Universidade de São Paulo

SSC0951 – Desenvolvimento de Código Otimizado

Atividade 5

Nome	**Nº USP**
João Pedro Rodrigues Freitas	11316552
Rafael Kuhn Takano	11200459
Vinícius Santos Monteiro	11932463

Dezembro

2022

1 Análises de desempenho utilizando otimizações

1.1 Introdução

O objetivo dessa atividade é fazer uma comparação entre o tempo gasto na multiplicação de matrizes utilizando um algoritmo básico e o na multiplicação utilizando a biblioteca numpy.

Ainda, comparar o tempo gasto na ordenação utilizando um mergeSort padrão e utilizando caches no mergeSort.

1.2 Metodologia

Para análise do experimento, utilizamos a biblioteca timeit para calcularmos o tempo gasto na execução das funções utilizadas.

Ainda, utilizamos a biblioteca functools para podermos utilizar cache e a biblioteca numpy para realizar algumas operações.

1.2.1 Multiplicação de matrizes

Definiu-se um tamanho fixo de 200 para as matrizes.

Foram geradas 4 matrizes, sendo duas, listas de listas, e duas, matrizes do numpy.

As matrizes simples (listas de listas) foram multiplicadas utilizando um algoritmo padrão de multiplicação, enquanto as matrizes do numpy foram multiplicadas utilizando a operação dot do numpy.

Os seguintes resultados foram obtidos com 10 testes para cada operação:

Time (s)	name
8.836	defaultMatrixMultiplication
0.056	numpyMatrixMultiplication

Tabela 1: Multiplicação de matrizes.

Com base nos resultados, o numpy mostra-se muito mais eficiente, utilizando apenas 0,63% do tempo comparado à multiplicação padrão.

1.2.2 Merge Sort

Definiu-se um tamanho fixo de 500.000 para os arrays.

Foram gerados 2 arrays com valores aleatórios.

O primeiro array foi ordenado utilizando o algoritmo padrão do mergeSort. O segundo, no entanto, foi ordenado pelo mergeSort utilizando cache na política Least Recently Used (LRU). Para utilizar a lru_cache da biblioteca functools, foi necessário criar um decorador para realizar a conversão de lista para tupla, pois listas não são hashable.

Os seguintes resultados foram obtidos com 10 testes para cada operação:

Time (s)	name
19.645	mergeSort
1.927	cachedMergeSort

Tabela 2: Ordenação mergeSort.

Com base nos resultados, o uso de caches mostra-se extremamente útil, pois a ordenação com caches utilizou apenas 9,8% do tempo comparado ao mergeSort padrão.