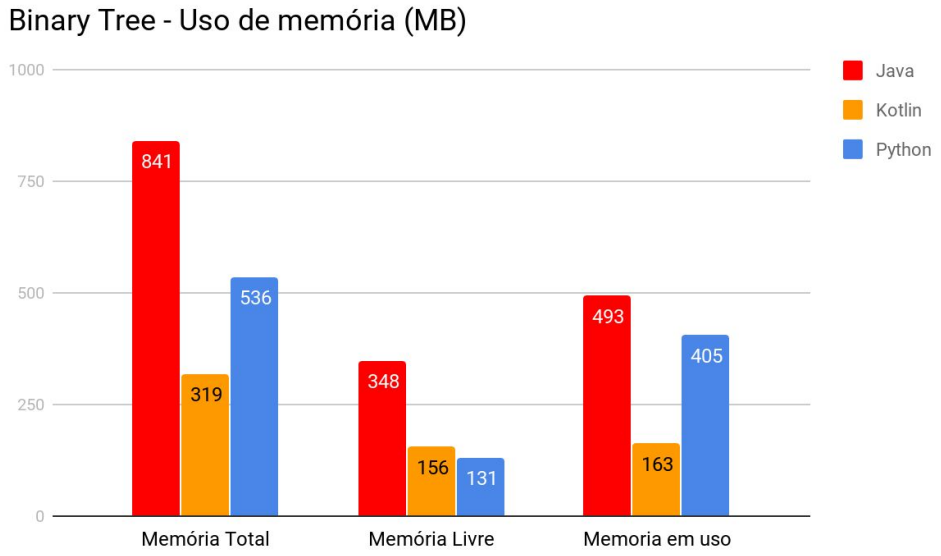
Para executar o programa, basta apenas compilar o arquivo *compile_all.py* e será gerada uma tabela no formato .CSV que conterá linhas com as informações referentes a: (1) Benchmark; (2) Consumo de energia (em Joules); (3) CPU (em J); e (4) Tempo (em milissegundos).

3. Resultados

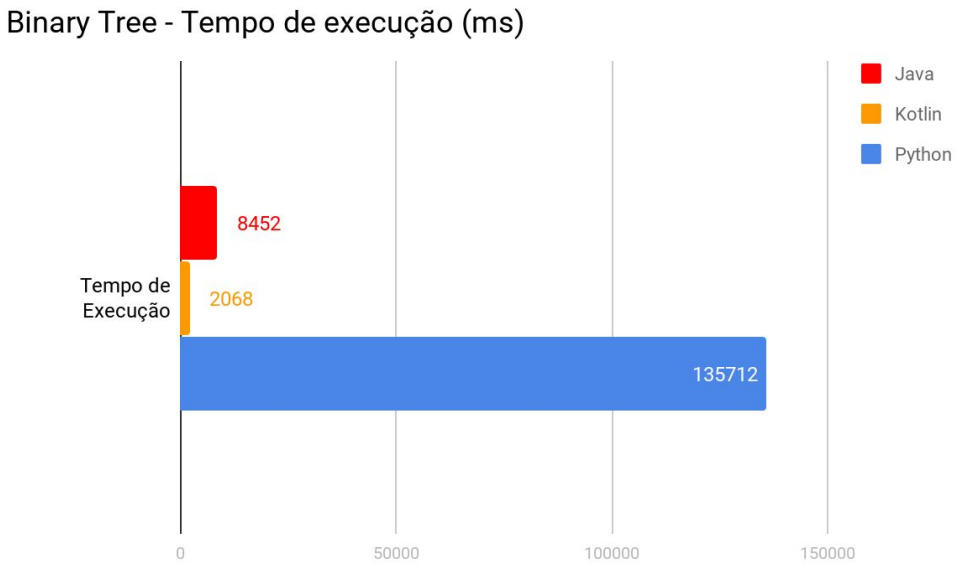
Primeiramente, vale salientar que alguns dos resultados relacionados ao uso de memória entre a aplicação em Java e a em Kotlin foram muito próximos, visto que o segundo foi projetado para interoperar totalmente com o Java, e a versão JVM de biblioteca padrão depende da biblioteca de Classes Java; enquanto que Python é uma linguagem interpretada e funcional, de tipagem dinâmica e forte.

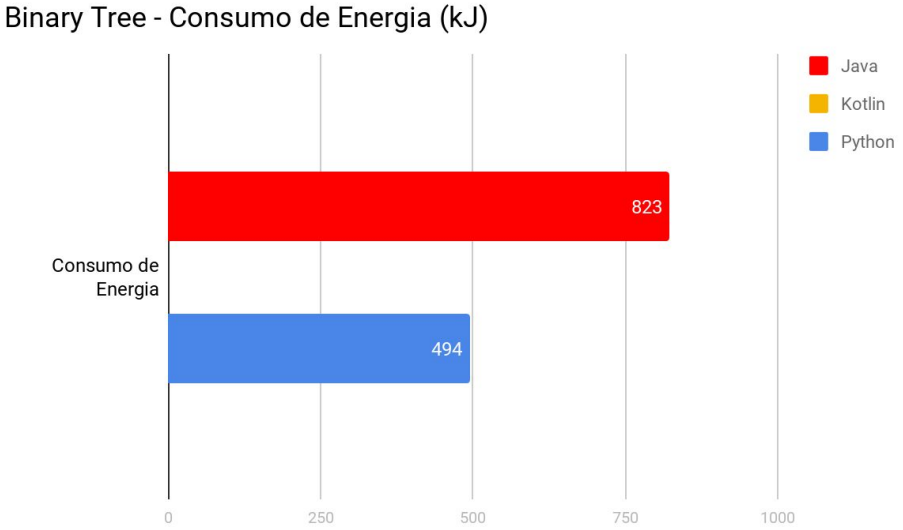
3.1 Binary Trees

Para o teste Binary Tree foi usado a profundidade (*depth*) em 21 e houve variação no uso de memória a cada execução. Os resultados são baseados na média.



Para execução da tarefa os códigos em *Java* e *Kotlin* foram implementados com *Pool de Threads*. Em Python, foi implementado algo similar, o *Multiprocessing*. Todavia, houve uma diferença drástica entre eles, levando a execução do *benchmark* em Java durar até 4 vezes mais do que a execução em *Kotlin* e em *Python* durar 19 vezes mais que em *Java*. Deve-se levar em conta que para esse problema a implementação do código em Python não foi usado o padrão Thread Pool para concorrência dos processos. O teste em Kotlin foi mais rápido que em C. Abaixo seguem os resultados obtidos:

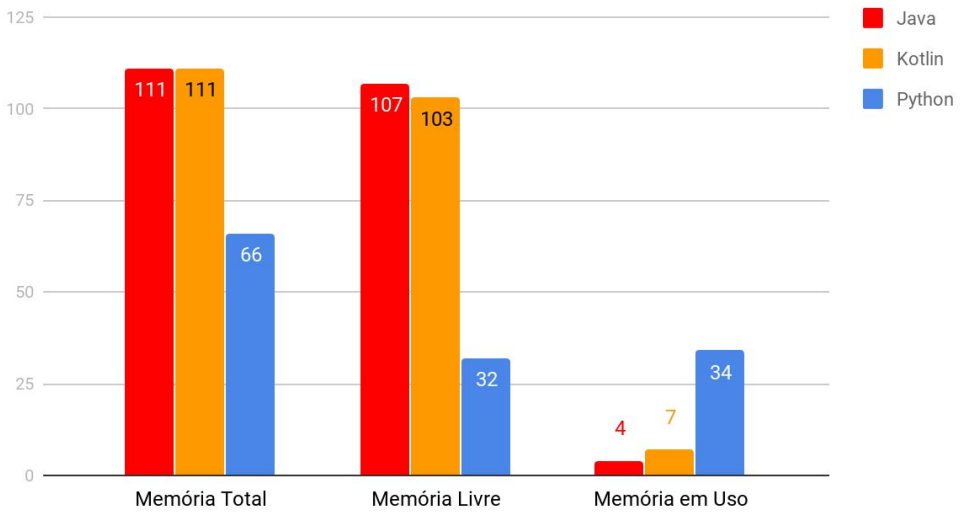




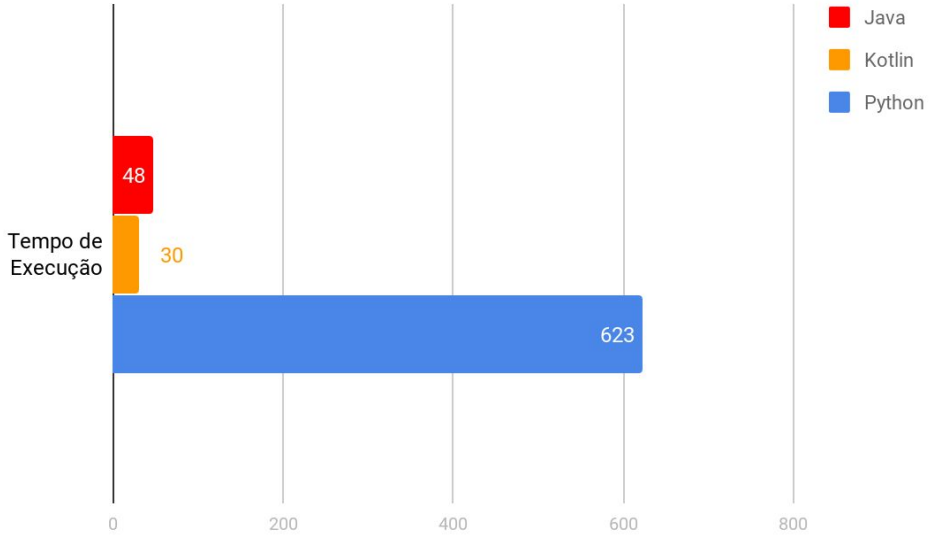
3.2 Fannkuch-redux

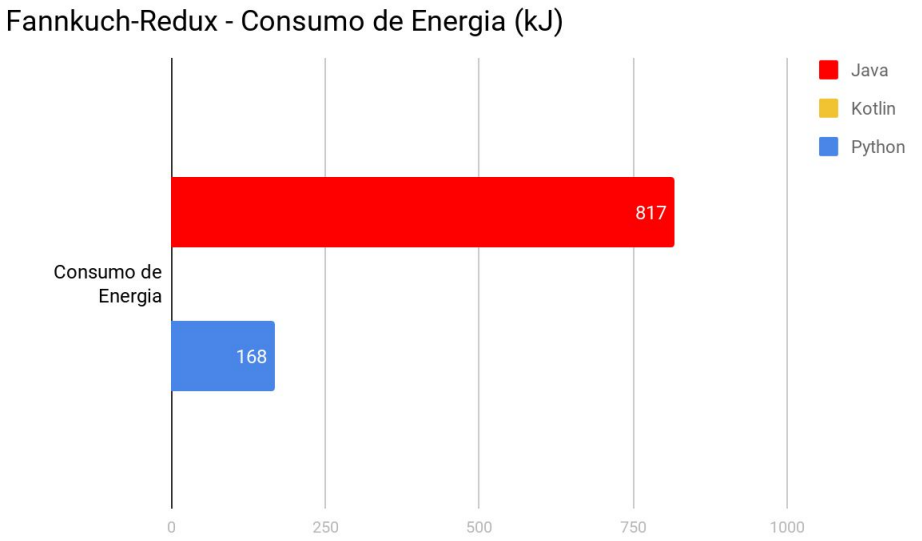
Para o teste Fannkuch-redux foi usado n = 9 e houve variação no uso de memória a cada execução. Java e Kotlin foram parecidos no teste, e Python usou mais memória para executar a tarefa. A Execução foi muito mais lenta em Python do que em Java e Kotlin.

Fannkuch - Uso de memória (MB)



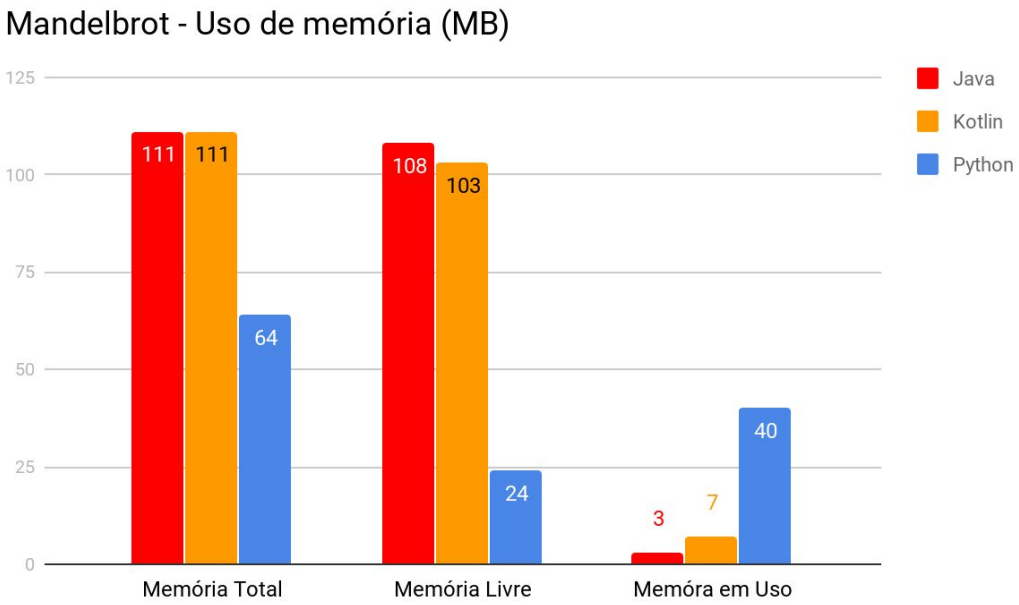
Fannkuch - Tempo de execução (ms)

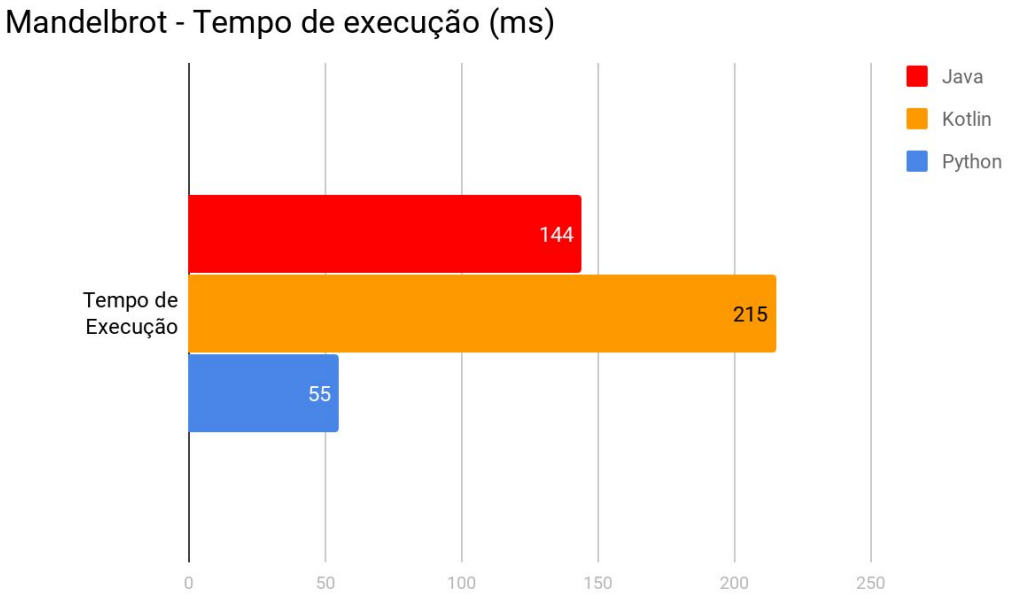


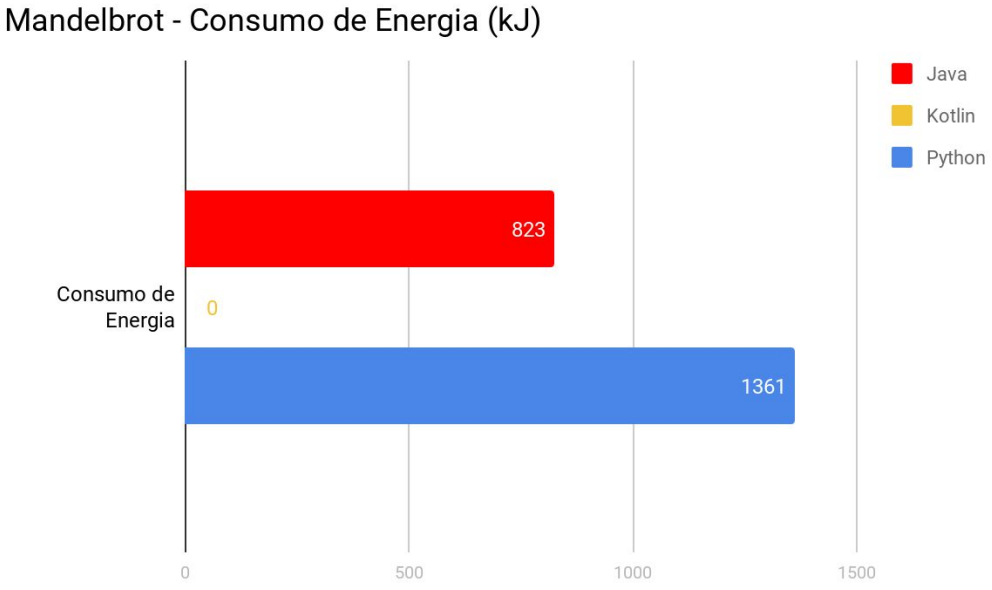


3.3 Mandelbrot

Para o teste Mandelbrot foi usado n = 1600 e houve variação no uso de memória a cada execução. Para o uso de memória Java e Koltin permaneceram similares e Python teve mais memória em uso durante a execução. Vejamos que para esse caso, a implementação em Python executou a tarefa em menos tempo que Java e Kotlin. Vale também ressaltar que Kotlin também foi o que mais demorou durante a execução do problema, embora tenha usado menos memória.

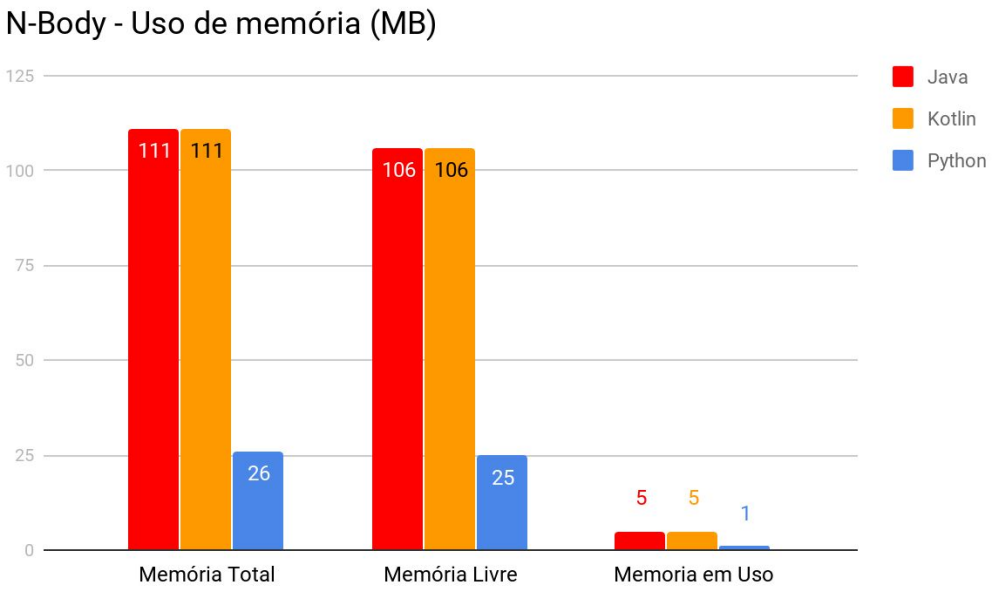




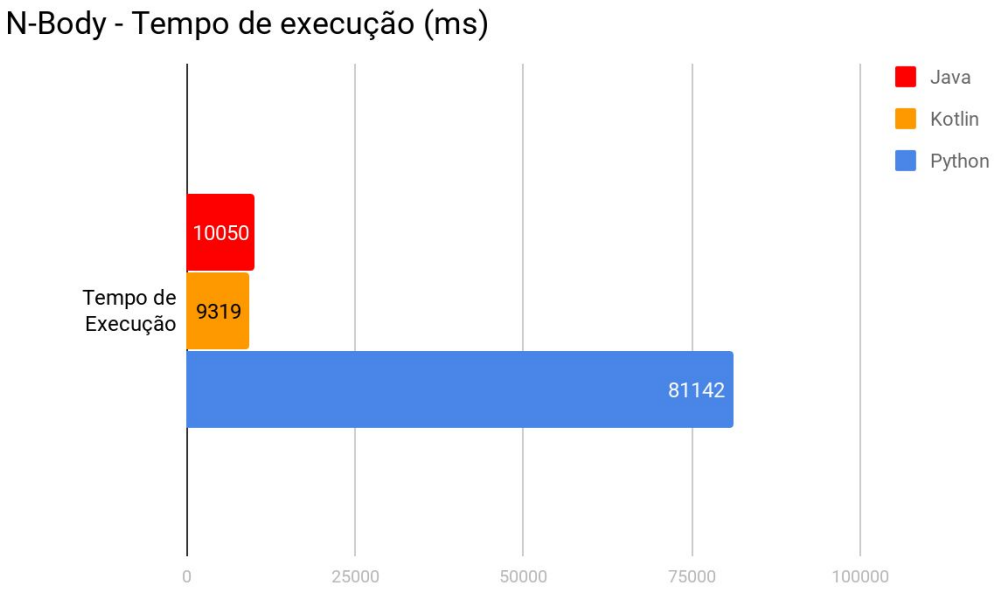


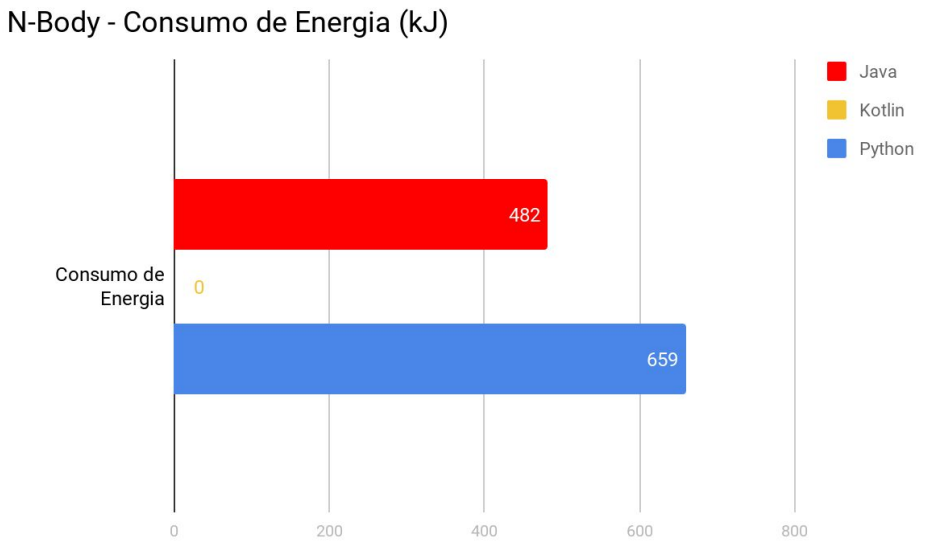
3.4 NBody

Para o n-body, foi utilizado o n = 50000000. Os resultados para Java e kotlin foram muito próximos com relação ao uso de memória, e Python também usou menos memória nesse teste.



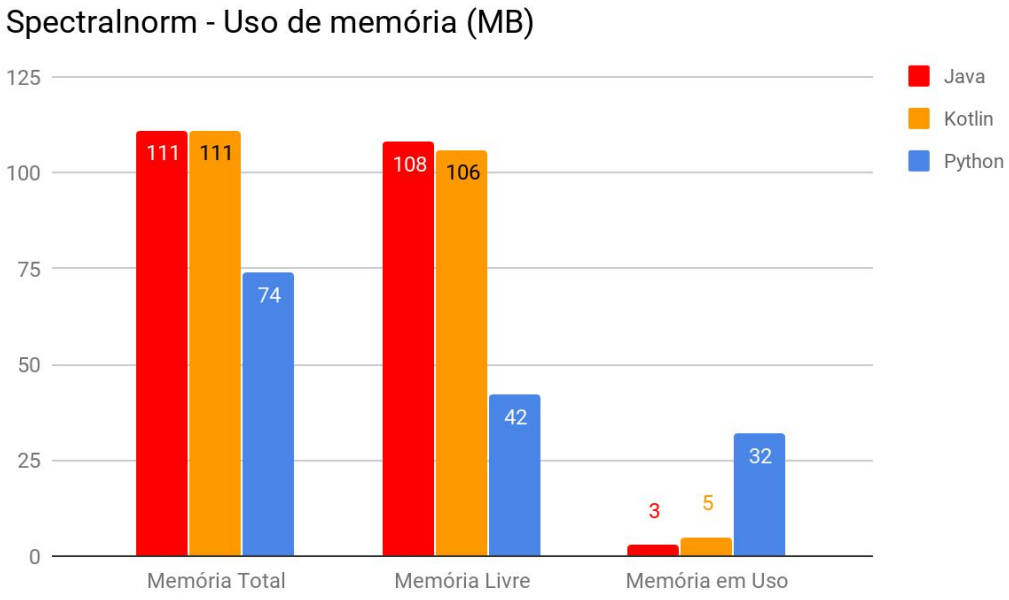
O tempo de execução foi um pouco menor para Kotlin com relação à aplicação em Java. Também deve ser levado em conta que a implementação do código para esse problema em python não foi feito com padrão Thread Pool para concorrência dos processos.

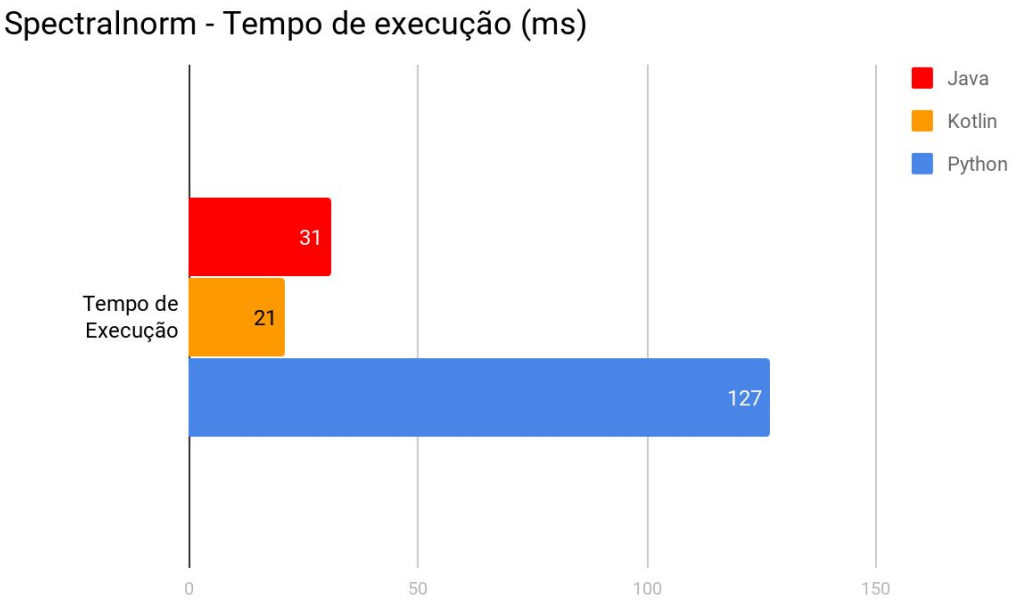


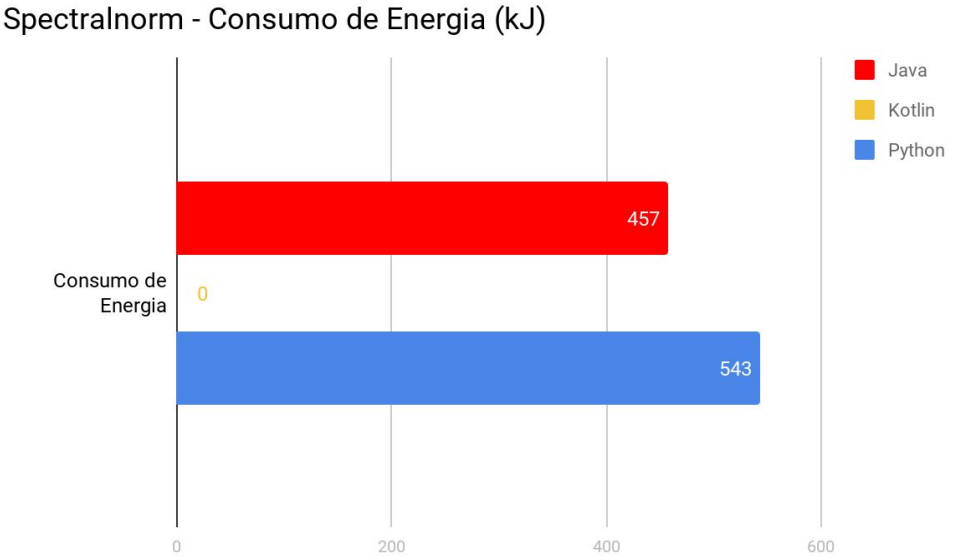


3.5 Spectralnorm

Para o Spectralnorm, foi utilizado o n = 100. Os resultados foram similares com relação ao uso de memória em Java e Koltin, mas com relação a Python não podemos dizer o mesmo. Todavia Mais uma vez Java e Kotlin deixam mais memória reservado para o programa. Os tempos tiveram os mesmos tiveram situações parecidas.




Spectralnorm - Tempo de execução (ms)
Java
Kotlin
Python
Tempo de Execução
31
21
127
0
50
100
150


Spectralnorm - Consumo de Energia (kJ)
Java
Kotlin
Python
Consumo de Energia
457
0
543
0
200
400
600

4. Considerações Finais

Com este trabalho e a realização de diversos testes e medições a respeito do consumo de energia, memória e tempo, foi possível realizar uma análise mais detalhada a respeito do funcionamento de certas linguagens de programação. Se compararmos *Python* e *Java*, há quem sempre apontará Python como melhor linguagem, por ser mais fácil e mais rápida para determinados problemas. Entretanto, isso não implica dizer que por ser mais rápida em certos casos, ela será sempre a mais eficiente. Por meio dos testes realizados, foi possível comprovar a eficiência de *Java* e, principalmente, da nova linguagem *Kotlin*, que está se mostrando cada vez mais poderosa. Com relação ao consumo de energia, podemos ver que Java geralmente consome menos energia para executar os Benchmarks com relação a Python, consumindo apenas mais energia no teste de Binary Trees. Os teste Kotlin para consumo de energia não foram relatados porquê não foi possível realizar os testes de medição em tempo hábil pelo RAPL.

Nos testes realizados com *Java,* notou-se que era sempre armazenado uma mesma quantidade de memória para seu uso, permitindo que o programador se preocupe apenas com a lógica da aplicação deixado os detalhes da alocação da memória a cargo da JVM. A memória da JVM é dividida em 3 partes: *Heap memory, Non-Heap* e *Other*. O tamanho da memória alocada é a quantidade total alocada pela JVM enquanto a memória usada é a quantidade que está realmente em uso. A quantidade exata de memória alocada é determinada pelas estratégias internas da JVM. Para trabalhos futuros, consultar o link anexo em *Java Memory Management for Java Virtual Machine* nas referências. Lá explica em detalhes o funcionamento das partições de memória da JVM. [9].

5. Referências

[1] Kotlin Language Documentation. Link: https://kotlinlang.org/docs/kotlin-docs.pdf

[2] Wikipédia: Java. Link: https://pt.wikipedia.org/wiki/Java_(linguagem_de_programa%C3%A7%C3%A3o)

[3] Vaz, Welton. "Python: Como Surgiu a Linguagem e o Seu Cenário Atual." *Eu Sou Dev*, 11 Jan. 2019, eusoudev.com.br/python-como-surgiu/.

[4] Documentação: Java. Link: https://docs.oracle.com/javase/7/docs/api/java/lang/Runtime.html

[5] Benchmarks Game Team: https://benchmarksgame-team.pages.debian.net/benchmarksgame

[6] Intel RAPL Tool. Link: https://github.com/greensoftwarelab/Energy-Languages#the-operations

[7] Energy Efficient Languages. Link: https://jaxenter.com/energy-efficient-programming-languages-137264.html

[8] "Which Programming Languages Use the Least Electricity?'. Link: https://thenewstack.io/which-programming-languages-use-the-least-electricity/

[9] Java Memory Management for Java Virtual Machine (JVM). Link: https://betsol.com/2017/06/java-memory-management-for-java-virtual-machine-jvm/