INE 5645 – PROGRAMAÇÃO PARALELA E DISTRIBUIDA – PROVA 2 – 13/11/2017

ALUNO _____

1. **Sockets - Indicar (Verdade/Falso):** (2.0)
   (a) (Verdade/Falso) A comunicação entre processos consiste em transmitir uma mensagem entre o *socket* de um processo e o socket de outro processo.
   (b) (Verdade/Falso) *Sockets* é um conceito de redes, implementados com um par (IP, porta).
   (c) (Verdade/Falso) Para que um processo receba mensagens, seu *socket* deve estar vinculado ao IP de um computador e a uma porta local desse computador em que é executado.
   (d) (Verdade/Falso) As mensagens enviadas para um endereço IP e uma porta específica, só podem ser recebidas por um processo cujo *socket* esteja associada a esse IP e a esse número de porta.
   (e) (Verdade/Falso) Processos não podem usar o mesmo *socket* para enviar e receber mensagens.
   (f) (Verdade/Falso) Um processo pode fazer uso de uma porta para receber mensagens.
   (g) (Verdade/Falso) Um processo pode compartilhar portas com outros processos no mesmo computador.
   (h) (Verdade/Falso) Qualquer número de processos podem enviar mensagens para a mesma porta. Isto não ocorre em *IP Multicast.*

2. Indique (Correto/Errado) (1.0)

(**C**) As **arquiteturas** apropriadas para *Paralelismo de dados* é "SIMD" e para *Paralelismo de tarefas* é "MIMD".

(**E**) SIMD significa que as unidades paralelas têm instruções distintas, então cada uma delas pode fazer algo diferente em um dado momento.

(**E**) MIMD significa que todas as unidades paralelas compartilham a mesma instrução, mas a realizam em diferentes elementos de dados.

(**C**) MIMD significa que as unidades paralelas têm instruções distintas, então cada uma delas pode fazer algo diferente em um dado momento.

3. **Esboce, de forma resumida, o modelo heterogêneo de plataforma utilizado na programação paralela com OpenCL/CUDA.** (1.0)

Qualquer desenho que identique a heterogeinidade de plataforma **CPU + GPU**.

4. **OpenCL -** Um objeto de programa encapsula o *código-fonte de um kernel*, sendo este identificados no código-fonte por meio da palavra-chave `__kernel`. O que significa, o seguinte código, executado em OpenCL ? (1.0)

```
__kernel void ArrayDiff (
          __global const int* a,
          __global const int* b,
          __global int* c )
{
  int id = get_global_id(0);
  c[id] = a[id] - b[id];
}
```

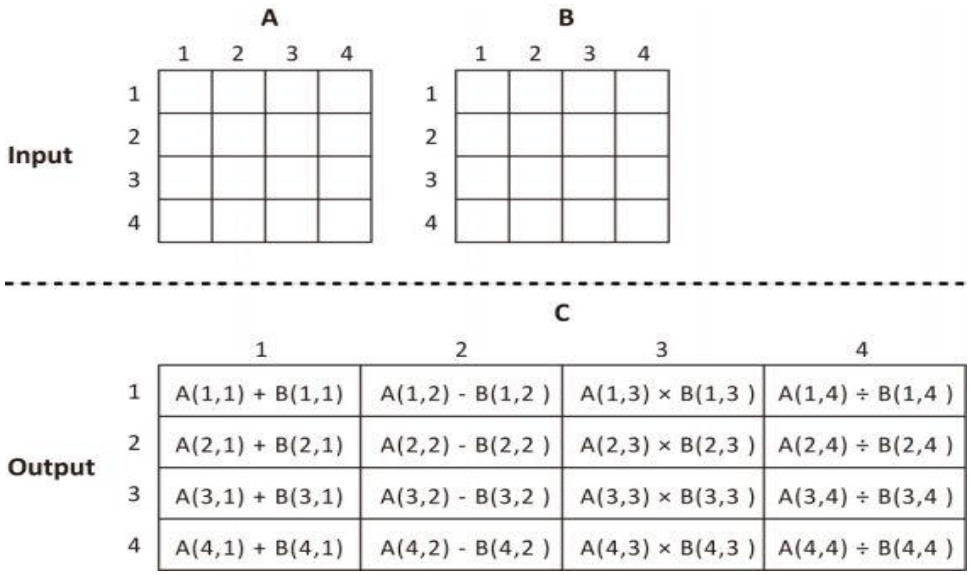
Significa o código que é paralelizado na GPU. O kernel é instanciado (como se fiosse uma classe Java) e surgem várias threads processadas em paralelo, executadas em núcleos distintos na GPU. Para cada dessas threads, o código corresponde a um id= 0, 1, 2, 3, ..., n. Sendo n o tamanho do vetor que deve ser fornecido no programa que usa o kernel.

5. **O modelo de programação descreve as abordagens possíveis para a execução de código OpenCL. O modelo de programação em OpenCL é semelhante ao utilizado em CUDA. *Kernels* podem ser executados de dois modos distintos:** (1.0)

• (Verdade/Falso) - Paralelismo de dados: um mesmo kernel é instanciado através de múltiplos núcleos processadores sobre dados distintos.

• (Verdade/Falso) - Paralelismo de tarefas: kernels distintos devem ser instanciados para a execução de um único *work-item* (thread) *que executará múltiplas operações*.

6. **Observe a figura abaixo com a entrada Input de duas matrizes A e B, e que as operações sobre os elementos de A e de B são feitas conforma o quadro Output. O codigo amostra realize as operações aritméticas básicas: adição, subtraçao, multiplicação e divisão.**



Como mostra a figura acima, para resolver este problema, podemos utilizar a programação paralela visualizando um modelo de paralelismo de dados ou um modelo de paralelismo de tarefas. Para cada problema, o modelo de paralelismo deve ser escolhido.

(a) Em qual modelo de paralelismo o código abaixo é apropriado, visualizando as operações sobre as linhas das matrizes A, B e C? Explique brevemente o porquê. (0.5)

```
1. __kernel void dataParallel(__global float* A, __global float* B, __global float* C)
2. {
3. int base = 4*get_global_id(0);
4. C[base+0] = A[base+0] + B[base+0];
5. C[base+1] = A[base+1] - B[base+1];
6. C[base+2] = A[base+2] * B[base+2];
7. C[base+3] = A[base+3] / B[base+3];
8. }
```

Quando o kernel é instanciado, *work-items* (threads) são criados para o paralelismo de dados nas linhas das matrizes. Cada destes work-items (threads) executa o mesmo kernel em paralelo. Este código trata cada linha de dados como um *work-group* OpenCL (grupo de threads equivalente ao conceito de bloco de threads do CUDA) para executar a computação paralela.

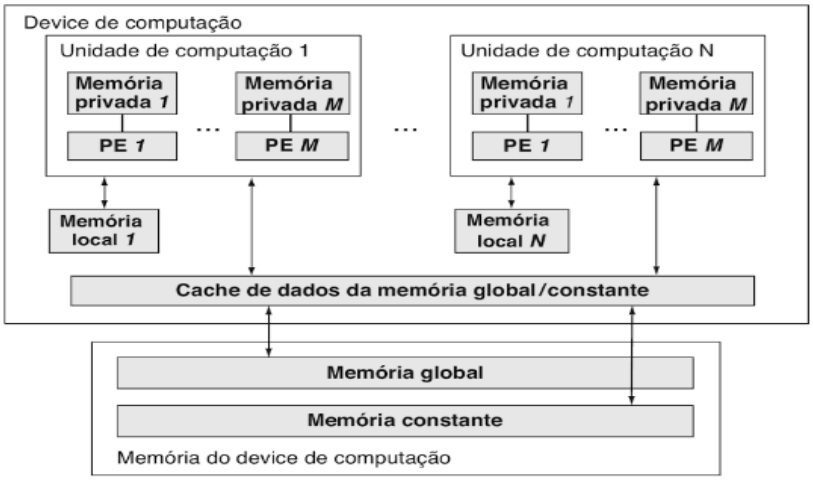
(b) Em qual modelo de paralelismo o código abaixo é apropriado, visualizando as operações sobre as colunas das matrizes A, B e C? Explique brevemente o porquê. (0.5)

```
1.  __kernel void taskParallelAdd(__global float* A, __global float* B, __global float*
    C)
2.  {
3.  int base = 0;
4.
5.  C[base+0] = A[base+0] + B[base+0];
6.  C[base+4] = A[base+4] + B[base+4];
7.  C[base+8] = A[base+8] + B[base+8];
8.  C[base+12] = A[base+12] + B[base+12];
9.  }
10.
11. __kernel void taskParallelSub(__global float* A, __global float* B, __global float*
    C)
12. {
13. int base = 1;
14.
15. C[base+0] = A[base+0] - B[base+0];
16. C[base+4] = A[base+4] - B[base+4];
17. C[base+8] = A[base+8] - B[base+8];
18. C[base+12] = A[base+12] - B[base+12];
19. }
20.
21. __kernel void taskParallelMul(__global float* A, __global float* B, __global float*
    C)
22. {
23. int base = 2;
24.
25. C[base+0] = A[base+0] * B[base+0];
26. C[base+4] = A[base+4] * B[base+4];
27. C[base+8] = A[base+8] * B[base+8];
28. C[base+12] = A[base+12] * B[base+12];
29. }
30.
31. __kernel void taskParallelDiv(__global float* A, __global float* B, __global float*
    C)
32. {
33. int base = 3;
34.
35. C[base+0] = A[base+0] / B[base+0];
36. C[base+4] = A[base+4] / B[base+4];
37. C[base+8] = A[base+8] / B[base+8];
38. C[base+12] = A[base+12] / B[base+12];
39. }
```

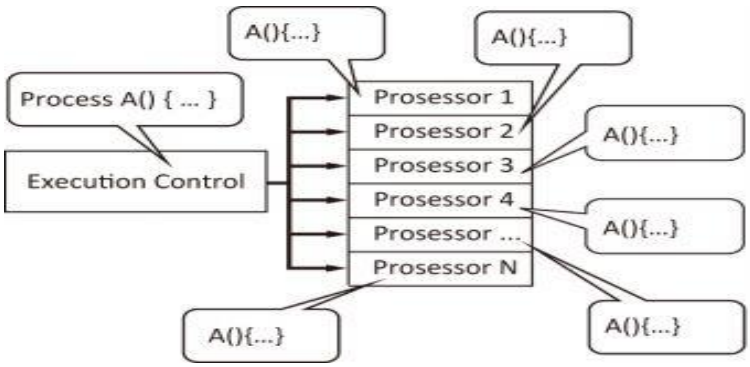
O código acima mostra a versão paralela com tarefas do mesmo problema que o item anterior. Nesta segunda alternativa de código, as tarefas são agrupadas de acordo ao tipo de operação aritmética sendo operada, utilizando-se, portanto, o modelo de paralelismo de tarefas.

7. Indique qual modelo (**plataforma**/**execução**/**memória**), está mostrado na figura seguinte: (0.5)

O modelo de memória para OpenCL é representado.

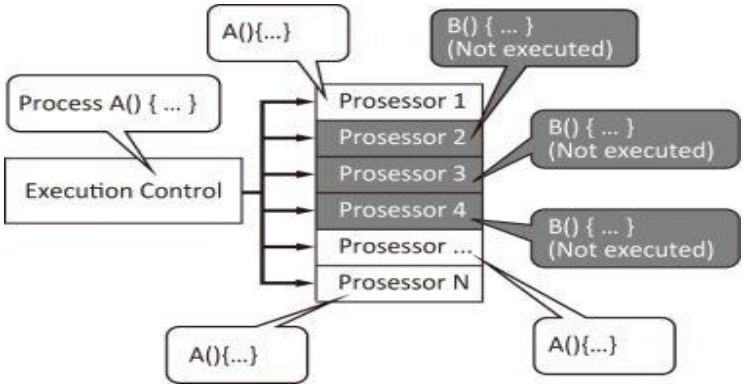


8. O que significa a figura abaixo: (1.0)



Representa o uso eficiente de GPU para paralelismo de dados. Quando vários processadores executam a mesma tarefa, o número de tarefas, igual ao número de processadores, pode ser executado ao mesmo tempo, de uma vez.

9. O que significa a figura seguinte: (1.0)



- Representa o uso ineficiente de GPU para paralelismo de dados. As GPUs funcionam muito bem para paralelismo de dados (códigos iguais) e são incapazes de executar tarefas de códigos diferentes em paralelo, junto com tarefas de códigos iguais, ao mesmo tempo. A figura mostra o caso em que tarefas A e B estão programadas para serem executadas em paralelo na GPU. Como processadores só podem processar o mesmo conjunto de instruções nos núcleos (códigos iguais, a unidade de controle envia uma única instrução), os processadores agendados para processar a Tarefa B (códigos diferentes de A) devem estar no modo inativo até que a Tarefa A esteja concluída (quando a unidade de controle poderá enviar um outra instrução para a tarefa B).

10. Complete a correspondência entre os elementos de processamento em CUDA e OpenCL respectivamente: (0.5)

CUDA	***OpenCL***
Host	*Host*
Device	*Device*
kernel	*kernel*
Thread	***workitem***
Bloco (de threads)	***workgroup***
Grid	*NDRange*

Veja o quadro comparativo abaixo:

CUDA	**OpenCL**
Streamming Multiprocessor (SM)	Compute Unit (CU)
Streamming Processor (SP)	Processing Element (PE)
host	*host*
device	*device*
kernel	*kernel*
thread	*workitem*
bloco	*workgroup*
grid	*NDRange*