Question

Dans l'approche somme des paires voisines, après l'étape 2, le tableau initial [3, 9, 6, 7, 10, 2, 14, 16, 8, 20, 22, 24, 26, 28, 30, 32] subit des modifications. Quelle est la nouvelle valeur des indices 0 à 3 dans le tableau ? / In the Neighbored Pair Sum approach, after step 2, the initial array [3, 9, 6, 7, 10, 2, 14, 16, 8, 20, 22, 24, 26, 28, 30, 32] undergoes modifications. What is the new value of indices 0 to 3 in the array?

Options

a) [25, 9, 13, 7] / [25, 9, 13, 7]

b) [12, 4, 16, 8] / [12, 4, 16, 8]

c) [42, 2, 30, 16] / [42, 2, 30, 16]

d) [67, 9, 13, 7] / [67, 9, 13, 7]

Réponse / Answer

a) [25, 9, 13, 7] / [25, 9, 13, 7]

Question à choix multiples / Multiple Choice Question

Dans l'approche somme des paires entrelacées, après l'étape 2, le tableau initial [3, 9, 6, 7, 10, 2, 14, 16, 8, 20, 22, 24, 26, 28, 30, 32] subit des modifications. Quelle est la nouvelle valeur des indices 0 à 3 dans le tableau ?

In the Interleaved Pair Sum approach, after step 2, the initial array [3, 9, 6, 7, 10, 2, 14, 16, 8, 20, 22, 24, 26, 28, 30, 32] undergoes modifications. What is the new value of indices 0 to 3 in the array?

a) [11, 29, 28, 31]

b) [47, 59, 72, 79]

c) [119, 138, 72, 79]

d) [257, 138, 72, 79]

Réponse correcte / Correct Answer

b) [47, 59, 72, 79]

20 Questions à Choix Multiples sur Lecture 5 (Français/Anglais)

Question 1:

Quelle est la mémoire avec la plus faible latence dans l'architecture CUDA ?

Which memory has the lowest latency in the CUDA architecture?

a) Mémoire globale / Global memory

b) Mémoire partagée / Shared memory

c) Mémoire constante / Constant memory

d) Mémoire en cache L2 / L2 cached memory

Réponse / Answer: b) Mémoire partagée / Shared memory

Question 2:

Quel type d'accès provoque un conflit de banque dans la mémoire partagée ?

Which type of access causes a bank conflict in shared memory?

a) Accès parallèle / Parallel access

b) Accès sériel / Serial access

c) Accès diffusé / Broadcast access

d) Accès coalescé / Coalesced access

Réponse / Answer: b) Accès sériel / Serial access

Question 3:

Quel est le rôle principal de la mémoire constante dans CUDA ?

What is the main role of constant memory in CUDA?

a) Fournir un stockage temporaire pour les threads / Provide temporary storage for threads

b) Servir de mémoire en lecture-écriture rapide / Act as fast read-write memory

c) Fournir une mémoire en lecture optimisée pour les threads d'un warp / Provide optimized read-only memory for warp threads

d) Stocker les résultats des calculs / Store computation results

Réponse / Answer: c) Fournir une mémoire en lecture optimisée pour les threads d'un warp / Provide optimized read-only memory for warp threads

---

Multiple-Choice Questions - Lecture 5

Question 1

Which memory type in CUDA has the lowest latency? / Quel type de mémoire dans CUDA a la latence la plus faible?

- a. Global Memory / a. Mémoire globale
- b. Shared Memory / b. Mémoire partagée
- c. Constant Memory / c. Mémoire constante
- d. Texture Memory / d. Mémoire de texture

Correct Answer: b

Question 2

What is the main benefit of shared memory in CUDA? / Quel est le principal avantage de la mémoire partagée dans CUDA?

- a. Faster global memory writes / a. Écritures plus rapides en mémoire globale
- b. Reduction in bank conflicts / b. Réduction des conflits de banque
- c. Low latency and high bandwidth / c. Faible latence et large bande passante
- d. Increases global memory size / d. Augmente la taille de la mémoire globale

Correct Answer: c

Question 3

What is a bank conflict in CUDA shared memory? / Qu'est-ce qu'un conflit de banque dans la mémoire partagée CUDA?

- a. Multiple threads accessing the same bank address / a. Plusieurs threads accédant à la même adresse de banque
- b. All threads accessing the same memory location / b. Tous les threads accédant à la même adresse mémoire
- c. Excessive usage of shared memory / c. Utilisation excessive de la mémoire partagée
- d. Inefficient memory allocation / d. Allocation inefficace de mémoire

Correct Answer: a

Question 4

How can bank conflicts be avoided in shared memory? / Comment éviter les conflits de banque dans la mémoire partagée?

- a. Use padding between memory elements / a. Utiliser du rembourrage entre les éléments de mémoire

- b. Allocate more global memory / b. Allouer plus de mémoire globale
- c. Synchronize threads before accessing memory / c. Synchroniser les threads avant d'accéder à la mémoire
- d. Use read-only cache instead / d. Utiliser à la place un cache en lecture seule

Correct Answer: a

Question 5

What does __syncthreads() do in CUDA? / Que fait __syncthreads() dans CUDA?

- a. Terminates a kernel / a. Termine un kernel
- b. Synchronizes threads within a block / b. Synchronise les threads au sein d'un bloc
- c. Allocates shared memory dynamically / c. Alloue de la mémoire partagée de manière dynamique
- d. Optimizes global memory access / d. Optimise l'accès à la mémoire globale

Correct Answer: b

Multiple-Choice Questions - Lecture 5

Question 1:

Which memory type in CUDA has the lowest latency? / Quel type de mémoire dans CUDA a la latence la plus faible?

a. Global Memory / a. Mémoire globale

b. Shared Memory / b. Mémoire partagée

c. Constant Memory / c. Mémoire constante

d. Texture Memory / d. Mémoire de texture

Réponse / Answer : b)

Question 2:

What is the main benefit of shared memory in CUDA? / Quel est le principal avantage de la mémoire partagée dans CUDA?

a. Faster global memory writes / a. Écritures plus rapides en mémoire globale

b. Reduction in bank conflicts / b. Réduction des conflits de banque

c. Low latency and high bandwidth / c. Faible latence et large bande passante

d. Increases global memory size / d. Augmente la taille de la mémoire globale

Réponse / Answer : c)

Question 3:

What is a bank conflict in CUDA shared memory? / Qu'est-ce qu'un conflit de banque dans la mémoire partagée CUDA?

a. Multiple threads accessing the same bank address / a. Plusieurs threads accédant à la même adresse de banque

b. All threads accessing the same memory location / b. Tous les threads accédant à la même adresse mémoire

c. Excessive usage of shared memory / c. Utilisation excessive de la mémoire partagée

d. Inefficient memory allocation / d. Allocation inefficace de mémoire

Réponse / Answer : a)

Question 4:

How can bank conflicts be avoided in shared memory? / Comment éviter les conflits de banque dans la mémoire partagée?

a. Use padding between memory elements / a. Utiliser du rembourrage entre les éléments de mémoire

b. Allocate more global memory / b. Allouer plus de mémoire globale

c. Synchronize threads before accessing memory / c. Synchroniser les threads avant d'accéder à la mémoire

d. Use read-only cache instead / d. Utiliser à la place un cache en lecture seule

Réponse / Answer : a)

Question 5:

What does __syncthreads() do in CUDA? / Que fait __syncthreads() dans CUDA?

a. Terminates a kernel / a. Termine un kernel

b. Synchronizes threads within a block / b. Synchronise les threads au sein d'un bloc

c. Allocates shared memory dynamically / c. Alloue de la mémoire partagée de manière dynamique

d. Optimizes global memory access / d. Optimise l'accès à la mémoire globale

Réponse / Answer : b)

Question 6:

What is the purpose of padding in shared memory? / Quel est l'objectif du rembourrage dans la mémoire partagée?

a. To reduce memory usage / a. Réduire l'utilisation de la mémoire

b. To avoid bank conflicts / b. Éviter les conflits de banque

c. To increase latency / c. Augmenter la latence

d. To allocate memory dynamically / d. Allouer la mémoire dynamiquement

Réponse / Answer : b)

Question 7:

What type of access pattern minimizes bank conflicts in shared memory? / Quel type de modèle d'accès minimise les conflits de banque dans la mémoire partagée?

a. Column-major / a. Colonne principale

b. Random / b. Aléatoire

c. Row-major / c. Ligne principale

d. Mixed / d. Mixte

Réponse / Answer : c)

Question 8:

What does the __shared__ qualifier in CUDA indicate? / Que signifie le qualificateur __shared__ dans CUDA?

a. Global memory / a. Mémoire globale

b. Shared memory / b. Mémoire partagée

c. Constant memory / c. Mémoire constante

d. Read-only memory / d. Mémoire en lecture seule

Réponse / Answer : b)

Question 9:

What is the effect of __threadfence() in CUDA? / Quel est l'effet de __threadfence() dans CUDA?

a. Ensures thread synchronization within a warp / a. Assure la synchronisation des threads dans un warp

b. Ensures memory consistency across the grid / b. Assure la cohérence mémoire dans la grille

c. Reduces latency in memory accesses / c. Réduit la latence dans les accès mémoire

d. Enables dynamic memory allocation / d. Permet l'allocation de mémoire dynamique

Réponse / Answer : b)

Question 10:

How is constant memory declared in CUDA? / Comment la mémoire constante est-elle déclarée dans CUDA?

a. Using __global__ / a. En utilisant __global__

b. Using __shared__ / b. En utilisant __shared__

c. Using __constant__ / c. En utilisant __constant__

d. Using __device__ / d. En utilisant __device__

Réponse / Answer : c)

Multiple-Choice Questions - Lecture 5

Question 1:

Which memory type in CUDA has the lowest latency? / Quel type de mémoire dans CUDA a la latence la plus faible?

a. Global Memory / a. Mémoire globale

b. Shared Memory / b. Mémoire partagée

c. Constant Memory / c. Mémoire constante

d. Texture Memory / d. Mémoire de texture

Réponse / Answer : b

Question 2:

What is the main benefit of shared memory in CUDA? / Quel est le principal avantage de la mémoire partagée dans CUDA?

a. Faster global memory writes / a. Écritures plus rapides en mémoire globale

b. Reduction in bank conflicts / b. Réduction des conflits de banque

c. Low latency and high bandwidth / c. Faible latence et large bande passante

d. Increases global memory size / d. Augmente la taille de la mémoire globale

Réponse / Answer : c

Question 3:

What is a bank conflict in CUDA shared memory? / Qu'est-ce qu'un conflit de banque dans la mémoire partagée CUDA?

a. Multiple threads accessing the same bank address / a. Plusieurs threads accédant à la même adresse de banque

b. All threads accessing the same memory location / b. Tous les threads accédant à la même adresse mémoire

c. Excessive usage of shared memory / c. Utilisation excessive de la mémoire partagée

d. Inefficient memory allocation / d. Allocation inefficace de mémoire

Réponse / Answer : a

Question 4:

How can bank conflicts be avoided in shared memory? / Comment éviter les conflits de banque dans la mémoire partagée?

a. Use padding between memory elements / a. Utiliser du rembourrage entre les éléments de mémoire

b. Allocate more global memory / b. Allouer plus de mémoire globale

c. Synchronize threads before accessing memory / c. Synchroniser les threads avant d'accéder à la mémoire

d. Use read-only cache instead / d. Utiliser à la place un cache en lecture seule

Réponse / Answer : a

Question 5:

What does __syncthreads() do in CUDA? / Que fait __syncthreads() dans CUDA?

a. Terminates a kernel / a. Termine un kernel

b. Synchronizes threads within a block / b. Synchronise les threads au sein d'un bloc

c. Allocates shared memory dynamically / c. Alloue de la mémoire partagée de manière dynamique

d. Optimizes global memory access / d. Optimise l'accès à la mémoire globale

Réponse / Answer : b

Question 6:

What is the purpose of padding in shared memory? / Quel est l'objectif du rembourrage dans la mémoire partagée?

a. To reduce memory usage / a. Réduire l'utilisation de la mémoire

b. To avoid bank conflicts / b. Éviter les conflits de banque

c. To increase latency / c. Augmenter la latence

d. To allocate memory dynamically / d. Allouer la mémoire dynamiquement

Réponse / Answer : b

Question 7

What type of access pattern minimizes bank conflicts in shared memory? / Quel type de modèle d'accès minimise les conflits de banque dans la mémoire partagée?

a. Column-major / a. Colonne principale

b. Random / b. Aléatoire

c. Row-major / c. Ligne principale

d. Mixed / d. Mixte

Réponse / Answer : c

Question 8:

What does the __shared__ qualifier in CUDA indicate? / Que signifie le qualificateur __shared__ dans CUDA?

a. Global memory / a. Mémoire globale

b. Shared memory / b. Mémoire partagée

c. Constant memory / c. Mémoire constante

d. Read-only memory / d. Mémoire en lecture seule

Réponse / Answer : b

Question 9:

What is the effect of __threadfence() in CUDA? / Quel est l'effet de __threadfence() dans CUDA?

a. Ensures thread synchronization within a warp / a. Assure la synchronisation des threads dans un warp

b. Ensures memory consistency across the grid / b. Assure la cohérence mémoire dans la grille

c. Reduces latency in memory accesses / c. Réduit la latence dans les accès mémoire

d. Enables dynamic memory allocation / d. Permet l'allocation de mémoire dynamique

Réponse / Answer : b

Question 10:

How is constant memory declared in CUDA? / Comment la mémoire constante est-elle déclarée dans CUDA?

a. Using __global__ / a. En utilisant __global__

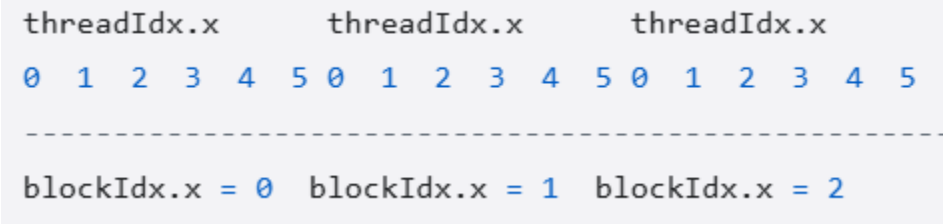
b. Using __shared__ / b. En utilisant __shared__

c. Using __constant__ / c. En utilisant __constant__

d. Using __device__ / d. En utilisant __device__

Réponse / Answer : c

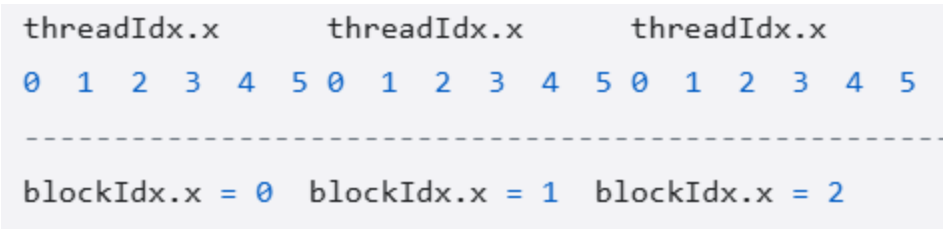
Question 1: Dans la configuration illustrée, combien de threads y a-t-il par bloc / In the configuration illustrated, how many threads are there per block?



a) 1
b) 2
c) 3
d) 6

Réponse/Answer : d) 6

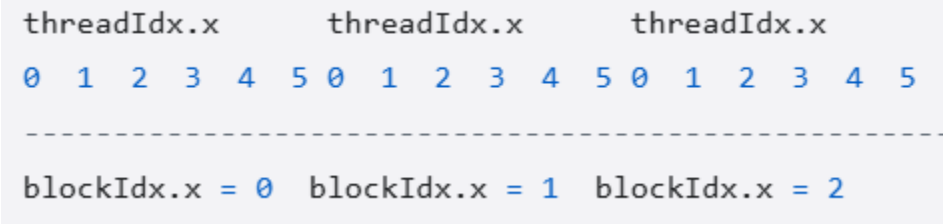
Question 2: Quel est l'index global du thread avec threadIdx.x = 4 dans le bloc où blockIdx.x = 1, en utilisant la configuration illustrée ci-dessous ? / What is the global index of the thread with threadIdx.x = 4 in the block where blockIdx.x = 1, using the configuration illustrated below?



a) 4
b) 5
c) 10
d) 11

Réponse/Answer: c) 10

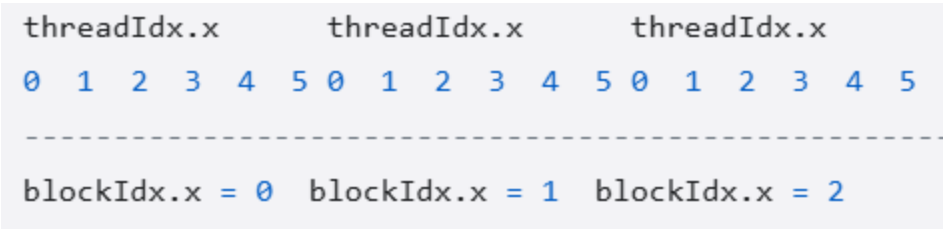
Question 3: Combien de blocs sont configurés dans la grille / How many blocks are configured in the grid?



a) 1
b) 2
c) 3
d) 6

Réponse/Answer: c) 3

Question: Dans cette configuration, quelle est la valeur de blockDim.x ? / In this configuration, what is the value of blockDim.x?



a) 1
b) 2
c) 3
d) 6

Réponse/Answer: d) 6

Questions sur Loop Unrolling / Questions on Loop Unrolling

Question 1

Pourquoi utilise-t-on le Loop Unrolling en CUDA C ? / Why is Loop Unrolling used in CUDA C?

a) Réduire la latence des branchements conditionnels / Reduce latency of conditional branching.
b) Améliorer l'utilisation de la mémoire globale / Improve global memory usage.
c) Diminuer la taille du code binaire généré / Decrease the size of the binary code.
d) Éviter l'utilisation de la mémoire partagée / Avoid shared memory usage.

Réponse/Answer: a) Réduire la latence des branchements conditionnels / Reduce latency of conditional branching.

Question 2

Quel est le résultat de l'exécution du code suivant ? / What is the output of the following code?

```c
#include <stdio.h>

__global__ void loopUnrollingExample(int *arr) {
    int idx = threadIdx.x;
    arr[idx] = idx * idx;  // Première itération
    arr[idx + 1] = (idx + 1) * (idx + 1);  // Deuxième itération
}

int main() {
    int arr[8], *d_arr;
    cudaMalloc(&d_arr, 8 * sizeof(int));
    loopUnrollingExample<<<1, 4>>>(d_arr);
    cudaMemcpy(arr, d_arr, 8 * sizeof(int), cudaMemcpyDeviceToHost);
    cudaFree(d_arr);

    for (int i = 0; i < 8; i++) printf("%d ", arr[i]);
    return 0;
}
```

a) 0 1 4 9 16 25 36 49
b) 0 1 4 9 16 25 36 0
c) 0 1 4 9 0 0 0 0
d) 0 0 0 0 0 0 0 0

Réponse/Answer: b) 0 1 4 9 16 25 36 0.

Question 3

Quel facteur d'enroulement serait le plus efficace pour optimiser ce code ? / What unrolling factor would be most effective for optimizing this code?

```c
__global__ void reduceKernel(int *data) {
   int idx = threadIdx.x + blockIdx.x * blockDim.x;
   int sum = 0;

   for (int i = 0; i < 4; i++) {
      sum += data[idx + i];
   }
   data[idx] = sum;
}
```

a) 2
b) 4
c) 8
d) Aucun, la boucle est déjà optimale / None, the loop is already optimal.

Réponse/Answer: b) 4.

Question 4

Dans quel cas le Loop Unrolling est-il inefficace ? / When is Loop Unrolling ineffective?

a) Lorsque le nombre d'itérations est très élevé / When the number of iterations is very high.
b) Lorsque le nombre d'itérations est inconnu à la compilation / When the iteration count is unknown at compile-time.
c) Lorsque la mémoire partagée est saturée / When shared memory is saturated.
d) Lorsque des threads dans un warp divergent fortement / When threads in a warp diverge heavily.

Réponse/Answer: b) Lorsque le nombre d'itérations est inconnu à la compilation / When the iteration count is unknown at compile-time.

Question 5

Identifiez l'erreur dans le code suivant utilisant le Loop Unrolling. / Identify the error in the following code using Loop Unrolling.

```c
__global__ void loopUnrollingError(int *arr, int N) {
   int idx = threadIdx.x + blockIdx.x * blockDim.x;
```

```
    if (idx < N) {
        arr[idx] = idx;
        arr[idx + 1] = idx + 1;  // Potentielle erreur
    }
}
```

a) Le facteur d'enroulement est trop grand / The unrolling factor is too large.
b) idx + 1 peut dépasser la limite N / idx + 1 might exceed the limit N.
c) Le tableau arr est mal alloué / The array arr is incorrectly allocated.
d) Aucun problème / No issue.

Réponse/Answer: b) idx + 1 peut dépasser la limite N / idx + 1 might exceed the limit N.

Question 6

Comment améliorer ce code en utilisant le Loop Unrolling ? / How can this code be improved using Loop Unrolling?

```c
__global__ void sumArray(int *data, int N) {
    int idx = threadIdx.x + blockIdx.x * blockDim.x;
    int sum = 0;

    for (int i = idx; i < N; i += blockDim.x) {
        sum += data[i];
    }
    data[idx] = sum;
}
```

a) Diviser la boucle en plusieurs itérations non conditionnelles / Divide the loop into multiple unconditional iterations.
b) Utiliser `__syncthreads()` pour synchroniser les threads / Use `__syncthreads()` to synchronize threads.
c) Utiliser un facteur d'enroulement fixe pour traiter plusieurs éléments par thread / Use a fixed unrolling factor to process multiple elements per thread.
d) Ajouter une vérification de limites pour éviter les erreurs d'accès mémoire / Add boundary checks to avoid memory access errors.

Réponse/Answer: c) Utiliser un facteur d'enroulement fixe pour traiter plusieurs éléments par thread / Use a fixed unrolling factor to process multiple elements per thread.

Questions à choix multiples sur les matrices transposées (Français/Anglais)

Question 1 / Question 1:

Quelle est la transposée de la matrice d'origine suivante, représentée en mémoire par :
What is the transpose of the following original matrix, represented in memory as:
[Matrice d'origine / Original Matrix] = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
si elle est de dimension (3 × 4) (3 lignes, 4 colonnes) / if it has dimensions (3 × 4) (3 rows, 4 columns)?

a) [0, 4, 8, 1, 5, 9, 2, 6, 10, 3, 7, 11]

b) [0, 1, 2, 4, 5, 6, 8, 9, 10, 3, 7, 11]

c) [0, 3, 6, 9, 1, 4, 7, 10, 2, 5, 8, 11]

d) [0, 8, 1, 9, 2, 10, 3, 11, 4, 5, 6, 7]

Réponse / Answer : a)

Question 2 / Question 2:

Si la matrice transposée est donnée par :
If the transposed matrix is given as:
[Matrice transposée / Transposed Matrix] = [0, 4, 8, 1, 5, 9, 2, 6, 10, 3, 7, 11]
Quelle était la disposition d'origine si la matrice est de dimension (4 × 3) (4 lignes, 3 colonnes)?
What was the original layout if the matrix has dimensions (4 × 3) (4 rows, 3 columns)?

a) [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

b) [0, 4, 8, 1, 5, 9, 2, 6, 10, 3, 7, 11]

c) [0, 3, 6, 9, 1, 4, 7, 10, 2, 5, 8, 11]

d) [0, 8, 1, 9, 2, 10, 3, 11, 4, 5, 6, 7]

Réponse / Answer : a)
