Computação Heterogénea de Alto Desempenho Lab 2

Manuel Santos - 2019231352

05 October 2023

Exercício 1

Para resolver este exercício, comecei por implementar uma versão sequencial do código.

De seguida, analisei o código construído e identifiquei secções possíveis de paralelizar. A secção escolhida em causa foi o ciclo *for* principal (linha 69), pois é onde ocorre a maioria dos cálculos.

A diretiva escolhida foi a *acc kernels*.

De seguida, construí um ficheiro bash, que quando chamado com o argumento *"matrix_dim"*, corre 5 vezes cada uma das versões (sequencial e paralela). O facto de correr 5 veze serve como warmup para os cálculos e como forma de excluir outliers nos tempos.

Na tabela 1 é possível vizualizar os tempos de execução de cada uma das versões.

Dimensão Matrizes	**Sequencial**	**Paralela**
10×10	0.001286	63.369261
$10^2 \times 10^2$	0.001592	63.658546
$10^3 \times 10^3$	0.005274	63.551128
$10^4 \times 10^4$	0.035823	63.528542
$10^5 \times 10^5$	0.330609	64.606889
$10^6 \times 10^6$	3.186949	69.079642
$10^7 \times 10^7$	31.814746	103.658617
$10^8 \times 10^8$	316.887779	445.347368
$10^9 \times 10^9$	N/A	N/A

Table 1: Tempos de computação, em ms

Pela análise dos resultados, é possível concluir que os resultados da versão sequencial são bastante melhores aos obtidos com a versão paralela.

Esta diferença deve-se ao facto de na versão paralela o tempo de computação ser consideravelmente menor face ao tempo de transferência dos daos entre Host-Device e Device-Host.

Em suma, o overhead de transferência é tão grande que anula qualquer benefício de paralelização.

Exercício 2

Alínea a)

Para resolver este exercício, utilizei o código fornecido na aula pelo professor.

Este código utiliza a diretiva *acc kernels* no *loop for* principal.

Na tabela 2 estão registados os tempos de computação obtidos.

Dimensão Vetores	Sequencial	Paralela
10	0.000692	57.660828
10^2	0.003598	59.191677
10^3	0.032891	56.263539
10^4	0.326033	56.047478
10^5	3.301003	56.359085
10^6	15.475167	57.912639
10^7	81.690587	62.675758
10^8	735.022999	98.675815
10^9	7253.093165	343.076298
10^100	10224.700966	460.299380

Table 2: Tempos de computação, em ms

Com é possível constatar, a partir de uma determinada dimensão dos vetores, a versão paralela demonstra ser mais vantajosa.

Alínea b)

Sabendo que o *speedup* é dado por $s = \frac{T_original}{T_acelerado}$, obteve-se o gráfico visto na figura 1.

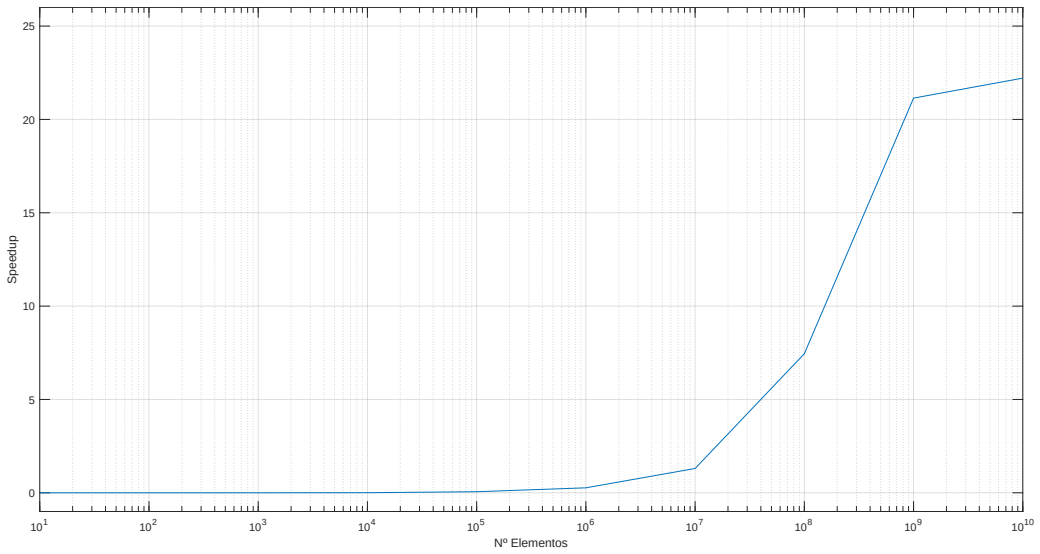

Figure 1: Curva de Speedup

Tal como constatado na alína anterior, a partir de uma determinada dimensão dos vetores, a versão paralela demonstra ser mais vantajosa. Para os valores menores, a paralelização não é vantajosa devido aos overheads de transferência dos dados entre Host-Device e Device-Host.

Exercício 3

Para resolver este exercício, comecei por implementar a versão sequencial.

Ao tentar implementar a versão paralela, utilizando inicialmente a diretiva *acc kernels*, não fui bem sucedido, devido a um erro de execução.

Alínea a)

Para melhorar eficiência da transferência de dados, é possível, por exemplo, copiar para o Device apenas os blocos de dados necessários aos cálculos, neste caso, os elementos que ocupam a parte triangular inferior da matriz.

Alínea b)

Para otimizar o trabalho realizado em cada iteração, de forma a que este seja equilibrado pelas diferentes threads da GPU,podemos efetuar as somas em pares de blocos: bloco 0 + bloco n, bloco 1 + bloco n-1, ...

Exercício 4

Neste exercício, temos um programa OpenACC que falha (produz um valor diferente de zero para error count).

Ao analisar o código, percebemos que temos um ciclo inicial onde inicializamos a nossa matriz, um ciclo paralelizado onde alteramos os valores da mesma e, por fim, um ciclo onde verificamos se os valores resultantes estão corretos ou não.

Apesar de este código fazer sentido, há um problema: não estamos a copiar a matriz inicializada para a GPU. Portanto, iremos sobrepor valores incorretos à matriz que inicializámos anteriormente.

Assim, uma solução para este problema pode passar por adicionar a diretiva *data copy* no pragma, de forma a garantirmos que estamos a adicionar e a escrever os dados no local certo.

Exercício 7

Alínea a)

Como visto na figura 2, o resultado apresentado não é mais do que a matriz inicial.

```
Jacobi relaxation Calculation: 10 x 10 mesh
    0, 1.000000
1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
  100, 1.000000
1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
 total: 0.105197 s
```

Figure 2: Output

Alínea b)

Citando o documento *"OpenACC Programming and Best Practices Guide"*, disponível no site da OpenACC:

"The update directive provides a way to explicitly update the values of host or device memory with the values of the other. This can be thought of as syncrhonizing the contents of the two memories. The update directive accepts a device clause for copying data from the host to the device and a self directive for updating from the device to local memory, which is the host memory, except in the case of nested OpenACC regions."

Alínea c)

Após a introdução da diretiva *acc update self(A[0 : m * n])* na linha antes de *if (iter % 100 == 0)*, o erro foi corrigido, como visto na figura 3.

Esta diretiva é necessária nesta posição para atualizar os dados no Host, e ter a certeza que estes são os corretos.

```
Jacobi relaxation Calculation: 10 x 10 mesh
    0, 1.000000
1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00
0.00 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
  100, 1.000000
1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00
0.00 0.49 0.67 0.74 0.77 0.77 0.74 0.67 0.49 0.00
0.00 0.28 0.45 0.54 0.58 0.58 0.54 0.45 0.28 0.00
0.00 0.17 0.30 0.38 0.42 0.42 0.38 0.30 0.17 0.00
0.00 0.11 0.20 0.26 0.29 0.29 0.26 0.20 0.11 0.00
0.00 0.07 0.14 0.18 0.20 0.20 0.18 0.14 0.07 0.00
0.00 0.05 0.09 0.12 0.14 0.14 0.12 0.09 0.05 0.00
0.00 0.03 0.05 0.07 0.08 0.08 0.07 0.05 0.03 0.00
0.00 0.01 0.03 0.03 0.04 0.04 0.03 0.03 0.01 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
 total: 0.071104 s
```

Figure 3: Output Corrigido