Exercices pour la préparation de l'examen de mi-session/Mid-term exam preparation exercises

Questions à choix multiples pour l'examen de mi-session/Multiple-choice questions for the mid-term exam.

Question 1
Comment les threads sont-ils organisés dans CUDA? / How are threads organized in CUDA?

a) Par matrice / By matrix
b) Par grilles et blocs / By grids and blocks **(Correct)**
c) Par segments / By segments
d) Par sous-groupes / By subgroups

Question 2
Quel est le rôle principal des blocs de threads dans CUDA? / What is the main role of thread blocks in CUDA?

a) Assurer la synchronisation globale / Ensure global synchronization
b) Permettre le calcul parallèle / Enable parallel computation **(Correct)**
c) Gérer la mémoire locale / Manage local memory
d) Exécuter des instructions séquentielles / Execute sequential instructions

Question 3
Combien de threads peut contenir un warp dans CUDA? / How many threads can a warp contain in CUDA?

a) 16
b) 32 **(Correct)**
c) 64
d) 128

Question 4
Quels threads partagent la mémoire partagée dans CUDA? / Which threads share shared memory in CUDA?

a) Tous les threads de la grille / All threads in the grid
b) Tous les threads d'un bloc / All threads in a block **(Correct)**
c) Tous les threads d'un warp / All threads in a warp
d) Tous les threads d'un programme / All threads in a program

Question 5
Pourquoi utiliser la mémoire partagée dans CUDA? / Why use shared memory in CUDA?

a) Pour stocker des données à long terme / To store long-term data

b) Pour accélérer l'accès aux données dans un bloc / To speed up data access within a block **(Correct)**
c) Pour synchroniser les threads entre les blocs / To synchronize threads across blocks
d) Pour réduire l'utilisation des registres / To reduce register usage

Question 6
Quelle est la taille maximale d'un bloc dans CUDA (en nombre de threads)? / What is the maximum size of a block in CUDA (in terms of threads)?
a) 512
b) 1024 **(Correct)**
c) 2048
d) 4096

Question 7
Quel type de mémoire est accessible par tous les blocs dans une grille CUDA? / What type of memory is accessible by all blocks in a CUDA grid?
a) Mémoire globale / Global memory **(Correct)**
b) Mémoire partagée / Shared memory
c) Mémoire locale / Local memory
d) Mémoire de registre / Register memory

Question 8
Quel est le rôle d'un kernel dans CUDA? / What is the role of a kernel in CUDA?
a) Synchroniser les threads / Synchronize threads
b) Gérer l'allocation de mémoire / Manage memory allocation
c) Exécuter des fonctions sur le GPU / Execute functions on the GPU **(Correct)**
d) Définir la taille des blocs / Define block size

Question 9
Comment les threads dans un bloc communiquent-ils entre eux? / How do threads in a block communicate with each other?
a) En utilisant la mémoire globale / Using global memory
b) En utilisant des registres locaux / Using local registers
c) En utilisant la mémoire partagée / Using shared memory **(Correct)**
d) En exécutant des instructions en série / By executing instructions serially

Question 10
Quelle fonction est utilisée pour synchroniser les threads dans un bloc CUDA? / Which function is used to synchronize threads within a CUDA block?
a) __cudaDeviceSynchronize()
b) __syncthreads() **(Correct)**
c) __threadFence()
d) __deviceBarrier()

Question 11
Quel est le rôle du registre dans CUDA? / What is the role of the register in CUDA?

a) Gérer la mémoire globale / Manage global memory
b) Stocker les variables locales pour chaque thread / Store local variables for each thread **(Correct)**
c) Partager des données entre les threads / Share data between threads
d) Synchroniser les threads dans un bloc / Synchronize threads within a block

Question 12
Quelle est la fonction de la mémoire constante dans CUDA? / What is the function of constant memory in CUDA?

a) Fournir une mémoire rapide et locale pour chaque thread / Provide fast, local memory for each thread
b) Stocker des valeurs qui ne changent pas pendant l'exécution du kernel / Store values that do not change during kernel execution **(Correct)**
c) Partager des données entre les grilles / Share data across grids
d) Optimiser la latence des threads / Optimize thread latency

Question 13
Que se passe-t-il si les threads d'un warp n'exécutent pas les mêmes instructions? / What happens if threads in a warp do not execute the same instructions?

a) Ils sont annulés / They are canceled
b) Ils sont synchronisés automatiquement / They are synchronized automatically
c) Ils divergent et exécutent tous les chemins possibles / They diverge and execute all possible paths **(Correct)**
d) Ils attendent la fin du kernel / They wait for the kernel to finish

Question 14
Comment les threads sont-ils numérotés dans un bloc dans CUDA? / How are threads numbered within a block in CUDA?

a) Par numéro de bloc global / By global block number
b) Par `threadIdx.x`, `threadIdx.y`, `threadIdx.z` / By `threadIdx.x`, `threadIdx.y`, `threadIdx.z` **(Correct)**
c) Par numéro de warp / By warp number
d) Par `blockIdx.x` et `blockIdx.y` / By `blockIdx.x` and `blockIdx.y`

Question 15
Quelle est la différence entre la mémoire globale et la mémoire partagée dans CUDA? / What is the difference between global memory and shared memory in CUDA?

a) La mémoire partagée est accessible par tous les threads d'une grille / Shared memory is accessible by all threads in a grid
b) La mémoire globale est plus rapide que la mémoire partagée / Global memory is faster than shared memory
c) La mémoire partagée est plus rapide et est limitée à un bloc / Shared memory is faster and limited to a block **(Correct)**
d) La mémoire globale est utilisée pour la synchronisation des threads / Global memory is used for thread synchronization

Question 16
Quelle méthode est la plus efficace pour réduire une grande quantité de données dans CUDA? / What is the most efficient method to reduce a large amount of data in CUDA?

a) Réduction par blocs / Block-wise reduction **(Correct)**
b) Réduction en série / Serial reduction
c) Réduction par grilles / Grid-wise reduction
d) Réduction inter-warp / Inter-warp reduction

Question 17
Quel est l'avantage de la programmation SIMT dans CUDA? / What is the advantage of SIMT programming in CUDA?

a) Elle permet une exécution séquentielle des threads / It allows sequential execution of threads
b) Elle permet à plusieurs threads d'exécuter la même instruction en parallèle / It allows multiple threads to execute the same instruction in parallel **(Correct)**
c) Elle réduit la consommation de mémoire globale / It reduces global memory usage
d) Elle synchronise automatiquement les threads / It automatically synchronizes threads

Question 18
Dans quelle situation utiliseriez-vous la fonction `cudaMallocHost` dans CUDA? / In what situation would you use the `cudaMallocHost` function in CUDA?

a) Lorsque vous voulez allouer de la mémoire globale sur le GPU / When you want to allocate global memory on the GPU
b) Lorsque vous voulez allouer de la mémoire accessible par le CPU et le GPU / When you want to allocate memory accessible by both the CPU and GPU **(Correct)**
c) Lorsque vous voulez synchroniser les threads dans un bloc / When you want to synchronize threads within a block
d) Lorsque vous voulez réduire la latence de la mémoire partagée / When you want to reduce shared memory latency

Question 19
Quel type de parallélisme est utilisé par les warps dans CUDA? / What type of parallelism is used by warps in CUDA?

a) SIMD (Single Instruction, Multiple Data) / SIMD (Single Instruction, Multiple Data) **(Correct)**

b) MIMD (Multiple Instruction, Multiple Data) / MIMD (Multiple Instruction, Multiple Data)

c) SISD (Single Instruction, Single Data) / SISD (Single Instruction, Single Data)

d) MISD (Multiple Instruction, Single Data) / MISD (Multiple Instruction, Single Data)

Question 20
Que fait la fonction `cudaDeviceSynchronize()` dans CUDA? / What does the `cudaDeviceSynchronize()` function do in CUDA?

a) Elle synchronise les threads dans un bloc / It synchronizes threads within a block

b) Elle synchronise les blocs dans une grille / It synchronizes blocks within a grid

c) Elle attend que tous les threads sur le GPU terminent leur exécution / It waits for all threads on the GPU to finish execution **(Correct)**

d) Elle alloue de la mémoire partagée sur le GPU / It allocates shared memory on the GPU

Question 21
Quel est le rôle de la fonction __syncthreads() dans CUDA? / What is the role of the __syncthreads() function in CUDA?

A) Synchroniser les threads au sein d'un bloc / Synchronize threads within a block

B) Arrêter l'exécution des threads / Stop thread execution

C) Partager les registres entre les threads / Share registers between threads

D) Synchroniser les blocs au sein d'une grille / Synchronize blocks within a grid

Correct Answer: A

Question 22
Comment les threads peuvent-ils accéder aux données constantes dans CUDA? / How can threads access constant data in CUDA?

A) Via la mémoire globale / Via global memory

B) Via la mémoire constante / Via constant memory

C) Via la mémoire partagée / Via shared memory

D) Via les registres / Via registers

Correct Answer: B

Question 23
Que se passe-t-il si une fonction kernel dans CUDA dépasse les limites de la mémoire allouée? / What happens if a CUDA kernel function exceeds the allocated memory limits?

A) Les threads redémarrent automatiquement / Threads automatically restart
B) Un dépassement de mémoire se produit et le programme échoue / A memory overrun occurs, and the program fails
C) Les données non utilisées sont ignorées / Unused data is ignored
D) Les threads continuent à s'exécuter sans problème / Threads continue to execute without issues
Correct Answer: B

Question 24
Quelle fonction dans CUDA est utilisée pour allouer de la mémoire sur le GPU? / Which function in CUDA is used to allocate memory on the GPU?

A) cudaFree
B) cudaMemcpy
C) cudaMalloc
D) cudaDeviceSynchronize
Correct Answer: C

Question 25
Combien de threads sont exécutés en parallèle dans un warp dans CUDA? / How many threads are executed in parallel in a warp in CUDA?

A) 8
B) 16
C) 32
D) 64
Correct Answer: C

Question 26
Quelle est la fonction principale utilisée pour libérer la mémoire allouée sur le GPU dans CUDA? / What is the primary function used to free memory allocated on the GPU in CUDA?

A) cudaFree
B) cudaMalloc
C) cudaDeviceSynchronize
D) cudaMemset
Correct Answer: A

Question 27
Que signifie la divergence de threads dans un warp en CUDA? / What does thread divergence in a warp in CUDA mean?

A) Les threads suivent des chemins d'exécution différents / Threads follow different execution paths
B) Les threads se synchronisent dans un warp / Threads synchronize in a warp
C) Les threads se partagent les registres / Threads share registers
D) Les threads se réinitialisent automatiquement / Threads automatically reset
Correct Answer: A

Question 28
Comment la latence mémoire peut-elle être masquée dans un programme CUDA? / How can memory latency be hidden in a CUDA program?

A) En utilisant la mémoire constante / By using constant memory
B) En augmentant le nombre de threads par bloc / By increasing the number of threads per block
C) En utilisant la mémoire partagée / By using shared memory
D) En utilisant des warps inactifs pour effectuer d'autres calculs / By using idle warps to perform other computations
Correct Answer: D

Question 29
Quel est l'avantage principal d'utiliser la mémoire partagée dans CUDA? / What is the main advantage of using shared memory in CUDA?

A) Augmenter la capacité mémoire globale / Increase overall memory capacity
B) Réduire la latence d'accès à la mémoire / Reduce memory access latency
C) Permettre aux threads de partager des registres / Allow threads to share registers
D) Permettre l'allocation dynamique de mémoire / Allow dynamic memory allocation
Correct Answer: B

Question 30
Pourquoi l'optimisation des accès à la mémoire globale est-elle importante dans CUDA? / Why is optimizing global memory access important in CUDA?

A) Pour maximiser l'utilisation des registres / To maximize register usage
B) Pour minimiser la latence et améliorer les performances / To minimize latency and improve performance
C) Pour synchroniser les warps plus efficacement / To synchronize warps more efficiently
D) Pour allouer plus de mémoire partagée / To allocate more shared memory
Correct Answer: B

Questions à développement/Development Questions:

Question 1.

Combien de threads sont créés avec la configuration suivante ? / How many threads are created with the following configuration?

```
dim3 grid(8, 4);
dim3 block(16, 8);
```

Solution (Français) :

La configuration ci-dessus crée :
8 blocs dans la direction x et 4 blocs dans la direction y, soit un total de 8 x 4 = 32 blocs dans la grille.
Chaque bloc contient 16 threads dans la direction x et 8 threads dans la direction y, soit un total de 16 x 8 = 128 threads par bloc.
Le nombre total de threads créés est donc : Total threads = Nombre de blocs x Nombre de threads par bloc = 32 x 128 = 4096.

Solution (English) :

The configuration above creates:
8 blocks in the x direction and 4 blocks in the y direction, resulting in 8 x 4 = 32 blocks in the grid.
Each block contains 16 threads in the x direction and 8 threads in the y direction, which gives 16 x 8 = 128 threads per block.
The total number of threads created is: Total threads = Number of blocks x Number of threads per block = 32 x 128 = 4096

Question 2

Quelle est la configuration de la grille et des blocs pour une matrice 1024 x 1024 avec 32 threads par dimension de bloc ? / What is the grid and block configuration for a 1024 x 1024 matrix with 32 threads per block dimension?

Solution (Français) :

Pour une matrice de 1024 x 1024, si chaque dimension de bloc contient 32 threads :
Le nombre de blocs par dimension dans la grille est donné par : Nombre de blocs = 1024 / 32 = 32 blocs par dimension.
La configuration de la grille sera donc dim3 grid(32, 32) et la configuration des blocs sera dim3 block(32, 32).

Solution (English) :

For a 1024 x 1024 matrix, if each block dimension contains 32 threads:
The number of blocks per dimension in the grid is given by: Number of blocks = 1024 / 32 = 32 blocks per dimension.

The grid configuration will be dim3 grid(32, 32) and the block configuration will be dim3 block(32, 32).

Question 3

Montrez les différentes étapes de l'algorithme Neighbored Pair Sum pour le tableau suivant et complétez la fonction neighboredPairSum(). / Show the different steps of the Neighbored Pair Sum algorithm for the following table and complete the neighboredPairSum() function: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]

Solution

Code:

```
__global__ void neighboredPairSum(int* input, int n, int step) {
    int tid = threadIdx.x;  // Thread ID in the block
    int offset = 1 << step;  // Calculate the offset for each step

    if (tid % (2 * offset) == 0 && (tid + offset) < n) {
        input[tid] += input[tid + offset];
    }
    __syncthreads(); // Thread synchronization
}
```

Question 4.

Montrez les différentes étapes de l'approche par paires entrelacées pour le tableau suivant et complétez la fonction neighboredPairSum(). / Show the different steps of the Neighbored Pair Sum algorithm for the following table and complete the neighboredPairSum() function: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]

Solution:

```
Initial array:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Array state after step 1 :
10 12 14 16 18 20 22 24 9 10 11 12 13 14 15 16
Array state after step 2 :
28 32 36 40 18 20 22 24 9 10 11 12 13 14 15 16
Array state after step 3 :
64 72 36 40 18 20 22 24 9 10 11 12 13 14 15 16
Array state after step 4 :
136 72 36 40 18 20 22 24 9 10 11 12 13 14 15 16
Final reduced sum: 136
```

Code

```
__global__ void interleavedpairsSum(int* input, int n, int step) {
    int tid = threadIdx.x;
    int index = blockDim.x * blockIdx.x + tid;

    int offset = blockDim.x / (2 << step);  // Dynamically adjust offset for each
step

    if (tid < offset && (index + offset) < n) {
```

```
        input[index] += input[index + offset];
    }
    __syncthreads();  // Ensure all threads finish before moving to the next step
}
```

Question 5.
Comparez l'approche par paires voisines et l'approche interlacée en termes d'efficacité dans les algorithmes de réduction / Compare the neighbor-pair and interleaved approaches in terms of efficiency in reduction algorithms.

Solution:
Neighbor-pair approach involves adjacent elements, and each thread works on pairs of neighboring elements, which can lead to better coalesced memory access. However, it may suffer from thread divergence if the reduction size is unevenly divisible by the number of threads.
Interleaved-pair approach reduces the number of active threads by half in each step, which can reduce the number of memory accesses. However, it may lead to more uncoalesced memory accesses in the early steps, as non-adjacent elements are accessed.

In general, interleaved reduction is often preferred for its simplicity and fewer divergent threads in large arrays.

Question 6.
Décrivez la technique d'unrolling et comment elle peut être appliquée pour optimiser les performances dans une réduction / Describe the unrolling technique and how it can be applied to optimize performance in a reduction.

Solution:
Loop unrolling involves manually duplicating the loop body to reduce the overhead of loop control instructions. By performing multiple reductions in a single iteration, this technique can increase instruction-level parallelism and reduce the number of synchronization barriers.
Code Example with unrolling:

```
__global__ void unrolledReduction(int *input) {
    int tid = threadIdx.x;
    for (int stride = blockDim.x / 4; stride > 0; stride /= 4) {
        input[tid] += input[tid + stride];
        input[tid] += input[tid + stride * 2];
        input[tid] += input[tid + stride * 3];
        __syncthreads();
    }
}
```