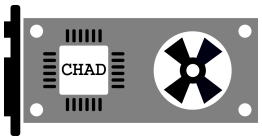


Computação Heterogénea de Alto Desempenho

Mestrado em Engenharia Eletrotécnica e de Computadores

LAB 4

– CUDA –

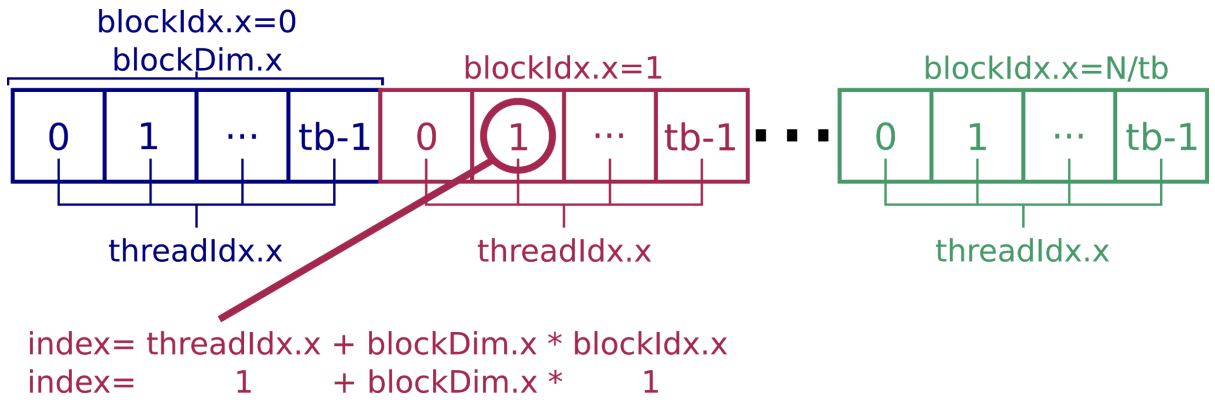
1. Analise o código seguinte que representa uma operação de multiplicação em que o resultado é guardado num vector de inteiros (este exemplo está disponível como material de apoio). Neste programa identifica-se:

1.1. Função que executa na GPU (*CUDA kernel*): Funções que executam na GPU são especificadas com a cláusula " **__global__** ".

```
-----------------------------------------------------------------
// Device code
__global__ void device_func_name(int * device_buffer, int N)
{
      // Thread identifier (1 dimensional problem)
      int index= threadIdx.x + blockIdx.x * blockDim.x;
      // CODE
      if(index<N)
            device_buffer[index]=index*2;
}
-----------------------------------------------------------------
```

1.1.1 A função que executa na GPU deve ter um índice associado que identifica a thread no contexto do programa. A seguir é demonstrado um exemplo elucidativo de como em CUDA se calcula o índice da thread. Um vector com N elementos pode ser agrupado em blocos, com a dimensão *blockDim*, com um determinado número de *threads* (*tb - threads per block*):

- Índice da *thread* dentro do bloco: **`threadIdx.x`**;
- Índice global da *thread*: **`index= threadIdx.x + blockIdx.x * blockDim.x`**;
- Dimensão do bloco: **`blockDim.x`**.



1.1.2. Descrição do algoritmo: No exemplo usado está definida a inicialização de um vetor que multiplica o índice do vetor por 2.

```
...
      // CODE
      if(index<N)
            device_buffer[index]=index*2;
      ...
```

1.2 Código que executa no CPU (host code): Para além do código que qualquer programa requer, temos também as configurações necessárias para executar o programa na GPU, nomeadamente a alocação/transferência de dados na GPU e lançamento das funções que vão executar nela.

1.2.1 Alocação de memória na GPU: A função `cudaMalloc` aloca memória no dispositivo. Na chamada da função deve-se indicar o ponteiro (`device_buffer`) e o tamanho desejado (`sizeof(int)*N`).

```
...
  // Allocate buffer in the device
  int *device_buffer=NULL;
  err=cudaMalloc(device_buffer, sizeof(int)*N);
  ...
```

1.2.2 Lançamento das CUDA kernels: Aqui o utilizador deve definir a estrutura sobre a qual o programa vai iterar. Para tal, é necessário indicar o número de blocos (`blocksPerGrid`) e o número de threads por bloco (`threadsPerBlock`).

```
...
  // Launch device function
  int threadsPerBlock=256;
  int blocksPerGrid=N/256;
  device_func_name      <<<blocksPerGrid      ,      threadsPerBlock>>>
  (device_buffer,
  N);
  ...
```

1.2.3 Transferência dos dados da GPU para o host: Depois do programa executar é pertinente recolher o resultado do dispositivo. Para tal, é necessário usar a função `cudaMemcpy` para transferir os dados da GPU para o host.

```
...
  // Copy data from device memory to host memory
  int * host_buffer=(int *)malloc(sizeof(int)*N);
  err=cudaMemcpy(host_buffer,device_buffer, sizeof(int)*N,
  cudaMemcpyDeviceToHost);
  ...
```

1.2.4 Libertar memória alocada na GPU: Para libertar memória alocada na GPU deve usar a função `cudaFree`.

```
...
  // Free device buffers
  cudaFree(device_buffer);
  ...
```

2. Tendo em conta a análise do ponto anterior, escreva um novo programa, em CUDA, que implemente a soma de vetores: $v_1 + v_2 = v_3$.

2.1 Implemente uma versão do programa onde os vectores v_1 e v_2 são inicializados no host e transferidos para a GPU.

2.2 Implemente uma versão do programa onde os v_1 e v_2 são inicializados na GPU.

2.3 Assuma agora que os elementos a somar são inteiros positivos e não ultrapassam a gama dinâmica 0-256. Implemente uma versão do programa que otimize a gestão de memória na GPU, explorando de forma mais eficiente a largura de banda disponível (dica: precisa de 32-bit para representar estes dados?).

3. A linguagem CUDA permite expressar problemas em três dimensões (x, y, z). Implemente um programa em CUDA para somar duas matrizes e guardar o resultado numa terceira. Tenha em atenção que deve expressar as duas dimensões deste problema na CUDA kernel (`threadIdx.y`, `blockIdx.y`, etc.).

NOTA 1: O aluno pode e deve utilizar o programa Nvidia Visual Profiler para analisar o perfil de execução de cada programa em CUDA (correr o comando nvvp na linha de comandos para abrir o programa).

NOTA 2: O aluno deve consultar o guia de programação disponibilizado pela NVIDIA para mais informações (https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html).