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

Trabalho 2 – 2024-1

Instruções Gerais

Leia com atenção o enunciado do trabalho e as instruções para a entrega. Em caso de dúvidas, não invente. Pergunte!

- O trabalho deve ser entregue até a data marcada no EaD.
- Trabalhos entregues com atraso perderão um ponto por dia (ou fração) de atraso.
- Trabalhos que não compilem (isto é, que não produzam um executável) não serão considerados! Ou seja, receberão grau zero.
- Os trabalhos podem ser feitos em grupos de no máximo dois alunos.
- Alguns grupos poderão ser chamados para apresentações orais / demonstrações dos trabalhos entregues.
- Importante: Somente ***UM*** integrante do grupo deve fazer a carga dos arquivos fontes no EAD.

Objetivo

O objetivo deste trabalho é implementar a operação do produto de matrizes a partir da implementação da versão otimizada do produto de matrizes. Nenhuma mudança deve ser realizada no módulo `matrix_lib_test.c` do programa base já implementado e testado. Apenas a função `matrix_matrix_mult` do módulo `matrix_lib.c` deve sofrer alteração.

Obs.: a função de teste será entregue junto com esse texto.

O programa de teste deve obter o tempo logo após o início da função `main` e o tempo logo antes do retorno final da função `main` para fazer o cálculo do tempo de execução total do programa (impresso antes do seu encerramento). E, também deve obter o tempo antes da chamada de uma função da biblioteca e depois do retorno da função para calcular os tempos parciais da execução de cada função. Todas as tomadas de tempo e impressão das medidas de tempo devem ser realizadas no programa de teste.

Parte I:

Aprimorar o módulo escrito utilizando a plataforma CUDA da Nvidia, chamado `matrix_lib.c`, implementado em sala de aula, com a utilização de processamento. As duas funções de operações aritméticas com matrizes estão descritas a seguir.

Crie um módulo escrito em linguagem C, chamado `matrix_lib.c`, que implemente duas funções para fazer operações aritméticas com matrizes, conforme descrito 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. O resultado da operação deve ser armazenado na matriz de entrada. Em caso de sucesso, a função deve retornar o valor 1. Em caso de erro, a função deve retornar 0. Esta função irá executar no host e chamar uma função no device.

b. Função `int matrix_matrix_mult(struct matrix *matrixA, struct matrix * matrixB, struct matrix * matrixC)`

Essa função recebe três matrizes como argumentos de entrada e calcula o valor do produto da matriz A pela matriz B *utilizando o algoritmo otimizado apresentado em aula*. O resultado da operação deve ser armazenado na matriz C. Em caso de sucesso, a função deve retornar o valor 1. Em caso de erro, a função deve retornar 0. Esta função irá executar no host e chamar uma função no device.

c. 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)

d. Ambas funções devem chamar funções no *device* para realizar a computação necessária. O módulo principal irá criar duas variáveis globais `threadsPerBlock` e `blocksPerGrid`, representando respectivamente o número de *threads* por bloco e o número de blocos para ser utilizado na chamada das funções que irão executar no *device*.

Obs.: em lugar algum desse texto está escrito que o número de threads é múltiplo ou divisível pelo número de linhas, colunas ou qualquer outra dimensão das matrizes.

Parte II:

Obs.: essa parte foi entregue pronta. Aqui temos a sua descrição.

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.

A função `scalar_matrix_mult` deve ser chamada com os seguintes argumentos:

- o valor escalar fornecido e
- a primeira matriz (A).

O resultado (armazenado 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 floats_256_2.0f.dat floats_256_5.0f.dat
result1.dat result2.dat 256 4096
```

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;
- 256 é o número de *threads* por bloco;
- 4096 é o número de blocos.

A função 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 `clock` disponível em `<time.h>`. Essa função retorna o tempo aproximado de processador usado pelo programa. Para calcular a diferença de tempo (delta) entre duas marcas de tempo `t0` e `t1`, deve-se usar a macro `timedifference_msec`, definida no módulo `timer.h`, fornecido.

Observação 1:

Todas as práticas da disciplina serão executadas no ambiente Linux. Como o objetivo é usar esse trabalho como base para os próximos trabalhos desta disciplina, este trabalho deve ser compilado e executado no ambiente Linux. Para fazer a compilação do programa no Linux, deve-se usar o NVCC, com os seguintes argumentos:

```
$ nvcc -o matrix_lib_test matrix_lib_test.c matrix_lib.cu
```

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.

Uma máquina virtual padrão VMware com o sistema Linux está disponível na área de download do site da Equipe de Suporte do DI, em:

URL: `http://suporte.inf.puc-rio.br/download/vms/VMCCPP-FC23-64-DI-PUC-Rio-V1.0.zip`

Para executar a máquina virtual, basta baixar o VMware Workstation Player gratuitamente a partir do site da VMware.

URL: `https://www.vmware.com/br/products/workstation-player/workstation-player-evaluation.html`

Observação 2:

Apenas o programa fonte `matrix_lib.c` deve ser carregado no site de EAD da disciplina até o prazo de entrega. Não é necessário incluir os outros arquivos com código fonte. Se você for utilizar algumas funções auxiliares (você vai precisar usar), essas funções e seus protótipos devem ser incluídos no arquivo `matrix_lib.c`.

No campo destinado a observações, informe os nomes dos componentes do grupo