

**PUC-Rio – Departamento de Informática**  
**Ciência da Computação**  
**Introdução à Arquitetura de Computadores**  
**Prof.: Anderson Oliveira da Silva**



## **Trabalho 2 – 2022.2**

### **Parte I:**

Aprimorar o módulo escrito em linguagem C, chamado *matrix\_lib.c*, implementado no trabalho 1, com a utilização de **instruções vetoriais (AVX/FMA)** usando a biblioteca Intel Intrinsics. As duas funções de operações aritméticas com matrizes estão descritas abaixo.

- a. Função *int scalar\_matrix\_mult(float scalar\_value, struct matrix \*matrix)*

Essa função recebe um valor escalar e uma matriz como argumentos de entrada e calcula o produto do valor escalar pela matriz utilizando instruções vetoriais (AVX). O resultado da operação deve ser retornado na matriz de entrada. Em caso de sucesso, a função deve retornar o valor 1. Em caso de erro, a função deve retornar 0.

- b. Função *int matrix\_matrix\_mult(struct matrix \*matrixA, struct matrix \*matrixB, struct matrix \*matrixC)*

Essa função recebe 3 matrizes como argumentos de entrada e calcula o valor do produto da matriz A pela matriz B utilizando instruções vetoriais (AVX/FMA). O **algoritmo otimizado de produto de matrizes** apresentado em aula deve ser utilizado nesta função. O resultado da operação deve ser retornado na matriz C. Em caso de sucesso, a função deve retornar o valor 1. Em caso de erro, a função deve retornar 0.

O tipo estruturado matrix é definido da seguinte forma:

```
struct matrix {  
    unsigned long int height;  
    unsigned long int width;  
    float *rows;  
};
```

Onde:

height = número de linhas da matriz (múltiplo de 8)  
width = número de colunas da matriz (múltiplo de 8)  
rows = sequência de linhas da matriz (height\*width elementos)

### **Parte II:**

Crie um programa em linguagem C, chamado *matrix\_lib\_test.c*, que implemente um código para testar a biblioteca *matrix\_lib.c*. Esse programa deve receber um valor escalar float, a dimensão da primeira matriz (A), a dimensão da segunda matriz (B) e o nome de quatro **arquivos binários** de floats na linha de comando de execução. O programa deve inicializar as duas matrizes (A e B) respectivamente a partir dos dois primeiros **arquivos binários** de floats e uma terceira matriz (C) com zeros. **Ambas as inicializações devem utilizar instruções vetoriais (AVX)**. A função *scalar\_matrix\_mult* deve ser chamada com os seguintes argumentos: o valor escalar fornecido e a primeira matriz (A). O resultado (retornado na matriz A) deve ser armazenado em um **arquivo binário** usando o nome do terceiro arquivo de floats. Depois, a função *matrix\_matrix\_mult* deve

ser chamada com os seguintes argumentos: a matriz A resultante da função *scalar\_matrix\_mult*, a segunda matriz (B) e a terceira matriz (C). O resultado (retornado na matriz C) deve ser armazenado com o nome do quarto **arquivo binário** de floats.

Exemplo de linha de comando:

```
matrix_lib_test 5.0 8 16 16 8 floats_256_2.0f.dat floats_256_5.0f.dat result1.dat result2.dat
```

Onde,

5.0 é o valor escalar que multiplicará a primeira matriz;

8 é o número de linhas da primeira matriz;

16 é o número de colunas da primeira matriz;

16 é o número de linhas da segunda matriz;

8 é o número de colunas da segunda matriz;

*floats\_256\_2.0f.dat* é o nome do arquivo de floats que será usado para carregar a primeira matriz;

*floats\_256\_5.0f.dat* é o nome do arquivo de floats que será usado para carregar a segunda matriz;

*result1.dat* é o nome do arquivo de floats onde o primeiro resultado será armazenado;

*result2.dat* é o nome do arquivo de floats onde o segundo resultado será armazenado.

O programa principal deve cronometrar o tempo de execução geral do programa (overall time) e o tempo de execução das funções *scalar\_matrix\_mult* e *matrix\_matrix\_mult*. Para marcar o início e o final do tempo em cada uma das situações, deve-se usar a função padrão *gettimeofday* disponível em *<sys/time.h>*. Essa função trabalha com a estrutura de dados *struct timeval* definida em *<sys/time.h>*. Para calcular a diferença de tempo (delta) entre duas marcas de tempo *t0* e *t1*, deve-se usar a função *timedifference\_msec*, implementada no módulo *timer.c*, fornecido no roteiro do trabalho 1.

### Observação 1:

O programa deve ser desenvolvido em linguagem C e com a biblioteca Intel Intrinsics. A compilação do programa fonte deve ser realizada com o compilador GCC, usando os seguintes argumentos:

```
gcc -std=c11 -mfma -o matrix_lib_test matrix_lib_test.c matrix_lib.c timer.c
```

Onde,

*matrix\_lib\_test* = nome do programa executável.

*matrix\_lib\_test.c* = nome do programa fonte que tem a função *main()*.

*matrix\_lib.c* = nome do programa fonte do módulo de funções de matrizes.

*timer.c* = nome do programa fonte do módulo do cronômetro.

O servidor do DI está disponível para acesso remoto, conforme informado anteriormente, e pode ser usado para executar o programa de teste.

### Observação 2:

O programa deve ser executado com matrizes de dimensões 1024 x 1024 (4MB por matriz) e 2048 x 2048 (16MB por matriz) e imprimir na tela até no máximo 256 elementos da matriz A, B e C, além dos tempos parciais de execução das duas funções da biblioteca e o tempo total da execução do programa (overall time). Deve-se imprimir também o modelo do processador usado no teste (output do comando *lscpu*). Esses dados devem ser copiados da tela e armazenados no arquivo de relatório de execução do programa chamado *relatorio.txt*.

**Observação 3:**

Apenas os programas fontes *matrix\_lib.c*, *matrix\_lib.h*, *matrix\_lib\_test.c* e o relatório de execução do programa (*relatorio.txt*) devem ser carregados no site de EAD da disciplina até o prazo de entrega. **Não devem ser carregados arquivos compactados (ex: .zip, .rar, .gz, .tgz, etc).**

**Importante: Cada integrante do grupo deve fazer o carregamento dos arquivos no EAD.**

**Prazo de entrega: 15/9/2022 – 12:00h.**

**Prazo limite para entrega: 15/9/2022 – 23:59h.**