Computação de Alto Desempenho 2014/15
2º Teste – 5/6/2015
Duração: 2h00

O teste é sem consulta e em caso de dúvida na interpretação do enunciado, deve explicitar todos os pressupostos na elaboração das suas respostas.

1. Considere que tem uma aplicação que demora demasiado tempo a executar, pelo que se pretende fazer a sua execução numa plataforma computacional com GPUs. Diga, justificando, quais as estratégias de optimização do desempenho que teria de considerar na adaptação dessa aplicação, por forma a tirar o melhor partido possível da execução em GPU.

2. O algoritmo da "*bi-segmentação*" é um dos algoritmos muito usados no processamento de imagem, e.g. num dos passos da identificação de objectos em imagens na área de Ciências dos Materiais. Por exemplo, partindo de uma imagem tomográfica 2D de um material composto, o algoritmo usa como base dois valores *L1* e *L2* entre 0 e 255 (0- cor preta; 255- cor branca), representando cada um a cor de um de dois materiais A e B presentes na imagem e que devem ser diferenciados. Assim, com base nesses dois valores, o algoritmo aplica a seguinte operação a todos os pixels da imagem 2D:

```
if ( cor_pixel < L1 )
   cor_pixel = 0;   //pixel fica com cor preta (material A)
else if (cor_pixel > L2 ) {
   cor_pixel = 255; //pixel fica com cor branca (material B)
else
   cor_pixel = 128; //pixel fica com cor cinzenta (indeterminado)
```

Implemente o algoritmo acima descrito usando ***Cuda C***, considerando que a imagem 2D tem as dimensões de N*N bytes, sendo que cada byte contém o valor da cor do pixel nessa posição. Para tal, preencha o seguinte esqueleto de código em C, com o seu código necessário em ***Cuda C***, assumindo que há espaço no *device* para pelo menos duas imagens:

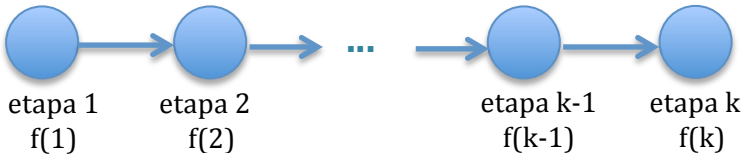
```
#include <stdio.h>
#include <stdlib.h>

#define N ...
unsigned char material[N][N]; // matriz com imagem inicial do material
unsigned char *pt_img;        // apontador para a imagem processada

// kernel function

int main (int argc, char *argv[])
{
   FILE *f;
   // Leitura da imagem
   f = fopen("input.bin", "r");
   fread(material, sizeof(unsigned char), N*N, f);
   fclose(f);
```

```
    // o seu código CUDA C

    // Escrita da imagem processada
    f= fopen("simulacao.bin", "w");
    fwrite(pt_img, sizeof(float), N*N, f);

    return 0;
}
```

3. Considere uma plataforma computacional constituída por ***N*** computadores pessoais interligados por uma rede local a 1 Gbps, no qual está instalado o *MPI*. Explique como poderia implementar as seguintes funcionalidades nessa plataforma, indicando que *funções de comunicação do MPI* usaria em cada caso:

a) Processamento de um vetor de inteiros ***V*** de dimensão ***d*** *(**d** muito maior que **N**)* para determinar quantos elementos do vetor são iguais a um valor ***val***; esse número deve ficar disponível em todos os nós (e não apenas no nó de rank 0).

b) Implementação de um *pipeline de **k** etapas (**k < N**)* para processamento de um fluxo (*stream*) de dados, no qual cada *etapa i* do *pipeline* aplica uma *função f(int etapa) com argumento i* para transformação dos dados, tal como indicado na figura:



Assuma que é *o nó com rank 0* que está continuamente a receber os dados de um canal *f_in* e que os dados processados pelo pipeline são escritos no canal *f_out*. A linha de comando do programa poderia ser:

```
mpiexec -np k_etapas –hostfilename stream_processing f_in f_out
```

4. O problema do cálculo da *figura de Mandelbrot* tem como base a seguinte função:

```
#define max_iterations 255
int compute_point(double ci, double cr) {
    int iterations = 0;
    double zi = 0;
    double zr = 0;
    while ((zr*zr + zi*zi < 4) && (iterations < max_iterations))
    {
        double nr, ni;
        nr = zr*zr - zi*zi + cr; ni = 2*zr*zi + ci;
        zi = ni; zr = nr;
        iterations ++;
    }
```

```
    return iterations;
}
```

Pretende-se, com base nessa função, construir o chamado *histograma de Mandelbrot*, i.e. saber para quantos pontos é o número de iterações igual a 1, para quantos é 2, etc, até para quantos é 255. Para o efeito, o algoritmo deve ser paralelizado usando as primitivas do MPI, de modo a executar num conjunto de 16 computadores pessoais interligados por uma rede local a 1 Gbps, no qual está instalado o MPI.
Suponha que a linha de comando seria:

```
mpiexec -np 16 –hostfilename mandelbrot_histogram x1 y1 x2 y2 L H
```

em que

- *x1*, *y1* são as coordenadas do canto superior esquerdo da área rectangular considerada.
- *x2*, *y2* são as coordenadas do canto inferior direito da área rectangular considerada.
- L é o número de pixels na horizontal.
- H é o número de pixels na vertical.

Descreva a aplicação da metodologia de Foster ao desenvolvimento dessa solução paralela, justificando, para cada fase, as opções tomadas.

5. Diga em que situações haveria vantagem em ter uma implementação híbrida (OpenMP+MPI) para paralelização de um problema particular, em vez de uma implementação em CUDA. Pode considerar como exemplo o problema de aplicar uma função a todos os elementos de uma matriz de bytes, tal como descrito na *pergunta 2*.