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

Trabalho 5 – 2021.2

Parte I:
Criar os módulos escritos em linguagem C, chamados *matrix_lib_vh.c* e *matrix_lib_ve.c*, com processamento vetorial e paralelo em uma única Vector Engine, usando a biblioteca AVEO da NEC. As duas funções de operações aritméticas com matrizes e outras auxiliares 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 processamento vetorial e paralelo da AVEO. *Cada thread executada com processamento vetorial em uma Vector Engine (VE) deve calcular o resultado do produto entre o valor escalar e parte dos elementos da matriz em função da quantidade de threads paralelas.* 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 processamento vetorial e paralelo da AVEO. *Cada função thread executada com processamento vetorial em uma Vector Engine (VE) deve calcular o resultado de N linhas da matriz C, onde N é o número de linhas da matriz C dividido pelo número de threads disparadas em paralelo.* 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 *void set_ve_execution_node(int num_node)*

   Essa função recebe o número de identificação da Vector Engine (VE) na qual devem ser disparadas as threads em paralelo durante o processamento das operações aritméticas com as matrizes e deve ser chamada pelo programa principal antes das outras funções. O número máximo de Vector Engine (VE) disponíveis é 4 e os valores de num_node variam de 0 a 3. Se a função for chamada com um valor de num_node inválido, o valor configurado deverá ser 0 (zero). Caso não seja chamada, o valor default 0 (zero) deve ser configurado.

d. Função *void set_number_threads(int num_threads)*

   Essa função recebe o número de threads que devem ser disparadas em paralelo durante o processamento das operações aritméticas com as matrizes e deve ser chamada pelo programa principal antes das outras funções. O número máximo de threads da Vector Engine (VE) da NEC é 8. Se a função for chamada com um valor superior a 8, o valor máximo do número de threads deve ser usado pelo módulo. Caso não seja chamada, o valor default do número de threads configurado no módulo deve ser 1.

e. Função *int init_proc_ve_node(void)*

   Essa função cria um processo na VE identificada por ve_execution_node e carrega a biblioteca dinâmica matrix_lib_ve.so no processo criado. Retorna 1 em caso de sucesso e 0 (zero) em caso de erro na criação do processo ou erro no carregamento da biblioteca dinâmica.

f. Função *int close_proc_ve_node(void)*

   Essa função descarrega a biblioteca dinâmica matrix_lib_ve.so do processo criado na VE de execução e destrói o processo criado na VE de execução. Retorna 1 em caso de sucesso e 0 (zero) em caso de erro no descarregamento da biblioteca dinâmica ou erro na destruição do processo.

g. Função *int load_ve_matrix(struct matrix *matrix)*

   Essa função carrega a matriz fornecida no processo criado na VE de execução. Para isso, a função aloca a memória necessária na VE e armazena o ponteiro no campo *ve_rows* da *struct matrix*. Depois, sincroniza os elementos de *vh_rows* com *ve_rows*. Retorna 1 em caso de sucesso e 0 (zero) em caso de erro alocação da memória na VE ou erro na sincronização (cópia) dos elementos de *vh_rows* com *ve_rows*.

h. Função *int unload_ve_matrix(struct matrix *matrix)*

   Essa função descarrega a matriz fornecida do processo criado na VE de execução. Para isso, a função sincroniza (copia) os elementos de *ve_rows* com *vh_rows* e, depois, libera a memória alocada na VE e armazena o valor NULL no campo *ve_rows* da *struct matrix*. Retorna 1 em caso de sucesso e 0 (zero) em caso de erro na sincronização (cópia) dos elementos de *ve_rows* com *vh_rows* ou erro na liberação da memória alocada na VE.

i. Função *int sync_vh_ve_matrix(struct matrix *matrix)*

   Essa função sincroniza a memória do VH com a memória da VE fazendo a cópia dos elementos de *vh_rows* para *ve_rows*, previamente alocados com sucesso. Retorna 1 em caso de sucesso e 0 (zero) se *vh_rows* ou *ve_rows* forem NULL ou se ocorrer erro na cópia dos elementos de *vh_rows* para *ve_rows*.

j. Função *int sync_ve_vh_matrix(struct matrix *matrix)*

   Essa função sincroniza a memória da VE com a memória do VH fazendo a cópia dos elementos de *ve_rows* para *vh_rows*, previamente alocados com sucesso. Retorna 1 em caso de sucesso e 0 (zero) se *vh_rows* ou *ve_rows* forem NULL ou se ocorrer erro na cópia dos elementos de *ve_rows* para *vh_rows*.

O tipo estruturado matrix é definido da seguinte forma:

```
struct matrix {
        unsigned long int height;
        unsigned long int width;
        float *vh_rows;
        void *ve_rows;
};
```

Onde:

height = número de linhas da matriz (múltiplo de 8)
width = número de colunas da matriz (múltiplo de 8)
vh_rows = sequência de linhas da matriz (height*width elementos alocados no vector host)
ve_rows = sequência de linhas da matriz (height*width elementos alocados na vector engine)

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 identificador da VE de execução, o número de threads a serem disparadas na VE de execução**, 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_ve_execution_node* deve ser chamada com o valor do identificador da VE de execução fornecido na linha de comando. A função *set_number_threads* deve ser chamada com o valor do número de threads a serem disparadas fornecido 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 1 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;
1 é o número de identificação da VE de execução;
8 é o número de threads a serem disparadas na VE de execução;
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 AVEO da NEC. A compilação do programa fonte da VE deve ser realizada com o compilador NCC, usando os seguintes argumentos:

/opt/nec/ve/bin/ncc -fpic -shared -lveohmem -o matrix_lib_ve.so matrix_lib_ve.c -fopenmp

Onde,
matrix_lib_ve.so = nome do módulo de funções de matrizes (carregado na VE de execução).
matrix_lib_ve.c = nome do programa fonte do módulo de funções de matrizes (do VE).

E, a compilação do programa fonte do VH deve ser realizada com o compilador GCC, usando os seguintes argumentos:

gcc -o matrix_lib_test matrix_lib_test.c matrix_lib_vh.c timer.c -I/opt/nec/ve/veos/include -L/opt/nec/ve/veos/lib64 -Wl,-rpath=/opt/nec/ve/veos/lib64 -lveo

Onde,
matrix_lib_test = nome do programa executável do vector host (VH).
matrix_lib_test.c = nome do programa fonte que tem a função main().
matrix_lib_vh.c = nome do programa fonte do módulo de funções de matrizes (do VH).
timer.c = nome do programa fonte do módulo do cronômetro.

O servidor do DI está disponível para acesso remoto, conforme informado anteriormente, e deve ser usado para acessar o Jump Server da NEC. A partir do Jump Server da NEC, pode-se acessar o Vector Host (VH) para compilar e executar o programa de teste.

Como o VH está sendo utilizado em conjunto com alunos de outra disciplina do DI, os alunos de cada disciplina terão *um horário diário para uso exclusivo do VH* e *outro para uso compartilhado do VH*. O horário de uso exclusivo dos alunos de INF1029 é de **8h às 13h** e o horário de uso compartilhado é de **18h às 8h**. O horário de **13h às 18h** é de uso exclusivo dos alunos da outra disciplina.

Observação 2:

As matrizes A, B e C devem ser alocadas simultaneamente e por completo na memória da VE de execução, que tem 48GB 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_ve.c*, *matrix_lib_vh.c*, *matrix_lib.h* e *matrix_lib_test.c* 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. **Cada integrante do grupo deve fazer a carga.**

Prazo de entrega: 30/11/2021 – 12:00h. **Prazo limite para entrega: 30/11/2021 – 23:59h.**