INE5645-Programação Paralela e Distribuída – Aula 17/09/2018
Nome _____

Para paralelizar códigos de programas, tudo que necessitamos é de uma construção sintática denominada *kernel* .
Seja o kernel:

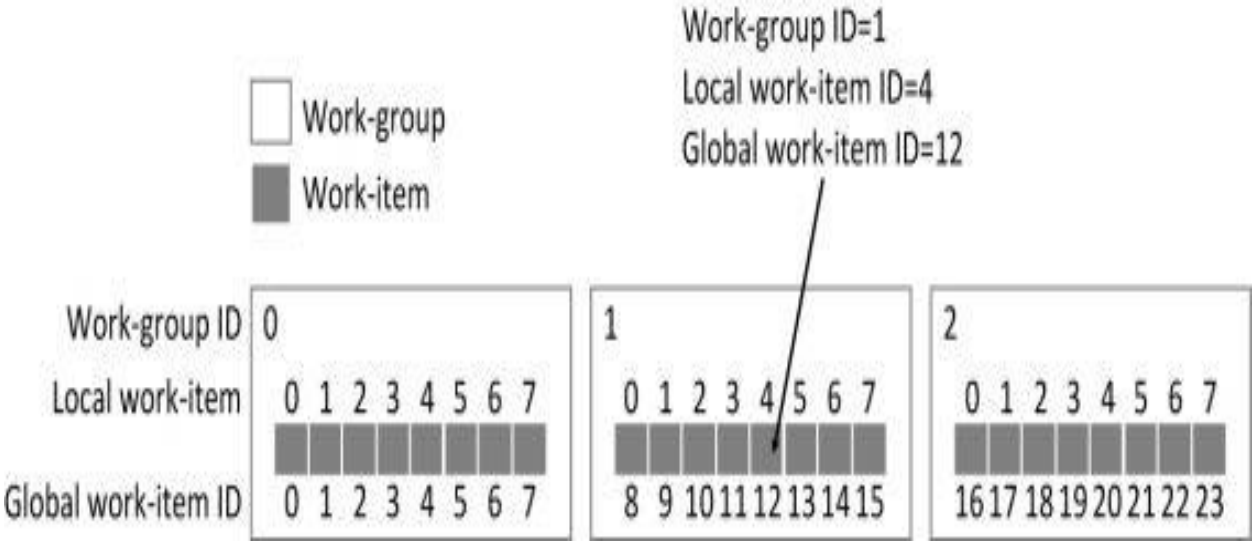
```
__kernel void SAXPY ( __global float* x,
                      __global float* y,
                                float a )
{
    const int i = get_global_id (0);

    y [i] += a * x [i];
}
```

Para OpenCL, responder as questões:

1. Um kernel é a trecho de código C code. Especificamente é um função marcada com o qualificador ( ).
2. O qualificador __kernel declara uma função para ser um kernel que pode ser executado por uma aplicação sobre um OpenCL device(dispositivo) conhecido como uma *Graphical Processing Unit*
   ( ).
3. As seguintes regras se aplicam a funções que são declaradas com __Kernel:
   (a) Ela pode ser executada somente sobre um device ( ).
   (b) Ela pode ser chamada pelo lado Host ( ).
4. No kernel acima ( ), indica que tem-se memória global, ou simplesmente, memória alocada pelo device ( ).
5. (Verdade/Falso) Neste exemplo, estamos usando somente memória global.
6. (Verdade/Falso) Sobre o kernel *SAXPY,* pode-se dizer que a ausência da descrição de um loop (como no caso sequencial) é uma decisão de projeto em OpenCL.
7. Seja entendermos, primeiro, como o código de um kernel é executado.
   (a) (Verdade/Falso) A hipótese básica é muitas instâncias de um kernel são executadas em paralelo, cada uma executando em um work-item (thread).
   (b) (Verdade/Falso) Múltiplos *work-items* formam *um* ***grupo de threads*** onde, executadas juntas, formam um *work-group (um bloco de threads) de OpenCL.*

8. (Verdade/Falso) Dentro de um *work-group*, instâncias de kernel são executadas em paralelo.
9. (Verdade/Falso) No sentido de identificar uma instância de kernel, o ambiente de *runtime* do OpenCL provê um *id*.
10. (Verdade/Falso) Dentro do kernel, nós obtemos um *id* com um `get_global_id`, o qual retorna o *id* do work-item corrente.
11. (Verdade/Falso) O paralelismo se inicia com muitas instâncias, tantas, quantas foram os elementos de nosso vetor, e cada *work-item* processa, exatamente, uma entrada.
12. (Verdade/Falso) Múltiplos kernels podem se apresentar em um único arquivo, chamado de *programa*.
13. *About Work-group* ID and *Work-item* ID

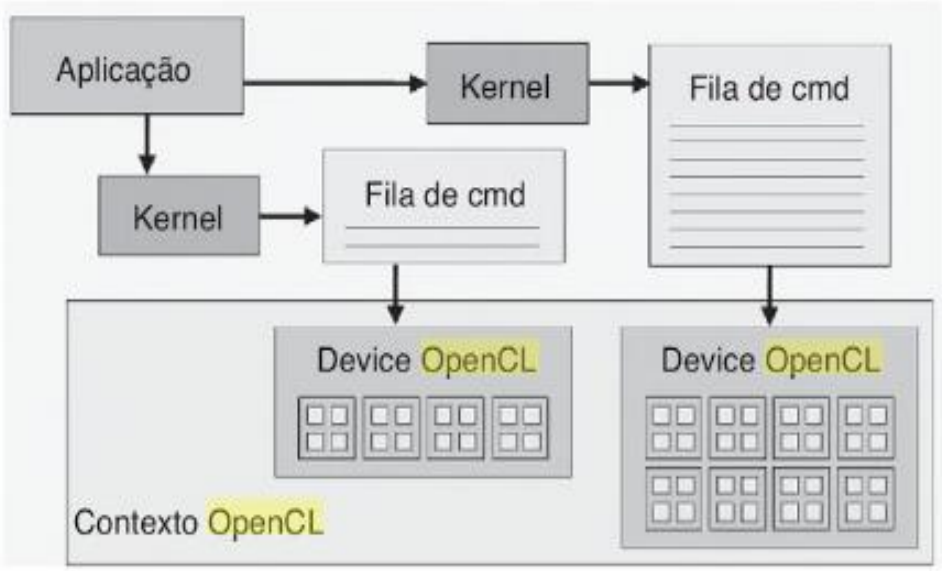


A terminologia na figura acima é relativa ao OpenCL.

14. Em relação a CUDA:

(Verdade/Falso) Um *work-item* em OpenCL corresponde a uma thread em CUDA.
(Verdade/Falso) Um *work-group* corresponde a um conjunto de *work-itens*, e em CUDA chama-se _____ de threads.

15. Sobre o *contexto* OpenCL para um *device GPU*:



- (Verdade/Falso) A execução de *kernels* em uma aplicação OpenCL só é possível após a definição de um ***contexto***.
- (Verdade/Falso) Um ***contexto*** engloba um *conjunto de dispositivos (devices)* e *kernels*, além de outras estruturas necessárias para a operação da aplicação, como *filas de comandos*, *objetos de programa* e *objetos de memória (buffers)*.

16.O código sequencial em C é paralelizado no kernel seguinte:

```
void ArrayDiff (const int* a, const int* b, int* c, int n)
{
  for (int i = 0; i < n; ++i)
  {
    c[i] = a[i] - b[i];
  }
}
```

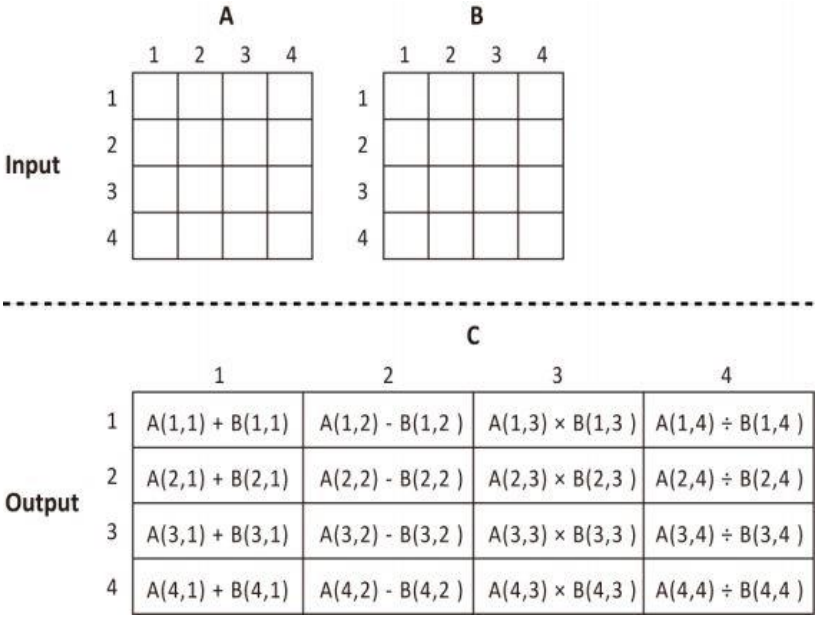
```
__kernel void ArrayDiff (
    __global const int* a,
    __global const int* b,
    __global int* c )
```

```
{
  int id = get_global_id(0);
  c[id] = a[id] - b[id];
}
```

No kernel acima, quantos threads deverão existir para realizar a computação paralela da operação entre os vetores a, b e c ?

Resposta: _____

16. Seja o problema a seguir:



No kernel a seguir:

```
__kernel void dataParallel (
                __global float* A,

                __global float* B,

                __global float* C )
{
     int id = 4*get_global_id(0);
```

```
        C[id+0] = A[id+0] + B[id+0];
        C[id+1] = A[id+1] - B[id+1];
        C[id+2] = A[id+2] * B[id+2];
        C[id+3] = A[id+3] / B[id+3];
    }
```

Quantas threads (*work-itens*) deverão existir na computação paralela para as matrizes A e B redundar na matriz C ?

Resposta: _____

17. O modelo de paralelismo no kernel acima, corresponde ao paralelismo de _____(tarefa/dados).

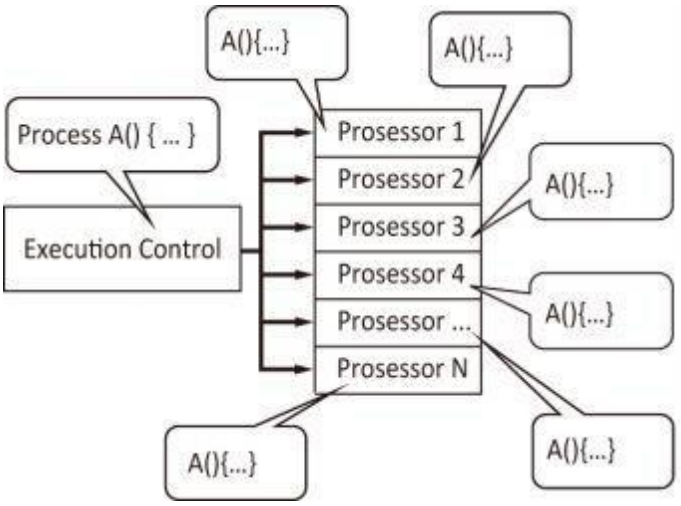
Seja as operações de soma das primeiras colunas de A e B:

[A(1,1) + B(1,1)], [A(2,1) + B(2,1)], [A(3,1) + B(3,1)], [A(4,1) + B(4,1)]

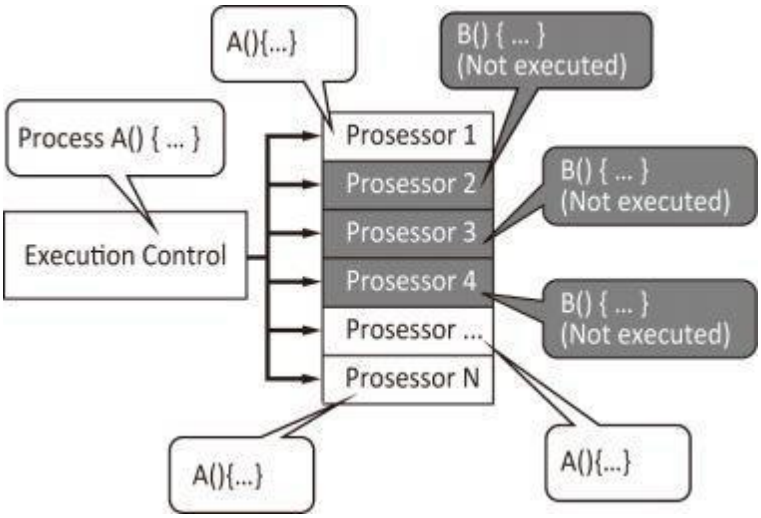
Estas operações de soma correspondem a um conjunto de (quantas?) _____ threads dispostas como na figura dos núcleos (ALUs) uma GPU:

Cntlr	A(1,1) + B(1,1)	A(2,1) + B(2,1)	A(3,1) + B(3,1)	A(4,1) + B(4,1)
Cache				
Cntrl	A(1,2) - B(1,2)	A(2,2) - B(2,2)	A(3,2) - B(3,2)	A(4,2) - B(4,2)
Cache				
Cntrl	A(1,3) * B(1,3)	A(2,3) * B(2,3)	A(3,3) * B(3,3)	A(4,3) * B(4,3)
Cache				
Cntrl	A(1,4) / B(1,4)	A(2,4) / B(2,4)	A(3,4) / B(3,4)	A(4,4) / B(4,4)
Cache				

18. **OpenCL – Dadas as figuras abaixo, o que significam cada uma delas ?**



Resposta: Representa o uso eficiente de GPU para paralelismo de dados. Exemplifica o uso da arquitetura SIMD (Single Instruction Multiple Data). 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.



Resposta: 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, 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).

19. (ERRADA/CERTA)

( ) (**Paralelismo de Tarefas**) Uma técnica de programação que divide uma quantidade de dados em partes menores que podem ser operadas em paralelo. A mesma operação é executada simultaneamente (isto é, em paralelo). E
( ) (**Paralelismo de Dados**) Uma técnica de programação que divide uma grande quantidade de dados em partes menores que podem ser operadas em paralelo. A mesma operação é executada simultaneamente (isto é, em paralelo). C
( ) As **arquiteturas** apropriadas para *Paralelismo de dados* é "MIMD" e para *Paralelismo de tarefas* é "SIMD". E
( ) **SIMD** significa que as unidades paralelas têm instruções distintas, então cada uma delas pode fazer algo diferente em um dado momento. E
( ) **SIMD** significa que todas as unidades paralelas compartilham a mesma instrução, mas a realizam em diferentes elementos de dados. C
( ) No **modelo de paralelismo de tarefas**, para cada operação a ser executada, deve ser definido um kernel para executar uma determinada operação na arquitetura MIMD. C

20. **OpenCL -** Dê um exemplo, envolvendo uma operação sobre números, que seja executada com **paralelismo de dados**.

Seja a sequência de números inteiros: 1, 2, 3, 4, 5, 6, 7, 8, 9. Tome a operação da raiz quadrada de cada valor, A operação é a mesma (a raiz quadrada), mas o dado (o valor) é diferente.