Exercício Programa 4
MAC0431 - Introdução à Computação Paralela e Distribuida

Fernando Omar Aluani 6797226
Jefferson Serafim Ascaneo 6431284

1 Detalhes da Implementação

Nós decidimos que nosso programa de *matrix shift* executa de uma vez só um *row_shift(R)* e um *column_shift(C)*. Dessa forma, o código é simplificado e o programa em si pode ser usado como uma ferramenta, executando-o várias vezes em cadeia para atingir uma sequência de *shifts* desejada.

1.1 O Algoritmo

O algoritmo em si é bem simples: ele lê uma matriz passada ao programa, executa de uma vez só uma operação de *row_shift(R)* e um *column_shift(C)* juntamente (já que a ordem entre elas não importa), e salva a matriz resultante num outro arquivo, com o mesmo nome do arquivo original mais o sufixo "*_result*".

Para tal nós seguimos o exemplo do **VectorAdd**, fazendo algumas modificações: criamos uma classe para representar uma matriz $M \times N$ e nos ajudar com as operações que executamos com as matrizes (do lado da CPU); mudamos os parâmetros passados/recebidos do kernel da GPU; e mais importante, mudamos a quantidade de *work items* nos quais seriam executados o kernel para que ele fosse executado $M \times N$ vezes, numa grade, de tal forma que o kernel fosse executado para cada elemento da matriz.

Quanto ao kernel em si, também alteramos ele para receber os parâmetros que especificamos e aplicar um *row_shift(R)* e um *column_shift(C)* de uma vez só. Juntando com o fato que o kernel é executado para cada elemento da matriz, o código nele fica bem simples:

```
unsigned int i = get_global_id(0);
unsigned int j = get_global_id(1);

unsigned int a = (i - (R)) % M; //row shift
unsigned int b = (j + (C)) % N; //column shift

matriz_resultante[a*N + b] = matriz_original[i*N + j];
```

E desse jeito também os *shifts* funcionam como mostrados no enunciado:

- *row_shift(R)* tem um sentido "para cima", de tal forma que com $R > 0$ uma linha irá "subir"na matriz;
- *column_shift(C)* tem um sentido "para direita", de tal forma que com $C > 0$ uma coluna irá se deslocar para a direita na matriz;

Todo o código do programa (a pasta com os arquivos, como mostrado no enunciado) está incluso neste arquivo, na pasta **MatrixShift**.

1.2 Entrada

O programa recebe três argumentos pela linha de comando ao ser executado, no formato:

./MatrixShift <input-matrix> <row-shift-amount> <column-shift-amount>

Onde:

- *input-matrix*: é o caminho para um arquivo de texto contendo a matriz de entrada. Cada linha do arquivo é uma linha da matriz, e em cada linha os números (inteiros ou reais) devem estar separados por espaços (espaços separam colunas);
- *row-shift-amount*: número de *shifts* de linhas a fazer. É o R mostrado anteriormente (*row_shift(R)*);
- *column-shift-amount*: número de *shifts* de colunas a fazer. É o C mostrado anteriormente (*column_shift(C)*);

Note que o programa supõe que o arquivo da matriz de entrada é bem formado. Isto é, que ele segue essa especificação e não tem valores não-numéricos, nem linhas de tamanhos (quantidade de valores) diferentes, etc. Passar um arquivo de entrada mal-formado irá fazer o programa *crashar* com algum erro ou retornar resultados estranhos.

1.3 Saída

O programa tem duas saídas. Durante a execução ele imprime alguns valores na saída padrão, mostrando o andamento da execução do mesmo, e a matriz de entrada e a resultante depois das operações.

Fora isso, ele também salva a matriz resultante em um arquivo com o mesmo nome do arquivo original de entrada mais o sufixo "*_result*".

2 Conclusão

2.1 Pontos Positivos

- Os exemplos, principalmente o **VectorAdd** ajudaram bastante a entender como executar o código na GPU. Só precisamos pesquisar coisas sobre OpenCL na internet para entender direito a assinatura de algumas funções do OpenCL;
- A **AMD APP SDK** também ajuda bastante a setar o ambiente de desenvolvimento. Sua instalação é simples e não tivemos problemas em compilar e rodar o programa até numa VM com Linux rodando sobre um computador com CPU Intel e GPU NVidia;
- Nossa decisão de como aplicar os *shifts* deixou o algoritmo bem simples;
- Mensagens de erro quando o programa tenta compilar o kernel mas não consegue ajudaram a achar os problemas;

2.2 Pontos Negativos

Complicação em compilar o programa dentro da hierarquia de pastas da AMD APP SDK, e rodar ele em outra pasta ainda nessa hierarquia. Felizmente com um pequeno script bash para copiar os arquivos e rodar os comandos necessários de qualquer outra pasta no sistema resolveu esse problema.

Tipos dos parâmetros a serem passados para o kernel. Foi o maior problema que tivemos ao fazer o programa. Achavámos que podiamos passar os argumentos de um jeito, mas ou não era possível, como passar *int* sendo que tinha que ser *int**; ou deixaria o código mais complexo, como passar a matriz no formato *float***. Depois passamos a matriz como um grande vetor (*float**) e tratamos ele como uma matriz no código do kernel por simplicidade.