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

Trabalho 4 – 2022.2

Parte I:
O objetivo deste trabalho é aprimorar o módulo escrito em linguagem C, chamado matrix_lib.c, implementado no trabalho 3, com a utilização de paralelismo usando a biblioteca CUDA NVIDIA e a GPGPU NVIDIA Tesla C2075. Ambas as operações aritméticas com matrizes devem ser implementadas com processamento paralelo: produto de um escalar por uma matriz (scalar_matrix_mult) e produto de duas matrizes (matrix_matrix_mult). Também deve ser implementado um programa de teste deste módulo fazendo uso do módulo de cronômetro fornecido para medir o tempo global de execução do programa de teste, assim como os tempos parciais de execução de cada uma das funções implementadas.

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;
        int alloc_mode;
};
```

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)
alloc_mode = FULL_ALLOC (valor inteiro 1) ou PARTIAL_ALLOC (valor inteiro 0)

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.cu*, que implemente um código para testar a biblioteca *matrix_lib.cu*. 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**, **a quantidade máxima de memória que pode ser alocada na GPGPU**, 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 1024*** *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;
1024 é a quantidade máxima de memória em MiB que pode ser alocado na GPGPU;
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 demonstrado em sala de aula, e pode ser usado para executar o programa de teste.

Observação 2:

A alocação de memória na CPU e na GPGPU deve ser realizada no código matrix_lib_test.cu. O programa deve tentar alocar as matrizes A, B e C simultaneamente e por completo na memória da CPU e, quando possível, também na memória da GPGPU NVIDIA Tesla C2075, limitando-se ao valor máximo para alocação de memória na GPGPU informado na linha de argumentos.

Se for viável fazer a alocação completa da três matrizes na CPU e na GPGPU, o programa deve atribuir o valor FULL_ALLOCATION no campo alloc_mode da três matrizes.

Se não for viável fazer a alocação simultânea e completa das matrizes A, B e C na CPU, o programa principal deve emitir uma notificação de erro de alocação de memória na CPU e encerrar sua execução.

Se não for viável fazer a alocação simultânea e completa das matrizes A, B e C na GPGPU, o programa deve tentar alocar simultaneamente a matriz B por completo e o equivalente a uma das linhas da matriz A e uma das linhas da matriz C na GPGPU. Se tiver sucesso nessa alocação, o programa deve atribuir o valor FULL_ALLOCATION no campo alloc_mode da matriz B e o valor PARTIAL_ALLOC no campo alloc_mode das matrizes A e C.

Se não for viável fazer a alocação completa da matriz B e a alocação parcial das matrizes A e C simultaneamente na GPGPU, o programa principal deve emitir uma notificação de erro de alocação de memória na GPGPU e encerrar sua execução.

Observação 3:

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 4:

Apenas os programas fontes *matrix_lib.cu*, *matrix_lib.h*, *matrix_lib_test.cu* 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: 10/11/2022 – 12:00h.

Prazo limite para entrega: 10/11/2022 – 23:59h.