Institut Supérieur des Sciences Appliquées et de Technologie de Sousse	ALGORITHMIQUE ET ARCHITECTURES PARALLELES TRAVAUX PRATIQUES N°1 PROF : TAHAR ALIMI	Section : FIA2-GL A. A. : 2022-2023 Cuda

Exercice 1

Ecrire un programme qui additionne deux vecteurs des valeurs de taille seize avec les éléments de type float
- Utilisation d'un bloc et des threads avec l'indice : ThreadIdx.x

Exercice 2

Ecrire un programme qui additionne deux matrices des valeurs de taille quatre sur quatre avec les éléments de type float.

- Utilisation de la structure Dim3 pour le passage des arguments

- Utilisation d'un bloc et des threads avec les indices : ThreadIdx.x, ThreadIdx.y

Exercice 3

Ecrire un programme qui additionne deux matrices dans une grille bidimensionnelle de taille deux sur deux avec une organisation bidimensionnelle des blocs (2x2)

- Utilisation de la structure Dim3 pour le passage des arguments

- Utilisation des blocs avec les 'indices : BlockIdx.x, BlockIdx.y

- Utilisation des threads avec les 'indices : ThreadIdx.x, ThreadIdx.y

Exercice 4

Ecrire un programme qui multiplie deux matrices carrées (A et B) de taille N*N avec les éléments de type float.
Les valeurs dans les matrices sont initialisées dans la fonction principale (CPU) par la fonction rand().
Pour comparer les performances écrire deux partitions de cet algorithme, une s'exécutant dans la partie CPU et une dans la partie GPU.
Utiliser les événements (CUDA) pour calculer le temps d'exécution de chaque partition.

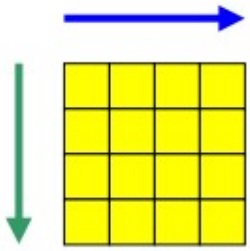
Correction

Exercice 1

```
#include __global__ void addVect(float* in1, float* in2, float* out)
{
	int i = threadIdx.x;
	out[i] = in1[i] + in2[i];
}
int main()
{
	 float v1[]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16};
	float v2[]={1.0,1.1,1.2,1.3,1.4,1.5,1.6,1.7,1.8,1.9,2.0,2.1,2.2,2.3,2.4,2.5};
	int memsize = sizeof(v1); int vsize = memsize/sizeof(float);
	float res[vsize];
	float* Cv1; cudaMalloc((void **)&Cv1,memsize);
	float* Cv2;

	cudaMalloc((void **)&Cv2,memsize);
	float* Cres; cudaMalloc((void **)&Cres,memsize);

	cudaMemcpy(Cv1,v1,memsize,cudaMemcpyHostToDevice);
	cudaMemcpy(Cv2,v2,memsize,cudaMemcpyHostToDevice);

	addVect<<>>(Cv1,Cv2,Cres);

	cudaMemcpy(res,Cres,memsize,cudaMemcpyDeviceToHost);
	int i=0;
	printf("res= { ");
	for(i=0;i<vsize ;i++)
	{
		printf("%2.2f ", res[i]);
	}

	printf("}\n");
}
```

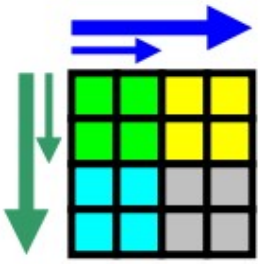
Exercice 2

```
#include __global__ void addVect(float* in1, float* in2, float* out)
{
	int i = threadIdx.x + threadIdx.y*blockDim.x;
	out[i] = in1[i] + in2[i];
}

 int main()
{
	float v1[]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16};
	float v2[]={1.0,1.1,1.2,1.3,1.4,1.5,1.6,1.7,1.8,1.9,2.0,2.1,2.2,2.3,2.4,2.};
	int memsize = sizeof(v1);
	int vsize = memsize/sizeof(float);
	float res[vsize]; float* Cv1; dim3 BlXY(vsize/4,4);

	cudaMalloc((void **)&Cv1,memsize);
	float* Cv2; cudaMalloc((void **)&Cv2,memsize);
	float* Cres; cudaMalloc((void **)&Cres,memsize);

	cudaMemcpy(Cv1,v1,memsize,cudaMemcpyHostToDevice);
	cudaMemcpy(Cv2,v2,memsize,cudaMemcpyHostToDevice);

	addVect<<>>(Cv1,Cv2,Cres);

	cudaMemcpy(res,Cres,memsize,cudaMemcpyDeviceToHost);
	int i=0; printf("res= { ");
	for(i=0;i<vsize ;i++)
	{
		printf("%2.2f ", res[i]);
	}
	printf("}\n");
}
```



Exercice 3

```
#include __global__ void addVect(float* in1, float* in2, float* out)
{
	int i = threadIdx.x + threadIdx.y*blockDim.x + blockIdx.x + blockIdx.y*gridDim.x;
	out[i] = in1[i] + in2[i];
}

int main()
{
	float v1[]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16};
	float v2[]={1.0,1.1,1.2,1.3,1.4,1.5,1.6,1.7,1.8,1.9,2.0,2.1,2.2,2.3,2.4,2.5};
	int memsize = sizeof(v1);
	int vsize = memsize/sizeof(float);
	int bsize = vsize/4;
	float res[vsize];
	float* Cv1;
	dim3 BlXY(bsize/2,2); // block 2x2
	dim3 GrXY(2,2); // grid 2x2

	cudaMalloc((void **)&Cv1,memsize);
	float* Cv2;
	cudaMalloc((void **)&Cv2,memsize);
	float* Cres;
	cudaMalloc((void **)&Cres,memsize);
	cudaMemcpy(Cv1,v1,memsize,cudaMemcpyHostToDevice);
	cudaMemcpy(Cv2,v2,memsize,cudaMemcpyHostToDevice);

	addVect<<>>(Cv1,Cv2,Cres);

	cudaMemcpy(res,Cres,memsize,cudaMemcpyDeviceToHost);

	int i=0; printf("res= { ");
	for(i=0;i<vsize ;i++)
	{
		printf("%2.2f ", res[i]);
	}
	printf("}\n");
}
```



Exercice 4

```
#define DIM 256
__global__ void matrix_mul(int* dev_A, int* dev_B, int* dev_C, int Width)
{
	 // 2D thread ID
	int tx = threadIdx.x;
	int ty = threadIdx.y;
	int Pvalue =0;

	for(int k=0;k<Width ;++k)
	{
		int Ael=dev_A[ty*Width + k];
		int Bel=dev_B[k*Width +tx];
		Pvalue += Ael*Bel;
	}
}
int* random_block(int size)
{
	int *ptr; ptr = (int *)malloc(size*sizeof(int));
	for (int i=0 ; i<size ; i++)
		ptr[i] = rand();
	return ptr;
}
int main(void)
{
	int *buffA= (int *)random_block(DIM*DIM);
	int *buffB= (int *)random_block(DIM*DIM);
	int *buffC = (int *) malloc(DIM*DIM*sizeof(int));
	int *dev_A, *dev_B, *dev_C;
	float elapsedTime; int impl=0;
	cudaEvent_t start,stop;
	printf("Test GPU or CPU [0,1]:");
	scanf("%d",&impl);
	cudaEventCreate(&start);
	cudaEventCreate(&stop);
	cudaEventRecord(start,0);
	cudaMalloc((void **)&dev_A,DIM*DIM*sizeof(int));
	cudaMalloc((void **)&dev_B,DIM*DIM*sizeof(int));
	cudaMalloc((void **)&dev_C,DIM*DIM*sizeof(int));
	cudaMemcpy(dev_A,buffA,DIM*DIM*sizeof(int),cudaMemcpyHostToDevice);
	cudaMemcpy(dev_B,buffB,DIM*DIM*sizeof(int),cudaMemcpyHostToDevice);
	dim3 dimBlock(DIM,DIM);
	dim3 dimGrid(1,1);
	if(impl==0) // GPU
		matrix_mul<<>>(dev_A,dev_B,dev_C,DIM);
	else // CPU
		for(int i=0 ; i<DIM ; i++)
			for(int j=0 ; j<DIM ; j++)
				for(int k=0 ; k<DIM ; k++)
					buffC[j + i*DIM] += buffA[k+j*DIM]*buffB[j+k*DIM];
	cudaMemcpy(buffC,dev_C,DIM*DIM*sizeof(int),cudaMemcpyDeviceToHost);
	cudaEventRecord(stop,0);
	cudaEventSynchronize(stop);
	cudaEventElapsedTime(&elapsedTime,start,stop);
	if(impl==0)
		printf("GPU time to multiply %d*%d matrix: %3.2f ms\n",DIM,DIM,elapsedTime);
	else
		printf("CPU time to multiply %d*%d matrix: %3.2f ms\n",DIM,DIM,elapsedTime);

	cudaFree(dev_A);
	cudaFree(dev_B);
	cudaFree(dev_C);
	free(buffA);
	free(buffB);
	free(buffC);
}
```