# Mini EP6 - Múltiplicação de Matrizes e Memória *Cache*

Lucas de Sousa Rosa & Alfredo Goldman

04 de outubro de 2024

## Introdução

A memória *cache* é uma memória de rápido acesso, localizada dentro dos processadores. Sua velocidade de acesso em comparação à RAM chega a ser da ordem de centenas de vezes mais rápida por estar fisicamente muito mais próxima da CPU; e ser implementada usando SRAM que, embora encareça a produção por *byte*, permite a construção de memórias mais rápidas[1]. Existem heurísticas de detecção automática do tamanho do *cache*, conforme discutido em[2], mas também é possível detectar o *cache* usando os recursos do SO (veja o comando lscpu).

## Problema

Considere o seguinte código (multiplicação de matrizes $n \times n$):

```
        for (i = 0; i < n; ++i)
            for (j = 0; j < n; ++j)
            {
                double sum = 0;
                for (k = 0; k < n; ++k)
                    sum += A(i, k)*B(k, j);
                C(i, j) = sum;
            }
```

presente em `matrix_dgemm_0`, dentro do arquivo `matrix.c`. Seu objetivo é utilizar os conhecimentos recém adquiridos a respeito da *cache* para otimizar o código acima.

### Primeira Otimização

Você deverá implementar na função `matrix_dgemm_1` uma versão mais rápida (e ainda correta) do código acima, apenas usando as noções de localidade de acesso à memória *cache* vistas em aula.

**Dica**: Lembre-se que a ordem que se itera sobre uma matriz (coluna depois linha ou linha depois coluna) pode acabar invalidando *cache*. Assim, procure uma forma de garantir que a iteração sobre `A` e `B`, no código acima, possa melhor aproveitar o *cache*.

### Usando Blocagem

É possível melhorar ainda mais o acesso ao cache usando uma técnica chamada blocagem. Aqui as matrizes são particionadas em matrizes menores, e a multiplicação é feita em etapas. Por exemplo, podemos particionar as matrizes $A = [\, A_{11} \; A_{12} \,]$ e $B = [\, B_{11} \; B_{21} \,]^\top$, e então calcular a matriz $C$ como:

$$C = AB = \begin{bmatrix} A_{11} & A_{12} \end{bmatrix} \begin{bmatrix} B_{11} \\ B_{21} \end{bmatrix} = A_{11}B_{11} + A_{12}B_{21} \tag{1}$$

[1] `https://people.freebsd.org/~lstewart/articles/cpumemory.pdf`
[2] `https://stackoverflow.com/questions/2576762/measure-size-and-way-order-of-l1-and-l2-caches`

como ilustrado na Figura 1. Por fim, é possível realizar o particionamento de diversas maneiras[3], mas aqui você deve buscar um particionamento que otimize o acesso à memória *cache*.

Sua tarefa é implementar em `matrix_dgemm_2` uma versão ainda mais otimizada de sua `matrix_dgemm_1`, agora usando blocagem.

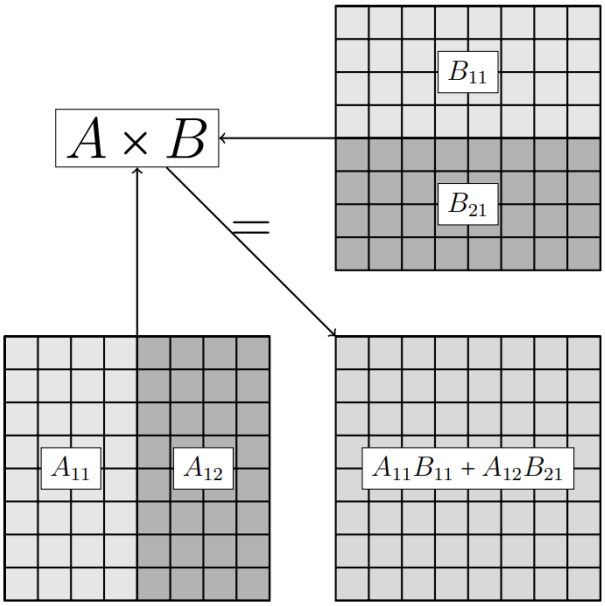


Figura 1: Ilustração da técnica de blocagem.

## Entrega

Você deverá entregar um relatório sucinto em PDF, respondendo às seguintes perguntas:

1. Mostre, com embasamento estatístico, a variação do tempo entre as funções `matrix_dgemm_0`, `matrix_dgemm_1` e `matrix_dgemm_2`. Explique o porquê.
2. Como você usou a blocagem para melhorar a velocidade da multiplicação de matrizes?

Evitem `.doc`, `.odt`, `.docx`, pois dificultam a correção. Além disso, você deverá entregar o arquivo `matrix.c`, contendo as funções `matrix_dgemm_1` e `matrix_dgemm_2` implementadas.

## Código Fornececido

Além deste enunciado, você deve baixar o arquivo compactado `miniep6.zip` disponível no e-Disciplinas. Você deverá modificar apenas as funções `matrix_dgemm_1` e `matrix_dgemm_2` com a sua implementação, conforme descrito acima. O programa fornecido contém testes, na qual a sua implementação deverá passar. Para executar os testes, basta executar o comando `make test` no terminal na pasta contendo os arquivos o `Makefile`.

Para facilitar a coleta de amostras estatísticas, também é fornecido um binário `main` que recebe como entrada os seguintes argumentos:

```
main <ARGS>
onde <ARGS> pode ser a combinação de
--matrix-size <NUM> Tamanho da matriz quadrada. (n)
--algorithm <NUM>   Algoritmo a ser utilizado. (0 = dgemm_0,
                                                1 = dgemm_1,
                                                2 = dgemm_2)
```

[3] `https://en.wikipedia.org/wiki/Block_matrix`

Para compilá-lo, basta rodar `make` no terminal na pasta contendo os arquivos o `Makefile`.