Universidade do Minho

Departamento de Informática

Projeto: Fase 2

(2.º Semestre/2023-2024)

Experimentação em Engenharia de Software

pg53944 João Pedro Moreira Brito
pg52690 José Pedro Gomes Ferreira

Contents

1 Introdução

O presente relatório descreve o projeto desenvolvido no âmbito da disciplina de Experimentação em Engenharia de Software, que visa explorar e analisar a performance de diferentes linguagens de programação. Este projeto foi dividido em duas fases distintas, cada uma com os seus objetivos específicos e metodologias de análise.

Na Fase 1, intitulada "Monitorização da performance de linguagens de programação", realizou-se a instalação, execução e análise de diversos benchmarks. Através destes benchmarks, o grupo foi desafiado a explorar as ferramentas de monitorização,RAPL e PowerCap, para investigar o impacto dos limites de consumo de energia na execução dos benchmarks.

Na Fase 2, denominada "Análise dos Resultados", o foco recai sobre a interpretação e avaliação dos dados coletados na fase anterior. Utilizando métodos estatísticos, o grupo tem como objetivo analisar o efeito do PowerCap no consumo de energia e tempo de execução dos benchmarks e comparar o desempenho de diferentes linguagens e de versões de compiladores/interpretadores.

Este relatório aborda detalhadamente cada fase do projeto, apresentando os procedimentos realizados, os resultados obtidos e as conclusões alcançadas. Por meio desta análise, esperamos contribuir para um melhor entendimento da performance de linguagens de programação e fornecer insights relevantes para futuras pesquisas e desenvolvimentos na área de Engenharia de Software.

2 Metodologia Usada e Ambiente de Teste

Neste projeto, foi crucial estabelecer um ambiente de teste padronizado e robusto, garantindo a confiabilidade dos resultados obtidos. Para isso, utilizamos uma metodologia cuidadosamente planeada e configuramos um ambiente de teste consistente para todas as linguagens de programação avaliadas.

Em relação aos benchmarks selecionados, utilizamos os seguintes repositórios:

- **Python:** Utilizamos o benchmark pyperformance, uma ferramenta amplamente reconhecida na comunidade Python para avaliar o desempenho da linguagem em diferentes cenários.
- **Haskell:** Optamos pelo benchmark NoFib, um conjunto de testes desenvolvidos pela comunidade Haskell para avaliar a performance de implementações GHC em uma variedade de algoritmos e programas.
- **Java:** Utilizamos o benchmark Dacapo, disponível em https://github.com/dacapobench/, que oferece uma coleção de programas Java representativos para análise de desempenho.

Além dos benchmarks, é fundamental destacar as especificações do ambiente de teste, que influenciam diretamente nos resultados produzidos. O ambiente de teste foi configurado da seguinte forma:

- **Processador:** Intel(R) Core(TM) i7-10875H CPU @ 2.30GHz;
- **Sistema Operativo:** Ubuntu 22.04;
- **Versão do Pyperformance:** 1.1, executado 10 vezes para cada limite de energia;
- **Versão do Nofib:** Não temos a informação específica da versão utilizada, porém realizamos 10 execuções para cada limite de energia;
- **Versão do Dacapo:**23.11, executado 10 vezes para cada limite de energia;
- **Versão do Python:** 3.10;
- **Versão do GHC:** 8.8.4;
- **Powercap ideal do Hardware:** 22.

Estas especificações garantiram a consistência dos resultados e permitiram uma análise precisa do desempenho das linguagens de programação sob diferentes condições de energia.

3 Análise de Resultados

Para realizar a análise de resultados para o pyperformance, o grupo deparou-se com alguns valores negativos em colunas que não deviam possuir valores negativos, por isso decidiu-se realizar um tratamento de dados e remover todas as linhas com valores negativos nas colunas "Package" e "Core(s)". Já para as outras benchmarks, como não se obteve outliers, nem missing values nos ficheiros .csv produzidos a partir do RAPL, o grupo optou por não realizar qualquer tratamento de dados. Assim, começou por criar, num ambiente jupyter notebook, diversos gráficos que permitissem analisar os resultados obtidos. Contudo, esta análise de resultados, irá se focar maioritariamente no impacto do powercap no consumo de energia e no tempo de execução dos programas.

3.1 pyperformance

Para o benchmark pyperformance, o grupo optou por executar programa a programa no RAPL, devido a este benchmark possuir diversos programas associados a ele. O grupo também testou este benhchmark com 3 versões diferentes do Python, para os diferentes programas presentes no benchmark de forma a tentar perceber e comparar o desempenho das versões 3.7, 3.8, 3.10 e 3.12 do Python.

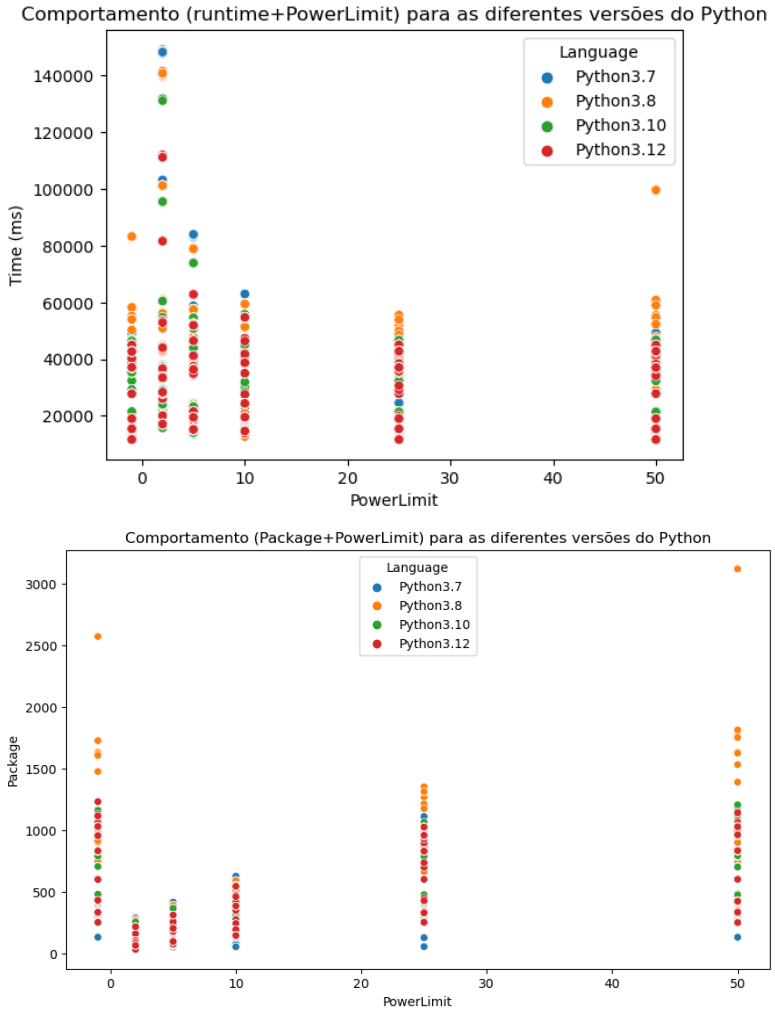

Figure 1: Gráficos da Relação entre PowerLimit e Tempo de Execução e da Relação entre PowerLimit e consumo de energia para os programas nas diferentes versões do python

Analisando os gráficos apresentados, podemos perceber que a versão 3.12 do Python possui, para grande parte dos programas, melhor tempo de execução e melhor consumo de energia comparando com as outras versões. O mesmo acontece com a versão 3.10 quando comparada com a versão 3.8 e 3.7, tendo para grande parte dos programas e para qualquer PowerLimit, melhor tempo de execução e melhor consumo de energia que as versões 3.7 e 3.8. Estes dados reforçam a importância de manter as versões do Python atualizadas, pois cada nova versão traz aprimoramentos que beneficiam tanto a performance quanto a eficiência energética.

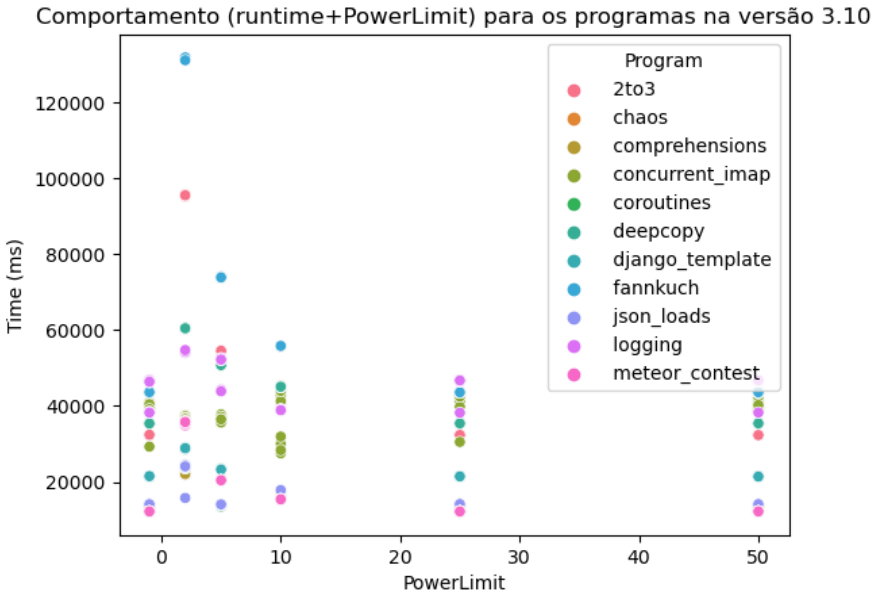

Figure 2: Gráficos da Relação entre PowerLimit e Tempo de Execução para os programas na versão 3.10 do Python

Fazendo agora a análise dos programas que se encontram na benchmark com a versão 3.10 do Python, podemos notar que para a maioria dos programas, a aplicação de um limite de energia diferente de "-1" (que indica sem limite) resulta num aumento no tempo de execução. No entanto, alguns programas mostram pouca ou nenhuma variação significativa ao modificar-se o limite de energia. Aqui está uma visão geral dos resultados para cada programa:

- 2to3: A aplicação de qualquer limite de energia aumenta significativamente o tempo de execução em comparação com o cenário sem limite (-1). Com o PowerLimit de 2, o tempo aumenta drasticamente para 81616 ms.
- chaos: O programa mostra um aumento considerável no tempo de execução com PowerLimit 2 (26031 ms) e depois uma diminuição gradual à medida que o limite aumenta, voltando a tempos próximos do cenário sem limite.
- comprehensions: Similar ao programa "chaos", há um aumento no tempo com PowerLimit 2 (17149 ms), mas tempos relativamente estáveis para os outros limites, com valores próximos ao sem limite.
- concurrent_imap: O comportamento é semelhante, com um pico no tempo de execução para PowerLimit 2 (29354 ms) e tempos mais estáveis para os outros limites.
- coroutines: Mostra uma pequena variação com tempos de execução relativamente estáveis independentemente do limite de energia imposto.

- deepcopy: O tempo de execução aumenta com a aplicação de limites de energia, especialmente notável com PowerLimit 2 (44498 ms), mas estabiliza em torno de 35361 ms para o cenário sem limite.
- django_template: A aplicação de limites de energia resulta em variações consideráveis no tempo de execução, especialmente com PowerLimit 2 (27867 ms).
- fannkuch: Mostra um aumento significativo no tempo de execução com PowerLimit 2 (111197 ms), mas uma diminuição progressiva com limites mais altos.
- json_loads: O programa tem tempos de execução relativamente estáveis, com pouca variação entre diferentes limites de energia.
- logging: Similar a "deepcopy", mostra um aumento notável no tempo de execução com a aplicação de PowerLimit 2 (43935 ms), mas tempos estáveis para outros limites.
- meteor_contest: O programa mostra um aumento significativo com PowerLimit 2 (33180 ms), mas tempos muito mais baixos e próximos do cenário sem limite para outros valores de limite.

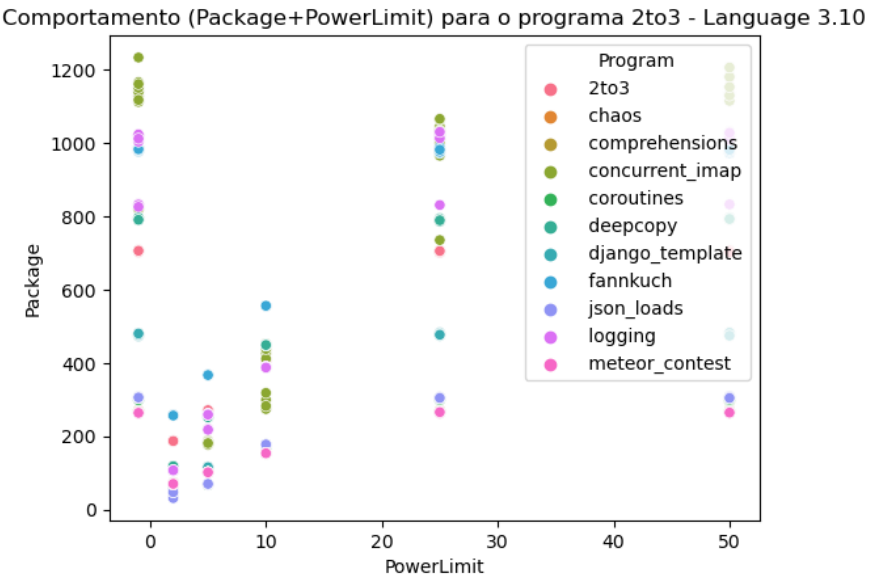

Figure 3: Gráficos da Relação entre PowerLimit e Consumo de energia para os programas na versão 3.10 do Python

Analisando os dados acima, podemos notar que, em geral, a aplicação de limites de energia reduz significativamente o consumo de energia em comparação com o cenário sem limite (-1). No entanto, a magnitude dessa redução varia conforme o programa e o valor do limite de energia imposto. Aqui está uma visão geral dos resultados para cada programa:

- 2to3: Sem limite, o consumo de energia é de 600.90. Com a aplicação de um PowerLimit de 25, o consumo é quase igual ao sem limite (599.37), mostrando pouca variação para limites altos, mas uma redução significativa com limites baixos (PowerLimit 2: 159.83).

- chaos: Sem limite, o consumo é de 265.88. Com PowerLimit 25, é de 265.22, mostrando pouca variação para limites altos e uma redução significativa para limites baixos (PowerLimit 2: 50.96).
- comprehensions: O consumo sem limite é de 267.42 e com PowerLimit 25 é de 266.02, indicando uma redução mínima para limites altos, mas uma redução significativa para limites baixos (PowerLimit 2: 33.78).
- concurrent_imap: Sem limite, o consumo é de 130.52. Com PowerLimit 25, o valor mostrado é anômalo (-260909.63), indicando um possível erro nos dados. Para os outros limites, há uma redução significativa (PowerLimit 2: 41.12).
- coroutines: Sem limite, o consumo é de 298.53. Com PowerLimit 25, é de 298.10, mostrando pouca variação para limites altos e uma redução significativa para limites baixos (PowerLimit 2: 38.54).
- deepcopy: Sem limite, o consumo é de -261238.86, um valor anômalo indicando erro nos dados. Para PowerLimit 25, é de 786.67, mostrando que para limites altos o consumo é maior.
- django_template: Sem limite, o consumo é de 421.20. Com PowerLimit 25, é de 422.72, mostrando pouca variação para limites altos e uma redução significativa para limites baixos (PowerLimit 2: 54.53).
- fannkuch: Sem limite, o consumo é de 831.70. Com PowerLimit 25, é de 826.51, mostrando pouca variação para limites altos e uma redução significativa para limites baixos (PowerLimit 2: 217.61).
- json_loads: Sem limite, o consumo é de 303.89. Com PowerLimit 25, é de 303.26, mostrando pouca variação para limites altos e uma redução significativa para limites baixos (PowerLimit 2: 31.04).
- logging: Sem limite, o consumo é de 790.18. Com PowerLimit 25, é de 787.64, mostrando pouca variação para limites altos e uma redução significativa para limites baixos (PowerLimit 2: 85.92).
- meteor_contest: Sem limite, o consumo é de 251.08. Com PowerLimit 25, é de 249.89, mostrando pouca variação para limites altos e uma redução significativa para limites baixos (PowerLimit 2: 65.11).

3.2 nofib

No caso do nofib, como o grupo estava a ter algumas adversidades a executar programa a programa, devido a dependências presentes entre eles, optou por executar a benchmark como um único programa para o RAPL.

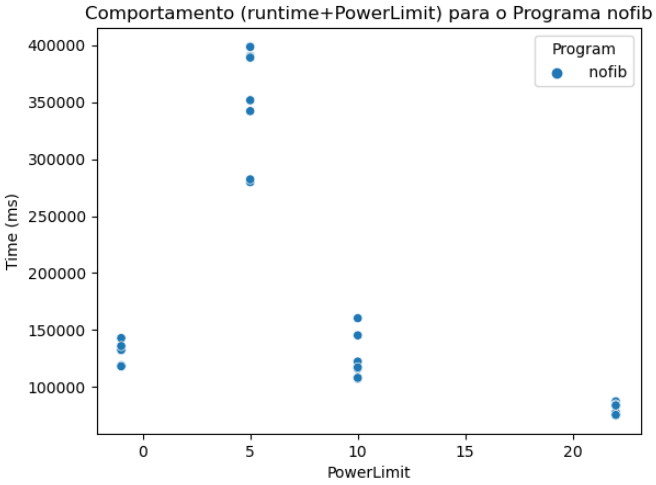

Figure 4: Gráfico da Relação entre PowerLimit e Tempo de Execução para o nofib

Como podemos ver no gráfico acima, para PowerLimit 22, o tempo de execução é de 75288 ms, que é o valor mais baixo obtido na execução do programa para todos os PowerLimits. Representando uma diferença de -36.3% em relação ao tempo de execução sem PowerLimit, que é 118132 ms. Este resultado não era esperado pelo grupo, contudo acredita-se que esta diferença deve-se a uma combinação complexa de fatores, incluindo algoritmos de gestão de energia, características da carga de trabalho e comportamento dinâmico do processador.

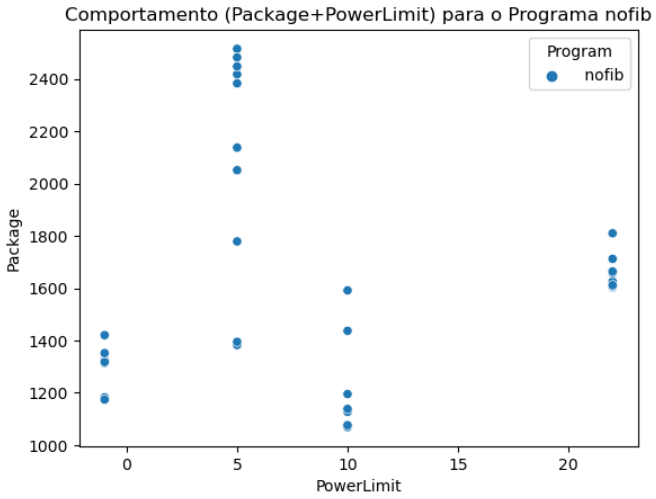

Figure 5: Gráfico da Relação entre PowerLimit e Consumo de Energia para o nofib

Já sobre o consumo de energia, pode-se visualizar que para um PowerLimit igual a 10, o consumo de energia é o mais baixo, tendo o valor de 1068.749268 packages, que representa uma redução de aproximadamente 8.92% em relação ao consumo de energia sem PowerLimit, que é de 1174.110413 packages. Pode-se ver também que, embora

o PowerLimit 22 possa resultar em um tempo de execução mais curto, o consumo de energia obtém valores altos comparado com outros PowerLimits, isto pode dever-se a uma variedade de razões, incluindo ineficiências no algoritmo de gestão de energia e overheads adicionais.

3.3 Dacapo

Para o Dacapo, tal como no pyperformance, o grupo optou por executar programa a programa no RAPL, devido a este benchmark possuir diversos programas associados a ele.

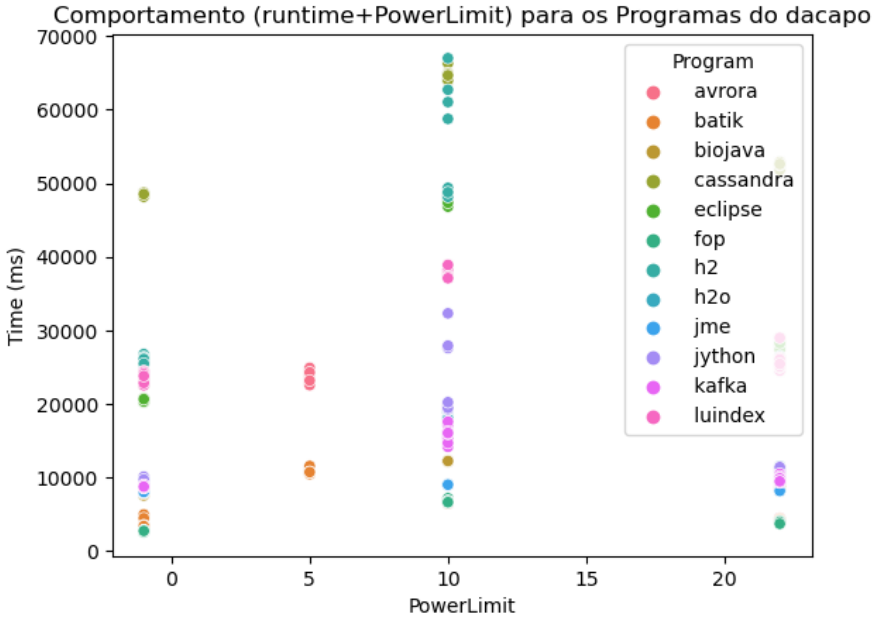

Figure 6: Gráfico da Relação entre PowerLimit e Tempo de Execução para o Dacapo

Ao analisar o gráfico acima, observou-se uma variação significativa nos tempos de execução entre os diferentes programas e os valores de PowerLimit. Alguns programas mostram uma resposta clara ao ajuste do PowerLimit, enquanto outros parecem menos sensíveis a essa configuração, o que permite reconhecer que o impacto do PowerLimit pode variar consideravelmente entre diferentes programas e cenários de uso.

Entre os programas analisados, identificou-se que o programa "batik" teve o menor tempo de execução com PowerLimit 22, registando um tempo de aproximadamente 4101 ms. Ao comparar esse valor com o tempo de execução sem PowerLimit (-1), que foi de cerca de 3282 ms, observou-se uma diferença percentual de aproximadamente 24.98% a mais do tempo de execução sem PowerLimit.

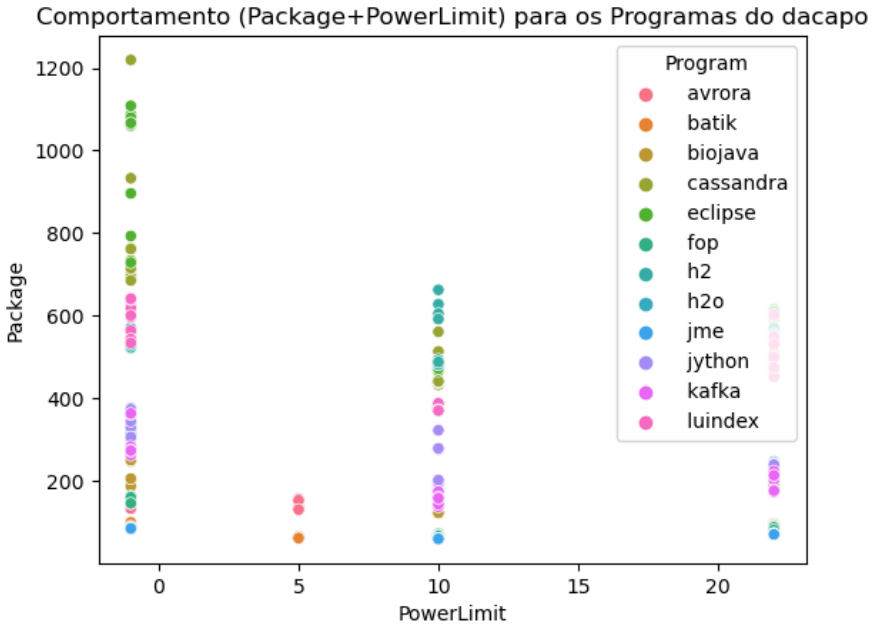

Figure 7: Gráfico da Relação entre PowerLimit e Consumo de Energia para o Dacapo

Analisando o gráfico acima, é visível que o consumo de energia varia consideravelmente entre os diferentes programas e ajustes de PowerLimit. Alguns programas demonstram ser mais eficientes energeticamente em determinadas configurações de PowerLimit, enquanto outros mostram uma sensibilidade menor a essas mudanças.

Pode-se visualizar também o programa "jme" teve o menor consumo de energia com PowerLimit, registando aproximadamente 58.28 com PowerLimit 10. Isso sugere que o programa "jme" é mais eficiente energeticamente quando o PowerLimit é ajustado para 10 em comparação com outras configurações.

Ao comparar os consumos de energia com PowerLimit com aqueles sem PowerLimit, observa-se diferenças percentuais significativas. Por exemplo, para o programa "jme", o consumo de energia com PowerLimit 10 foi cerca de 29.54% menor em comparação com o consumo sem PowerLimit. Essa redução no consumo de energia destaca a eficácia do ajuste do PowerLimit na otimização do consumo de energia e na promoção da eficiência energética dos programas.

4 Comparação do Desempenho das Diferentes Linguagens

Comparando o desempenho das diferentes linguagens, destaca-se, primeiramente, que a linguagem Haskel, das 3 linguagens analisadas é a que possui pior desempenho tanto em termos de tempo de execução quanto em consumo de energia, especialmente quando os ajustes de PowerLimit são aplicados. O que sugere que o ambiente de execução de Haskell pode não ser tão otimizado para a gestão de energia em comparação com as outras linguagens analisadas.

A linguagem Python, especialmente nas versões 3.12 e 3.10, mostrou-se superior em termos de eficiência energética e tempo de execução, com melhorias significativas observadas em relação às versões anteriores, 3.7 e 3.8.

Para a linguagem Java, utilizando o benchmark Dacapo, observou-se que alguns programas, como "batik" e "jme", responderam bem aos ajustes de PowerLimit, apresentando bons resultados tanto em termos de tempo de execução quanto em eficiência energética. No entanto, a variação significativa nos resultados sugere que a sensibilidade ao PowerLimit pode depender muito do tipo de programa em execução.

Analisando os dados apresentados anteriormente, pode-se concluir que o Python demonstrou ser a escolha mais eficiente em termos de performance e consumo energético, que o Java também mostrou boa eficiência em certos cenários específicos, enquanto o Haskell teve o pior desempenho em geral.

5 Conclusão

As conclusões tiradas ao comparar as linguagens de programação foram feitas unicamente a partir dos dados obtidos nos testes realizados. É importante ressaltar que estas conclusões podem não estar 100% corretas, uma vez que para uma comparação precisa das linguagens seria necessário testar os mesmos programas nas mesmas condições para as diferentes linguagens.

É também relevante mencionar que o grupo, composto por apenas por dois elementos, enfrentou desafios significativos na gestão de recursos. Este fator limitou a profundidade e o alcance dos benchmarks realizados, com mais tempo e recursos, o grupo acredita que poderia ter realizado uma análise ainda mais detalhada e abrangente, explorando diversos aspetos adicionais que afetam o desempenho e o consumo de energia.

Por fim, a realização deste trabalho permitiu ao grupo aprender como utilizar a ferramenta RAPL, proporcionando um entendimento prático sobre a medição e a gestão de consumo de energia em diferentes programas, para além disso, consegui-se, também, perceber o impacto do PowerLimit no desempenho e na eficiência energética dos programas. Para futuras abordagens ao tema, o grupo poderia fazer uma investigação mais extensa, incluindo uma variedade maior de linguagens de programação e tipos de carga de trabalho, bem como um estudo mais detalhado das razões subjacentes às variações observadas no impacto do PowerLimit.

.

Bibliography

[1] https://github.com/python/pyperformance

[2] https://gitlab.haskell.org/ghc/nofib

[3] https://github.com/dacapobench/dacapobench/releases/tag/v23.11-chopin