On the Energy Efficiency of Sorting Algorithms

André Nunes
Universidade do Minho
Portugal
a85635@alunos.uminho.pt

João Silva
Universidade do Minho
Portugal
pg50495@alunos.uminho.pt

Marco Sampaio
Universidade do Minho
Portugal
pg47447@alunos.uminho.pt

Abstract

Este documento apresenta o estudo prático "A Eficiência Energética de Algoritmos de Ordenação", que foi desenvolvido no contexto da disciplina "Experimentação em Engenharia de Software", sob a orientação do professor João Saraiva.

O propósito central deste projeto é analisar a eficiência de distintas linguagens de programação utilizando algoritmos de ordenação variados. Neste sentido, foram realizadas uma série de medições empregando duas linguagens, C e Python, com diversos tamanhos de entrada para alguns algoritmos em particular: selectionSort, insertionSort, bubbleSort e radixSort.

Os resultados obtidos foram examinados e as conclusões foram baseadas tanto nas métricas apuradas como na análise estatística. O intuito dessas avaliações é determinar qual das linguagens demonstra um desempenho superior em diferentes cenários.

Keywords Green Software, Programming Languages, Sorting Algorithms

1 Introdução

Neste documento, apresentamos a pesquisa realizada intitulada "A Eficiência Energética de Algoritmos de Ordenação", efetuada no contexto da disciplina "Experimentação em Engenharia de Software", sob a supervisão do professor João Saraiva, durante o ano acadêmico de 2022/2023.

A eficiência dos sistemas de software não se mede apenas em termos de velocidade de processamento, mas também pelo seu consumo energético. No âmbito deste projeto, o nosso foco principal foi analisar a performance de diferentes linguagens de programação ao implementar uma variedade de algoritmos de ordenação. Para isso, realizamos uma série de medições, usando duas linguagens em particular: C e Python. Aplicamos diferentes tamanhos de entrada (10.000, 20.000 e 30.000) para quatro algoritmos específicos: SelectionSort, InsertionSort, BubbleSort e RadixSort.

Os dados obtidos através destas medições foram então submetidos a uma análise estatística aprofundada, com o objetivo de determinar qual das linguagens apresentava melhor performance em diferentes cenários. Estas avaliações não só nos permitiram identificar a linguagem de programação mais eficiente, mas também aperfeiçoar nossa compreensão das relações entre algoritmos, linguagens de programação e consumo de energia.

Perfil ES'23, 2023
2023.

2 Metodologia

Neste estudo, foram implementados os algoritmos de ordenação Bubble Sort, Insertion Sort, Radix Sort e Selection Sort em C e Python. O consumo energético e desempenho foram comparados entre as linguagens. Foi usado um conjunto de dados de entrada pré-definido. A biblioteca RAPL e o pacote lm-sensors foram usados para monitorar o consumo de energia e a temperatura durante a execução dos programas. O tempo de execução e uso de memória também foram monitorizados. A biblioteca powercap foi utilizada para limitar o consumo de energia em valores como 30W, 40W, 50W e 60W. O estudo visou analisar a eficiência energética, tempo de execução e uso de memória dos algoritmos em diferentes cenários e linguagens de programação.

2.1 Implementação de Algoritmos de Ordenação

Foram implementados os seguintes algoritmos de ordenação: Bubble Sort, Insertion Sort, Radix Sort e Selection Sort. As versões em C e Python de cada algoritmo foram testadas para comparar seu consumo energético. As duas linguagens foram utilizadas no estudo para verificar se as diferenças de eficiência energética e tempo de execução, geralmente associadas a essas linguagens, poderiam ser observadas e medidas em resultados reais. C é conhecida por ser altamente eficiente em termos de energia e tempo de execução, enquanto Python é geralmente considerada menos eficiente nesses aspectos. Ao comparar o consumo energético dos algoritmos implementados em ambas as linguagens, foi possível verificar empiricamente se essas características se mantêm e quais são as diferenças reais entre elas em relação à eficiência energética.

Antes dos testes, foi gerado um único conjunto de dados de entrada que foi utilizado em todas as execuções dos algoritmos. Essa abordagem garantiu que o conjunto de entrada fosse sempre o mesmo, eliminando qualquer influência da geração dos dados no consumo energético dos algoritmos, assim como inconsistências nos resultados . Dessa forma, o estudo concentrou-se exclusivamente na análise do consumo energético durante a execução de cada algoritmo.

A implementação dos algoritmos de ordenação foi obtida a partir do repositório do GitHub disponível no seguinte endereço: https://github.com/diptangsu/Sorting-Algorithms.

Esse repositório fornece uma coleção abrangente de implementações de algoritmos de ordenação em diferentes linguagens de programação.

Utilizando esse recurso, foi possível ter acesso a uma base sólida e confiável de implementações já existentes dos algoritmos de interesse. A utilização dessas implementações disponíveis economizou tempo e esforço, permitindo focar nas análises do consumo energético dos algoritmos de ordenação em diferentes contextos e linguagens.

2.2 Monitorização do Consumo de Energia

Implementamos um código em C que utiliza a biblioteca RAPL (Running Average Power Limiting) da Intel, e o pacote lm-sensors para medir, limitar e monitorizar o consumo de energia e a temperatura durante a execução de um programa especificado.

- Monitorização da temperatura com lm-sensors: Antes de cada execução do programa, o código verifica a temperatura do processador utilizando o pacote lm-sensors e um comando de shell incorporado sensors | grep 'Package'. Se a temperatura for maior que 75°C, o código espera (usando a função sleep()) e verifica novamente a temperatura até um máximo de 4 segundos ou até que a temperatura caia abaixo de 75°C.
- Execução e medição do programa especificado:
  O programa a ser executado é passado como argumento para este programa em C. O comando é executado um número determinado de vezes (especificado pelo usuário), e cada execução é monitorizada individualmente. Antes de cada execução, a função rapl_before(fp, core) é chamada para começar a medição de energia, e depois de cada execução, a função rapl_after(fp, core) é chamada para terminar a medição. Estas funções estão definidas na biblioteca RAPL.
- Monitorização do uso de Energia:
  A função rapl_before(fp, core) registra a energia consumida antes da execução do programa e rapl_after(fp, core) registra a energia consumida depois da execução do programa. A diferença entre essas duas medidas é o consumo de energia do programa durante a sua execução. Esta informação é então escrita num arquivo de saída.
- Monitorização do tempo de execução e uso de memória:
  Além da monitorização de energia e da temperatura, o código também registra o tempo de execução do programa e o uso de memória. O tempo de execução é medido usando funções da biblioteca time.h. O uso de memória é monitorizado usando a função getrusage() da biblioteca sys/resource.h.

As leituras de energia através da interface RAPL são atualizadas aproximadamente a cada milissegundo (1 kHz). Isto proporciona uma estimativa de energia muito granular e de alta frequência, permitindo um acompanhamento preciso do consumo de energia durante a execução do programa.

2.3 Limitação do Consumo de Energia com PowerCap

A biblioteca powercap é usada para limitar o consumo de energia. Primeiro, o arquivo de controle de tipo de energia é aberto para habilitar o controle de energia. Em seguida, o arquivo da zona de energia é aberto para habilitar essa zona de energia. Por fim, abre-se o arquivo de restrição para definir o limite de energia. O limite de energia é então definido para um valor específico que vamos variando. Fizemos testes para 30W, 40W, 50W e 60W.

3 Benchmarking do Desempenho da Implementação de Algoritmos de Ordenação

No contexto da análise de dados do nosso estudo, o grupo começou por indentificar os outliers da medições efetuadas de modo a que os dados utilizados fossem o mais coerentes possível. Recorrendo à técnica de boxplot, que exibe a distribuição dos dados por meio de quartis, o grupo conseguiu identificar possíveis valores discrepantes. Após a identificação e remoção dos outliers, foi possível prosseguir com a análise das estatísticas básicas e das relações entre os dados. Nessa etapa, foram calculadas medidas descritivas, como média, mediana, desvio padrão e variância, que forneceram informações importantes sobre a tendência central, a dispersão e a forma da distribuição dos dados.Além disso, foram exploradas as relações entre as variáveis presentes nos dados. Isso pode ser feito utilizando a análise de correlação, que mede a intensidade e a direção da relação linear entre duas variáveis. A matriz de correlação permite identificar se há associações positivas, negativas ou nulas entre as variáveis, auxiliando na compreensão dos padrões e tendências presentes nos dados. Após a análise das estatísticas básicas e das relações entre os dados, o grupo prosseguiu para a identificação de similaridade entre grupos. Para realizar esta tarefa, recorremos a técnicas de clustering como o K-means, DBSCAN, Hierarchical Clustering. No seguimento de todo este tratamento de dados foi feita uma interpretação de todos estes resultados de modo a encontrar os algoritmos de ordenação mais eficientes.

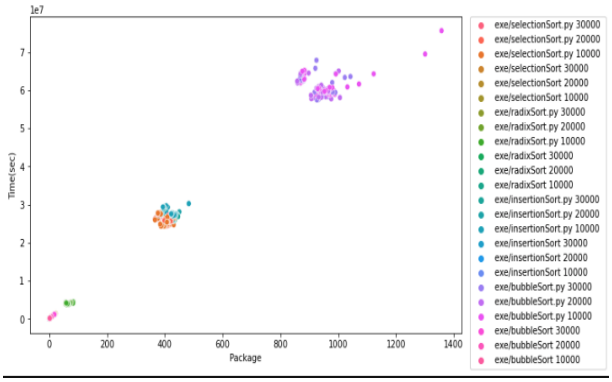

Figure 1. Scatter-plot on Time and Packkage

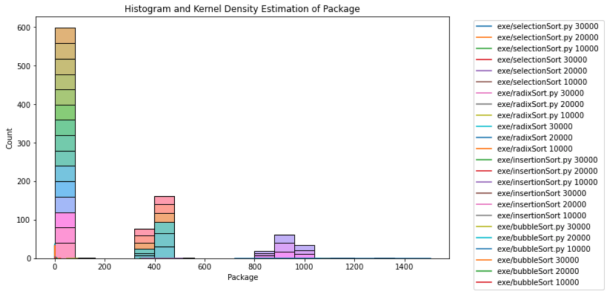

Figure 2. Histograma e Estimativa de Densidade de Kernel de Package

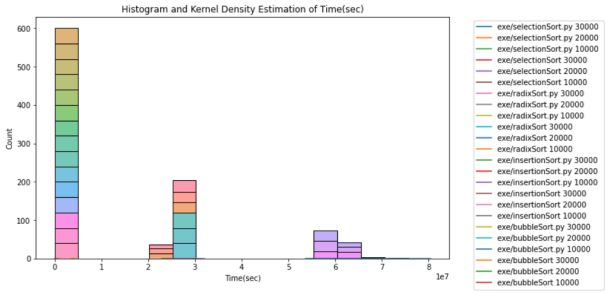

Figure 3. Histograma e Estimativa de Densidade de Kernel de Tempo

Como podemos perceber pelos histogramas e o "scatter-plot" o tempo e o consumo energético estão correlacionados facto que depois podemos comprovar pelo cálculo da sua correlação. É notório que os algoritmos que demoram mais tempo a executar são também aqueles que consomem mais energia. Em qualquer das análises que fizemos são identificados sempre 3 grupos distintos facto que se deve aos 3 tamanhos de inputs usados nas medições. Isto pode ser comprovado quando utilizamos os algoritmos de clustering que identificaram sempre 3 clusters bem denotados.

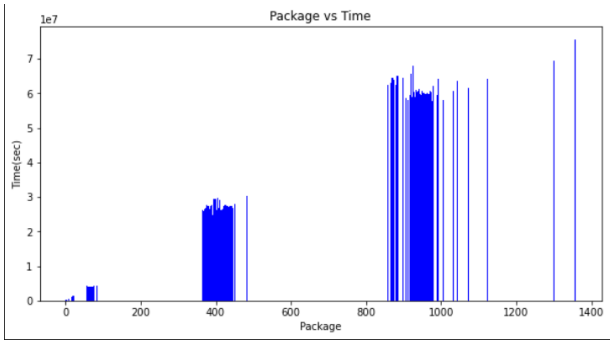

Figure 4. Histograma Package vs Tempo

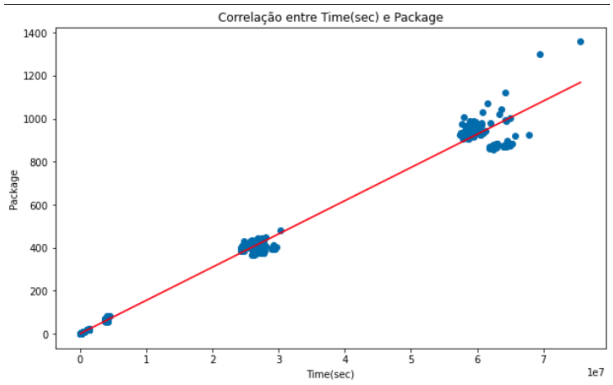

Figure 5. Correlação entre Tempo e Package

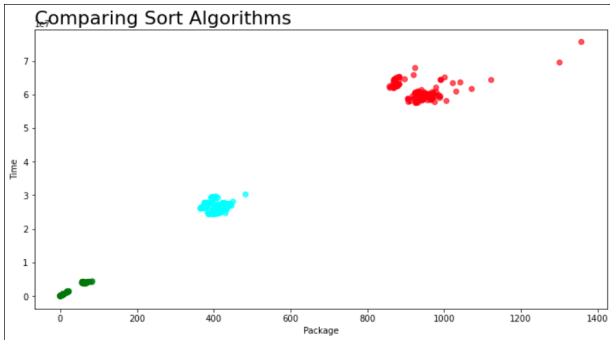

Figure 6. Clustering com Kmeans

4 Ranking de Algoritmos de Ordenação

Tempo	Consumo energético
RadixSort	RadixSort
SelectionSort	SelectionSort
InsertionSort	InsertionSort
BubbleSort	BubbleSort

Table 1. Tabela de Ranking dos algoritmos

Nos algoritmos em estudo conseguimos aferir que o consumo energético e o tempo de execução estão relacionados.

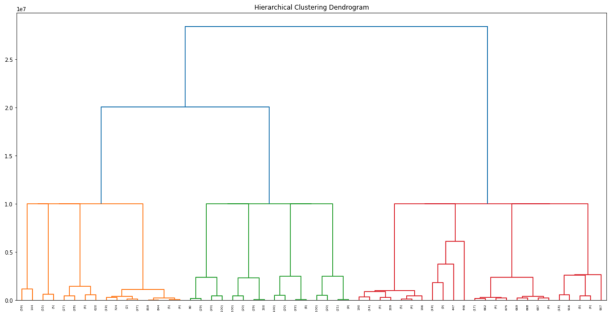

Figure 7. Clustering Aglomerativo Hierárquico

O Radix Sort é um algoritmo de ordenação estável com uma complexidade de O(d * n), onde n é o tamanho da lista e d é o número de dígitos. O Radix Sort é conhecido por ter um bom desempenho em termos de consumo de energia, especialmente quando o número de dígitos é pequeno. No entanto, a sua eficiência pode ser reduzida quando o número de dígitos é grande. O Selection Sort tem uma complexidade de $O(n^2)$, onde n é o tamanho da lista a ser ordenada. Este algoritmo não é eficiente em termos de consumo de energia, pois envolve um grande número de comparações e trocas. Geralmente, não é recomendado para grandes conjuntos de dados.
O Insertion Sort tem uma complexidade de $O(n^2)$, semelhante ao Selection Sort. Ele também envolve um grande número de comparações e trocas, o que não o torna eficiente em termos de consumo de energia.
O Bubble Sort também tem uma complexidade de tempo de $O(n^2)$, e assim como o Selection Sort e o Insertion Sort, envolve muitas comparações e trocas. Portanto, não é eficiente em termos de consumo de energia, especialmente para conjuntos de dados grandes.
Tudo isto pode ser comprovado com os testes e análise visual dos dados efetuadas. Podemos perceber que quanto maior o input maior será o tempo de execução e portanto o seu consumo energético. Podemos também perceber que o número de trocas e comparações efetuadas pelo algoritmo aumenta substancialmente o seu consumo energético mas nem sempre o seu tempo de execução.
Em resumo, com base nas características e complexidades dos algoritmos de ordenação mencionados, o Radix Sort tende a ser mais eficiente em termos de consumo de energia e tempo de execução do que o Selection Sort, Insertion Sort e Bubble Sort. Outras estatísticas interessantes estudas fora o pico de uso de memória dos algoritmos e a carga nos cores. Ambos os parâmetros estão também relacionados com a complexidade dos algoritmos e comportam se de igual modo que as faladas anteriormente. Resumidamente quanto maior a complexidade algoritmica, maior são os custos para a máquina uma vez que tem que fazer um maior esforço para chegar à solução pretendida.
Em relação ao consumo energético, em geral, a linguagem C é conhecida por ter uma vantagem em termos de eficiência energética. Isto deve-se ao fato de que o C é uma linguagem de programação de baixo nível, que oferece maior controlo sobre os recursos do sistema e permite uma otimização mais precisa do código. A programação em C permite a utilização direta de recursos do hardware, resultando num consumo de energia potencialmente mais eficiente em comparação com linguagens de alto nível, como o Python.
Em relação ao tempo de execução, a linguagem C também tem uma vantagem em muitos casos. Por ser uma linguagem compilada, o código em C é convertido em instruções de máquina antes da execução, o que geralmente resulta numa execução mais rápida. Além disso, a sintaxe e as estruturas de dados do C são mais leves e simples em comparação com o Python, o que pode resultar num desempenho superior para tarefas intensivas em termos de processamento.

5 Influência do Powercap nos Resultados

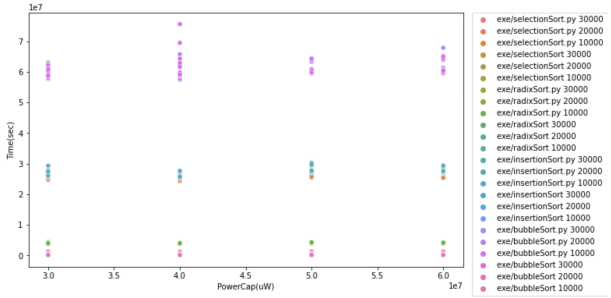

Figure 8. Scatter-plot on Time and Powercap

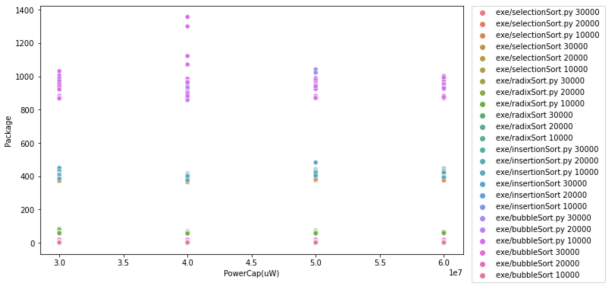

Figure 9. Scatter-plot on Package and Powercap

Considerando os resultados acima apresentados podemos constatar que relativamente a eficiência energética o maior Powercap(60W) foi o que o que ofereceu melhores resultados, embora sejam relativamente similares com o menor Powercap(30W). Por outro lado nos critérios de Tempo de execução e Peak Memory Usage os resultados foram completamente opostos uma vez que o menosr Powercap(30W) oferece os melhores resultados. Os limites energéticos intermédios usados nas medições foram os que obtiveram piores resultados. Com estas elações retiradas da análise visual dos gráficos

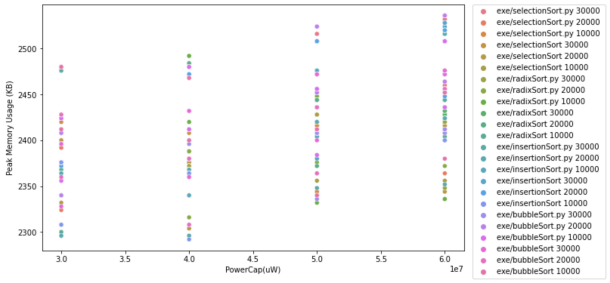

Figure 10. Scatter-plot on Peak Memory Usage and Powercap

podemos concluir que o menor Powercap é o que oferece os melhores resultados no geral. O que faz todo o sentido uma vez que ao limitarmos o consumo energético temos um menor desperdício energético.

Reduzir o limite de energia impõe restrições mais rigorosas aos algoritmos, forçando-os a operar com eficiência máxima para atender a essas restrições.

Em relação ao tempo de execução, algoritmos otimizados para limites de energia mais baixos tendem a ser mais rápidos. Eles são projetados para realizar cálculos de forma mais eficiente, aproveitando ao máximo os recursos disponíveis. Isso permite que o algoritmo complete as tarefas num menor tempo de execução.

Relativamente ao uso de memória, a restrição de energia mais baixa também incentiva uma gestão eficiente dos recursos de memória. Isso resulta numa melhor utilização dos recursos de memória disponíveis.

Além disso, a redução do limite de energia leva a um menor consumo energético no geral. Isso é especialmente benéfico em dispositivos alimentados por bateria ou em ambientes com restrições de energia, onde a eficiência energética é crucial.

Portanto, ao impor um limite de energia menor, é possível obter resultados superiores em termos de tempo de execução, uso de memória e consumo energético. Essa abordagem garante que os algoritmos operem com máxima eficiência, atendendo aos requisitos rigorosos de restrição de energia.

6 Trabalho Relacionado

É relevante mencionar a análise de dados como um componente essencial deste estudo. A análise de dados desempenha um papel fundamental na compreensão e avaliação dos resultados obtidos durante os testes de consumo energético dos algoritmos de ordenação.

Durante o estudo, foram recolhidos dados detalhados sobre o consumo energético de cada algoritmo de ordenação em diferentes cenários. Estes dados foram cuidadosamente registados e organizados para permitir uma análise abrangente e precisa. Através da aplicação de técnicas de análise de dados, foi possível examinar e interpretar as informações recolhidas, obtendo insights valiosos sobre o desempenho energético dos algoritmos e a comparação entre as suas implementações em C e Python.

Assim, a análise de dados desempenhou um papel crucial neste estudo, fornecendo suporte empírico para as conclusões alcançadas. Ao considerar o trabalho relacionado, é importante reconhecer a importância da análise de dados como uma abordagem comum em estudos semelhantes, onde a compreensão aprofundada dos dados recolhidos é essencial para obter conclusões significativas e fundamentadas.

7 Conclusões

Com base nos testes realizados, ficou comprovado que a implementação dos algoritmos de ordenação em C apresentou uma maior eficiência energética em comparação com a implementação em Python. Esses resultados destacam a importância de considerar a escolha da linguagem de programação ao buscar otimizar o consumo energético.

Além disso, a análise do consumo energético dos algoritmos de ordenação revelou-se uma área de grande potencial e relevância para o futuro. Com a crescente conscientização ambiental e a necessidade de soluções mais sustentáveis, o estudo e a otimização do consumo energético em algoritmos e aplicações serão cada vez mais valorizados.

Em suma, os testes confirmaram a superioridade energética da implementação em C em relação ao Python. A análise do consumo energético promete ser uma área em crescimento, com potencial para contribuir significativamente para a eficiência e a sustentabilidade em desenvolvimento de software e sistemas.