

Devoir No. 6 et Lab. # 6

PRÉ-REQUIS

- Vous devez avoir complété le Pré-Lab 3 et lu les chapitres 2 à 5 de la référence [2]
- À faire avant de commencer le lab (Voir annexe A)
 1. Ajouter “cl.exe” aux variables d’environnement du système

PARTIE 1 - MULTIPLICATION MATRICE-MATRICE

Objectif : Écrire une multiplication matricielle $C = A \times B$. Par souci de simplicité, nous allons uniquement utiliser des matrices carrées dont les éléments sont des single-precision floating- point.

Méthodologie :

Suivez les instructions ci-dessous :

- A. Écrire une fonction “host-stub ”qui allouera la mémoire pour les matrices d’entrées et la matrice de sortie, fera le transfert de ces matrices vers le Device, lancera le kernel, fera le transfert de la matrice de sortie vers l’Host et libèrera la mémoire des matrices d’entrées et de sortie sur le Device. Laissez les configurations d’exécution vide pour cette étape.

Cette fonction a quelque peu changé depuis mon laboratoire #5 :

```
// Helper function for using CUDA to add vectors in parallel.
cudaError_t multiplyWithCuda(float* c, const float* a, const float* b, unsigned int size, kernelType type)
{
    float* dev_a = 0;
    float* dev_b = 0;
    float* dev_c = 0;
    cudaError_t cudaStatus;

    // Choose which GPU to run on, change this on a multi-GPU system.
    cudaStatus = cudaSetDevice(0);
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaSetDevice failed! Do you have a CUDA-capable GPU installed?");
        goto Error;
    }

    // Allocate GPU buffers for three vectors (two input, one output)
    cudaStatus = cudaMalloc((void**)&dev_c, (size * size) * sizeof(float));
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaMalloc failed!");
        goto Error;
    }

    cudaStatus = cudaMalloc((void**)&dev_a, (size * size) * sizeof(float));
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaMalloc failed!");
        goto Error;
    }

    cudaStatus = cudaMalloc((void**)&dev_b, (size * size) * sizeof(float));
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaMalloc failed!");
        goto Error;
    }

    // Copy input vectors from host memory to GPU buffers.
    cudaStatus = cudaMemcpy(dev_a, a, (size * size) * sizeof(float), cudaMemcpyHostToDevice);
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaMemcpy failed!");
        goto Error;
    }

    cudaStatus = cudaMemcpy(dev_b, b, (size * size) * sizeof(float), cudaMemcpyHostToDevice);
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaMemcpy failed!");
        goto Error;
    }
}
```

```
// Launch a kernel on the GPU
if (size == 4) {
    dim3 threadsParBlock(size, size, 1);
    dim3 nombreDeBlock(1, 1, 1);
    switch (type) {
        case KERNEL_ELEMENT:
            multiplyKernelElement << <nombreDeBlock, threadsParBlock >> > (dev_c, dev_a, dev_b, size);
            break;
        case KERNEL_ROW:
            multiplyKernelRow << <nombreDeBlock, threadsParBlock >> > (dev_c, dev_a, dev_b, size);
            break;
        case KERNEL_COLUMN:
            multiplyKernelColumn << <nombreDeBlock, threadsParBlock >> > (dev_c, dev_a, dev_b, size);
            break;
        default:
            break;
    }
}

else if (size >= minMatrixWidth && size <= maxMatrixWidth) {
    dim3 threadsParBlock(size, size, 1);
    dim3 nombreDeBlock((size / blockWidth), (size / blockWidth), 1);

    switch (type) {
        case KERNEL_ELEMENT:
            multiplyKernelElement << <nombreDeBlock, threadsParBlock >> > (dev_c, dev_a, dev_b, size);
            break;
        case KERNEL_ROW:
            multiplyKernelRow << <nombreDeBlock, threadsParBlock >> > (dev_c, dev_a, dev_b, size);
            break;
        case KERNEL_COLUMN:
            multiplyKernelColumn << <nombreDeBlock, threadsParBlock >> > (dev_c, dev_a, dev_b, size);
            break;
        default:
            break;
    }
}

else {
    printf("Undefined behaviour. Please use matrix size between 8 and 64\n");
    return cudaErrorInvalidValue;
}

else {
    printf("Undefined behaviour. Please use matrix size between 8 and 64\n");
    return cudaErrorInvalidValue;
}

// Check for any errors launching the kernel
cudaStatus = cudaGetLastError();
if (cudaStatus != cudaSuccess) {
    fprintf(stderr, "addKernel launch failed: %s\n", cudaGetErrorString(cudaStatus));
    goto Error;
}

// cudaDeviceSynchronize waits for the kernel to finish, and returns
// any errors encountered during the launch.
cudaStatus = cudaDeviceSynchronize();
if (cudaStatus != cudaSuccess) {
    fprintf(stderr, "cudaDeviceSynchronize returned error code %d after launching addKernel!\n", cudaStatus);
    goto Error;
}

// Copy output vector from GPU buffer to host memory.
cudaStatus = cudaMemcpy(c, dev_c, (size * size) * sizeof(float), cudaMemcpyDeviceToHost);
if (cudaStatus != cudaSuccess) {
    fprintf(stderr, "cudaMemcpy failed!");
    goto Error;
}

Error:
    cudaFree(dev_c);
    cudaFree(dev_a);
    cudaFree(dev_b);

    return cudaStatus;
}
```

Comme vous pouvez constater, dépendamment de la grosseur de la matrice entrée, le kernell sera calculé différemment. Dans ce programme, je peux faire la multiplicaton matricielle sur des matrices de 4*4, et tout autre matrice carré qui ont une dimension entre 8 et 64. Si une taille plus grande est rentré, un message est écrit pour indiquer que ce calcul de matrice n'est pas pris en compe dans ce programme.

Pour rendre ce code plus agréable à lire, j'ai créé un enum au début de mon programme pour indiquer les différents type de calcul il est possible de faire avec les threads.

```
typedef enum {  
    KERNEL_ELEMENT = 0,  
    KERNEL_ROW,  
    KERNEL_COLUMN  
} kernelType;
```

Lorsqu'on appelle la fonction multiplyWithCuda, il est donc important de spécifier quel type d'opération qu'on veut faire tel qu'énoncé dans le cahier dans le laboratoire.

J'ai aussi créé des variables constantes nous permettant de rapidement changer différentes fonctionnalités du programme :

```
const int blockDim = 8;  
const int minMatrixWidth = 8;  
const int maxMatrixWidth = 64;
```

J'ai aussi corrigé l'allocation de la mémoire pour correspondre au float.

- B. Écrivez un kernel où chaque thread calculera un seul élément de la matrice de sortie. Ajouter les paramètres d'exécution dans la fonction « Host-Stub » nécessaire à ce kernel.

```
_global__ void multiplyKernelElement(float* c, const float* a, const float* b, int length)
{
    int row = (blockIdx.y * blockDim.y) + threadIdx.y;
    int column = (blockIdx.x * blockDim.x) + threadIdx.x;
    int indexmatC = (row * length) + column;

    int linearIndexRow;
    int linearIndexColumn;

    float sum=0;
    //Cette condition permet de rester dans les limites
    if (row < length && column < length){
        for (int index = 0; index < length; index++) {
            linearIndexRow = (row * length) + index;
            linearIndexColumn = (index * length) + column;
            sum += a[linearIndexRow] * b[linearIndexColumn];
        }
        c[indexmatC] = sum;
    }
}
```

Cette fonction permet de calculer un élément de la matrice de sortie. Contrairement au devoir 5, chaque thread doit quand même faire une loop car la matrice de sortie est une somme de toutes les multiplications rangées par colonnes de chaque élément des matrices A et B.

- C. Écrivez un kernel où chaque thread calculera tous les éléments d'une ligne de la matrice de sortie. Ajouter les paramètres d'exécution dans la fonction « Host-Stub » nécessaire à ce kernel.

```
_global__ void multiplyKernelRow(float* c, const float* a, const float* b, int length)
{
    int row = (blockIdx.y * blockDim.y) + threadIdx.y;
    int indexMatC = (row * length);

    int linearIndexRow;
    int linearIndexColumn;

    //Cette condition permet de rester dans les limites
    if (row < length) {
        for (int column = 0; column < length; column++) {
            float sum = 0;
            for (int index = 0; index < length; index++) {
                linearIndexRow = (row * length) + index;
                linearIndexColumn = (index * length) + column;
                sum += a[linearIndexRow] * b[linearIndexColumn];
            }
            c[(indexMatC + column)] = sum;
        }
    }
}
```

Pour faire ceci, il est inévitable de faire une loop imbriquée dans une autre loop, car on doit calculer tous les éléments de la matrice de sortie. En tant que tel, le calcul reste très similaire que la question précédente dans la loop imbriquée, sauf que la première loop va garder la colonne respectif de la matrice de sortie, car on doit calculer une rangée complète dans ce thread.

- D. Écrivez un kernel où chaque thread calculera tous les éléments d'une colonne de la matrice de sortie. Ajouter les paramètres d'exécution dans la fonction « Host-Stub » nécessaire à ce kernel.

```
_global__ void multiplyKernelColumn(float* c, const float* a, const float* b, int length)
{
    int column = (blockIdx.x * blockDim.x) + threadIdx.x;
    int linearIndexRow;
    int linearIndexColumn;

    //Cette condition permet de rester dans les limites
    if (column < length) {
        for (int row = 0; row < length; row++) {
            float sum = 0;
            for (int index = 0; index < length; index++) {
                linearIndexRow = (row * length) + index;
                linearIndexColumn = (index * length) + column;
                sum += a[linearIndexRow] * b[linearIndexColumn];
            }
            c[(row*length)+column] = sum;
        }
    }
}
```

Cette fois-ci, l'implémentation est très similaire que la question précédente, sauf qu'il faut calculer une colonne pour la matrice de sortie. Pour faire ceci, mathématiquement il faut calculer chaque rangée de la matrice A pour une colonne de la matrice B, et ceci donnera une colonne de la matrice C.

E. Analysez les avantages et inconvénients de chaque kernel ci-dessus.

Comme il est possible de voir dans les captures d'écran plus bas, faire un calcul élément par élément est énormément plus rapide que le calcul colonne ou rangée.

Pour A à E,

A. Validez que le kernel est fonctionnel et effectue le bon calcul en utilisant un simple programme en CUDA C où :

a. Les matrices doivent avoir une taille variable 4x4, 8x8, 32x32 et 64x64.

Matrice 4*4

Élément :

```
// Add vectors in parallel.
cudaError_t cudaStatus = multiplyWithCuda(c, a, b, arraySize, KERNEL_ELEMENT);

Matrix A:
{ 0.01 2.25 0.77 3.23 }
{ 2.34 1.92 1.40 3.58 }
{ 3.29 2.99 0.70 3.44 }
{ 2.84 2.05 1.22 0.06 }

Matrix B:
{ 0.37 1.46 0.59 0.66 }
{ 3.95 1.78 0.48 0.02 }
{ 0.04 1.51 2.13 2.28 }
{ 2.41 2.43 0.66 2.65 }

Matrix C:
{ 16.73 13.05 4.87 10.39 }
{ 17.12 17.66 7.66 14.30 }
{ 21.31 19.52 7.13 12.94 }
{ 9.35 9.79 5.28 4.86 }
```

Rangée :

```
// Add vectors in parallel.
cudaError_t cudaStatus = multiplyWithCuda(c, a, b, arraySize, KERNEL_ROW);
```



```
Matrix A:
{ 0.01 2.25 0.77 3.23 }
{ 2.34 1.92 1.40 3.58 }
{ 3.29 2.99 0.70 3.44 }
{ 2.84 2.05 1.22 0.06 }

Matrix B:
{ 0.37 1.46 0.59 0.66 }
{ 3.95 1.78 0.48 0.02 }
{ 0.04 1.51 2.13 2.28 }
{ 2.41 2.43 0.66 2.65 }

Matrix C:
{ 16.73 13.05 4.87 10.39 }
{ 17.12 17.66 7.66 14.30 }
{ 21.31 19.52 7.13 12.94 }
{ 9.35 9.79 5.28 4.86 }
```

Colonne :

```
// Add vectors in parallel.
cudaError_t cudaStatus = multiplyWithCuda(c, a, b, arraySize, KERNEL_COLUMN);
```

```
Matrix A:
{ 0.01 2.25 0.77 3.23 }
{ 2.34 1.92 1.40 3.58 }
{ 3.29 2.99 0.70 3.44 }
{ 2.84 2.05 1.22 0.06 }

Matrix B:
{ 0.37 1.46 0.59 0.66 }
{ 3.95 1.78 0.48 0.02 }
{ 0.04 1.51 2.13 2.28 }
{ 2.41 2.43 0.66 2.65 }

Matrix C:
{ 16.73 13.05 4.87 10.39 }
{ 17.12 17.66 7.66 14.30 }
{ 21.31 19.52 7.13 12.94 }
{ 9.35 9.79 5.28 4.86 }
```

Matrice 32*32

Élément :

```
// Add vectors in parallel.  
cudaError_t cudaStatus = multiplyWithCuda(c, a, b, arraySize, KERNEL_ELEMENT);
```

Matrice A :

```
Matrix A:  
0.01 2.25 0.77 2.33 2.34 1.92 1.40 3.58 3.29 2.99 0.70 3.44 2.84 2.05 1.22 0.06 0.37 1.46 0.59 0.66 3.95 1.78 0.48 0.02 0.04 1.51 2.13 2.28 2.41 2.43 0.66 2.65  
1.80 1.41 0.23 2.43 3.13 3.21 2.08 1.21 3.50 2.91 3.82 3.70 2.16 0.57 1.85 0.94 3.45 0.84 3.12 3.37 3.99 4.00 2.45 1.57 1.06 1.19 3.36 0.09 1.50 0.37 2.71 0.22  
0.04 3.68 1.10 1.09 2.35 2.76 3.35 2.91 1.94 0.82 2.97 1.87 1.83 3.80 2.98 0.43 2.40 1.54 2.94 2.44 2.29 1.45 0.61 0.90 1.70 3.21 2.07 3.96 3.81 1.38 0.68 2.63  
1.97 0.25 2.80 2.02 0.59 3.80 0.57 3.62 2.77 1.21 1.71 0.28 3.87 2.73 0.61 3.51 3.29 2.33 0.77 0.71 3.27 1.90 0.62 2.02 2.93 1.62 1.12 2.27 2.73 3.02 2.89 1.90  
0.49 1.47 3.34 0.14 2.07 2.65 1.70 0.42 3.80 3.69 2.20 1.38 1.89 1.50 3.39 1.27 1.82 1.80 3.93 1.19 2.96 2.27 0.78 3.05 3.36 1.59 2.00 3.56 0.11 3.98 2.29 0.20  
2.13 0.78 3.37 2.51 2.63 0.79 3.37 0.49 0.44 2.29 1.67 1.26 3.76 1.14 1.35 0.56 2.93 2.34 2.83 2.40 2.99 1.01 0.58 0.01 0.24 3.22 3.41 0.84 0.46 2.21 0.06 0.46 1.82  
0.01 2.74 1.27 0.30 1.75 0.81 2.78 1.16 1.75 0.93 2.31 2.13 2.51 0.64 2.62 3.85 2.78 3.70 0.76 1.34 0.71 3.98 1.83 3.99 0.39 2.50 0.38 1.75 3.73 0.19 1.38 1.58 1.16  
3.91 3.08 1.64 0.81 2.51 2.42 1.81 1.87 2.39 2.54 3.42 3.32 2.50 2.88 2.26 1.50 0.74 2.95 2.22 3.62 0.97 0.76 2.42 2.79 2.34 1.41 1.98 0.32 2.96 2.45 2.48 2.76  
2.22 0.60 2.30 3.47 3.65 2.46 2.91 0.17 2.67 3.91 1.26 2.28 1.22 0.70 4.43 3.48 3.40 2.98 0.62 1.31 0.32 0.31 2.56 3.28 2.18 1.79 1.64 1.19 1.86 2.00 0.61 1.29  
1.95 1.26 3.31 3.84 3.49 2.90 1.20 3.78 0.51 0.26 3.14 2.10 2.44 3.82 0.29 3.50 2.62 1.29 0.42 2.02 0.91 1.16 3.68 2.20 2.65 0.46 1.97 1.52 1.99 3.17 0.24 1.53  
2.75 2.13 2.43 1.58 0.02 2.83 0.40 2.49 3.45 1.97 2.99 1.99 1.52 3.14 2.21 4.34 3.82 2.52 0.71 1.50 0.53 2.97 3.81 2.45 0.11 1.32 2.22 0.56 0.53 3.39 3.46 2.39  
2.89 3.42 0.06 0.51 2.83 2.24 0.87 0.26 0.68 2.50 1.36 1.28 1.47 2.64 3.21 3.23 2.11 2.44 3.19 3.60 0.58 2.52 1.61 1.01 0.55 3.42 0.26 1.71 2.29 1.21 2.19 0.90  
1.25 0.44 3.24 0.54 1.10 3.15 3.58 3.16 2.98 2.46 1.44 3.43 0.91 1.45 0.21 0.90 2.22 3.33 1.21 1.71 0.34 0.22 3.04 1.04 0.57 0.93 0.50 0.50 3.73 0.32 0.19 0.23  
1.35 0.66 1.50 0.73 3.78 3.35 2.14 2.37 2.77 1.19 0.84 0.02 2.10 3.82 0.59 0.96 2.34 1.02 2.74 3.78 1.74 3.56 0.03 3.76 2.44 2.34 0.57 0.89 1.33 0.02 1.67  
0.33 2.64 3.42 0.26 2.34 2.65 2.77 3.21 1.12 1.74 0.57 2.76 0.91 1.11 0.12 3.47 2.58 2.83 3.44 2.21 3.70 0.24 1.10 0.58 3.93 2.48 1.17 3.69 1.47 2.78 0.87 0.62  
0.96 2.09 3.61 0.43 3.61 1.77 0.32 3.13 0.69 3.00 3.10 3.48 0.84 1.83 0.02 3.00 0.46 1.62 1.24 3.97 0.15 1.01 0.76 0.99 0.61 2.48 3.54 3.37 0.83 3.12 2.58 0.63  
3.64 1.82 3.60 2.66 0.60 2.55 2.28 1.69 3.77 2.16 3.21 2.15 0.82 1.50 1.40 0.15 3.18 0.79 0.28 1.56 2.36 0.28 0.70 0.62 2.58 1.82 2.42 2.70 1.77 2.74 1.59 3.34  
2.37 0.80 3.80 3.50 1.57 0.95 0.74 3.58 2.30 1.77 2.51 2.83 0.28 1.14 0.14 1.63 3.58 2.84 2.91 3.58 1.57 1.59 3.62 1.23 1.55 2.28 1.41 1.93 2.98 2.95 0.26 0.56  
0.80 1.89 2.72 3.64 1.47 2.07 0.44 3.63 0.81 2.07 2.62 1.75 2.75 0.36 0.32 3.00 0.11 1.42 1.23 2.79 0.57 1.58 0.27 2.70 2.90 0.79 2.78 2.46 2.62 1.18 0.39 2.18  
0.20 3.90 1.86 3.88 0.91 2.12 2.72 2.77 1.49 1.55 2.08 3.24 0.68 0.24 1.24 3.14 0.85 1.29 1.94 0.84 1.76 2.03 2.42 3.03 1.60 1.40 2.93 3.77 2.49 3.99 0.21 2.25  
3.45 1.55 1.23 1.14 0.14 2.51 1.56 0.07 0.12 1.82 3.43 3.75 0.61 2.91 2.45 3.23 3.50 2.33 3.28 0.32 3.17 1.85 2.36 2.36 2.28 3.26 2.69 3.37 3.04 1.07 1.37 0.93  
1.08 1.49 3.48 0.47 2.90 2.52 3.58 0.96 0.73 2.14 3.61 3.12 3.86 0.79 3.61 3.19 3.81 3.40 3.56 2.51 0.74 2.47 1.06 1.00 2.93 3.15 0.68 2.86 1.92 0.33 3.23 3.42 0.50  
3.23 2.29 1.89 3.05 0.77 2.91 3.98 0.06 2.59 2.77 0.70 2.29 1.10 2.71 3.36 2.23 1.48 3.12 0.93 2.55 0.93 0.82 3.88 1.13 0.09 3.08 0.61 1.29 0.04 1.30 2.58 1.37  
1.66 1.67 0.36 2.28 1.80 3.25 2.63 3.50 2.14 2.28 0.22 1.86 1.04 1.71 0.45 1.95 3.05 2.26 2.84 3.31 3.46 3.40 1.17 3.46 2.58 2.99 3.31 3.06 3.49 3.63 3.65 2.09  
1.76 0.43 1.42 3.29 1.71 0.68 0.00 2.63 3.09 0.25 0.36 1.54 2.74 3.23 0.62 0.57 0.19 1.38 0.87 3.91 2.50 0.21 3.66 2.55 1.73 0.08 0.97 1.57 2.62 3.13 0.09 3.30  
0.55 2.54 2.20 0.93 1.85 0.86 1.20 3.13 1.37 1.92 0.43 0.41 3.94 3.02 0.60 1.83 2.85 3.99 3.10 2.57 2.48 3.78 2.75 1.16 2.27 1.16 2.19 2.85 0.99 2.19 1.97 3.81  
2.68 3.06 2.49 1.78 3.20 2.98 3.96 0.82 2.52 3.64 2.37 3.67 0.02 1.96 1.80 3.56 3.76 0.73 1.01 0.81 2.92 3.76 2.50 0.17 2.57 0.14 2.23 1.28 1.79 0.25 3.98 0.51  
0.40 0.20 0.69 2.64 1.33 1.26 2.61 1.64 1.71 0.89 2.71 0.67 1.99 0.23 3.21 1.74 0.02 1.36 1.50 0.51 3.31 2.93 3.56 2.57 2.60 2.06 2.94 2.95 3.83 1.86 1.45 2.32  
3.62 3.28 1.82 0.43 3.48 3.05 0.16 1.67 3.74 0.27 0.66 0.85 0.47 2.36 0.47 2.32 2.76 2.93 3.66 2.68 1.19 3.24 1.51 3.99 3.09 0.82 0.68 1.55 1.72 0.25 0.78  
3.84 0.76 1.53 1.22 0.48 0.79 1.88 3.08 3.04 3.52 0.01 1.79 1.71 3.21 1.85 1.98 0.00 0.95 3.24 0.37 3.69 3.58 3.43 0.24 1.96 2.53 2.99 3.02 1.02 3.64 3.17 0.80  
2.21 2.20 2.09 3.16 2.82 1.00 2.93 2.21 3.66 3.50 3.19 1.22 1.58 1.93 2.21 2.61 0.00 2.07 3.02 0.88 0.62 0.23 2.68 0.13 1.38 2.79 3.19 1.75 3.93 2.54 1.18 1.91  
3.50 0.32 0.40 0.57 0.57 3.67 1.23 1.50 3.26 0.19 3.26 0.96 3.59 1.03 1.08 0.83 0.92 3.38 3.50 3.55 3.38 2.44 0.16 0.69 0.74 1.42 1.80 0.74 3.58 2.49 2.19 3.65
```

Matrice B :

```
Matrix B:  
2.09 2.92 4.00 3.88 3.47 3.36 0.59 1.50 1.14 0.68 3.90 2.97 2.87 0.12 2.58 3.25 2.94 3.27 0.50 2.88 1.50 2.60 3.66 3.56 0.13 3.09 1.80 0.03 0.91 1.67 0.63 3.68  
2.01 0.97 3.02 3.63 0.85 0.13 3.08 0.91 2.23 3.23 3.41 1.33 2.16 3.97 3.48 2.59 0.91 3.25 1.48 1.47 0.14 2.25 1.59 0.48 1.30 3.60 0.81 3.76 0.50 3.40 3.58 1.78  
0.89 2.54 2.24 2.50 1.69 0.07 3.46 0.56 1.07 1.17 2.73 2.25 3.84 2.43 3.15 1.69 3.27 1.96 0.54 2.07 0.08 0.93 2.99 1.34 3.61 1.85 3.07 0.20 2.52 0.10 2.73 3.88  
2.00 2.83 1.06 2.29 2.59 2.39 1.05 3.38 1.84 2.99 1.41 0.06 1.91 2.51 1.62 1.53 0.11 2.10 2.39 3.54 1.73 3.73 1.53 2.13 0.04 3.90 3.46 1.75 2.80 1.33 2.41 1.99  
0.24 0.16 1.51 0.93 3.75 3.44 3.86 2.44 2.34 0.31 1.63 1.22 1.99 3.32 1.50 1.22 3.13 3.73 2.35 0.86 1.12 2.24 3.50 1.00 1.21 2.91 3.47 0.92 0.76 1.79 3.56 3.20  
2.12 3.98 0.24 2.48 1.03 0.83 0.68 0.46 0.93 1.98 0.84 0.20 0.32 1.35 1.12 2.06 1.96 1.65 3.70 2.40 1.17 1.90 1.61 3.81 2.19 3.34 3.18 3.21 2.37 3.48 1.27 3.76 1.07  
3.77 2.88 0.51 3.10 1.89 0.70 0.11 0.17 1.82 3.45 0.54 0.06 0.85 3.27 3.19 3.12 2.63 1.19 3.13 3.69 3.37 0.97 2.00 0.09 0.30 0.12 1.42 2.86 1.64 0.72 1.19 0.73  
2.98 0.54 0.23 1.04 1.44 3.23 0.64 3.60 0.53 0.12 0.73 3.27 1.00 2.28 3.39 2.16 2.66 3.48 3.73 3.78 0.42 0.61 2.23 3.20 3.58 2.09 3.49 3.29 1.55 2.20 2.69 1.80  
1.13 3.00 3.93 2.28 2.38 1.06 0.53 0.97 2.40 1.95 1.42 0.92 1.91 2.77 3.79 2.68 3.93 0.95 2.05 0.94 3.59 1.94 3.31 2.15 2.28 1.35 3.19 0.68 1.16 0.88 3.69 2.96  
3.23 2.29 2.75 1.08 0.07 0.26 2.89 3.44 1.37 0.18 1.04 0.08 3.35 2.47 2.16 2.27 3.11 2.56 2.70 3.04 3.47 0.63 1.60 3.15 0.63 1.75 0.17 2.03 1.08 2.07 0.90  
3.93 2.03 2.45 0.08 0.97 0.42 3.44 1.35 0.97 1.16 0.20 3.97 0.69 0.28 0.88 1.12 3.05 2.10 0.60 3.31 1.05 0.55 0.87 2.25 0.71 2.58 0.43 1.01 3.84 1.24 0.23 1.13 2.78  
3.49 2.27 3.47 3.23 3.18 0.90 2.54 2.48 2.38 1.27 0.78 0.92 0.93 0.17 0.96 1.02 1.63 1.10 1.09 3.60 1.52 0.91 0.03 0.21 2.42 1.42 1.44 2.67 3.90 0.89 0.58 1.39  
3.81 2.66 0.81 1.13 1.42 2.13 1.16 0.88 0.40 0.43 2.84 3.38 0.69 3.28 0.02 1.97 2.42 3.74 1.16 2.42 2.90 3.50 3.65 3.32 2.42 1.22 1.34 2.60 1.72 2.76 1.58 3.97  
0.41 1.65 0.11 2.75 1.04 0.68 0.91 0.76 3.45 3.05 0.49 1.93 0.22 0.84 3.34 2.26 3.85 2.30 3.51 1.63 1.61 1.65 2.63 3.29 3.78 1.13 0.01 2.49 0.20 2.62 1.69 3.32  
2.42 0.80 1.66 3.16 2.40 3.23 0.02 0.42 1.58 1.11 1.27 3.30 3.41 1.25 0.79 2.26 1.88 3.98 1.73 1.47 1.68 1.54 2.32 3.51 1.35 1.57 3.80 2.04 1.67 2.43 3.41 0.83  
0.60 0.83 0.35 1.27 1.80 2.99 2.52 1.33 0.32 2.46 0.57 1.12 3.30 1.76 3.98 0.88 0.43 2.08 1.06 0.93 2.31 3.08 0.42 3.88 1.48 3.62 2.09 0.89 1.21 1.91 1.24 1.66 3.06  
3.37 1.56 1.71 2.90 2.84 0.37 3.33 0.91 3.39 2.55 1.44 1.90 0.38 1.66 0.75 1.37 3.65 0.92 1.24 0.62 1.37 0.73 1.09 0.82 0.69 1.34 0.26 0.91 3.95 3.24 1.79 3.65  
0.72 1.93 1.89 2.80 0.90 3.75 1.11 1.20 3.29 3.28 2.52 3.18 0.03 1.04 1.16 3.81 3.44 2.07 2.83 1.02 3.68 1.97 0.37 3.30 0.31 1.13 1.54 0.57 2.06 1.73 0.56 3.90  
0.30 0.02 3.66 1.65 3.02 0.75 0.41 1.69 2.71 2.11 3.30 3.35 1.09 2.00 2.48 0.60 3.88 2.15 0.98 0.27 0.82 0.79 0.77 0.80 3.83 2.37 2.96 1.18 2.72 3.14 3.69 3.71  
3.38 1.04 0.42 0.25 3.62 0.59 3.57 1.44 3.83 1.53 3.33 2.74 3.55 1.52 1.73 2.70 1.07 1.93 2.11 3.21 3.18 2.99 1.95 3.31 2.24 1.12 1.39 0.22 3.65 3.05 1.47 1.13  
0.84 3.30 0.20 3.50 1.00 1.62 0.58 3.50 2.43 1.13 0.79 1.81 3.05 1.73 2.55 3.22 0.70 1.29 3.85 2.84 0.79 3.45 0.19 2.97 0.14 0.52 0.33 2.21 2.28 2.03 1.08 0.81  
1.22 0.26 1.65 0.53 1.01 3.28 2.04 1.07 3.95 3.67 0.49 1.14 2.83 1.88 1.18 3.38 2.10 3.24 2.07 0.27 3.32 2.22 0.41 2.04 3.79 2.89 1.90 3.81 3.68 2.20 1.46 3.54  
2.64 0.49 2.03 0.45 1.12 2.19 3.17 1.48 3.28 1.98 1.86 1.44 1.77 3.24 1.52 2.53 1.23 1.93 3.35 0.74 0.04 0.44 0.16 0.89 1.02 1.64 2.83 1.77 0.53 2.18 3.43 3.23  
0.87 3.70 1.25 0.57 2.43 3.04 3.82 2.15 1.55 0.05 0.95 3.54 2.37 0.61 0.07 2.14 3.67 2.61 0.90 1.62 3.86 0.39 1.60 0.02 1.34 1.31 2.50 1.00 1.16 1.23 3.59 1.91  
1.31 2.70 1.84 2.33 1.65 3.69 0.48 2.66 0.59 0.99 0.45 0.41 2.76 3.31 1.96 1.81 2.61 1.67 2.56 0.40 3.32 1.47 3.51 2.08 3.85 1.58 2.90 1.52 1.50 3.25 3.41 0.21 0.26  
2.97 1.45 0.94 3.28 3.13 1.10 2.25 0.70 0.50 3.03 1.66 1.12 0.68 2.18 0.96 1.27 0.90 2.63 2.79 2.68 2.57 0.22 1.32 3.81 1.28 1.23 0.65 1.47 1.11 3.84 0.94 1.04  
2.50 3.02 2.33 1.08 2.85 3.35 2.86 0.09 2.80 2.52 0.51 2.88 1.52 1.20 3.46 2.37 1.98 0.12 2.69 2.21 1.40 2.29 3.06 0.09 0.00 2.19 2.99 1.43 3.18 3.34 2.90 1.19  
1.15 0.62 0.41 1.56 0.39 2.67 0.65 2.76 2.68 0.95 3.55 3.03 2.28 1.06 0.45 1.07 2.93 3.91 1.95 0.86 3.20 2.45 1.39 1.93 3.99 1.49 3.79 0.88 2.52 3.00 0.58 3.72  
1.83 5.57 0.19 2.62 2.97 0.40 2.60 0.80 0.94 2.93 3.98 3.02 3.57 2.19 3.57 3.48 2.46 2.88 2.92 0.61 3.02 0.61 1.62 2.46 2.87 3.32 3.54 0.25 1.58 2.41 1.95 1.81  
3.05 2.32 1.34 2.33 1.44 0.90 0.33 0.38 3.99 0.18 1.59 0.43 1.66 1.74 1.08 1.81 0.90 1.05 2.57 3.57 3.52 1.80 1.65 0.93 3.86 2.96 1.37 2.89 2.41 2.49 3.58 2.73  
0.94 0.44 2.13 2.86 2.27 0.06 2.04 1.83 3.88 3.56 2.81 0.98 0.41 1.07 1.87 2.43 0.20 1.54 0.47 3.99 2.96 0.16 0.02 2.72 3.83 2.83 3.00 0.56 2.71 2.45 2.65 3.90  
0.28 3.63 3.45 2.51 3.56 2.30 0.61 3.59 2.30 1.05 0.70 2.50 3.34 2.68 3.43 1.79 3.93 2.64 2.84 0.23 3.26 2.99 3.90 3.34 1.26 0.47 0.82 1.17 3.79 3.37 1.00 0.86
```

Matrice C :

```
Matrix C:  
115.31 122.85 89.07 117.14 114.76 107.13 85.01 103.48 118.60 113.65 86.02 106.86 110.85 114.08 111.23 117.48 121.38 137.25 133.00 116.29 117.33 119.75 100.92 114.01 117.46 94.55 117.04 96.14 121.42 114.85 121.94 115.02 )  
145.01 125.33 93.56 123.07 153.73 123.03 127.57 116.21 157.91 145.27 99.76 136.91 128.82 120.27 134.54 138.91 152.40 156.30 134.98 130.46 136.59 128.05 116.70 123.72 130.33 121.41 143.00 109.04 155.78 138.50 157.24 150.29 )  
137.50 124.73 101.66 139.33 137.33 118.06 103.31 107.35 143.97 145.36 105.93 141.60 123.67 137.76 120.79 135.89 102.02 168.40 146.39 118.06 138.02 115.35 131.66 133.07 140.53 111.74 139.36 121.05 138.59 151.97 147.40 143.60 )  
117.70 131.23 126.97 118.97 90.21 110.35 138.83 125.60 105.08 133.36 125.50 136.56 112.10 134.66 106.12 152.45 133.65 126.10 149.87 108.18 136.62 146.27 159.53 107.06 138.83 104.53 141.93 137.78 141.21 162.83 )  
124.71 127.36 116.30 125.51 132.06 113.46 102.95 102.64 146.06 121.40 105.57 126.02 132.23 125.08 122.11 127.20 158.30 155.82 125.88 118.01 146.89 116.08 119.78 121.26 163.04 112.01 143.75 103.46 136.10 135.20 157.31 150.81 )  
118.91 110.49 97.77 117.59 127.56 107.87 96.66 96.45 114.17 109.36 93.08 114.78 100.44 114.03 111.65 110.64 136.10 110.06 149.58 111.78 122.92 98.25 114.76 100.22 114.76 95.45 93.45 100.71 95.62 126.15 114.55 112.97 122.24 )  
125.17 110.00 107.08 125.76 134.57 114.03 117.36 92.14 139.17 130.82 117.34 145.06 121.34 130.49 114.29 112.24 130.60 154.04 158.02 111.29 116.33 148.12 93.73 115.26 129.92 134.35 110.48 96.42 129.19 129.93 132.01 102.90 )  
139.36 130.76 121.47 128.48 150.65 124.34 128.31 106.99 151.
```

Rangée :

```
// Add vectors in parallel.
cudaError_t cudaStatus = multiplyWithCuda(c, a, b, arraySize, KERNEL_ROW);
```

Matrice A :

```
Matrix A:
0.01 2.25 0.77 3.23 2.34 1.92 1.40 3.58 3.29 2.99 0.70 3.44 2.84 2.05 1.22 0.06 0.37 1.46 0.59 0.66 3.95 1.78 0.48 0.02 0.04 1.51 2.13 2.28 2.41 2.43 0.66 2.65 }
1.80 1.41 0.23 2.43 3.13 3.21 2.08 1.21 3.50 2.91 3.82 3.70 2.16 0.57 1.85 0.94 3.45 0.84 3.12 3.37 3.99 4.00 2.45 1.57 1.06 1.19 3.36 0.09 1.50 0.37 2.71 0.22 }
0.04 3.68 1.10 1.09 2.35 2.76 3.35 2.91 1.94 0.82 2.97 1.87 1.83 3.80 2.98 0.43 3.24 1.54 2.94 2.44 2.29 1.45 0.61 0.90 1.70 3.21 0.07 3.96 3.01 1.38 0.68 2.63 }
1.97 2.05 2.89 2.02 0.59 3.80 0.57 3.62 2.77 1.21 1.71 0.28 3.87 2.73 0.61 3.51 3.29 2.03 3.77 0.71 3.27 1.90 0.62 2.02 2.93 1.62 1.12 2.27 2.73 0.82 2.89 1.90 }
0.49 1.47 3.34 0.14 2.07 2.65 1.70 0.42 3.80 3.69 2.20 1.38 1.89 1.50 3.39 1.27 1.82 2.09 3.93 1.19 2.96 2.27 0.78 3.05 3.36 1.59 2.00 3.56 0.11 3.98 2.29 0.20 }
2.13 0.78 3.37 2.51 2.63 0.79 3.37 0.49 0.44 2.97 1.26 3.76 1.14 1.35 0.56 2.93 3.34 2.83 2.40 2.99 1.01 0.58 0.01 0.24 3.22 3.41 0.84 0.46 2.21 0.06 0.46 1.82 }
3.01 2.74 2.17 0.30 1.75 0.81 2.78 1.16 1.75 0.93 2.31 2.13 2.51 0.64 2.02 3.85 2.78 3.70 0.76 1.34 0.71 3.98 1.83 3.99 0.39 2.50 0.38 1.75 3.73 0.19 3.58 1.16 }
0.91 3.64 0.81 2.51 2.42 1.81 1.07 2.39 2.54 3.49 3.32 2.50 2.88 2.26 1.50 0.74 2.95 2.22 3.62 0.97 0.76 2.42 2.79 2.34 1.41 1.98 0.32 2.96 2.45 2.48 2.76 }
3.02 0.68 2.30 2.47 3.65 2.46 2.01 0.17 2.76 3.91 1.26 2.28 1.22 0.70 0.43 3.40 2.40 2.98 0.62 1.21 0.36 0.20 2.18 1.70 1.64 1.10 1.86 2.00 0.61 1.29 }
0.95 1.26 3.31 3.84 3.49 2.90 1.20 3.78 0.51 0.26 1.34 2.10 2.48 2.46 3.82 0.29 3.50 2.62 1.29 0.42 2.02 0.91 1.16 3.68 2.20 2.65 0.46 1.97 1.52 1.90 3.47 2.04 1.59 }
2.75 2.13 2.43 1.58 0.02 2.83 0.48 2.49 3.45 1.97 2.90 1.99 1.52 3.14 2.21 1.43 0.82 2.56 0.71 1.58 0.53 2.97 3.81 2.45 0.11 1.32 0.22 2.56 0.53 3.39 3.46 2.30 }
2.89 3.42 0.06 0.51 2.83 2.47 0.87 0.26 0.68 2.50 1.36 1.28 1.47 2.64 3.21 1.23 2.11 2.44 3.19 3.60 0.58 2.52 1.61 1.01 0.55 3.42 0.26 1.71 2.29 1.21 2.19 0.90 }
1.25 0.44 3.23 0.54 1.14 3.15 3.58 3.16 2.98 2.46 1.44 3.43 0.91 3.45 0.92 1.00 2.41 3.94 0.22 0.33 2.10 1.71 0.38 1.04 3.57 0.93 0.59 2.50 0.73 0.32 0.19 0.23 }
1.35 3.66 1.59 1.73 3.78 3.35 2.14 3.37 2.77 1.59 1.04 0.02 2.10 3.82 1.59 0.96 3.24 1.02 2.74 3.78 1.74 3.56 0.03 3.76 2.41 3.14 2.31 0.57 0.89 1.53 0.02 1.67 }
0.33 2.64 3.42 0.26 3.24 2.65 2.77 3.21 2.12 2.74 0.57 2.76 2.91 3.11 0.12 3.47 2.58 2.83 0.34 2.21 3.79 0.24 1.10 0.58 3.93 2.48 1.17 3.69 1.47 2.78 0.87 0.62 }
0.96 2.09 3.61 0.43 3.61 1.77 0.32 1.33 0.69 3.90 3.10 3.48 0.84 1.83 0.02 3.60 0.46 1.62 1.24 3.97 0.15 1.01 0.76 0.99 0.61 2.48 3.54 3.76 0.78 3.12 2.58 2.63 }
3.64 1.82 0.69 2.66 0.60 2.55 2.28 1.69 3.77 2.16 2.31 2.15 1.02 1.59 1.40 0.15 3.18 0.79 0.28 1.56 2.36 0.28 0.79 0.62 2.58 1.82 2.42 2.79 1.77 2.74 1.59 3.34 }
2.37 0.80 3.80 3.50 1.57 3.95 0.74 3.58 2.30 1.77 2.51 2.83 2.20 1.14 1.04 1.63 3.58 2.84 2.91 3.58 1.57 1.59 3.62 1.23 1.55 2.28 1.41 2.93 2.98 2.95 2.96 0.56 }
0.30 1.09 2.72 3.64 1.47 2.07 0.44 3.63 0.81 2.07 2.62 1.75 2.75 0.36 0.32 0.30 0.11 1.42 1.23 2.79 0.57 1.58 0.27 2.70 2.90 0.79 2.78 2.46 2.62 2.18 0.39 2.18 }
0.20 3.90 1.86 3.88 2.91 2.12 1.22 2.77 1.49 1.55 2.00 3.24 0.68 0.24 1.24 3.14 0.85 1.29 1.94 0.04 1.76 2.03 2.42 3.03 1.60 1.40 3.37 3.77 2.49 3.99 0.21 2.25 }
3.45 1.55 1.23 1.14 0.01 2.45 1.56 0.07 0.12 1.82 3.43 3.75 1.61 2.91 2.45 3.23 3.50 2.33 3.38 0.32 3.17 1.85 2.36 2.36 2.28 3.26 2.69 3.37 0.04 1.07 1.37 0.93 }
3.08 1.49 3.48 0.47 2.90 5.22 3.58 0.07 2.73 2.14 3.61 3.12 3.86 0.79 3.61 3.19 3.81 3.40 3.56 2.51 0.74 2.47 1.06 1.00 2.93 3.15 0.68 2.86 1.92 3.03 3.23 3.42 0.50 }
1.23 2.29 1.89 3.05 0.77 2.91 3.98 0.06 2.59 2.77 0.70 0.29 1.10 2.71 3.36 2.23 1.48 3.12 0.93 2.55 0.93 0.82 3.88 1.13 0.09 3.08 0.61 0.19 0.04 1.39 2.58 1.37 }
1.66 1.67 0.36 2.18 2.80 2.53 2.63 3.50 2.14 2.28 0.22 1.86 1.04 1.71 3.45 1.95 3.05 2.26 2.84 3.11 3.46 3.40 1.17 3.46 2.58 2.99 3.31 3.06 3.49 3.63 3.65 0.29 }
1.76 0.43 1.42 3.29 1.71 0.68 0.00 2.63 3.09 0.25 0.36 1.54 2.74 3.23 0.62 0.57 0.19 1.38 0.07 3.91 2.59 0.21 3.66 2.55 1.73 0.08 0.97 1.57 2.62 3.13 0.09 3.30 }
0.55 2.54 2.20 3.93 1.85 0.86 1.20 3.13 1.37 1.92 2.43 0.41 3.94 3.02 0.60 1.83 2.85 3.99 3.10 2.57 2.48 3.78 2.75 1.16 2.27 1.16 2.19 2.85 0.99 2.19 1.97 3.81 }
2.68 0.36 2.49 1.78 2.20 2.98 3.96 0.82 2.52 3.64 2.37 3.67 0.02 1.96 1.80 3.66 3.76 2.73 1.01 0.81 2.92 3.76 2.50 0.17 2.57 1.04 2.23 2.18 1.79 0.25 3.98 0.51 }
2.40 0.20 0.69 2.64 1.33 1.26 0.61 1.64 1.71 0.89 2.71 0.67 1.99 2.03 3.21 1.74 0.02 1.36 1.50 0.51 3.31 2.93 3.56 2.57 2.60 2.06 2.94 2.95 3.83 1.86 1.45 0.45 }
3.62 3.28 1.82 0.43 3.48 3.05 0.26 1.67 3.74 0.27 0.66 0.85 0.47 2.36 0.47 2.47 2.32 2.76 2.93 3.66 2.68 1.19 3.24 1.51 3.99 3.09 0.82 0.68 1.55 0.72 3.32 0.78 }
3.84 0.76 1.53 1.22 0.48 0.79 1.38 3.00 3.04 3.52 0.01 1.79 1.71 3.21 1.85 1.98 3.00 0.95 2.24 0.37 3.69 3.58 3.43 0.24 1.96 2.53 2.99 3.02 1.02 3.64 3.17 0.80 }
2.21 0.20 2.09 3.16 2.82 0.10 2.93 2.21 3.66 3.50 3.19 1.22 1.58 1.93 2.11 2.61 0.00 0.97 3.02 0.88 0.62 0.23 2.68 0.13 1.38 2.79 3.19 1.75 3.93 2.54 1.18 1.91 }
3.50 0.32 0.40 0.57 0.57 3.67 1.23 1.50 3.26 0.19 3.26 0.96 3.59 1.03 1.08 0.83 0.92 3.38 3.50 3.55 3.38 2.44 0.16 0.69 0.74 1.42 1.80 0.74 3.58 2.49 2.19 3.65 }
```

Matrice B :

```
Matrix B:
2.09 2.92 4.00 3.88 3.47 3.36 0.59 1.50 1.14 0.68 3.90 2.97 2.87 0.12 2.58 3.25 2.94 3.27 0.50 2.88 1.50 2.60 3.66 3.56 0.13 3.09 1.80 0.03 0.91 1.67 0.63 3.68 }
2.01 0.97 3.02 0.63 0.85 0.13 3.08 0.91 2.23 3.23 3.41 1.33 2.16 3.97 3.48 2.59 0.91 3.25 1.48 1.47 0.14 2.25 1.59 0.48 1.30 3.60 0.81 3.76 0.50 3.40 3.58 1.78 }
0.89 2.54 2.24 2.50 1.69 0.07 1.17 2.73 2.25 3.84 2.43 3.15 1.69 3.27 2.96 5.04 2.07 0.08 0.03 2.99 1.34 3.61 1.85 3.07 0.20 2.52 1.00 2.73 3.78 1.88 }
2.00 2.83 1.06 3.29 2.59 2.39 1.05 3.38 1.84 2.99 1.41 0.06 1.91 3.51 1.62 1.53 0.11 1.16 2.39 3.54 1.73 3.73 1.53 2.13 0.04 3.90 3.46 1.75 2.80 1.33 2.41 1.99 }
2.24 0.16 1.51 0.93 3.75 3.44 0.86 2.44 2.34 0.31 1.63 1.22 1.99 3.32 1.62 1.23 3.13 2.35 0.86 1.12 2.24 3.50 1.00 1.21 2.91 3.47 0.92 0.76 1.79 3.56 3.20 }
0.12 3.98 0.24 2.48 1.03 0.68 0.46 0.93 1.98 3.84 0.20 0.32 1.35 1.12 2.06 1.96 1.65 3.70 2.40 1.17 1.90 1.61 3.81 2.19 3.34 3.18 2.31 2.37 3.48 1.27 3.76 1.07 }
3.77 2.88 0.51 3.10 1.89 0.70 0.11 1.07 1.82 3.45 0.54 0.66 0.85 3.27 3.19 3.12 2.63 1.19 3.13 3.69 3.37 0.97 2.00 0.09 0.30 0.12 1.42 2.86 1.64 0.72 1.19 1.93 }
2.98 0.54 0.23 2.10 1.44 2.33 0.64 3.60 0.53 0.12 0.73 3.27 1.00 2.28 3.19 1.56 2.66 3.48 3.03 3.78 0.42 0.61 2.23 3.20 3.58 0.29 3.49 3.29 1.55 2.20 2.69 1.80 }
1.13 3.80 3.93 2.91 2.38 1.09 0.53 0.79 2.40 1.95 1.42 0.92 1.91 2.77 3.79 2.68 3.93 0.95 2.05 0.94 3.59 1.94 1.31 2.15 2.28 1.35 3.19 0.68 1.16 0.88 3.69 2.96 }
3.23 2.60 2.75 1.08 3.07 0.26 2.89 3.44 1.37 0.18 1.94 0.08 3.35 2.47 2.16 2.42 2.27 3.11 2.56 2.79 3.04 3.47 0.63 1.69 3.15 0.09 1.75 0.17 2.03 1.08 0.27 0.98 }
3.93 0.23 2.46 0.28 1.47 3.82 3.44 1.35 0.97 1.16 0.20 3.97 0.69 2.08 0.88 1.20 1.35 2.60 0.33 1.05 0.55 0.87 2.25 0.71 2.53 0.43 1.81 3.04 1.34 2.03 2.13 1.28 }
4.49 2.27 3.47 0.32 3.18 4.00 2.54 2.48 2.38 3.27 0.78 3.92 0.93 1.07 0.96 1.02 1.63 1.38 1.09 3.60 1.52 2.91 0.83 0.21 2.42 1.44 2.67 3.90 0.89 3.58 1.39 }
3.81 2.66 0.81 1.13 1.42 2.13 1.16 0.88 0.40 0.43 2.84 3.38 2.69 3.28 0.02 1.97 2.42 3.74 1.16 2.42 2.99 3.50 3.65 3.32 2.42 1.22 3.14 0.60 2.17 2.76 1.58 3.97 }
4.41 1.65 0.11 2.75 1.04 3.68 0.91 0.76 3.45 3.05 1.19 1.93 0.22 3.84 2.26 3.85 2.30 3.51 1.63 1.61 1.65 2.63 3.29 3.78 1.13 0.01 2.47 0.20 2.62 1.69 3.82 }
2.42 0.80 1.66 1.36 2.49 3.23 0.02 0.42 1.58 1.11 1.27 3.30 3.41 1.25 0.79 2.26 1.88 3.98 1.73 1.47 1.68 1.54 2.32 3.51 1.35 1.57 3.80 0.40 1.67 2.43 3.11 0.33 }
0.60 0.83 0.35 1.27 1.89 2.52 3.33 0.32 2.46 0.57 1.12 3.30 1.76 3.98 0.88 0.43 2.08 1.06 0.03 2.31 3.08 0.43 3.88 1.48 3.62 2.09 0.89 1.21 1.91 1.24 1.66 3.06 }
3.37 1.56 1.71 2.90 2.84 0.39 3.37 3.01 3.39 2.55 1.44 1.90 0.38 1.66 0.75 1.37 3.65 0.92 1.24 0.62 1.37 0.73 1.09 0.82 0.69 1.34 0.26 2.01 3.85 3.24 1.79 3.65 }
0.72 1.93 1.89 2.80 0.99 3.75 1.11 1.20 3.29 3.28 2.52 3.18 0.03 1.04 1.16 3.81 3.44 2.07 2.83 1.02 3.68 1.97 0.37 3.30 3.31 1.13 1.54 2.57 2.06 1.73 0.56 3.90 }
0.30 0.02 3.66 1.65 3.02 0.75 0.41 1.69 2.71 2.11 3.30 3.35 1.09 2.00 2.48 0.60 3.88 2.15 0.88 0.27 0.82 0.79 0.77 0.80 3.83 2.37 2.96 1.18 2.72 3.14 3.69 3.71 }
3.38 1.04 0.42 2.55 3.62 0.59 3.57 1.44 3.83 1.53 1.33 2.74 3.55 1.52 1.73 2.70 1.07 1.93 2.11 3.21 3.18 2.99 1.95 3.31 2.24 1.12 1.39 0.22 3.65 3.05 1.47 1.13 }
0.30 0.20 3.20 3.53 1.00 1.62 0.58 3.50 2.43 3.13 0.79 1.81 0.35 1.05 1.73 2.55 3.22 0.70 1.29 3.85 2.84 0.79 3.45 0.19 2.97 0.14 0.52 0.33 2.21 2.28 2.13 0.88 0.31 }
1.22 0.26 1.65 0.53 0.01 3.28 2.04 1.07 3.95 3.67 0.49 1.14 2.83 1.88 1.18 1.38 2.10 3.24 0.97 0.27 3.32 2.22 0.41 2.04 3.79 0.89 1.90 1.81 3.68 2.20 1.46 3.54 }
2.64 0.40 2.03 0.45 1.12 2.19 3.17 1.48 3.43 2.98 1.06 1.44 1.77 3.64 1.59 2.53 2.23 1.93 3.35 0.74 0.04 0.44 0.16 0.09 1.02 1.64 2.83 1.77 0.53 2.18 3.33 3.23 }
0.87 3.70 1.25 0.57 2.45 0.04 0.82 2.15 1.11 0.05 0.95 3.54 2.37 0.60 0.07 2.14 0.67 2.61 0.90 1.62 3.86 3.90 1.60 2.02 1.24 3.11 2.50 1.60 1.21 3.53 3.50 }
1.31 2.70 0.04 2.02 1.65 0.69 0.40 0.66 0.50 0.90 0.45 0.41 2.76 3.31 1.06 0.81 2.61 1.67 2.56 0.40 2.32 2.47 3.51 2.08 3.05 3.50 0.90 1.52 1.50 3.05 0.91 }
2.07 1.45 0.94 3.28 1.33 1.10 2.25 0.70 0.50 0.60 1.66 1.12 0.68 2.18 0.96 1.27 0.90 2.68 2.02 2.68 2.57 0.22 1.32 3.81 1.20 1.23 0.65 1.47 1.11 3.84 0.04 1.06 }
2.50 0.92 2.33 1.08 2.85 1.35 2.86 0.90 2.80 2.52 0.51 2.88 1.52 1.20 3.46 2.37 1.98 1.02 1.20 2.60 2.21 1.40 2.20 3.06 0.90 0.00 2.10 1.90 1.13 3.34 2.90 1.10 }
2.15 1.62 0.41 1.56 3.09 2.67 0.65 2.76 2.68 2.95 3.55 0.33 2.08 1.06 0.45 1.07 2.93 0.91 1.95 0.86 3.20 2.45 1.30 1.03 0.90 1.40 3.79 0.88 2.52 3.30 0.58 3.72 }
1.83 2.57 0.10 2.62 0.97 2.48 0.60 0.80 0.94 2.93 3.08 0.02 3.57 2.10 3.57 3.48 2.46 2.89 2.92 0.61 3.02 0.61 1.62 2.46 2.87 3.32 3.54 0.25 1.58 2.41 1.95 1.81 }
3.05 3.21 3.24 3.23 1.44 3.00 0.03 3.98 3.90 0.18 1.59 0.43 1.66 1.74 0.08 1.81 0.90 1.05 2.57 3.57 3.52 1.80 1.65 0.93 3.86 2.96 1.37 2.79 2.80 2.41 4.90 3.58 2.73 }
0.94 0.44 2.13 2.06 2.27 0.06 2.04 1.83 3.88 3.56 2.81 0.98 0.41 1.07 1.87 2.43 0.20 1.54 0.47 3.99 2.96 0.16 0.02 2.72 3.83 2.83 3.00 0.56 2.71 2.45 2.65 3.90 }
0.28 3.63 3.45 2.51 3.56 2.30 0.61 3.59 2.30 1.05 0.70 2.50 3.34 2.68 3.43 1.79 3.93 2.64 2.84 0.23 3.26 2.99 3.90 3.34 1.26 0.47 0.82 1.17 3.79 3.37 1.00 0.86 }
```

Matrice C :

```
Matrix C:
115.31 112.85 89.07 117.14 114.76 107.13 85.01 103.48 118.60 113.65 86.02 106.86 110.85 114.08 111.23 117.48 121.38 107.29 117.33 119.75 100.92 114.01 117.46 94.55 117.04 96.14 121.42 114.85 121.94 115.02 }
145.01 125.33 113.07 153.73 123.03 117.21 157.91 145.27 99.26 136.91 128.82 120.27 134.94 138.91 152.42 156.30 134.98 136.46 136.59 128.05 116.70 123.72 130.33 121.41 140.08 109.04 155.78 138.05 157.24 150.29 }
137.50 124.73 101.66 139.33 137.03 118.06 103.31 107.35 143.97 145.36 105.93 141.60 123.07 137.76 120.79 138.99 102.02 168.40 140.30 118.06 138.02 115.35 131.66 133.07 140.03 111.74 130.30 121.05 138.59 151.07 147.46 143.60 }
117.70 135.96 144.23 126.97 118.97 90.21 110.35 136.83 125.60 105.08 133.36 125.50 136.56 112.10 134.66 151.55 152.45 133.65 162.10 149.87 108.18 136.62 146.27 159.53 116.07 138.83 104.53 141.93 137.78 141.21 162.83 }
124.71 127.31 93.30 125.51 132.06 113.46 102.95 102.64 146.06 121.40 105.57 126.02 132.23 125.08 122.11 127.20 158.79 155.82 125.88 118.01 146.89 110.66 119.78 121.26 163.04 112.81 143.75 163.46 139.16 135.20 157.31 150.81 }
110.91 110.49 97.77 117.59 37.56 102.07 96.60 96.45 114.12 100.26 93.00 114.28 100.44 114.02 111.85 110.64 156.10 120.26 109.50 111.70 122.92 98.25 114.76 100.22 114.55 92.45 100.71 151.62 126.15 114.55 112.97 122.24 }
125.17 116.00 107.00 125.76 117.04 113.07 116.32 117.34 139.17 136.57 117.34 145.06 121.34 149.11 122.17 134.09 154.00 158.02 161.29 116.33 148.12 93.73 115.26 129.92 134.38 116.21 131.48 96.42 129.19 129.93 132.01 162.90 }
139.36 136.76 121.47 128.48 150.65 124.34 126.31 106.99 151.74 129.83 109.31 148.63 132.68 135.28 132.00 140.04 164.68 167.47 139.92 129.94 149.82 118.47 135.30 136.03 154.19 118.00
```


Colonne :

```
// Add vectors in parallel.
cudaError_t cudaStatus = multiplyWithCuda(c, a, b, arraySize, KERNEL_COLUMNN);
```

Matrice A :

```
Matrix A:
0.01 2.25 0.77 3.23 2.34 1.92 1.40 3.58 3.29 2.99 0.70 3.44 2.84 2.05 1.22 0.06 0.37 1.46 0.59 0.66 3.95 1.78 0.48 0.02 0.04 1.51 2.13 2.28 2.41 2.43 0.66 2.65 )
1.80 1.41 0.23 2.43 3.13 3.21 2.08 1.21 3.50 2.91 3.82 3.70 2.16 0.57 1.85 0.94 3.45 0.84 3.12 3.37 3.99 4.00 2.45 1.57 1.06 1.19 3.36 0.89 1.50 0.37 2.71 0.22 )
0.04 3.68 1.10 1.09 2.35 2.76 3.35 2.91 1.94 0.82 2.97 1.87 1.83 3.80 2.98 0.43 2.26 1.54 2.94 2.44 2.29 1.45 0.61 0.90 1.70 3.21 2.07 3.96 3.01 1.38 0.68 2.63 )
1.97 0.25 2.80 2.02 0.55 2.63 3.57 3.62 2.77 1.21 1.71 0.28 3.07 2.73 0.61 3.51 3.29 2.32 0.77 0.71 3.27 1.90 0.62 2.02 2.93 1.62 1.12 2.27 2.73 3.62 2.89 1.90 )
0.49 1.47 3.34 2.14 2.07 0.65 1.70 0.42 3.38 1.80 1.55 3.39 1.27 0.82 1.00 2.93 1.19 2.96 2.27 0.78 3.05 3.35 2.90 2.00 3.56 0.16 1.89 2.29 0.60 2.29 0.60 )
2.13 0.78 3.37 2.51 2.63 0.79 3.36 0.49 0.88 2.97 1.26 3.75 1.14 1.35 0.56 2.93 3.34 2.80 2.40 1.99 1.01 0.58 0.01 0.24 3.22 3.41 0.84 0.46 2.21 0.66 1.82 )
3.01 2.74 2.17 0.38 1.75 0.81 2.78 1.16 1.75 0.93 2.31 2.13 2.51 0.64 2.02 3.85 2.78 3.70 0.76 1.34 0.71 3.98 1.83 0.90 0.30 2.50 0.38 1.75 3.73 0.49 3.58 1.16 )
0.91 0.88 1.64 0.81 2.51 2.42 1.81 0.87 2.39 2.54 3.42 3.32 2.50 2.88 2.26 1.50 0.74 2.95 2.22 3.62 0.97 0.76 2.42 2.79 2.34 1.41 1.98 0.32 2.96 2.45 2.48 2.76 )
3.22 0.60 2.30 3.47 3.65 2.46 2.91 0.47 2.67 3.91 1.26 2.28 1.22 0.70 3.43 3.48 3.40 2.98 0.62 1.31 0.32 0.31 2.56 3.28 2.18 1.79 1.64 1.49 1.86 2.00 0.61 1.29 )
2.95 1.26 3.31 3.84 3.49 2.98 1.20 3.78 0.51 0.26 3.14 2.10 2.44 3.82 0.29 3.50 2.62 1.29 0.42 2.02 0.91 1.16 3.68 2.28 2.65 0.46 1.97 1.52 1.99 3.17 0.24 1.53 )
2.75 2.13 2.43 1.58 0.02 2.83 0.40 2.49 3.45 1.97 2.99 1.99 1.52 3.14 2.21 1.43 3.82 2.52 0.71 1.50 0.53 2.97 3.81 2.45 0.11 1.32 2.22 2.56 0.53 3.39 3.46 2.39 )
2.89 3.42 0.06 0.51 2.83 2.47 0.87 0.26 0.68 2.50 1.36 1.28 1.47 2.64 3.21 2.23 2.11 2.44 3.19 3.60 0.58 2.52 1.61 1.01 0.55 3.42 0.26 1.71 2.29 1.21 2.19 0.90 )
1.25 0.44 3.23 0.54 1.14 3.15 3.58 3.16 2.98 2.46 1.44 3.43 0.91 3.45 0.32 1.00 2.17 3.94 0.22 0.33 2.10 1.71 0.38 1.04 3.57 0.93 0.59 0.50 3.73 0.32 1.19 0.23 )
1.35 3.66 1.59 1.73 3.78 3.35 2.14 3.37 2.77 1.59 1.04 0.02 1.10 3.82 1.59 0.96 3.24 1.02 2.74 3.78 1.74 3.56 0.03 3.76 2.41 3.14 3.21 0.57 0.89 1.53 0.02 1.67 )
3.23 2.64 3.42 0.26 2.24 2.65 2.77 3.21 2.12 2.74 0.57 2.76 2.91 3.11 0.12 3.47 2.58 2.83 3.34 2.21 3.79 0.24 1.10 0.58 3.93 2.48 1.17 3.69 1.47 2.78 0.87 0.62 )
0.96 2.09 3.61 0.43 3.61 1.77 0.32 1.33 0.69 3.90 3.10 3.48 0.84 1.83 0.02 3.00 0.46 1.62 1.24 3.97 0.15 1.01 0.76 0.99 0.61 2.48 3.54 3.76 0.78 3.12 2.58 2.63 )
3.64 1.82 3.69 2.66 0.60 2.55 2.28 1.69 3.77 2.16 2.31 2.15 1.02 1.59 1.40 0.15 3.18 0.79 2.28 1.56 2.36 0.28 0.79 0.62 2.58 1.82 2.42 2.79 1.77 2.74 1.59 3.34 )
2.37 0.80 3.80 3.50 1.57 0.95 3.54 3.58 2.30 1.77 2.51 2.83 0.20 1.14 1.04 1.63 3.58 2.84 2.91 3.58 1.57 1.59 3.62 1.23 1.55 2.28 1.41 2.93 2.98 2.95 0.96 0.56 )
0.80 1.09 2.72 3.64 1.47 2.07 0.44 3.63 0.81 2.07 2.62 1.75 2.75 0.36 3.2 0.30 0.11 1.42 1.23 2.79 0.57 1.58 0.27 2.70 2.90 0.79 2.78 4.66 2.62 2.18 0.39 1.28 )
0.20 1.90 1.86 3.88 2.91 2.12 2.72 2.77 1.49 1.55 0.20 3.24 0.68 0.24 1.24 1.34 0.15 1.29 1.94 0.04 1.76 2.03 2.42 3.03 1.60 1.40 3.37 3.77 2.49 3.99 0.21 2.25 )
4.45 1.55 1.23 1.14 0.11 2.45 1.56 0.07 1.12 1.82 3.43 3.75 1.61 2.91 3.45 3.23 3.50 2.33 2.38 0.32 3.17 1.85 2.36 2.36 2.28 3.26 0.69 3.37 0.04 1.07 1.37 0.93 )
3.08 1.49 3.48 0.47 2.90 2.52 3.58 0.73 2.14 3.61 3.12 3.86 0.79 3.61 3.19 3.81 3.40 3.56 2.51 0.74 2.47 1.06 1.00 2.93 3.15 0.68 2.86 1.92 0.33 3.23 2.42 0.50 )
1.23 2.29 1.89 3.05 0.77 2.91 3.98 0.06 2.59 2.77 0.70 0.29 1.10 2.71 3.36 2.23 1.48 3.12 0.93 2.55 0.93 0.82 3.88 1.13 0.09 3.08 0.61 0.19 0.04 1.39 2.58 1.37 )
1.66 3.67 0.36 2.18 2.80 1.35 2.63 3.50 2.14 2.28 0.22 1.86 1.04 1.71 3.45 1.95 3.05 3.26 2.84 3.11 3.46 3.40 1.17 3.46 2.58 2.99 3.31 3.06 3.49 2.63 3.65 2.09 )
1.76 0.43 1.42 3.29 1.71 0.68 0.00 2.63 3.09 0.25 0.36 1.54 2.74 3.23 0.62 0.57 0.19 1.38 0.07 3.91 2.59 0.21 3.66 2.55 1.73 0.08 0.97 1.57 2.62 3.13 0.09 3.30 )
0.55 2.54 2.20 3.93 1.85 0.86 1.20 1.33 1.37 1.92 2.43 0.41 3.94 3.02 0.60 1.83 2.85 3.99 3.10 2.57 2.48 3.78 2.75 1.16 2.27 1.16 2.19 2.85 0.99 2.19 0.97 3.81 )
2.68 3.06 2.49 1.78 3.20 1.98 3.96 0.82 2.52 3.64 2.37 3.67 2.02 1.96 1.80 3.56 3.76 0.73 1.01 0.81 2.92 3.76 2.50 0.17 2.57 0.14 2.23 2.18 1.79 0.25 3.98 0.51 )
2.40 2.09 0.69 2.64 1.33 1.26 2.61 1.64 1.71 0.89 2.71 0.67 1.99 0.20 3.21 1.74 0.02 1.36 1.50 0.51 3.31 2.93 3.56 2.57 2.60 2.06 2.94 2.95 3.83 1.86 1.45 0.45 )
3.62 3.28 1.82 0.43 3.48 0.95 0.16 1.67 3.74 0.27 0.66 0.85 0.47 2.36 0.47 2.47 2.32 3.76 3.93 3.66 2.68 1.19 3.24 1.51 3.99 3.09 0.82 0.68 1.55 0.72 2.32 0.78 )
3.84 0.76 1.53 1.22 0.48 0.79 1.08 3.08 3.04 3.52 0.01 1.79 1.71 3.21 1.85 1.98 3.00 0.95 2.24 0.37 3.69 3.58 3.43 0.24 1.96 2.53 2.99 3.02 1.02 3.64 3.17 0.80 )
2.21 0.20 2.09 3.16 2.82 0.10 2.93 2.21 3.66 3.50 1.19 1.22 1.58 1.93 0.21 2.61 0.00 3.07 3.02 0.88 0.62 0.23 2.68 0.13 1.38 2.79 3.19 1.75 3.93 2.54 1.18 1.91 )
3.50 0.32 0.40 0.57 0.57 3.67 1.23 1.50 3.26 0.19 3.26 0.96 3.59 1.03 1.08 0.83 0.92 3.38 3.06 3.55 3.38 2.44 0.16 3.68 0.94 1.42 1.80 0.74 1.78 3.58 2.49 2.19 3.65 )
```

Matrice B :

```
Matrix B:
2.09 2.92 4.00 3.88 3.47 3.36 0.59 1.50 1.14 0.68 3.90 2.97 2.87 0.12 2.58 3.25 2.94 3.27 0.50 2.88 1.50 2.60 3.66 3.56 0.13 3.09 1.80 0.03 0.91 1.67 0.63 3.68 )
2.01 0.97 3.02 3.63 0.85 0.13 3.08 0.91 2.23 3.23 3.41 1.33 2.16 3.97 3.48 2.59 0.91 3.25 1.25 1.48 1.47 0.14 2.25 1.59 0.48 1.30 3.60 0.81 3.76 0.50 4.40 3.58 1.78 )
0.89 2.54 2.24 2.50 1.69 0.07 3.46 0.56 1.07 1.17 2.73 2.25 3.84 2.43 3.45 1.65 1.69 3.27 0.96 0.54 2.07 0.08 0.03 2.99 1.34 3.61 1.85 3.07 2.20 2.52 1.01 2.73 3.88 )
2.00 2.83 1.06 3.29 2.59 2.39 1.05 3.38 1.84 2.99 1.41 0.06 1.91 2.51 1.62 1.53 0.11 1.16 2.39 3.54 1.73 3.73 1.53 2.13 0.84 3.90 3.46 1.75 2.80 1.33 2.41 1.99 )
2.24 0.16 1.51 0.93 3.75 3.44 3.86 2.44 2.34 0.31 1.63 1.22 1.99 0.32 3.50 1.22 1.33 3.73 2.35 0.86 1.12 2.24 3.50 1.00 1.21 2.91 3.47 0.92 0.76 1.79 3.56 2.30 )
0.12 3.98 0.24 2.48 1.03 0.68 0.46 0.93 1.98 3.84 0.20 0.32 1.35 1.12 2.06 1.96 1.65 3.70 2.40 1.17 1.90 1.61 3.81 2.19 3.34 1.38 3.21 3.37 3.48 1.27 3.76 1.07 )
3.77 2.88 0.51 3.18 1.89 0.70 0.11 0.17 1.82 3.45 0.54 0.06 0.85 3.27 1.39 1.12 2.63 1.19 3.13 3.69 3.37 0.97 2.00 0.09 3.00 0.12 1.42 2.86 1.64 2.72 2.19 0.73 )
2.98 0.54 0.23 2.10 1.44 3.23 0.64 3.60 0.53 0.12 0.73 3.27 1.00 2.28 1.39 2.16 2.66 3.48 3.73 3.78 0.42 0.61 2.23 3.20 3.58 0.29 3.49 3.29 1.55 2.20 6.69 1.80 )
1.13 3.80 3.93 2.91 2.38 1.09 0.53 0.79 2.40 1.95 1.42 0.92 1.91 2.77 3.79 2.68 3.93 2.95 2.05 0.94 3.59 1.94 1.31 2.15 2.28 1.35 1.39 0.68 1.16 0.88 3.69 2.96 )
3.23 2.60 2.75 1.08 3.07 0.26 2.89 3.44 1.37 1.08 1.94 0.88 3.35 2.47 2.16 2.42 2.97 3.11 0.56 0.79 3.04 3.47 0.63 1.69 3.15 0.09 1.75 0.17 2.03 1.08 0.27 0.98 )
3.93 0.23 2.46 0.28 1.47 3.82 3.44 1.35 0.97 1.16 0.20 3.97 0.69 0.28 0.88 1.20 1.35 2.60 0.33 1.05 0.55 0.87 2.25 0.71 2.53 0.43 1.81 3.04 1.34 0.23 1.13 2.78 )
3.49 2.27 3.47 0.32 3.18 4.00 2.54 2.48 2.38 2.37 0.78 3.92 0.93 0.17 0.96 1.82 0.62 1.63 1.18 1.09 3.60 1.52 2.91 0.83 0.21 2.42 1.42 1.44 2.67 3.90 0.89 3.58 1.39 )
3.81 2.66 0.81 1.13 1.42 2.13 1.16 0.88 0.40 0.43 2.84 3.28 2.69 3.28 0.02 1.97 2.42 3.74 1.16 2.42 2.99 3.50 3.65 3.32 2.42 1.22 3.14 0.60 2.21 2.76 1.58 3.97 )
0.41 1.65 0.11 2.75 1.04 0.68 0.91 0.76 3.45 3.05 0.19 1.93 0.22 3.84 3.24 2.96 3.85 2.30 3.51 1.63 1.61 1.65 2.63 3.29 3.78 1.13 0.01 2.47 0.20 2.62 1.69 3.82 )
2.42 0.80 1.66 1.36 2.49 3.23 0.02 0.42 1.58 1.11 1.27 3.30 3.41 1.25 2.79 2.26 1.88 3.98 1.73 1.47 1.68 1.54 2.32 3.51 1.35 1.57 3.80 2.04 1.67 2.43 3.11 0.33 )
0.60 0.83 0.35 1.27 1.89 2.52 1.33 0.32 2.46 0.57 1.12 3.30 1.76 3.98 0.88 0.43 2.08 1.06 0.03 2.31 3.08 0.43 3.88 1.48 3.62 2.09 0.89 1.21 1.91 1.24 1.66 3.06 )
3.37 1.56 1.71 2.90 2.94 3.37 0.33 3.01 3.39 2.55 1.44 1.90 0.38 1.66 0.75 3.37 3.65 0.92 1.24 0.62 1.37 0.73 1.09 0.82 0.69 1.34 0.26 2.01 3.85 3.24 1.79 3.65 )
0.72 1.93 1.89 2.80 0.99 3.75 1.11 1.20 3.29 3.28 2.52 1.18 0.83 1.04 1.16 3.81 3.44 2.07 2.83 1.02 3.68 1.97 3.37 3.30 0.31 1.13 1.54 2.57 2.06 1.73 0.56 3.90 )
0.30 0.42 3.66 1.31 1.92 0.79 1.64 2.71 1.33 3.30 3.35 1.04 2.00 3.48 0.20 2.00 3.48 0.20 0.88 0.27 0.02 0.41 0.77 0.84 0.80 1.83 2.37 2.90 3.36 3.47 3.71 )
3.38 1.04 0.42 0.25 3.62 0.59 3.57 1.44 3.83 1.53 1.33 2.74 3.55 1.52 1.70 2.98 1.07 1.93 2.11 3.21 3.18 2.90 1.95 3.31 2.24 1.12 1.30 0.22 3.65 3.05 1.47 1.13 )
0.84 1.30 2.0 0.33 1.00 1.62 0.58 3.50 2.43 1.33 0.79 1.81 0.85 1.75 2.32 0.70 1.29 3.85 2.84 0.79 3.45 0.19 2.97 0.14 0.52 0.33 2.21 2.28 2.13 0.88 0.31 )
1.22 0.26 1.65 0.53 1.01 2.38 2.04 1.07 3.95 0.67 0.49 1.14 2.83 1.88 1.18 1.38 1.20 3.24 2.07 0.27 3.32 2.22 0.41 2.04 3.79 2.89 1.90 1.81 3.68 2.20 1.46 3.54 )
2.64 0.49 2.03 0.45 1.12 2.19 3.17 1.48 3.43 2.98 1.86 1.44 1.77 3.34 1.52 0.53 2.23 1.93 3.35 0.74 0.84 0.44 0.16 0.89 1.02 1.64 2.83 1.77 0.53 1.48 3.43 2.23 )
0.87 3.70 1.25 0.57 4.43 0.04 3.82 2.15 1.55 0.05 0.95 3.54 2.37 0.61 0.07 2.14 3.67 2.61 1.90 1.62 3.86 0.39 1.60 0.20 1.34 1.31 2.50 1.00 1.16 2.13 3.59 1.91 )
1.31 2.70 1.84 3.23 1.65 3.60 0.48 2.66 0.50 0.90 0.45 0.41 2.76 3.31 1.96 1.81 2.61 1.67 2.56 0.40 3.32 1.47 3.51 2.08 3.85 1.58 2.90 2.52 1.50 3.25 3.41 0.26 )
2.97 1.45 0.94 3.28 3.13 1.10 2.25 0.70 0.50 0.33 1.66 1.12 0.68 2.18 0.96 1.27 0.90 2.63 2.79 2.68 2.57 0.22 1.32 3.81 1.28 1.23 0.65 1.47 1.11 3.84 0.94 1.84 )
2.50 3.02 2.33 1.08 2.85 1.35 2.86 0.09 2.80 0.52 0.51 2.88 1.52 1.10 3.46 2.37 1.98 1.02 2.63 2.21 1.40 2.29 3.06 0.09 0.00 2.19 2.99 1.13 3.18 3.34 2.90 1.19 )
2.15 1.62 0.41 1.56 0.39 2.67 0.65 2.76 2.68 0.25 3.55 3.03 2.28 1.06 0.45 1.07 2.93 3.91 1.95 0.86 3.20 2.45 1.39 1.93 3.99 1.49 3.79 0.88 2.52 2.30 0.58 3.72 )
1.83 5.57 0.19 2.62 2.97 0.40 2.60 0.80 0.94 2.93 3.98 3.02 3.57 2.19 3.57 3.48 2.46 2.88 2.92 0.61 3.02 0.61 1.62 2.46 2.87 3.32 3.54 0.25 1.58 2.41 1.95 1.81 )
3.05 2.32 1.34 2.33 1.44 3.00 0.33 0.38 3.99 0.18 1.59 0.43 1.66 1.74 1.08 1.81 0.90 1.05 2.57 3.57 3.52 1.80 1.65 0.93 3.86 2.96 1.37 2.89 2.41 2.49 3.58 2.73 )
0.94 0.42 2.13 2.86 2.27 0.06 2.04 1.83 3.88 3.56 2.81 0.98 0.41 1.07 1.87 0.43 0.20 1.54 0.47 3.99 2.96 0.16 0.02 2.72 3.83 2.83 3.00 0.56 2.71 2.45 3.65 3.90 )
0.28 3.63 3.45 2.51 3.56 2.30 0.61 3.59 2.30 1.05 0.70 2.50 3.34 2.68 3.43 1.79 3.93 2.64 2.84 0.23 3.26 2.99 3.90 3.34 1.26 0.47 0.82 1.17 3.79 3.37 1.00 0.86 )
```

Matrice C :

```
Matrix C:
115.31 112.85 89.07 117.14 114.76 107.13 85.01 103.48 118.60 113.65 86.02 106.86 110.85 114.08 111.23 117.48 121.38 137.25 133.00 116.29 117.33 119.75 100.92 114.01 117.46 96.14 121.42 114.85 121.94 115.07 )
145.91 135.33 123.53 127.67 153.73 123.83 122.57 110.21 157.91 145.27 99.26 136.91 128.82 128.27 134.34 138.91 152.49 156.38 134.98 136.46 136.59 128.05 116.70 123.72 136.33 121.41 143.00 109.04 155.78 138.58 157.24 150.29 )
137.50 124.71 101.68 132.33 137.33 137.33 137.33 137.33 137.33 137.33 137.33 137.33 137.33 137.33 137.33 137.33 137.33 137.33 137.33 137.33 137.33 137.33 137.33 137.33 137.33 137.33 137.33 137.33 137.33 137.33 )
117.70 125.91 93.46 144.23 126.97 118.97 90.21 116.35 138.83 125.60 105.08 133.36 125.50 136.56 112.10 134.66 156.51 152.45 133.65 126.10 149.07 108.18 136.62 146.27 159.53 140.77 136.83 104.53 141.93 137.78 141.21 162.83 )
124.71 127.30 116.30 125.51 132.06 113.46 102.95 102.64 146.06 121.40 105.57 126.02 132.23 125.80 112.11 127.20 158.39 155.82 125.88 118.01 146.89 110.66 119.78 121.26 163.04 112.01 143.75 103.46 139.16 135.20 157.31 150.81 )
138.91 110.40 97.37 117.59 137.56 102.87 96.66 96.45 114.12 109.36 106.45 114.12 109.36 106.45 114.12 109.36 106.45 114.12 109.36 106.45 114.12 109.36 106.45 114.12 109.36 106.45 114.12 109.36 106.45 114.12 109.36 )
125.17 116.00 107.68 125.76 134.57 114.03 117.30 92.14 139.37 130.82 117.34 145.00 121.34 139.40 111.22 134.00 144.40 150.62 111.29 116.33 140.12 93.73 115.26 129.02 134.35 116.21 131.48 90.42 129.19 129.93 132.01 162.99 )
139.36 135.76 121.47 128.48 150.65 124.34 120.31 106.99 151.74 129.83 109.31 148.63 132.08 135.28 132.00 146.04 164.68 167.47 139.58 129.59 149.82
```

b. Les blocs doivent avoir une taille de 8x8 (à l'exception du cas de matrice 4x4)

B. Profilez votre kernel et discuter des résultats

Avec des matrices 32*32 :

NvProf :

Élément :

```
--21736== Profiling application: a
--21736== Profiling result:
Type      Time(%)      Time      Calls      Avg      Min      Max      Name
GPU activities: 62.24%  6.7520us    1  6.7520us  6.7520us  6.7520us multiplyKernelElement(float*, float const *, float const *, int)
                28.32%  3.0720us    2  1.5360us  1.2800us  1.7920us [CUDA memcpy HtoD]
                9.44%  1.0240us    1  1.0240us  1.0240us  1.0240us [CUDA memcpy DtoH]
API calls:    79.26%  221.22ms    3  73.740ms  2.2000us  221.21ms cudaMalloc
                20.52%  57.277ms    1  57.277ms  57.277ms  57.277ms cudaDeviceReset
                0.12%  332.70us    3  110.90us  42.400us  232.90us cudaMemcpy
                0.04%  124.00us    3  41.533us  2.3000us  117.40us cudaFree
                0.03%  75.500us    1  75.500us  75.500us  75.500us cudaLaunchKernel
                0.02%  47.500us    1  47.500us  47.500us  47.500us cudaDeviceSynchronize
                0.01%  15.600us   101    154ns    100ns   1.1000us cuDeviceGetAttribute
                0.01%  14.100us    3  4.7000us  200ns   13.200us cuDeviceGetCount
                0.00%  6.0000us    1  6.0000us  6.0000us  6.0000us cudaSetDevice
                0.00%  1.5000us    2    750ns   200ns   1.3000us cuDeviceGet
                0.00%  1.1000us    1  1.1000us  1.1000us  1.1000us cuDeviceGetName
                0.00%  400ns      1    400ns    400ns    400ns cuDeviceGetLuid
                0.00%  300ns      1    300ns    300ns    300ns cuDeviceTotalMem
                0.00%  300ns      1    300ns    300ns    300ns cudaGetLastError
                0.00%  200ns      1    200ns    200ns    200ns cuDeviceGetUuid
C:\Users\lacommap\Desktop\CudaC\Devoir6_Part1>
```

Rangée :

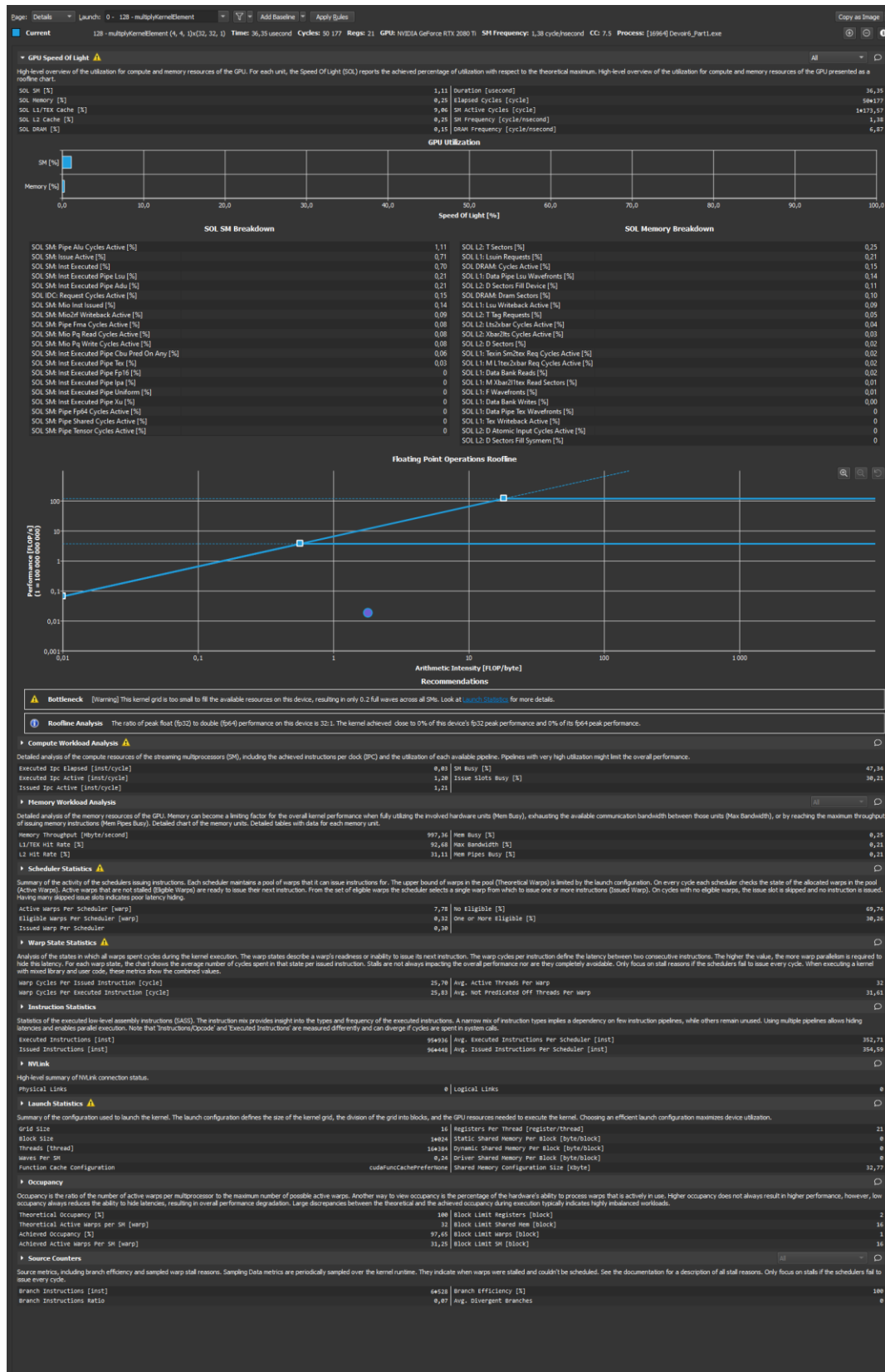
```
--22404== Profiling application: a
--22404== Profiling result:
Type      Time(%)      Time      Calls      Avg      Min      Max      Name
GPU activities: 96.30%  112.58us    1  112.58us  112.58us  112.58us multiplyKernelRow(float*, float const *, float const *, int)
                2.71%  3.1680us    2  1.5840us  1.3120us  1.8560us [CUDA memcpy HtoD]
                0.90%  1.0560us    1  1.0560us  1.0560us  1.0560us [CUDA memcpy DtoH]
API calls:    78.29%  206.80ms    3  68.933ms  2.0000us  206.79ms cudaMalloc
                21.47%  56.707ms    1  56.707ms  56.707ms  56.707ms cudaDeviceReset
                0.11%  288.30us    3  96.100us  38.400us  197.00us cudaMemcpy
                0.06%  150.40us    1  150.40us  150.40us  150.40us cudaDeviceSynchronize
                0.05%  124.30us    3  41.433us  2.0000us  117.10us cudaFree
                0.02%  57.200us    1  57.200us  57.200us  57.200us cudaLaunchKernel
                0.01%  13.800us   101    136ns    100ns    800ns cuDeviceGetAttribute
                0.00%  6.3000us    3  2.1000us  200ns    5.7000us cuDeviceGetCount
                0.00%  5.4000us    1  5.4000us  5.4000us  5.4000us cudaSetDevice
                0.00%  1.1000us    2    550ns   200ns    900ns cuDeviceGet
                0.00%  600ns      1    600ns    600ns    600ns cuDeviceGetName
                0.00%  400ns      1    400ns    400ns    400ns cuDeviceGetLuid
                0.00%  400ns      1    400ns    400ns    400ns cudaGetLastError
                0.00%  200ns      1    200ns    200ns    200ns cuDeviceTotalMem
                0.00%  100ns      1    100ns    100ns    100ns cuDeviceGetUuid
C:\Users\lacommap\Desktop\CudaC\Devoir6_Part1>
```

Colonne :

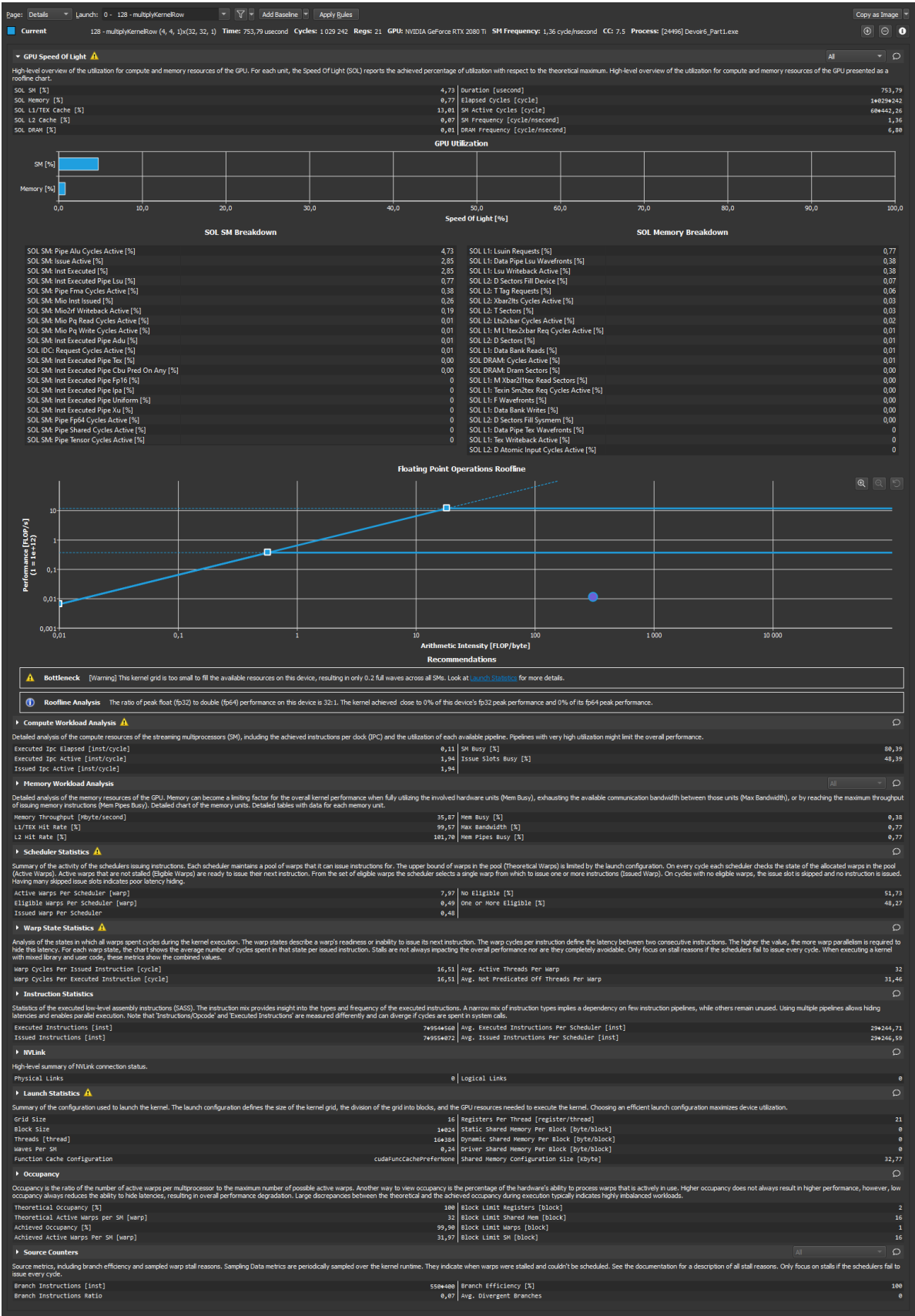
```
--1236== Profiling application: a
--1236== Profiling result:
Type      Time(%)      Time      Calls      Avg      Min      Max      Name
GPU activities: 96.60%  116.45us    1  116.45us  116.45us  116.45us multiplyKernelColumn(float*, float const *, float const *, int)
                2.57%  3.1040us    2  1.5520us  1.2800us  1.8240us [CUDA memcpy HtoD]
                0.82%  992ns      1    992ns    992ns    992ns [CUDA memcpy DtoH]
API calls:    80.31%  226.31ms    3  75.436ms  2.1000us  226.30ms cudaMalloc
                19.47%  54.857ms    1  54.857ms  54.857ms  54.857ms cudaDeviceReset
                0.10%  274.00us    3  91.333us  37.200us  184.40us cudaMemcpy
                0.05%  154.80us    1  154.80us  154.80us  154.80us cudaDeviceSynchronize
                0.04%  122.40us    3  40.800us  1.9000us  115.90us cudaFree
                0.02%  56.800us    1  56.800us  56.800us  56.800us cudaLaunchKernel
                0.01%  14.100us   101    139ns    100ns    600ns cuDeviceGetAttribute
                0.00%  6.5000us    3  2.1660us  200ns    5.7000us cuDeviceGetCount
                0.00%  5.8000us    1  5.8000us  5.8000us  5.8000us cudaSetDevice
                0.00%  1.2000us    2    600ns   200ns    1.0000us cuDeviceGet
                0.00%  600ns      1    600ns    600ns    600ns cuDeviceGetName
                0.00%  300ns      1    300ns    300ns    300ns cuDeviceGetLuid
                0.00%  300ns      1    300ns    300ns    300ns cudaGetLastError
                0.00%  200ns      1    200ns    200ns    200ns cuDeviceTotalMem
                0.00%  200ns      1    200ns    200ns    200ns cuDeviceGetUuid
C:\Users\lacommap\Desktop\CudaC\Devoir6_Part1>
```

NSight :

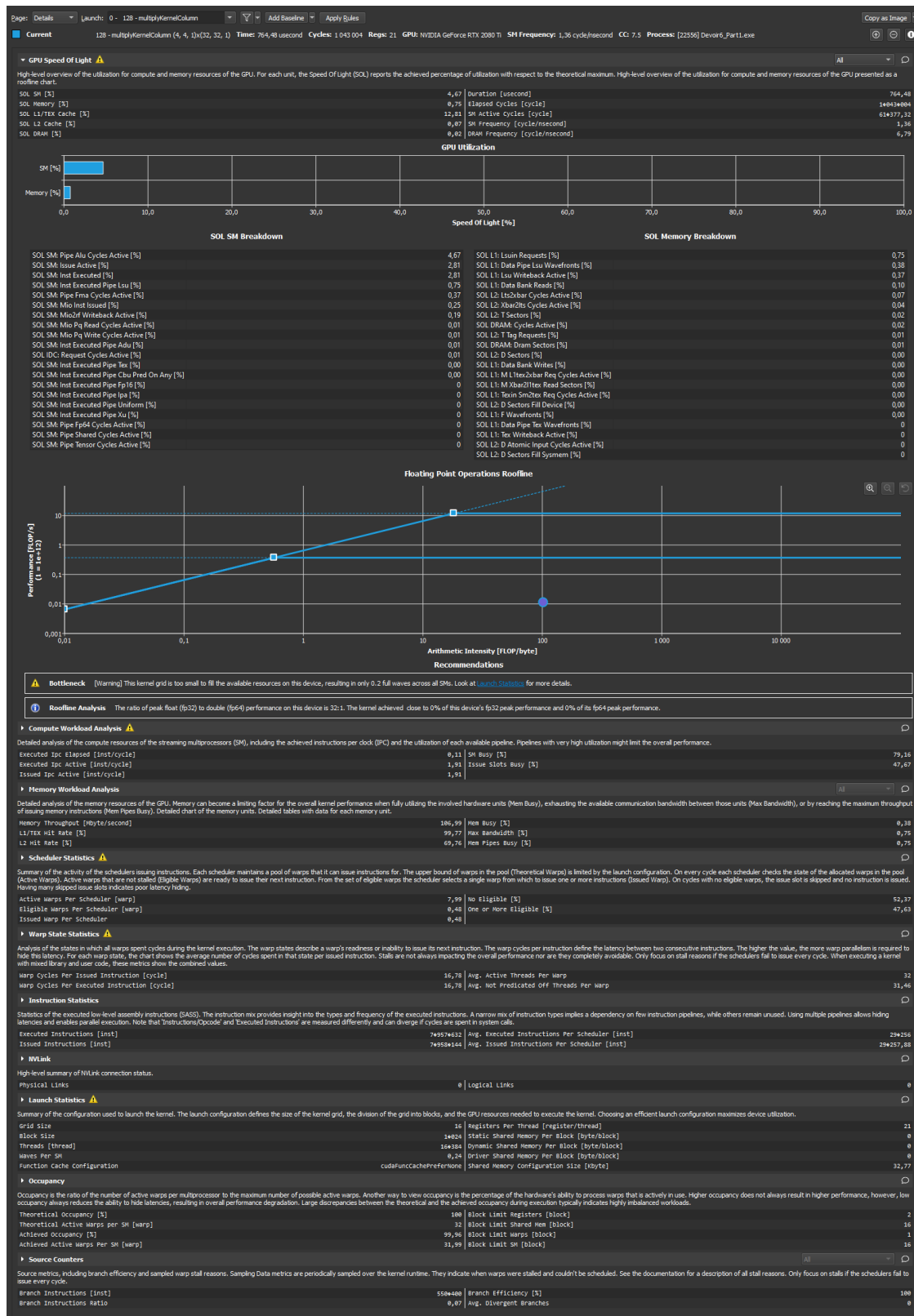
Élément :



Rangée :



Colonne :



PARTIE 2 – MULTIPLICATION MATRICIELLE OPTIMISÉE

Objectifs

Réimplémentez la multiplication matricielle de la PARTIE 1 en utilisant l'algorithme de tiling [chap. 4, 2]. En programmation parallèle, le tiling force plusieurs threads à se concentrer sur un sous ensemble de données d'entrée lors de chaque phase de sorte que ce sous ensemble de donnée puisse être stocké dans une mémoire spéciale, la shared memory par exemple, ce qui accélère grandement les accès mémoire. L'étudiant doit démontrer qu'avec des tiles de grande taille, les accès à la mémoire globale n'est plus un principal facteur limitant.

D'autre part, la capacité de raisonner en fonction des limitations matérielles en développant une application est un aspect clé de la pensée computationnelle. Les programmeurs CUDA doivent être conscient des limitations de chaque type de mémoire ainsi que des ressources propres à leur GPU puisqu'elles sont étroitement liées à l'implémentation. Une fois les capacités excédées, le GPU limite le nombre de threads pouvant être exécuter dans chaque SM. Les threads sont assignés aux SM sur une base de bloc par bloc. Chaque device CUDA dispose d'une limitation différente par rapport aux ressources disponible dans les SM, autant une limite de blocs par SM et une limite de threads par SM. La première limitation atteinte deviendra le facteur limitant.

Pour chaque Kernel, l'une ou plusieurs de ces limitations peut devenir le facteur limitant pour le nombre de threads qui sont utilisés simultanément dans le device CUDA.

MÉTHODOLOGIE

1. Implémentez la multiplication matricielle en suivant les mêmes instructions que pour la PARTIE 1. Toutefois, l'implémentation doit être faite en utilisant des techniques de tiling.

2. Votre implémentation doit être « transparent scalable » (voir section 3.4 à 3.6 dans [chap.3, 2]).
- Pour ce faire, vous devez connaître les propriétés de votre device. L'annexe B fournit un exemple d'une « host-stub function » qui permet d'extraire les propriétés.

```
Device 0: NVIDIA GeForce RTX 2080 Ti
Number of multiprocessors: 68
clock rate : 1545000
Compute capability : 7.5
Total amount of global memory: 11533888.00 KB
Total amount of constant memory: 64.00 KB
Total amount of shared memory per block: 48.00 KB
Total amount of shared memory per MP: 64.00 KB
Total number of registers available per block: 65536
Warp size: 32
Maximum number of threads per block: 1024
Maximum number of threads per multiprocessor: 1024
Maximum number of warps per multiprocessor: 32
Maximum Grid size : (2147483647,65535,65535)
Maximum block dimension : (1024,1024,64)

Sortie de C:\Users\lacommap\Desktop\CudaC\Devoir6Part2\x64\Debug\Devoir6Part2.exe (processus 7932). Code : 0.
Appuyez sur une touche pour fermer cette fenêtre. . .
```

3. L'implémentation doit être limitée à des matrices carrées 64x64. Toutefois, la taille des tuiles doit varier entre 2x2, 4x4, 8x8 et 16x16.
4. Validez chaque implementation.

Pour alléger l'affichage je prendrais des matrices 16*16.

Dans la fonction host stub, le nombre de threads par block est équivalent à la taille des tuiles. Par la suite, le nombre de bloc dans un grid est calculé en fonction du nombre de tuiles :

```
// Helper function for using CUDA to add vectors in parallel.
cudaError_t multiplyWithCuda(float* c, const float* a, const float* b, unsigned int size)
{
    float* dev_a = 0;
    float* dev_b = 0;
    float* dev_c = 0;
    cudaError_t cudaStatus;

    if (size > 64) {
        printf("Taille de la matrice limité à 64 par 64");
        return cudaErrorInvalidValue;
    }

    // Choose which GPU to run on, change this on a multi-GPU system.
    cudaStatus = cudaSetDevice(0);
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaSetDevice failed! Do you have a CUDA-capable GPU installed?");
        goto Error;
    }

    // Allocate GPU buffers for three vectors (two input, one output)
    cudaStatus = cudaMalloc((void**)&dev_c, (size * size) * sizeof(float));
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaMalloc failed!");
        goto Error;
    }

    cudaStatus = cudaMalloc((void**)&dev_a, (size * size) * sizeof(float));
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaMalloc failed!");
        goto Error;
    }

    cudaStatus = cudaMalloc((void**)&dev_b, (size * size) * sizeof(float));
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaMalloc failed!");
        goto Error;
    }
}
```

```
// Copy input vectors from host memory to GPU buffers.
cudaStatus = cudaMemcpy(dev_a, a, (size * size) * sizeof(float), cudaMemcpyHostToDevice);
if (cudaStatus != cudaSuccess) {
    fprintf(stderr, "cudaMemcpy failed!");
    goto Error;
}

cudaStatus = cudaMemcpy(dev_b, b, (size * size) * sizeof(float), cudaMemcpyHostToDevice);
if (cudaStatus != cudaSuccess) {
    fprintf(stderr, "cudaMemcpy failed!");
    goto Error;
}

dim3 threadsPerBlock(tileLength, tileLength, 1);
dim3 nombreDeBlock((size + tileLength - 1) / tileLength, (size + tileLength - 1) / tileLength);

multiplyKernel << nombreDeBlock, threadsPerBlock >> > (dev_c, dev_a, dev_b, size);

// Check for any errors launching the kernel
cudaStatus = cudaGetLastError();
if (cudaStatus != cudaSuccess) {
    fprintf(stderr, "addKernel launch failed: %s\n", cudaGetErrorString(cudaStatus));
    goto Error;
}

// cudaDeviceSynchronize waits for the kernel to finish, and returns
// any errors encountered during the launch.
cudaStatus = cudaDeviceSynchronize();
if (cudaStatus != cudaSuccess) {
    fprintf(stderr, "cudaDeviceSynchronize returned error code %d after launching addKernel!\n", cudaStatus);
    goto Error;
}

// Copy output vector from GPU buffer to host memory.
cudaStatus = cudaMemcpy(c, dev_c, (size * size) * sizeof(float), cudaMemcpyDeviceToHost);
if (cudaStatus != cudaSuccess) {
    fprintf(stderr, "cudaMemcpy failed!");
    goto Error;
}

Error:
    cudaFree(dev_c);
    cudaFree(dev_a);
    cudaFree(dev_b);

    return cudaStatus;
}
```

Pour le kernel, il existe plusieurs implémentations open-source qui font ce genre de calcul matriciel, je me suis alors inspiré de ce répertoire sur Github :

https://github.com/tgautam03/CUDA-C/blob/master/05_tiled_mat_mul/tiled_mat_mul_gpu.cu

```
__global__ void multiplyKernel(float* c, const float* a, const float* b, int length) {
    int threadY = threadIdx.y;
    int threadX = threadIdx.x;

    int row = (tileLength * blockIdx.y) + threadY;
    int column = (tileLength * blockIdx.x) + threadX;

    __shared__ float tileA[tileLength][tileLength];
    __shared__ float tileB[tileLength][tileLength];

    float sum = 0;

    int numTiles = (length + tileLength - 1) / tileLength;

    // Data loading
    for (int t = 0; t < numTiles; t++) {
        if (row < length && (t * tileLength + threadX) < length) {
            tileA[threadY][threadX] = a[row * length + (t * tileLength + threadX)];
        }
        else {
            tileA[threadY][threadX] = 0.0f;
        }

        if (column < length && (t * tileLength + threadY) < length) {
            tileB[threadY][threadX] = b[(t * tileLength + threadY) * length + column];
        }
        else {
            tileB[threadY][threadX] = 0.0f;
        }

        __syncthreads();

        for (int k = 0; k < tileLength; k++) {
            sum += tileA[threadY][k] * tileB[k][threadX];
        }

        __syncthreads();
    }

    if (row < length && column < length) {
        c[(row * length) + column] = sum;
    }
}
```

Essentiellement, cette technique permet d'exploiter la mémoire partagée de chaque bloc, ce qui réduit considérablement les accès à la mémoire globale et améliore les performances. Les variables « row » et « column » servent à calculer les indices de la matrice C finale que chaque thread est chargé de calculer.

Dans chaque itération, les données nécessaires sont d'abord exportées depuis la mémoire globale vers la mémoire partagée pour chaque tuile. Une fois toutes les données chargées, on s'assure de leur disponibilité grâce à la fonction `__syncthreads()`, qui synchronise tous les threads du bloc. Ensuite, les threads effectuent les calculs sur ces données en mémoire partagée. Avant de passer à la tuile suivante, on utilise à nouveau `__syncthreads()` pour garantir que tous les threads ont terminé leurs calculs.

Finalement, les résultats calculés sont écrits dans la mémoire globale pour remplir les éléments correspondants de la matrice C.

Tuile 2*2 :

```
Matrix A:
{ 0.01 2.25 0.77 3.23 2.34 1.92 1.40 3.58 3.29 2.99 0.70 3.44 2.84 2.05 1.22 0.06 }
{ 0.37 1.46 0.59 0.66 3.95 1.78 0.48 0.02 0.04 1.51 2.13 2.28 2.41 2.43 0.66 2.65 }
{ 1.80 1.41 0.23 2.43 3.13 3.21 2.08 1.21 3.50 2.91 3.82 3.70 2.16 0.57 1.85 0.94 }
{ 3.45 0.84 3.12 3.37 3.99 4.00 2.45 1.57 1.06 1.19 3.36 0.09 1.50 0.37 2.71 0.22 }
{ 0.04 3.68 1.10 1.09 2.35 2.76 3.35 2.91 1.94 0.82 2.97 1.87 1.83 3.80 2.98 0.43 }
{ 2.40 1.54 2.94 2.44 2.29 1.45 0.61 0.90 1.70 3.21 2.07 3.96 3.01 1.38 0.68 2.63 }
{ 1.97 0.25 2.80 2.02 0.59 3.80 0.57 3.62 2.77 1.21 1.71 0.28 3.87 2.73 0.61 3.51 }
{ 3.29 2.33 0.77 0.71 3.27 1.90 0.62 2.02 2.93 1.62 1.12 2.27 2.73 3.02 2.89 1.90 }
{ 0.49 1.47 3.34 0.14 2.07 2.65 1.70 0.42 3.80 3.69 2.20 1.38 1.89 1.50 3.39 1.27 }
{ 1.82 1.09 3.93 1.19 2.96 2.27 0.78 3.05 3.36 1.59 2.00 3.56 0.11 3.98 2.29 0.20 }
{ 2.13 0.78 3.37 2.51 2.63 0.79 3.37 0.49 0.44 2.97 1.26 3.76 1.14 1.35 0.56 2.93 }
{ 3.34 2.83 2.40 2.99 1.01 0.58 0.01 0.24 3.22 3.41 0.84 0.46 2.21 0.06 0.46 1.82 }
{ 3.01 2.74 2.17 0.30 1.75 0.81 2.78 1.16 1.75 0.93 2.31 2.13 2.51 0.64 2.02 3.85 }
{ 2.78 3.70 0.76 1.34 0.71 3.98 1.83 3.99 0.39 2.50 0.38 1.75 3.73 0.19 3.58 1.16 }
{ 0.91 3.08 1.64 0.81 2.51 2.42 1.81 1.87 2.39 2.54 3.42 3.32 2.50 2.88 2.26 1.50 }
{ 0.74 2.95 2.22 3.62 0.97 0.76 2.42 2.79 2.34 1.41 1.98 0.32 2.96 2.45 2.48 2.76 }

Matrix B:
{ 3.22 0.60 2.30 3.47 3.65 2.46 2.91 0.17 2.67 3.91 1.26 2.28 1.22 0.70 0.43 3.48 }
{ 3.40 2.98 0.62 1.31 0.32 0.31 2.56 3.28 2.18 1.79 1.64 1.19 1.86 2.00 0.61 1.29 }
{ 2.95 1.26 3.31 3.84 3.49 2.90 1.20 3.78 0.51 0.26 3.14 2.10 2.44 3.82 0.29 3.50 }
{ 2.62 1.29 0.42 2.02 0.91 1.16 3.68 2.20 2.65 0.46 1.97 1.52 1.99 3.17 2.04 1.53 }
{ 2.75 2.13 2.43 1.58 0.02 2.83 0.40 2.49 3.45 1.97 2.99 1.99 1.52 3.14 2.21 1.43 }
{ 3.82 2.52 0.71 1.50 0.53 2.97 3.81 2.45 0.11 1.32 0.22 2.56 0.53 3.39 3.46 2.39 }
{ 2.89 3.42 0.06 0.51 2.83 2.47 0.87 0.26 0.68 2.50 1.36 1.28 1.47 2.64 3.21 3.23 }
{ 2.11 2.44 3.19 3.60 0.58 2.52 1.61 1.01 0.55 3.42 0.26 1.71 2.29 1.21 2.19 0.90 }
{ 1.25 0.44 3.23 0.54 1.14 3.15 3.58 3.16 2.98 2.46 1.44 3.43 0.91 3.45 0.92 1.00 }
{ 2.17 3.94 0.22 0.33 2.10 1.71 0.38 1.04 3.57 0.93 0.59 0.50 3.73 0.32 0.19 0.23 }
{ 1.35 3.66 1.59 1.73 3.78 3.35 2.14 3.37 2.77 1.59 1.04 0.02 2.10 3.82 1.59 0.96 }
{ 2.34 1.02 2.74 3.78 1.74 3.56 0.03 3.76 2.41 3.14 2.31 0.57 0.89 1.53 0.02 1.67 }
{ 0.33 2.64 3.42 0.26 3.24 2.65 2.77 3.21 2.12 2.74 0.57 2.76 2.91 3.11 0.12 3.47 }
{ 2.58 2.83 0.34 2.21 3.79 0.24 1.10 0.58 3.93 2.48 1.17 3.69 1.47 2.78 0.87 0.62 }
{ 0.96 2.09 3.61 0.43 3.61 1.77 0.32 3.13 0.69 3.90 3.10 3.48 0.84 1.83 0.02 3.00 }
{ 0.46 1.62 1.24 3.97 0.15 1.01 0.76 0.99 0.61 2.48 3.54 3.76 0.78 3.12 2.58 2.63 }

Matrix C:
{ 70.80 70.44 60.55 55.63 53.52 71.79 58.92 74.93 70.70 69.21 45.94 61.33 58.39 76.84 39.43 51.63 }
{ 49.17 55.72 39.75 45.64 41.92 50.29 33.63 54.18 54.02 49.74 42.96 47.97 38.99 63.23 32.08 42.80 }
{ 75.95 76.97 64.24 58.84 65.36 87.00 65.14 83.46 76.66 76.39 53.56 62.77 57.28 88.26 47.16 62.48 }
{ 81.42 72.16 62.48 61.42 69.52 80.93 66.88 74.54 61.90 67.46 55.16 64.62 55.39 89.55 51.41 72.63 }
{ 77.62 83.65 58.76 56.78 68.43 71.75 58.12 76.98 66.41 77.38 51.61 68.67 54.89 88.44 48.11 60.74 }
{ 71.03 68.16 63.49 70.94 66.00 76.27 55.70 78.00 72.34 68.76 58.16 63.07 59.81 82.01 35.69 65.14 }
{ 65.69 67.09 63.95 69.06 62.88 71.89 68.78 67.00 56.89 69.80 46.87 77.88 55.18 88.18 47.56 66.43 }
{ 71.27 68.53 68.56 63.22 66.02 72.17 58.20 72.37 73.04 83.53 55.71 76.59 52.55 79.42 37.49 64.46 }
{ 65.70 70.98 62.60 48.89 67.59 73.68 51.46 77.55 62.28 64.62 53.95 66.86 54.37 81.99 35.07 60.77 }
{ 81.04 68.29 71.31 74.53 71.75 80.50 55.19 80.01 72.40 75.56 58.60 71.48 54.67 86.42 39.28 60.14 }
{ 70.25 64.43 50.70 67.98 61.70 68.88 41.78 63.04 61.80 61.66 58.96 55.29 53.30 74.84 39.85 63.85 }
{ 56.59 49.83 47.54 48.05 48.80 53.69 56.22 58.36 59.48 50.12 43.27 52.23 49.73 61.82 24.73 50.78 }
{ 63.07 65.14 60.88 64.05 63.59 68.71 50.39 67.48 56.20 74.25 57.66 64.72 49.31 77.56 38.70 69.60 }
{ 74.40 75.85 63.54 58.34 60.90 70.58 62.65 70.61 52.94 80.12 45.19 66.55 56.93 70.47 41.53 70.33 }
{ 77.63 83.83 66.00 64.84 72.50 80.28 59.14 85.22 76.03 79.55 57.21 69.55 60.86 90.57 43.11 64.04 }
{ 67.19 74.27 59.60 60.77 65.49 64.51 62.49 72.07 63.26 70.87 56.01 72.25 59.15 87.55 44.04 64.86 }
```


Tuile 4*4 :

```
Matrix A:
{ 0.01 2.25 0.77 3.23 2.34 1.92 1.40 3.58 3.29 2.99 0.70 3.44 2.84 2.05 1.22 0.06 }
{ 0.37 1.46 0.59 0.66 3.95 1.78 0.48 0.02 0.04 1.51 2.13 2.28 2.41 2.43 0.66 2.65 }
{ 1.80 1.41 0.23 2.43 3.13 3.21 2.08 1.21 3.50 2.91 3.82 3.70 2.16 0.57 1.85 0.94 }
{ 3.45 0.84 3.12 3.37 3.99 4.00 2.45 1.57 1.06 1.19 3.36 0.09 1.50 0.37 2.71 0.22 }
{ 0.04 3.68 1.10 1.09 2.35 2.76 3.35 2.91 1.94 0.82 2.97 1.87 1.83 3.80 2.98 0.43 }
{ 2.40 1.54 2.94 2.44 2.29 1.45 0.61 0.90 1.70 3.21 2.07 3.96 3.01 1.38 0.68 2.63 }
{ 1.97 0.25 2.80 2.02 0.59 3.80 0.57 3.62 2.77 1.21 1.71 0.28 3.87 2.73 0.61 3.51 }
{ 3.29 2.33 0.77 0.71 3.27 1.90 0.62 2.02 2.93 1.62 1.12 2.27 2.73 3.02 2.89 1.90 }
{ 0.49 1.47 3.34 0.14 2.07 2.65 1.70 0.42 3.80 3.69 2.20 1.38 1.89 1.50 3.39 1.27 }
{ 1.82 1.09 3.93 1.19 2.96 2.27 0.78 3.05 3.36 1.59 2.00 3.56 0.11 3.98 2.29 0.20 }
{ 2.13 0.78 3.37 2.51 2.63 0.79 3.37 0.49 0.44 2.97 1.26 3.76 1.14 1.35 0.56 2.93 }
{ 3.34 2.83 2.40 2.99 1.01 0.58 0.01 0.24 3.22 3.41 0.84 0.46 2.21 0.06 0.46 1.82 }
{ 3.01 2.74 2.17 0.30 1.75 0.81 2.78 1.16 1.75 0.93 2.31 2.13 2.51 0.64 2.02 3.85 }
{ 2.78 3.70 0.76 1.34 0.71 3.98 1.83 3.99 0.39 2.50 0.38 1.75 3.73 0.19 3.58 1.16 }
{ 0.91 3.08 1.64 0.81 2.51 2.42 1.81 1.87 2.39 2.54 3.42 3.32 2.50 2.88 2.26 1.50 }
{ 0.74 2.95 2.22 3.62 0.97 0.76 2.42 2.79 2.34 1.41 1.98 0.32 2.96 2.45 2.48 2.76 }

Matrix B:
{ 3.22 0.60 2.30 3.47 3.65 2.46 2.91 0.17 2.67 3.91 1.26 2.28 1.22 0.70 0.43 3.48 }
{ 3.40 2.98 0.62 1.31 0.32 0.31 2.56 3.28 2.18 1.79 1.64 1.19 1.86 2.00 0.61 1.29 }
{ 2.95 1.26 3.31 3.84 3.49 2.90 1.20 3.78 0.51 0.26 3.14 2.10 2.44 3.82 0.29 3.50 }
{ 2.62 1.29 0.42 2.02 0.91 1.16 3.68 2.20 2.65 0.46 1.97 1.52 1.99 3.17 2.04 1.53 }
{ 2.75 2.13 2.43 1.58 0.02 2.83 0.40 2.49 3.45 1.97 2.99 1.99 1.52 3.14 2.21 1.43 }
{ 3.82 2.52 0.71 1.50 0.53 2.97 3.81 2.45 0.11 1.32 0.22 2.56 0.53 3.39 3.46 2.39 }
{ 2.89 3.42 0.06 0.51 2.83 2.47 0.87 0.26 0.68 2.50 1.36 1.28 1.47 2.64 3.21 3.23 }
{ 2.11 2.44 3.19 3.60 0.58 2.52 1.61 1.01 0.55 3.42 0.26 1.71 2.29 1.21 2.19 0.90 }
{ 1.25 0.44 3.23 0.54 1.14 3.15 3.58 3.16 2.98 2.46 1.44 3.43 0.91 3.45 0.92 1.00 }
{ 2.17 3.94 0.22 0.33 2.10 1.71 0.38 1.04 3.57 0.93 0.59 0.50 3.73 0.32 0.19 0.23 }
{ 1.35 3.66 1.59 1.73 3.78 3.35 2.14 3.37 2.77 1.59 1.04 0.02 2.10 3.82 1.59 0.96 }
{ 2.34 1.02 2.74 3.78 1.74 3.56 0.03 3.76 2.41 3.14 2.31 0.57 0.89 1.53 0.02 1.67 }
{ 0.33 2.64 3.42 0.26 3.24 2.65 2.77 3.21 2.12 2.74 0.57 2.76 2.91 3.11 0.12 3.47 }
{ 2.58 2.83 0.34 2.21 3.79 0.24 1.10 0.58 3.93 2.48 1.17 3.69 1.47 2.78 0.87 0.62 }
{ 0.96 2.09 3.61 0.43 3.61 1.77 0.32 3.13 0.69 3.90 3.10 3.48 0.84 1.83 0.02 3.00 }
{ 0.46 1.62 1.24 3.97 0.15 1.01 0.76 0.99 0.61 2.48 3.54 3.76 0.78 3.12 2.58 2.63 }

Matrix C:
{ 70.80 70.44 60.55 55.63 53.52 71.79 58.92 74.93 70.70 69.21 45.94 61.33 58.39 76.84 39.43 51.63 }
{ 49.17 55.72 39.75 45.64 41.92 50.29 33.63 54.18 54.02 49.74 42.96 47.97 38.99 63.23 32.08 42.80 }
{ 75.95 76.97 64.24 58.84 65.36 87.00 65.14 83.46 76.66 76.39 53.56 62.77 57.28 88.26 47.16 62.48 }
{ 81.42 72.16 62.48 61.42 69.52 80.93 66.88 74.54 61.90 67.46 55.16 64.62 55.39 89.55 51.41 72.63 }
{ 77.62 83.65 58.76 56.78 68.43 71.75 58.12 76.98 66.41 77.38 51.61 68.67 54.89 88.44 48.11 60.74 }
{ 71.03 68.16 63.49 70.94 66.00 76.27 55.70 78.00 72.34 68.76 58.16 63.07 59.81 82.01 35.69 65.14 }
{ 65.69 67.09 63.95 69.06 62.88 71.89 68.78 67.00 56.89 69.80 46.87 77.88 55.18 88.18 47.56 66.43 }
{ 71.27 68.53 68.56 63.22 66.02 72.17 58.20 72.37 73.04 83.53 55.71 76.59 52.55 79.42 37.49 64.46 }
{ 65.70 70.98 62.60 48.89 67.59 73.68 51.46 77.55 62.28 64.62 53.95 66.86 54.37 81.99 35.07 60.77 }
{ 81.04 68.29 71.31 74.53 71.75 80.50 55.19 80.01 72.40 75.56 58.60 71.48 54.67 86.42 39.28 60.14 }
{ 70.25 64.43 50.70 67.98 61.70 68.88 41.78 63.04 61.80 61.66 58.96 55.29 53.30 74.84 39.85 63.85 }
{ 56.59 49.83 47.54 48.05 48.80 53.69 56.22 58.36 59.48 50.12 43.27 52.23 49.73 61.82 24.73 50.78 }
{ 63.07 65.14 60.88 64.05 63.59 68.71 50.39 67.48 56.20 74.25 57.66 64.72 49.31 77.56 38.70 69.60 }
{ 74.40 75.85 63.54 58.34 60.90 70.58 62.65 70.61 52.94 80.12 45.19 66.55 56.93 70.47 41.53 70.33 }
{ 77.63 83.83 66.00 64.84 72.50 80.28 59.14 85.22 76.03 79.55 57.21 69.55 60.86 90.57 43.11 64.04 }
{ 67.19 74.27 59.60 60.77 65.49 64.51 62.49 72.07 63.26 70.87 56.01 72.25 59.15 87.55 44.04 64.86 }
```

Tuile 8*8 :

```
Matrix A:
{ 0.01 2.25 0.77 3.23 2.34 1.92 1.40 3.58 3.29 2.99 0.70 3.44 2.84 2.05 1.22 0.06 }
{ 0.37 1.46 0.59 0.66 3.95 1.78 0.48 0.02 0.04 1.51 2.13 2.28 2.41 2.43 0.66 2.65 }
{ 1.80 1.41 0.23 2.43 3.13 3.21 2.08 1.21 3.50 2.91 3.82 3.70 2.16 0.57 1.85 0.94 }
{ 3.45 0.84 3.12 3.37 3.99 4.00 2.45 1.57 1.06 1.19 3.36 0.09 1.50 0.37 2.71 0.22 }
{ 0.04 3.68 1.10 1.09 2.35 2.76 3.35 2.91 1.94 0.82 2.97 1.87 1.83 3.80 2.98 0.43 }
{ 2.40 1.54 2.94 2.44 2.29 1.45 0.61 0.90 1.70 3.21 2.07 3.96 3.01 1.38 0.68 2.63 }
{ 1.97 0.25 2.80 2.02 0.59 3.80 0.57 3.62 2.77 1.21 1.71 0.28 3.87 2.73 0.61 3.51 }
{ 3.29 2.33 0.77 0.71 3.27 1.90 0.62 2.02 2.93 1.62 1.12 2.27 2.73 3.02 2.89 1.90 }
{ 0.49 1.47 3.34 0.14 2.07 2.65 1.70 0.42 3.80 3.69 2.20 1.38 1.89 1.50 3.39 1.27 }
{ 1.82 1.09 3.93 1.19 2.96 2.27 0.78 3.05 3.36 1.59 2.00 3.56 0.11 3.98 2.29 0.20 }
{ 2.13 0.78 3.37 2.51 2.63 0.79 3.37 0.49 0.44 2.97 1.26 3.76 1.14 1.35 0.56 2.93 }
{ 3.34 2.83 2.40 2.99 1.01 0.58 0.01 0.24 3.22 3.41 0.84 0.46 2.21 0.06 0.46 1.82 }
{ 3.01 2.74 2.17 0.30 1.75 0.81 2.78 1.16 1.75 0.93 2.31 2.13 2.51 0.64 2.02 3.85 }
{ 2.78 3.70 0.76 1.34 0.71 3.98 1.83 3.99 0.39 2.50 0.38 1.75 3.73 0.19 3.58 1.16 }
{ 0.91 3.08 1.64 0.81 2.51 2.42 1.81 1.87 2.39 2.54 3.42 3.32 2.50 2.88 2.26 1.50 }
{ 0.74 2.95 2.22 3.62 0.97 0.76 2.42 2.79 2.34 1.41 1.98 0.32 2.96 2.45 2.48 2.76 }

Matrix B:
{ 3.22 0.60 2.30 3.47 3.65 2.46 2.91 0.17 2.67 3.91 1.26 2.28 1.22 0.70 0.43 3.48 }
{ 3.40 2.98 0.62 1.31 0.32 0.31 2.56 3.28 2.18 1.79 1.64 1.19 1.86 2.00 0.61 1.29 }
{ 2.95 1.26 3.31 3.84 3.49 2.90 1.20 3.78 0.51 0.26 3.14 2.10 2.44 3.82 0.29 3.50 }
{ 2.62 1.29 0.42 2.02 0.91 1.16 3.68 2.20 2.65 0.46 1.97 1.52 1.99 3.17 2.04 1.53 }
{ 2.75 2.13 2.43 1.58 0.02 2.83 0.40 2.49 3.45 1.97 2.99 1.99 1.52 3.14 2.21 1.43 }
{ 3.82 2.52 0.71 1.50 0.53 2.97 3.81 2.45 0.11 1.32 0.22 2.56 0.53 3.39 3.46 2.39 }
{ 2.89 3.42 0.06 0.51 2.83 2.47 0.87 0.26 0.68 2.50 1.36 1.28 1.47 2.64 3.21 3.23 }
{ 2.11 2.44 3.19 3.60 0.58 2.52 1.61 1.01 0.55 3.42 0.26 1.71 2.29 1.21 2.19 0.90 }
{ 1.25 0.44 3.23 0.54 1.14 3.15 3.58 3.16 2.98 2.46 1.44 3.43 0.91 3.45 0.92 1.00 }
{ 2.17 3.94 0.22 0.33 2.10 1.71 0.38 1.04 3.57 0.93 0.59 0.50 3.73 0.32 0.19 0.23 }
{ 1.35 3.66 1.59 1.73 3.78 3.35 2.14 3.37 2.77 1.59 1.04 0.02 2.10 3.82 1.59 0.96 }
{ 2.34 1.02 2.74 3.78 1.74 3.56 0.03 3.76 2.41 3.14 2.31 0.57 0.89 1.53 0.02 1.67 }
{ 0.33 2.64 3.42 0.26 3.24 2.65 2.77 3.21 2.12 2.74 0.57 2.76 2.91 3.11 0.12 3.47 }
{ 2.58 2.83 0.34 2.21 3.79 0.24 1.10 0.58 3.93 2.48 1.17 3.69 1.47 2.78 0.87 0.62 }
{ 0.96 2.09 3.61 0.43 3.61 1.77 0.32 3.13 0.69 3.90 3.10 3.48 0.84 1.83 0.02 3.00 }
{ 0.46 1.62 1.24 3.97 0.15 1.01 0.76 0.99 0.61 2.48 3.54 3.76 0.78 3.12 2.58 2.63 }

Matrix C:
{ 70.80 70.44 60.55 55.63 53.52 71.79 58.92 74.93 70.70 69.21 45.94 61.33 58.39 76.84 39.43 51.63 }
{ 49.17 55.72 39.75 45.64 41.92 50.29 33.63 54.18 54.02 49.74 42.96 47.97 38.99 63.23 32.08 42.80 }
{ 75.95 76.97 64.24 58.84 65.36 87.00 65.14 83.46 76.66 76.39 53.56 62.77 57.28 88.26 47.16 62.48 }
{ 81.42 72.16 62.48 61.42 69.52 80.93 66.88 74.54 61.90 67.46 55.16 64.62 55.39 89.55 51.41 72.63 }
{ 77.62 83.65 58.76 56.78 68.43 71.75 58.12 76.98 66.41 77.38 51.61 68.67 54.89 88.44 48.11 60.74 }
{ 71.03 68.16 63.49 70.94 66.00 76.27 55.70 78.00 72.34 68.76 58.16 63.07 59.81 82.01 35.69 65.14 }
{ 65.69 67.09 63.95 69.06 62.88 71.89 68.78 67.00 56.89 69.80 46.87 77.88 55.18 88.18 47.56 66.43 }
{ 71.27 68.53 68.56 63.22 66.02 72.17 58.20 72.37 73.04 83.53 55.71 76.59 52.55 79.42 37.49 64.46 }
{ 65.70 70.98 62.60 48.89 67.59 73.68 51.46 77.55 62.28 64.62 53.95 66.86 54.37 81.99 35.07 60.77 }
{ 81.04 68.29 71.31 74.53 71.75 80.50 55.19 80.01 72.40 75.56 58.60 71.48 54.67 86.42 39.28 60.14 }
{ 70.25 64.43 50.70 67.98 61.70 68.88 41.78 63.04 61.80 61.66 58.96 55.29 53.30 74.84 39.85 63.85 }
{ 56.59 49.83 47.54 48.05 48.80 53.69 56.22 58.36 59.48 50.12 43.27 52.23 49.73 61.82 24.73 50.78 }
{ 63.07 65.14 60.88 64.05 63.59 68.71 50.39 67.48 56.20 74.25 57.66 64.72 49.31 77.56 38.70 69.60 }
{ 74.40 75.85 63.54 58.34 60.90 70.58 62.65 70.61 52.94 80.12 45.19 66.55 56.93 70.47 41.53 70.33 }
{ 77.63 83.83 66.00 64.84 72.50 80.28 59.14 85.22 76.03 79.55 57.21 69.55 60.86 90.57 43.11 64.04 }
{ 67.19 74.27 59.60 60.77 65.49 64.51 62.49 72.07 63.26 70.87 56.01 72.25 59.15 87.55 44.04 64.86 }
```

Tuile 16*16:

```
Matrix A:
{ 0.01 2.25 0.77 3.23 2.34 1.92 1.40 3.58 3.29 2.99 0.70 3.44 2.84 2.05 1.22 0.06 }
{ 0.37 1.46 0.59 0.66 3.95 1.78 0.48 0.02 0.04 1.51 2.13 2.28 2.41 2.43 0.66 2.65 }
{ 1.80 1.41 0.23 2.43 3.13 3.21 2.08 1.21 3.50 2.91 3.82 3.70 2.16 0.57 1.85 0.94 }
{ 3.45 0.84 3.12 3.37 3.99 4.00 2.45 1.57 1.06 1.19 3.36 0.09 1.50 0.37 2.71 0.22 }
{ 0.04 3.68 1.10 1.09 2.35 2.76 3.35 2.91 1.94 0.82 2.97 1.87 1.83 3.80 2.98 0.43 }
{ 2.40 1.54 2.94 2.44 2.29 1.45 0.61 0.90 1.70 3.21 2.07 3.96 3.01 1.38 0.68 2.63 }
{ 1.97 0.25 2.80 2.02 0.59 3.80 0.57 3.62 2.77 1.21 1.71 0.28 3.87 2.73 0.61 3.51 }
{ 3.29 2.33 0.77 0.71 3.27 1.90 0.62 2.02 2.93 1.62 1.12 2.27 2.73 3.02 2.89 1.90 }
{ 0.49 1.47 3.34 0.14 2.07 2.65 1.70 0.42 3.80 3.69 2.20 1.38 1.89 1.50 3.39 1.27 }
{ 1.82 1.09 3.93 1.19 2.96 2.27 0.78 3.05 3.36 1.59 2.00 3.56 0.11 3.98 2.29 0.20 }
{ 2.13 0.78 3.37 2.51 2.63 0.79 3.37 0.49 0.44 2.97 1.26 3.76 1.14 1.35 0.56 2.93 }
{ 3.34 2.83 2.40 2.99 1.01 0.58 0.01 0.24 3.22 3.41 0.84 0.46 2.21 0.06 0.46 1.82 }
{ 3.01 2.74 2.17 0.30 1.75 0.81 2.78 1.16 1.75 0.93 2.31 2.13 2.51 0.64 2.02 3.85 }
{ 2.78 3.70 0.76 1.34 0.71 3.98 1.83 3.99 0.39 2.50 0.38 1.75 3.73 0.19 3.58 1.16 }
{ 0.91 3.08 1.64 0.81 2.51 2.42 1.81 1.87 2.39 2.54 3.42 3.32 2.50 2.88 2.26 1.50 }
{ 0.74 2.95 2.22 3.62 0.97 0.76 2.42 2.79 2.34 1.41 1.98 0.32 2.96 2.45 2.48 2.76 }

Matrix B:
{ 3.22 0.60 2.30 3.47 3.65 2.46 2.91 0.17 2.67 3.91 1.26 2.28 1.22 0.70 0.43 3.48 }
{ 3.40 2.98 0.62 1.31 0.32 0.31 2.56 3.28 2.18 1.79 1.64 1.19 1.86 2.00 0.61 1.29 }
{ 2.95 1.26 3.31 3.84 3.49 2.90 1.20 3.78 0.51 0.26 3.14 2.10 2.44 3.82 0.29 3.50 }
{ 2.62 1.29 0.42 2.02 0.91 1.16 3.68 2.20 2.65 0.46 1.97 1.52 1.99 3.17 2.04 1.53 }
{ 2.75 2.13 2.43 1.58 0.02 2.83 0.40 2.49 3.45 1.97 2.99 1.99 1.52 3.14 2.21 1.43 }
{ 3.82 2.52 0.71 1.50 0.53 2.97 3.81 2.45 0.11 1.32 0.22 2.56 0.53 3.39 3.46 2.39 }
{ 2.89 3.42 0.06 0.51 2.83 2.47 0.87 0.26 0.68 2.50 1.36 1.28 1.47 2.64 3.21 3.23 }
{ 2.11 2.44 3.19 3.60 0.58 2.52 1.61 1.01 0.55 3.42 0.26 1.71 2.29 1.21 2.19 0.90 }
{ 1.25 0.44 3.23 0.54 1.14 3.15 3.58 3.16 2.98 2.46 1.44 3.43 0.91 3.45 0.92 1.00 }
{ 2.17 3.94 0.22 0.33 2.10 1.71 0.38 1.04 3.57 0.93 0.59 0.50 3.73 0.32 0.19 0.23 }
{ 1.35 3.66 1.59 1.73 3.78 3.35 2.14 3.37 2.77 1.59 1.04 0.02 2.10 3.82 1.59 0.96 }
{ 2.34 1.02 2.74 3.78 1.74 3.56 0.03 3.76 2.41 3.14 2.31 0.57 0.89 1.53 0.02 1.67 }
{ 0.33 2.64 3.42 0.26 3.24 2.65 2.77 3.21 2.12 2.74 0.57 2.76 2.91 3.11 0.12 3.47 }
{ 2.58 2.83 0.34 2.21 3.79 0.24 1.10 0.58 3.93 2.48 1.17 3.69 1.47 2.78 0.87 0.62 }
{ 0.96 2.09 3.61 0.43 3.61 1.77 0.32 3.13 0.69 3.90 3.10 3.48 0.84 1.83 0.02 3.00 }
{ 0.46 1.62 1.24 3.97 0.15 1.01 0.76 0.99 0.61 2.48 3.54 3.76 0.78 3.12 2.58 2.63 }

Matrix C:
{ 70.80 70.44 60.55 55.63 53.52 71.79 58.92 74.93 70.70 69.21 45.94 61.33 58.39 76.84 39.43 51.63 }
{ 49.17 55.72 39.75 45.64 41.92 50.29 33.63 54.18 54.02 49.74 42.96 47.97 38.99 63.23 32.08 42.80 }
{ 75.95 76.97 64.24 58.84 65.36 87.00 65.14 83.46 76.66 76.39 53.56 62.77 57.28 88.26 47.16 62.48 }
{ 81.42 72.16 62.48 61.42 69.52 80.93 66.88 74.54 61.90 67.46 55.16 64.62 55.39 89.55 51.41 72.63 }
{ 77.62 83.65 58.76 56.78 68.43 71.75 58.12 76.98 66.41 77.38 51.61 68.67 54.89 88.44 48.11 60.74 }
{ 71.03 68.16 63.49 70.94 66.00 76.27 55.70 78.00 72.34 68.76 58.16 63.07 59.81 82.01 35.69 65.14 }
{ 65.69 67.09 63.95 69.06 62.88 71.89 68.78 67.00 56.89 69.80 46.87 77.88 55.18 88.18 47.56 66.43 }
{ 71.27 68.53 68.56 63.22 66.02 72.17 58.20 72.37 73.04 83.53 55.71 76.59 52.55 79.42 37.49 64.46 }
{ 65.70 70.98 62.60 48.89 67.59 73.68 51.46 77.55 62.28 64.62 53.95 66.86 54.37 81.99 35.07 60.77 }
{ 81.04 68.29 71.31 74.53 71.75 80.50 55.19 80.01 72.40 75.56 58.60 71.48 54.67 86.42 39.28 60.14 }
{ 70.25 64.43 50.70 67.98 61.70 68.88 41.78 63.04 61.80 61.66 58.96 55.29 53.30 74.84 39.85 63.85 }
{ 56.59 49.83 47.54 48.05 48.80 53.69 56.22 58.36 59.48 50.12 43.27 52.23 49.73 61.82 24.73 50.78 }
{ 63.07 65.14 60.88 64.05 63.59 68.71 50.39 67.48 56.20 74.25 57.66 64.72 49.31 77.56 38.70 69.60 }
{ 74.40 75.85 63.54 58.34 60.90 70.58 62.65 70.61 52.94 80.12 45.19 66.55 56.93 70.47 41.53 70.33 }
{ 77.63 83.83 66.00 64.84 72.50 80.28 59.14 85.22 76.03 79.55 57.21 69.55 60.86 90.57 43.11 64.04 }
{ 67.19 74.27 59.60 60.77 65.49 64.51 62.49 72.07 63.26 70.87 56.01 72.25 59.15 87.55 44.04 64.86 }
```

5. Profilez chaque implementation.

Tuile 2*2 :

Nvprof :

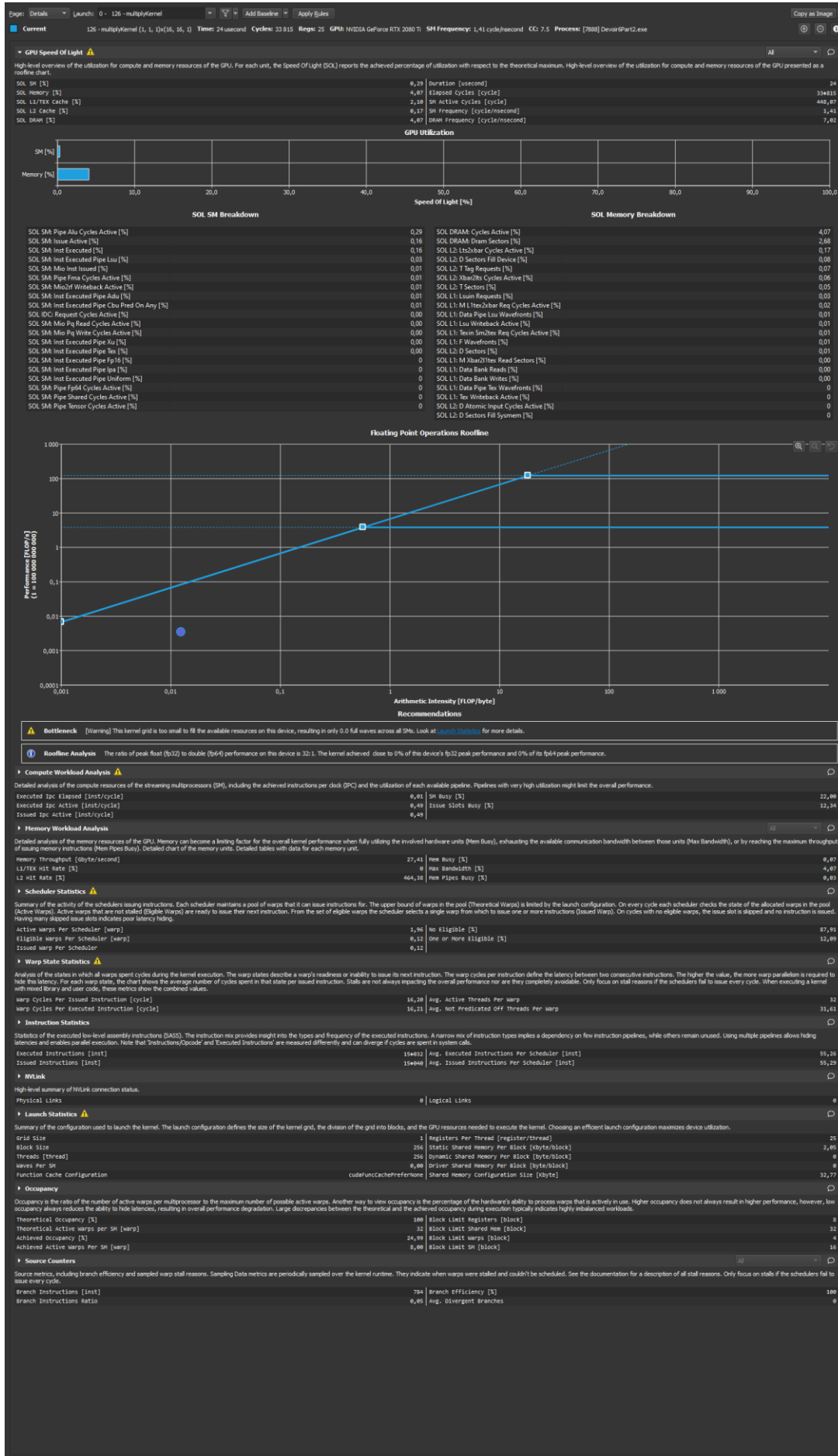
```
==23080== Profiling application: a
==23080== Profiling result:
   Type  Time(%)   Time     Calls   Avg        Min        Max  Name
GPU activities: 55.43%  4.8960us    1  4.8960us  4.8960us  4.8960us  multiplyKernel(float*, float const *, float const *, int)
              33.70%  2.9760us    2  1.4880us  1.1520us  1.8240us  [CUDA memcpy HtoD]
              10.87%   960ns      1    960ns    960ns    960ns    [CUDA memcpy DtoH]
API calls:    79.58%  200.94ms    3  66.980ms  5.7000us  200.93ms  cudaMalloc
              20.18%  50.947ms    1  50.947ms  50.947ms  50.947ms  cudaDeviceReset
              0.13%   330.50us    3  110.17us  51.500us  218.00us  cudaMemcpy
              0.05%   119.20us    3   39.733us  1.9000us  112.50us  cudaFree
              0.03%   68.100us    1   68.100us  68.100us  68.100us  cudaLaunchKernel
              0.02%   43.000us    1   43.000us  43.000us  43.000us  cudaDeviceSynchronize
              0.01%   15.600us   101    154ns    100ns    1.2000us  cuDeviceGetAttribute
              0.01%   14.200us    3   4.7330us  200ns    13.500us  cuDeviceGetCount
              0.00%   6.1000us    1   6.1000us  6.1000us  6.1000us  cudaSetDevice
              0.00%   1.7000us    2    850ns    200ns    1.5000us  cuDeviceGet
              0.00%   1.1000us    1   1.1000us  1.1000us  1.1000us  cuDeviceGetName
              0.00%    400ns     1    400ns    400ns    400ns    cuDeviceGetLuid
              0.00%    300ns     1    300ns    300ns    300ns    cuDeviceTotalMem
              0.00%    200ns     1    200ns    200ns    200ns    cuDeviceGetUuid
              0.00%    200ns     1    200ns    200ns    200ns    cudaGetLastError

C:\Users\lacommap\Desktop\CudaC\Devoir6Part2>
```

Nsight :

GEI1084-Architecture des ordinateurs et calcul accéléré

Jérémy POUPART and Messaoud AHMED OUAMEUR



Tuile 4*4 :

Nvprof :

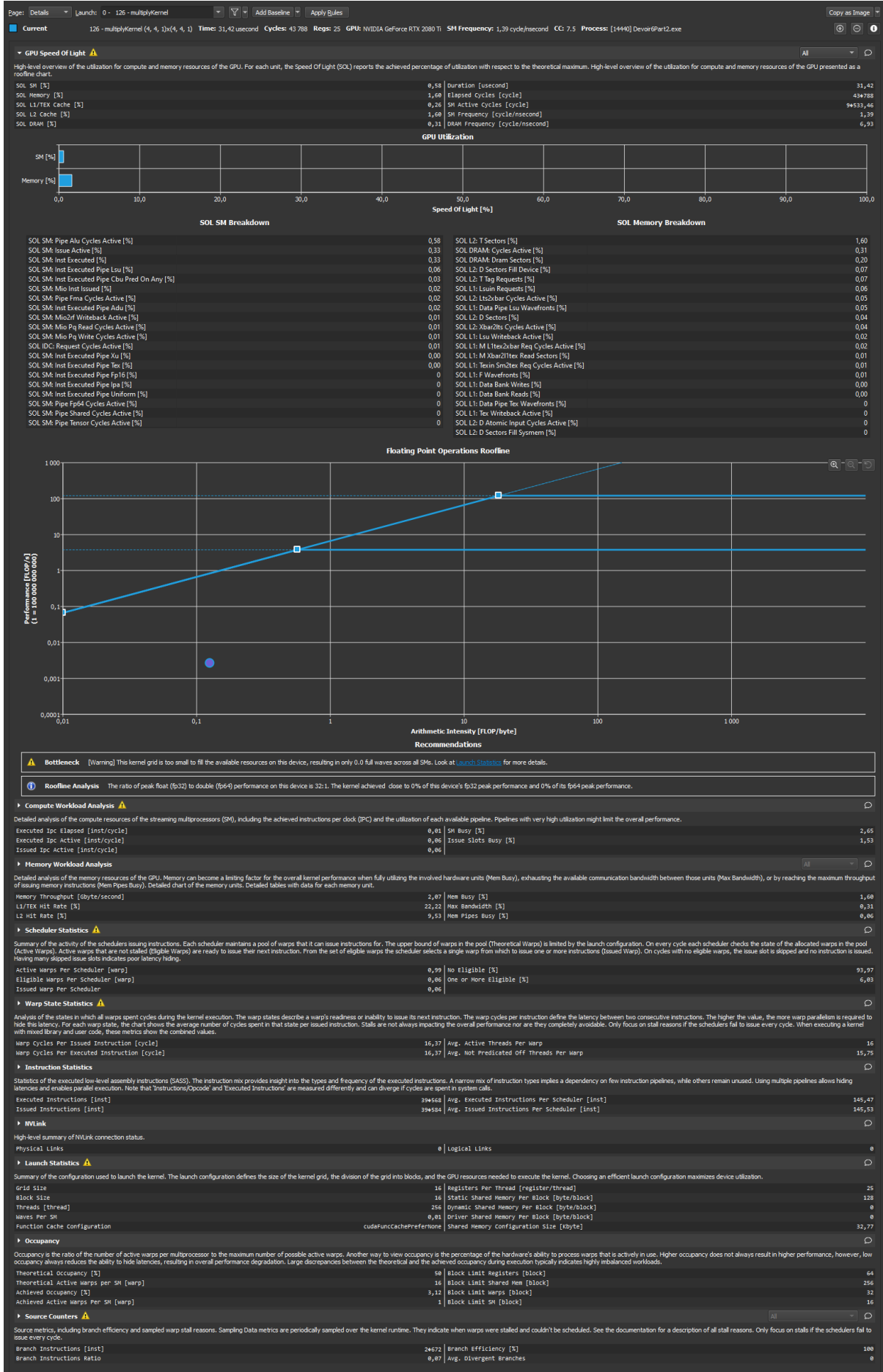
```
=6336== Profiling application: a
--6336== Profiling result:
Type      Time(%)      Time      Calls      Avg      Min      Max      Name
GPU activities: 52.20%  4.1600us      1  4.1600us  4.1600us  4.1600us  multiplyKernel(float*, float const *, float const *, int)
              35.35%  2.8170us      2  1.4080us  1.1210us  1.6960us  [CUDA memcpy HtoD]
              12.45%   992ns        1    992ns    992ns    992ns    [CUDA memcpy DtoH]
API calls: 79.29% 198.80ms      3  66.265ms  2.0000us  198.79ms  cudaMalloc
              20.48%  51.348ms      1  51.348ms  51.348ms  51.348ms  cudaDeviceReset
              0.14%  345.90us      3  115.30us  34.000us  259.20us  cudaMemcpy
              0.05%  117.90us      3   39.300us  1.9000us  111.70us  cudaFree
              0.02%  53.900us      1  53.900us  53.900us  53.900us  cudaLaunchKernel
              0.02%  40.700us      1  40.700us  40.700us  40.700us  cudaDeviceSynchronize
              0.01%  14.800us     101    146ns    100ns    1.0000us  cuDeviceGetAttribute
              0.00%  6.3000us      1  6.3000us  6.3000us  6.3000us  cudaSetDevice
              0.00%  6.1000us      3  2.0330us  300ns    5.4000us  cuDeviceGetCount
              0.00%  1.8000us      2    900ns    100ns    1.7000us  cuDeviceGet
              0.00%   800ns        1    800ns    800ns    800ns  cuDeviceGetName
              0.00%   300ns        1    300ns    300ns    300ns  cuDeviceTotalMem
              0.00%   300ns        1    300ns    300ns    300ns  cuDeviceGetLuid
              0.00%   200ns        1    200ns    200ns    200ns  cuDeviceGetUuid
              0.00%   200ns        1    200ns    200ns    200ns  cudaGetLastError

C:\Users\lacommap\Desktop\CudaC\Devoir6Part2>
```

Nsight :

GEI1084-Architecture des ordinateurs et calcul accéléré

Jérémy POUPART and Messaoud AHMED OUAMEUR



Tuile 8*8 :

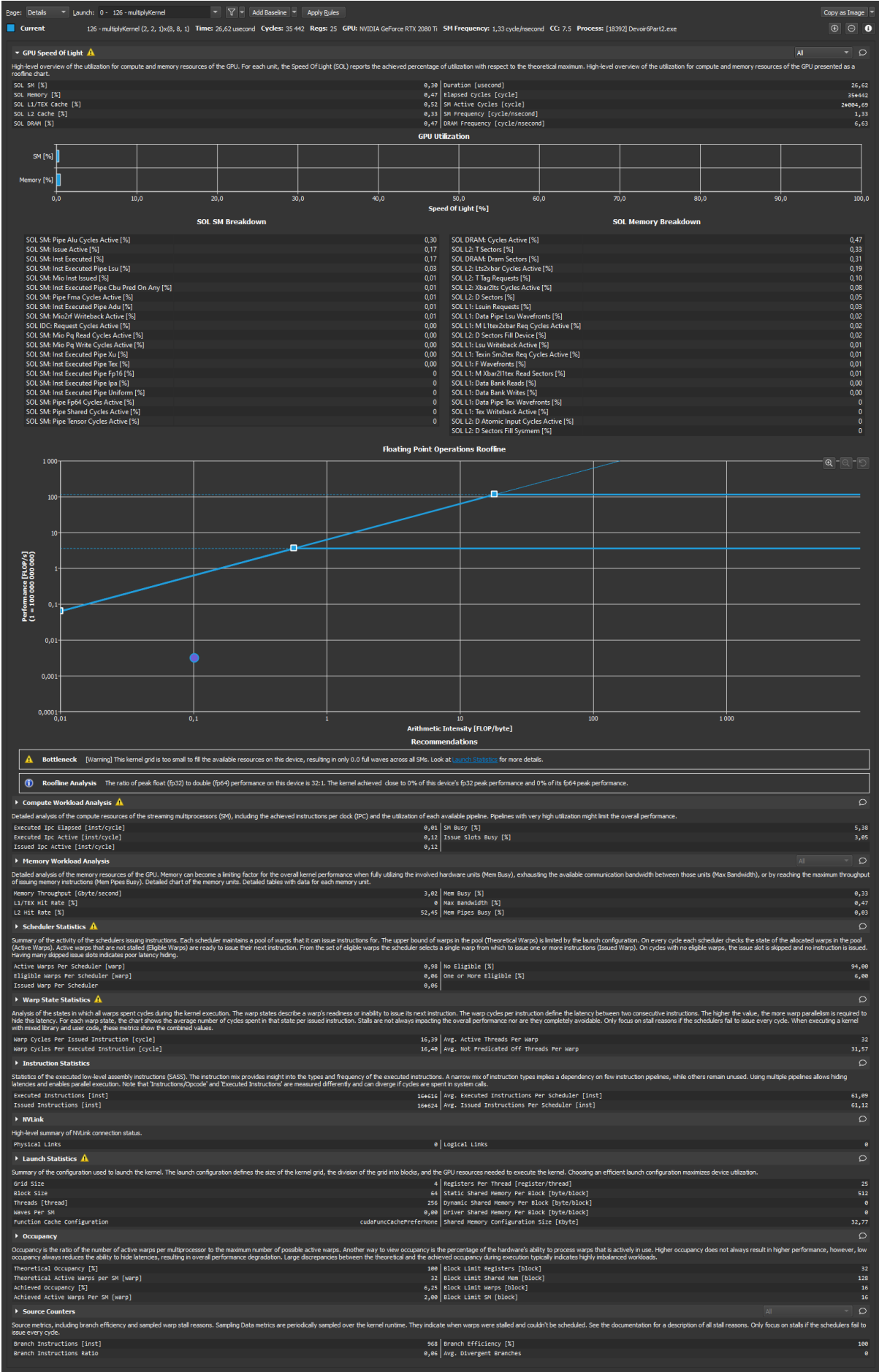
Nvprof :

```
--21300== Profiling application: a
--21300== Profiling result:
   Type  Time(%)   Time     Calls    Avg        Min        Max   Name
GPU activities:  47.06%  3.5840us      1  3.5840us  3.5840us  3.5840us  multiplyKernel(float*, float const *, float const *, int)
                40.76%  3.1040us      2  1.5520us  1.1200us  1.9840us  [CUDA memcpy HtoD]
                12.18%   928ns       1     928ns    928ns    928ns    [CUDA memcpy DtoH]
API calls:      79.02% 198.83ms      3  66.275ms  1.8000us  198.82ms  cudaMalloc
                20.77%  52.262ms     1  52.262ms  52.262ms  52.262ms  cudaDeviceReset
                0.11%   285.10us     3  95.033us  33.900us  201.60us  cudaMemcpy
                0.05%   114.50us     3  38.166us  1.8000us  108.30us  cudaFree
                0.02%   52.800us     1  52.800us  52.800us  52.800us  cudaLaunchKernel
                0.02%   40.000us     1  40.000us  40.000us  40.000us  cudaDeviceSynchronize
                0.01%   14.200us    101    140ns    100ns    800ns  cuDeviceGetAttribute
                0.00%   5.9000us     3  1.9660us   200ns   5.3000us  cuDeviceGetCount
                0.00%   5.5000us     1  5.5000us  5.5000us  5.5000us  cudaSetDevice
                0.00%   1.4000us     2     700ns    100ns   1.3000us  cuDeviceGet
                0.00%     600ns     1     600ns    600ns    600ns  cuDeviceGetName
                0.00%     300ns     1     300ns    300ns    300ns  cuDeviceTotalMem
                0.00%     300ns     1     300ns    300ns    300ns  cuDeviceGetLuid
                0.00%     200ns     1     200ns    200ns    200ns  cudaGetLastError
                0.00%     100ns     1     100ns    100ns    100ns  cuDeviceGetUuid

C:\Users\lacommap\Desktop\CudaC\Devoir6Part2>
```

Nsight :

GEI1084-Architecture des ordinateurs et calcul accéléré
Jérémy POUPART and Messaoud AHMED OUAMEUR



Tuile 16*16 :

Nvprof :

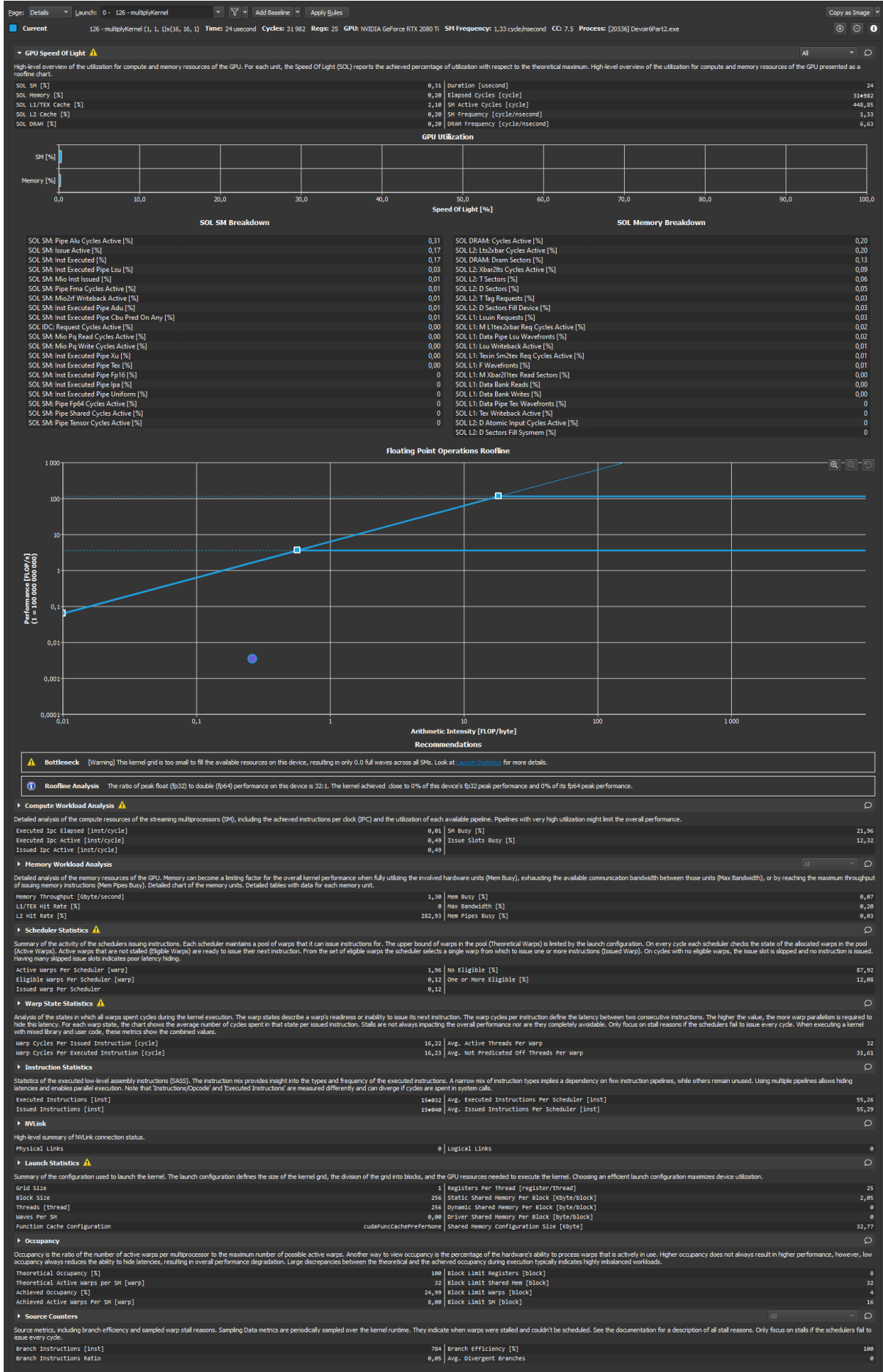
```
=15352== Profiling application: a
=15352== Profiling result:
Type      Time(%)      Time      Calls      Avg      Min      Max      Name
GPU activities: 49.79%  3.7760us    1  3.7760us  3.7760us  3.7760us  multiplyKernel(float*, float const *, float const *, int)
              37.13%  2.8160us    2  1.4080us  1.1200us  1.6960us  [CUDA memcpy HtoD]
              13.08%   992ns      1    992ns    992ns    992ns    [CUDA memcpy DtoH]
API calls:   79.88%  203.65ms    3  67.883ms  1.7000us  203.64ms  cudaMalloc
              19.89%  50.712ms    1  50.712ms  50.712ms  50.712ms  cudaDeviceReset
              0.13%   342.10us    3  114.03us  34.400us  256.10us  cudaMemcpy
              0.05%   118.00us    3  39.333us  1.6000us  112.30us  cudaFree
              0.02%   55.100us    1  55.100us  55.100us  55.100us  cudaLaunchKernel
              0.02%   41.300us    1  41.300us  41.300us  41.300us  cudaDeviceSynchronize
              0.01%   15.300us   101    151ns    100ns    1.4000us  cuDeviceGetAttribute
              0.00%   6.1000us    3  2.0330us  100ns    5.8000us  cuDeviceGetCount
              0.00%   5.8000us    1  5.8000us  5.8000us  5.8000us  cudaSetDevice
              0.00%   1.9000us    2    950ns    100ns    1.8000us  cuDeviceGet
              0.00%    700ns     1    700ns    700ns    700ns    cuDeviceGetName
              0.00%    400ns     1    400ns    400ns    400ns    cuDeviceGetLuid
              0.00%    300ns     1    300ns    300ns    300ns    cuDeviceGetTotalMem
              0.00%    200ns     1    200ns    200ns    200ns    cuDeviceGetUuid
              0.00%    200ns     1    200ns    200ns    200ns    cudaGetLastError

C:\Users\lacommap\Desktop\CudaC\Devoir6Part2>
```

Nsight :

GEI1084-Architecture des ordinateurs et calcul accéléré

Jérémy POUPART and Messaoud AHMED OUAMEUR



6. Discutez de la vitesse d'exécution du kernel versus les ressources utilisées (shared memory, registres, etc.).

7. Comparez et discutez des résultats par rapport à ceux obtenus à la PARTIE 1.

REFERENCES

[2] D. Kirk and W.-M. Hwu, *Programming Massively Parallel Processors- A Hands-on Approach*, 3rd Edition, Morgan Kaufmann, 2017. (Elsevier Science & Technology, ISBN: 978-0-12-811986-0).

