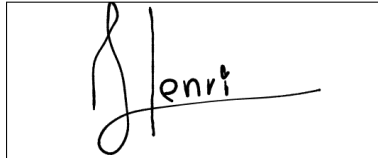


# MAC0219 - PROGRAMAÇÃO CONCORRENTE E PARALELA

## MINIEP5

Nome: João Henri Carrenho Rocha

Número USP: 11796378



*Sua assinatura atesta a autenticidade e originalidade de seu trabalho e que você se compromete a seguir o código de ética da USP em suas atividades acadêmicas, incluindo esta atividade.*

Exercício: MINIEP5

Data: 27/04/2023

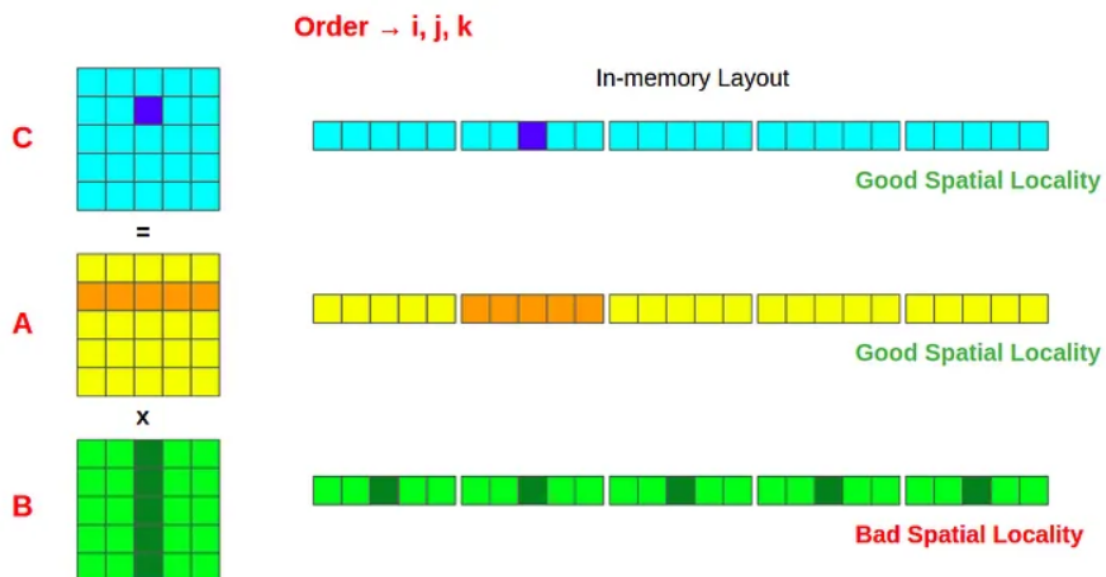
### SOLUÇÃO

Neste exercício, notou-se que é possível usar melhor a localidade do cache para acelerar a multiplicação de matrizes.

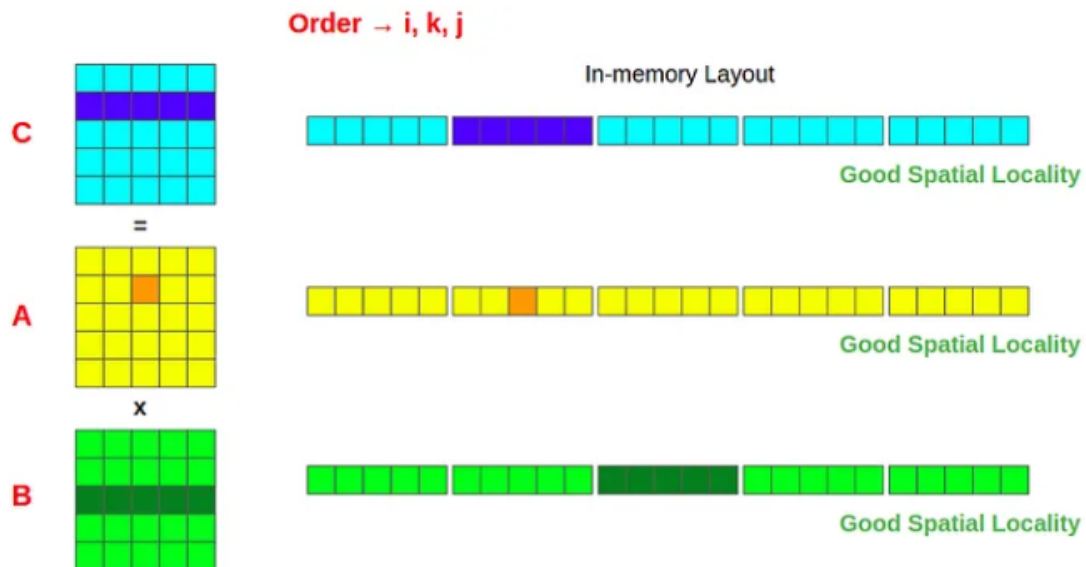
Para tal, foi necessário apenas inverter dois loops da matriz. C é uma linguagem que usa um sistema *row-major*, isto é, elementos consecutivos de uma linha de matriz residem um ao lado do outro na memória.

A memória cache lê uma linha de cache de dados por vez da RAM. Suponha que o tamanho da linha de cache seja de 64 bytes. Embora seja preciso acessar apenas uma variável do tipo **double** (8 bytes), o cache precisa carregar todos os 64 bytes.

Dessa maneira, ao iterar as multiplicações da maneira “linha por coluna”, isto é, os loops na sequência  $i, j, k$ , cada elemento da coluna gera um *cache miss*:



Invertendo os loops  $j$  e  $k$ , ou seja, fazendo  $i, k, j$ , obtém-se o mesmo resultado. Entretanto, dessa vez, aproveita-se a linha inteira da segunda matriz sequencialmente, aumentando o número de *cache hits* e consequentemente acelerando muito o programa:



A única diferença no algoritmo é que agora os elementos de  $C$  não são inteiramente calculados um a um. Na verdade, todos os elementos de uma determinada linha de  $C$  são calculados *juntos*. Foram feitas 20 multiplicações de matrizes com  $N = 2048$  por computador.

Para `matrix_dgemm_0` no meu computador:

51.798678s, 41.508107s, 39.122617s, 38.507272s, 38.568778s, 37.447434s,  
 37.716243s, 38.680080s, 38.929853s, 39.944641s, 38.230115s, 38.561267s,  
 38.485830s, 38.462983s, 38.555072s, 39.081195s, 39.333147s, 39.901625s,  
 38.319260s, 37.697126s

Média: 39.44256s | Intervalo de confiança (99.9%): 39.4426 ±2.185 (±5.54%)

Para `matrix_dgemm_1` no meu computador:

5.744465s, 5.885983s, 5.770540s, 5.538820s, 5.377906s, 5.472397s,  
 5.369920s, 5.392042s, 5.657295s, 5.686970s, 5.607804s, 5.314287s,  
 5.748245s, 5.424805s, 5.779050s, 6.023610s, 5.947006s, 5.373104s,  
 5.639492s, 5.690884s

Média: 5.62223s | Intervalo de confiança (99.9%): 5.6222 ±0.15 (±2.66%)

Para `matrix_dgemm.0` no meu notebook:

42.473415s, 47.014575s, 45.352803s, 39.966130s, 44.058574s, 40.039986s,  
47.417813s, 48.234117s, 39.615496s, 44.468866s, 40.289284s, 46.832806s,  
47.891253s, 46.961832s, 48.371872s, 47.221062s, 47.071594s, 48.900458s,  
45.850156s, 39.488204s

Média: 44.876015s | Intervalo de confiança (99.9%): 44.876 ±2.392 (±5.33%)

Para `matrix_dgemm.1` no meu notebook:

4.650185s, 4.745125s, 4.772345s, 4.670232s, 4.161879s, 4.497672s,  
4.016415s, 4.017777s, 4.071704s, 4.073371s, 4.084507s, 4.859599s,  
4.452582s, 4.445800s, 4.609205s, 4.422100s, 4.753575s, 4.512798s,  
4.528932s, 4.599953s

Média: 4.447288 | Intervalo de confiança (99.9%): 4.4473 ±0.2 (±4.50%)

Specs do computador:

Memory: 16.0 GiB

Processor: Intel® Core™ i7-8700K CPU @ 3.70GHz × 12

OS: Ubuntu 22.04.2 LTS

OS Type: 64-Bit

Specs do notebook:

Memory: 16 GB 2667 MHz DDR4

Processor: 2,3 GHz Intel Core i9 8-Core

OS: macOS 13.0.1 (22A400)

OS Type: 64-Bit

Os gráficos foram retirados de *Gunavaran Brihadiswaran - The impact of cache locality on performance in C through matrix multiplication.*

Disponível em: <https://levelup.gitconnected.com/c-programming-hacks-4-matrix-multiplication-are-we-doing-it-right-21a9f1cbf53>