PUC-Rio – Departamento de Informática
Ciência da Computação
Introdução à Arquitetura de Computadores
Prof.: Alexandre Meslin

Trabalho 4 – 2022.1

Parte I:
Criar o módulo escrito em linguagem C, chamado *matrix_lib.cu*, usando a biblioteca CUDA NVIDIA e processamento paralelo na GPGPU NVIDIA Tesla C2075, para implementar 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 CUDA. *Cada função kernel deve calcular o resultado do produto entre o valor escalar e um dos elementos da matriz (ou mais de um elemento se o dataset for maior que o número de threads do GRID).* 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. Fução *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 CUDA. *Cada função kernel deve calcular o resultado referente a um dos elementos da matriz C (ou mais de um elemento se o dataset for maior que o número de threads do GRID).* 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.

c. Função *int set_grid_size(int threads_per_block, int max_blocks_per_grid)*

   Essa função recebe o *número de threads por bloco* e o *número máximo de blocos por grid* que devem ser usados como parâmetros para disparar os threads (funções kernel) em paralelo durante o processamento das operações aritméticas com as matrizes e deve ser chamada pelo programa principal antes das outras funções. Caso não seja chamada, o valor default do número de threads por bloco do módulo é 256 e do número de blocos por grid é 4096. Os valores limites para a GPGPU NVIDIA Tesla C2075 são 1024 para o número de threads por bloco e 65535 para o número de blocos por grid. Se algum dos valores passados como argumento para a função extrapolar um dos valores máximos, os valores default deverão ser usados e a função deve retornar 0 para indicar erro. Caso contrário, os valores passados devem ser usados e a função deve retornar 1 para indicar que os valores foram aceitos com sucesso.

O tipo estruturado matrix é definido da seguinte forma:

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

Onde:

height = número de linhas da matriz (múltiplo de 8)
width = número de colunas da matriz (múltiplo de 8)
h_rows = sequência de linhas da matriz (height*width elementos alocados no host)
d_rows = sequência de linhas da matriz (height*width elementos alocados no device)

As alocações de memória no *host* e no *device* devem ser realizadas no programa principal, antes das chamadas das funções *scalar_matrix_mult* e *matrix_matrix_mult*.

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), **o número de threads por bloco a serem disparadas**, **o número máximo de blocos por GRID a serem usados**, 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. A função *set_grid_size* deve ser chamada com os respectivos valores dos argumentos passados na linha de comando. 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 de floats.

Exemplo de linha de comando:

matrix_lib_test 5.0 8 16 16 8 256 4096 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;
256 é o número de threads por bloco a serem disparadas;
4096 é o número máximo de blocos por GRID a serem usados;
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 CUDA da NVIDIA. A compilação do programa fonte deve ser realizada com o compilador NVCC, usando os seguintes argumentos:

nvcc -o matrix_lib_test matrix_lib_test.cu matrix_lib.cu timer.c

Onde,
matrix_lib_test = nome do programa executável.
matrix_lib_test.cu = nome do programa fonte que tem a função main().
matrix_lib.cu = 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:

As matrizes A, B e C devem ser alocadas simultaneamente e por completo na memória da GPGPU NVIDIA Tesla C2075 que tem 5GB de memória disponível. Se não for viável fazer a alocação, o programa principal deve emitir uma notificação de erro de alocação de memória para o usuário.

Observação 3:

O programa deve inicialmente ser testado com matrizes pequenas para facilitar a depuração, mas, a versão final deve ser testada com matrizes grandes, com dimensão 1024 x 1024.

Observação 4:

Apenas os programas fontes *matrix_lib.cu*, *matrix_lib.h* e *matrix_lib_test.cu* e a saída do programa com os tempos de execução das funções matriciais e tempo total devem ser carregados no site de EAD da disciplina até o prazo de entrega. **Apenas *UM* integrante do grupo deve fazer a carga.**

Prazo de entrega: 29/05/2022 – 23:59 h.