Nume, Prenume
Grupa
Data

Examen – PGPU
Master CI, 2019 - 2020
Nr. 1

1) Mentionati doua tipuri de intarzieri (overhead) ce pot aparea in programele CUDA.
0.3p

2) In CUDA accesul unui thread la memoria shared este mai rapid decat la memoria globala
0.3p deoarece……..

3) Care afirmatie este adevarata?
0.3p
a) cudaMemcpy() este o functie asincrona, inainte de startarea copierii datelor fiind necesara sincronizarea thread-urilor
b) thread-urile CUDA nu pot coopera prin intermediul memoriei comune
c) o variabila de tip __shared__ este vizibila tuturor thread-urilor dintr-un bloc
d) tipul de retur al unui kernel CUDA nu poate fi void

4) In CUDA, un apel de kernel este o operatie _____(sincrona/asincrona).
0.2p

5) Fie urmatoarele declaratii/definitii:
0.4p

```
__device__ int deviceVar;
__global__ void someFunc(int *input, int *output) {
   int someVar;
   int someArray[10];
   __shared__ int anotherVar[10];
   ….
}
```

In ce spatiu de memorie CUDA (shared/global/register/constant/texture) se aloca urmatoarele variabile?
deviceVar:_____; someVar:_____; someArray: :_____; anotherVar:_____

6) Fie urmatorul kernel:
1.5p

```
__global__ void accessFloat3(float3 *d_in, float3 *d_out)
{
   int index = blockIdx.x * blockDim.x + threadIdx.x;
   float3 a = d_in[index];
   a.x += 2;
   a.y += 2;
   a.z += 2;
   d_out[index] = a;
}
```

Care este problema cu acest kernel? Explicati cum ati rezolva aceasta problema

7) Fie urmatoarea secventa de cod:
1p

```
__global__ void my_kernel(int *out, int *in) {
  __shared__ int my_shared_variable;
  if(threadIdx.x == 0) my_shared_variable = 0;
  my_shared_variable += in[threadIdx.x + blockIdx.x * blockDim.x];
  if(threadIdx.x == 0) out[blockIdx.x] = my_shared_variable;
}
```

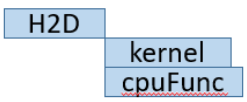
Cum ar trebui modificat acest program astfel incat, dupa executia kernelului folosind un grid de 4 blocuri cu 256 de thread-uri fiecare, vectorul `out` sa contina 4 valori corespunzatoare sumelor elementelor din fiecare partitie de 256 de elemente consecutive din vectorul `in` (out[0] este suma primelor 256 de elemente, out[1] este suma elementelor in[256]…in[511], s.a.m.d.)? Se considera ca vectorul `in` contine 1024 de elemente iar gridul si blocul sunt organizate liniar.

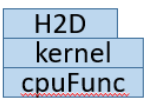
8) Un multiprocesor de pe un GPU suporta 32K registri si pana la 2048 de thread-uri. Cate warp-uri
0.4p pot fi executate concurent daca se utilizeaza blocuri de 1024 thread-uri si fiecare thread necesita 32 registri? Justificati.

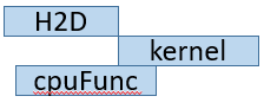
9) Fie urmatoarea secventa de cod:
0.5p

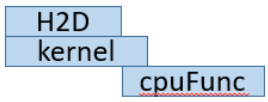
```
cudaMemcpyAsync(dev_a, host_a, size, cudaMemcpyHostToDevice, 0);
kernel<<<grid, block>>>(dev_a);
cpuFunc();
```

Care din urmatoarele variante ilustreaza corect timeline-ul de executie?

a) 

b) 

c) 

d) 

10) Completati cu notiunile corespunzatoare spatiile marcate prin "……":
0.4p

a) compilatorul CUDA se numeste ……
b) in CUDA, modificatorul …… marcheaza o functie kernel
c) in CUDA, id-ul unui thread poate fi accesat prin variabila interna ……
d) memoria …… a unui GPU este dispusa on-chip si poate fi accesata mult mai rapid decat memoria DRAM

11) Unified memory
0.3p

a) Permite realizarea unei singure alocari a memoriei pentru date ce vor fi disponibile atat pe GPU cat si pe CPU prin intermediul unui aceluiasi pointer, eliminand astfel transferul de date intre CPU si GPU
b) Permite ca prin intermediul aceleiasi variabile (pointer), sa se adreseze de pe GPU o zona de memorie alocata pe CPU, dar nu si invers.

c) Permite ca prin intermediul aceleiasi variabile (pointer), sa se adreseze de pe CPU o zona de memorie alocata pe GPU, dar nu si invers.
d) Nu elimina necesitatea transferului de date intre CPU si GPU

12) **0.5p** Se da un vector de numere reale, reprezentate in dubla precizie. Se doreste procesarea acestor elemente folosind CUDA, urmand pasii: (1) transfer vector intrare din memoria CPU in memoria GPU, (2) executie kernel ce realizeaza procesarea, (3) transfer vector iesire din memoria GPU in memoria CPU. Presupunand ca fiecare thread CUDA se va ocupa de un element din vector, descrieti pe scurt o solutie de accelerare (imbunatatire a performantelor) a unei astfel de aplicatii.

13) **0.3p** Care afirmatie este adevarata?
a) un thread CUDA are acces R/W la memoria constanta
b) un kernel CUDA este executat pe host
c) CUDA permite cooperarea intre thread-uri
d) CPU si GPU au un spatiu de memorie comuna

14) **0.5p** Ce reprezinta accesul fuzionat la memorie (coalescing)?

15) **0.3p** Care sunt diferentele dintre un thread GPU si unul CPU?

16) **0.5p** Ce este un branch divergent si cum este tratat pe GPU?

17) **1p** Dati doua exemple de operatii de reducere si mentionati doua metode prin care se poate realiza o astfel de operatie in paralel pe GPU.

18) **0.3p** Fie urmatoarea secventa de cod:

```
int main(int argc, char *argv[]) {
   …
   kernel1<<<GRID_SIZE, BLOCK_SIZE>>>(devParamIn, devParamOut);
   kernel2<<<GRID_SIZE, BLOCK_SIZE>>>(devParamIn, devParamOut);
   retVal = cpuFunction(hostParamIn, hostParamOut);
   …
}
```

Marcati cu ADEVARAT sau FALS urmatoarele afirmatii:

a. Functia `cpuFunction` este apelata numai dupa terminarea executiei `kernel2` __[A/F]__
b. Functia `cpuFunction` este apelata imediat dupa apelul kernel2, fara a astepta terminarea executiei acestuia __[A/F]__
c. Cele doua functii kernel se executa in paralel (simultan) pe GPU __[A/F]__

Oficiu 1p