Rubén Vera - 241456
Sara Soriano - 240007

REPORT LAB4

- **Create a code in CUDA to apply the four previously described transformations to an input grayscale.**

  Per tal de poder crear un codi en CUDA que apliqui els filtres *invert*, *detect*, *smooth* i *enhance*, el que hem anat fent és convertir el codi que teníem del lab anterior a codi en CUDA. Per fer-ho el primer que hem fet ha sigut assignar la memòria de cada variable al *device*, amb la funció cudaMalloc() i els paràmetres necessaris. El primer d'ells és la variable on es vol allocar la memòria i el segon el tamany d'aquest espai de memòria.

  A continuació, hem declarat la dimensió de la *grid* i dels *blocks* que utilitzarem per cridar a totes les funcions, tal i com se'ns va explicar a classe de teoria.

  A més a més, hem hagut de copiar les dades del *host* al *device* de la variable "image", de manera que tindrem en "d_image" les dades que conté "image" pero al device, és a dir, a la GPU. Per fer-ho, hem utilitzat cudaMemcpy() amb tercer paràmetre cudaMemcpyHostToDevice. Seguidament, hem cridat a cadascuna de les funcions, que executaran els filtres, amb els paràmetres corresponents del device per a que els filtres s'apliquin a la GPU. Posteriorment, hem hagut de copiar les dades del *device* al *host*, és a dir a la CPU, un cop aplicats els filtres, per tant, hem utilitzat la funció cudaMemcpy() amb tercer paràmetre cudaMemcpyDeviceToHost.
  Finalment, hem alliberat l'espai de cadascuna de les variables del *device* amb la funció de CUDA cudaFree().
  Per acabar amb la funció main() hem calculat el temps que es triga en aplicar tots els filtres, per fer-ho hem utilitzat les funcions de CUDA corresponents. Primer hem hagut de crear els diferents events, "start" i "stop", mitjançant cudaEventCreate(). Posteriorment, en la posició correcta em calculat el temps de cadascuna de les variables amb cudaEventRecord() i hem sincronitzat "stop" per tal d'assegurar-nos que tot havia finalitzat (cudaEventSynchronize()). Finalment, hem guardat la diferència del temps en la variable "runtime".

Rubén Vera - 241456
Sara Soriano - 240007

```
int main (int argc, char *argv[])
{
   int    nx,ny;
   char   filename[250];
   float runtime;
   cudaEvent_t start, stop;
   cudaEventCreate(&start);
   cudaEventCreate(&stop);

   /* Get parameters */
   if (argc != 4)
   {
      printf ("Usage: %s image_name N M \n", argv[0]);
      exit (1);
   }
   sprintf(filename, "%s.txt", argv[1]);
   nx  = atoi(argv[2]);
   ny  = atoi(argv[3]);

   printf("%s %d %d\n", filename, nx, ny);

   /* Allocate CPU and GPU pointers */

   int*   image=(int *) malloc(sizeof(int)*nx*ny);
   int*   image_invert  = (int *) malloc(sizeof(int)*nx*ny);
   int*   image_smooth  = (int *) malloc(sizeof(int)*nx*ny);
   int*   image_detect  = (int *) malloc(sizeof(int)*nx*ny);
   int*   image_enhance = (int *) malloc(sizeof(int)*nx*ny);

   int*   d_image, *d_image_invert, *d_image_smooth, *d_image_detect, *d_image_enhance;
   cudaMalloc((void **) &d_image, sizeof(int)*nx*ny);
   cudaMalloc((void **) &d_image_invert, sizeof(int)*nx*ny);
   cudaMalloc((void **) &d_image_smooth, sizeof(int)*nx*ny);
   cudaMalloc((void **) &d_image_detect, sizeof(int)*nx*ny);
   cudaMalloc((void **) &d_image_enhance, sizeof(int)*nx*ny);
```

```
   /* Read image and save in array imgage */
   readimg(filename,nx,ny,image);

   dim3 dimBlock(B,B,1);
   int dimgx = (nx+B-1)/B;
   int dimgy = (ny+B-1)/B;
   dim3 dimGrid(dimgx,dimgy, 1);

   cudaEventRecord(start);

   cudaMemcpy(d_image, image, sizeof(int)*nx*ny, cudaMemcpyHostToDevice);
   invert<<<dimGrid, dimBlock>>>(d_image, d_image_invert, nx, ny);
   smooth<<<dimGrid, dimBlock>>>(d_image, d_image_smooth, nx, ny);
   detect<<<dimGrid, dimBlock>>>(d_image, d_image_detect, nx, ny);
   enhance<<<dimGrid, dimBlock>>>(d_image, d_image_enhance, nx, ny);
   cudaMemcpy(image_invert, d_image_invert, sizeof(int)*nx*ny, cudaMemcpyDeviceToHost);
   cudaMemcpy(image_smooth, d_image_smooth, sizeof(int)*nx*ny, cudaMemcpyDeviceToHost);
   cudaMemcpy(image_detect, d_image_detect, sizeof(int)*nx*ny, cudaMemcpyDeviceToHost);
   cudaMemcpy(image_enhance, d_image_enhance, sizeof(int)*nx*ny, cudaMemcpyDeviceToHost);

   cudaEventRecord(stop);
   cudaEventSynchronize(stop);
   cudaEventElapsedTime(&runtime, start, stop);

   printf("Total time: %f\n", runtime);

   /* Save images */
   char fileout[255]={0};
   sprintf(fileout, "%s-inverse.txt", argv[1]);
   saveimg(fileout,nx,ny,image_invert);
   sprintf(fileout, "%s-smooth.txt", argv[1]);
   saveimg(fileout,nx,ny,image_smooth);
   sprintf(fileout, "%s-detect.txt", argv[1]);
   saveimg(fileout,nx,ny,image_detect);
   sprintf(fileout, "%s-enhance.txt", argv[1]);
   saveimg(fileout,nx,ny,image_enhance);

   /* Deallocate CPU and GPU pointers*/
   free(image);
   free(image_invert);
   free(image_smooth);
   free(image_detect);
   free(image_enhance);

   cudaFree(d_image);
   cudaFree(d_image_invert);
   cudaFree(d_image_smooth);
   cudaFree(d_image_detect);
   cudaFree(d_image_enhance);
}
```

Com en les funcions ja teniem allocada la memòria en le *device* no hem hagut de fer copys ni updates ja que farà en el main() en el moment corresponent. Tal i com se'ns ha ensenyat, primer hem de saber l'index unic de cada thread, per fer-ho, com s'indica al codi calculem indx = threadIdx.x + blockIdx.x * blockDim.x i d'igual manera per la y. Un cop sabem l'index fem dos ifs que reemplacen a dos for loops ja que volem que cada thread faci una posició diferent. Per tant, si l'índex del thread és major o igual a 0 i està dins la imatge, farem que calculi el valor de la seva posició, i per tant a aquella posició se li haurà aplicat el filtre. Aquesta estructura s'ha seguit amb totes les funcions tenint en compte les *boundary conditions*, les quals són que quan el thread tingui la "indx" o la "indy" igual a 0 o igual a nx-1 o ny-1 el valor serà 0. En cas contrari, farem el seu càlcul corresponent segons el filtre. A més, hem comprovat que els valors estiguin dins l'interval [0, 255]. Si no és així s'assigna el valor 0 si es un valor negatiu i si es un valor més gran a 255, s'assigna 255.

Rubén Vera - 241456
Sara Soriano - 240007

```
/*invert*/
__global__ void invert(int* image, int* image_invert, int nx, int ny){
  int indx = threadIdx.x + blockIdx.x * blockDim.x;
  int indy = threadIdx.y + blockIdx.y * blockDim.y;
  if(indx>=0 && indx < nx){
    if(indy>=0 && indy < ny){
      image_invert[pixel(indx,indy,nx)] = 255 - image[pixel(indx,indy,nx)];
    }
  }
}
```

```
/*smooth*/
__global__ void smooth(int* image, int* image_smooth, int nx, int ny){
 int indx = threadIdx.x + blockIdx.x * blockDim.x;
 int indy = threadIdx.y + blockIdx.y * blockDim.y;

 if(indx >= 0 && indx < nx){
   if(indy >= 0 && indy < ny){
     if(indx == 0 || indy == 0 || indx == nx-1 || indy == ny-1){
       image_smooth[pixel(indx,indy,nx)] = 0;
     }
     else{
       image_smooth[pixel(indx,indy,nx)] =(image[pixel(indx-1,indy+1,nx)] + image[pixel(indx,indy+1,nx)] +image[pixel(indx+1,indy+1,nx)] + image[pixel(indx-1,indy,nx)] + image[pixel(indx,indy,nx)] + image[pixel(indx+1,indy,nx)] + image[
pixel(indx-1,indy-1,nx)] + image[pixel(indx,indy-1,nx)] + image[pixel(indx+1,indy-1,nx)]);
       image_smooth[pixel(indx,indy,nx)] = (int) image_smooth[pixel(indx,indy,nx)] / 9;
       if(image_smooth[pixel(indx, indy, nx)] < 0){
         image_smooth[pixel(indx, indy, nx)] = 0;
       } else if (image_smooth[pixel(indx, indy, nx)] > 255){
         image_smooth[pixel(indx, indy, nx)] = 255;
       }
     }
   }
 }
}
```

```
/*detect*/
__global__ void detect(int* image, int* image_detect, int nx, int ny){
 int indx = threadIdx.x + blockIdx.x * blockDim.x;
 int indy = threadIdx.y + blockIdx.y * blockDim.y;

 if(indx>=0 && indx < nx){
   if(indy >= 0 && indy < ny){
     if(indx == 0 || indy == 0 || indx == nx-1 || indy == ny-1){
       image_detect[pixel(indx,indy,nx)] = 0;
     }
     else{
       image_detect[pixel(indx,indy,nx)] = image[pixel(indx-1,indy,nx)] + image[pixel(indx+1,indy,nx)] + image[pixel(indx,indy-1,nx)] + image[pixel(indx,indy+1,nx)] - 4 * image[pixel(indx,indy,nx)];
       if(image_detect[pixel(indx, indy, nx)] < 0){
         image_detect[pixel(indx, indy, nx)] = 0;
       } else if (image_detect[pixel(indx, indy, nx)] > 255){
         image_detect[pixel(indx, indy, nx)] = 255;
       }
     }
   }
 }
}
```

```
/*enhance*/
__global__ void enhance(int* image,int *image_enhance,int nx, int ny){
 int indx = threadIdx.x + blockIdx.x * blockDim.x;
 int indy = threadIdx.y + blockIdx.y * blockDim.y;

 if(indx >= 0 && indx < nx){
   if(indy >= 0 && indy < ny){
     if(indx == 0 || indy == 0 || indx == nx-1 || indy == ny-1){
       image_enhance[pixel(indx,indy,nx)] = 0;
     }
     else{
       image_enhance[pixel(indx,indy,nx)] = 9*image[pixel(indx,indy,nx)] -(image[pixel(indx-1,indy,nx)] +image[pixel(indx+1,indy,nx)] + image[pixel(indx,indy-1,nx)] + image[pixel(indx,indy+1,nx)]);
       if(image_enhance[pixel(indx, indy, nx)] < 0){
         image_enhance[pixel(indx, indy, nx)] = 0;
       } else if (image_enhance[pixel(indx, indy, nx)] > 255){
         image_enhance[pixel(indx, indy, nx)] = 255;
       }
     }
   }
 }
}
```

- **Optimize the previous code such that the four images can be generated concurrently on the GPU using streams.**

  Tal i com se'ns demanava en l'enunciat de la pràctica, per tal d'optimitzar el codi de l'apartat (exercici) anterior, hem utilitzat els streams i les asynchronous operations de CUDA.
  Per utilitzar-los, el primer que hem hagut de fer és assignar variables d'aquest tipus, el tipus stream, i després inicialitzar-les mitjançant la funció cudaStreamCreate(). A més, l'espai de les variables del *host* que posteriorment actualitzarem amb la informació del *device* l'hem hagut d'allocar mitjançant cudaMallocHost() amb els paràmetres necessaris.
  El següent canvi que hem realitzat és que a les funcions li passàvem l'stream on volíem que es fes l'aplicació del filtre de la imatge, assignant un stream a cada funció diferent per tal que es facin tots de manera concurrent a la GPU. A més, hem hagut de passar com a paràmetre la shared memory que utilitzem. Com en aquest cas no hi ha memòria compartida, hem posat un 0. A continuació, hem assignat un stream diferent a cada cudaMemcpyAsync(), de tal manera que les dades de cada imatge, un cop aplicat el filtre, es copiïn de la GPU a la CPU de manera concurrent. Finalment, la memòria que hem

allocat amb cudaMallocHost() l'haurem d'alliberar amb cudaFreeHost(), la resta de frees es mantenen com a l'exercici anterior.

```
/* main program */
int main (int argc, char *argv[])
{
   int    nx,ny;
   char   filename[250];
   float runtime;
   cudaEvent_t start, stop;
   cudaStream_t stream1;
   cudaStream_t stream2;
   cudaStream_t stream3;
   cudaStream_t stream4 ;
   cudaEventCreate(&start);
   cudaEventCreate(&stop);

   /* Get parameters */
   if (argc != 4)
   {
      printf ("Usage: %s image_name N M \n", argv[0]);
      exit (1);
   }
   sprintf(filename, "%s.txt", argv[1]);
   nx  = atoi(argv[2]);
   ny  = atoi(argv[3]);

   printf("%s %d %d\n", filename, nx, ny);
```

```
   /* Allocate CPU and GPU pointers */

   int*   image=(int *) malloc(sizeof(int)*nx*ny);

   int* image_invert, *image_smooth, *image_detect, *image_enhance;
   cudaMallocHost((void**) &image_invert, sizeof(int)*nx*ny);
   cudaMallocHost((void**) &image_smooth, sizeof(int)*nx*ny);
   cudaMallocHost((void**) &image_detect, sizeof(int)*nx*ny);
   cudaMallocHost((void**) &image_enhance, sizeof(int)*nx*ny);

   int*   d_image, *d_image_invert, *d_image_smooth, *d_image_detect, *d_image_enhance;
   cudaMalloc((void **) &d_image, sizeof(int)*nx*ny);
   cudaMalloc((void **) &d_image_invert, sizeof(int)*nx*ny);
   cudaMalloc((void **) &d_image_smooth, sizeof(int)*nx*ny);
   cudaMalloc((void **) &d_image_detect, sizeof(int)*nx*ny);
   cudaMalloc((void **) &d_image_enhance, sizeof(int)*nx*ny);
```

```
  /* Read image and save in array imgage*/

   readimg(filename,nx,ny,image);

   dim3 dimBlock(B,B,1);
   int dimgx = (nx+B-1)/B;
   int dimgy = (ny+B-1)/B;
   dim3 dimGrid(dimgx,dimgy, 1);

   cudaStreamCreate(&stream1);
   cudaStreamCreate(&stream2);
   cudaStreamCreate(&stream3);
   cudaStreamCreate(&stream4);

   cudaEventRecord(start);

   cudaMemcpy(d_image, image, sizeof(int)*nx*ny, cudaMemcpyHostToDevice);

   invert<<<dimGrid, dimBlock,0, stream1>>>(d_image, d_image_invert, nx, ny);
   smooth<<<dimGrid, dimBlock,0, stream2>>>(d_image, d_image_smooth, nx, ny);
   detect<<<dimGrid, dimBlock,0, stream3>>>(d_image, d_image_detect, nx, ny);
   enhance<<<dimGrid, dimBlock,0, stream4>>>(d_image, d_image_enhance, nx, ny);

   cudaMemcpyAsync(image_invert, d_image_invert, sizeof(int)*nx*ny, cudaMemcpyDeviceToHost, stream1);
   cudaMemcpyAsync(image_smooth, d_image_smooth, sizeof(int)*nx*ny, cudaMemcpyDeviceToHost,stream2);
   cudaMemcpyAsync(image_detect, d_image_detect, sizeof(int)*nx*ny, cudaMemcpyDeviceToHost,stream3);
   cudaMemcpyAsync(image_enhance, d_image_enhance, sizeof(int)*nx*ny, cudaMemcpyDeviceToHost,stream4);

   cudaEventRecord(stop);
   cudaEventSynchronize(stop);
   cudaEventElapsedTime(&runtime, start, stop);
```

```
    /* Save images*/
    char fileout[255]={0};
    sprintf(fileout, "%s-inverse.txt", argv[1]);
    saveimg(fileout,nx,ny,image_invert);
    sprintf(fileout, "%s-smooth.txt", argv[1]);
    saveimg(fileout,nx,ny,image_smooth);
    sprintf(fileout, "%s-detect.txt", argv[1]);
    saveimg(fileout,nx,ny,image_detect);
    sprintf(fileout, "%s-enhance.txt", argv[1]);
    saveimg(fileout,nx,ny,image_enhance);

   /* Deallocate CPU and GPU pointers*/
    free(image);

    cudaFreeHost(image_invert);
    cudaFreeHost(image_smooth);
    cudaFreeHost(image_detect);
    cudaFreeHost(image_enhance);

    cudaFree(d_image);
    cudaFree(d_image_invert);
    cudaFree(d_image_smooth);
    cudaFree(d_image_detect);
    cudaFree(d_image_enhance);

    cudaStreamDestroy(stream1);
    cudaStreamDestroy(stream2);
    cudaStreamDestroy(stream3);
    cudaStreamDestroy(stream4);

}
```

Finalment, volem recalcar les diferències de temps ja que, en el primer exercici hem obtingut un temps de 30 ms. En el segon exercici, un cop optimitzat el codi, hem reduït aquest temps a 5 ms. De manera que, un cop hem fet que es generin les imatges de manera concurrent a la GPU veiem que l'optimització ha sigut aconseguida amb éxit, utilitzant les diferents eines que té cuda per fer-ho possible.