PRACTICA 2

Autores: Íñigo Gómez Carvajal y Jon Zorrilla Gamboa

```
!nvcc --version
!nvidia-smi
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2021 NVIDIA Corporation
Built on Sun_Feb_14_21:12:58_PST_2021
Cuda compilation tools, release 11.2, V11.2.152
Build cuda_11.2.r11.2/compiler.29618528_0
Thu Nov 17 09:07:44 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   61C    P8    13W /  70W |      0MiB / 15109MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

```
!pwd
!ls -la .
```

```
/content
total 16
drwxr-xr-x 1 root root 4096 Nov 15 14:31 .
drwxr-xr-x 1 root root 4096 Nov 17 09:06 ..
drwxr-xr-x 4 root root 4096 Nov 15 14:30 .config
drwxr-xr-x 1 root root 4096 Nov 15 14:31 sample_data
```

```
!mkdir workcuda
!ls -la
```

```
total 20
drwxr-xr-x 1 root root 4096 Nov 17 09:07 .
drwxr-xr-x 1 root root 4096 Nov 17 09:06 ..
drwxr-xr-x 4 root root 4096 Nov 15 14:30 .config
drwxr-xr-x 1 root root 4096 Nov 15 14:31 sample_data
drwxr-xr-x 2 root root 4096 Nov 17 09:07 workcuda
```

```
%cd /content/workcuda/
!ls -la
!pwd
```

```
/content/workcuda
total 8
drwxr-xr-x 2 root root 4096 Nov 17 09:07 .
drwxr-xr-x 1 root root 4096 Nov 17 09:07 ..
/content/workcuda
```

PARTE 1: Programación GPGPU .

```
%%writefile suma0.cu

#include <iostream>
#include <math.h>

#include <time.h>
#include <sys/time.h>

void add(int n, float *x, float *y) {

    for (int i =0; i < n; i++ ){
        y[i]=x[i]+y[i];
    }
}


int main(void) {

    int N = 1 <<20;  // N = 2^20 = 1024*1024= 1.048.576
    struct timeval tv;
    unsigned long long start;

    float *x = new float[N];
    float *y = new float[N];

    for (int i =0; i < N; i++ ){
        x[i]= 1.0f;
        y[i]= 2.0f;
    }
    gettimeofday(&tv, NULL);
    start=tv.tv_sec*1000000+tv.tv_usec;

    add(N, x, y);

    gettimeofday(&tv, NULL);
    printf("Tiempo del cálculo : %lf ms\n", (tv.tv_sec*1000000+tv.tv_usec - start)/1000.0)
```

Se ha guardado correctamente

```
    for (int i=0; i <N; i++){
        maxError=fmax(maxError,fabs(y[i]-3.0f));
        if (y[i] != 3.0) contError++;
    }
    std::cout << "suma de " << N << " Elementos" << std::endl;
    std::cout << "Número de Errores: " <<contError << std::endl;
    std::cout << "Max error: " <<maxError << std::endl;


    delete [] x;
    delete [] y;
    return 0;
}
```

```
Writing suma0.cu
```

Comprobamos el fichero que se ha escrito

```
!ls -la
%cat suma0.cu
```

```
total 12
drwxr-xr-x 2 root root 4096 Nov 12 17:46 .
drwxr-xr-x 1 root root 4096 Nov 12 17:46 ..
-rw-r--r-- 1 root root 1091 Nov 12 17:46 suma0.cu

#include <iostream>
#include <math.h>

#include <time.h>
#include <sys/time.h>

void add(int n, float *x, float *y) {

    for (int i =0; i < n; i++ ){
        y[i]=x[i]+y[i];
    }
}


int main(void) {

    int N = 1 <<20;  // N = 2^20 = 1024*1024= 1.048.576
    struct timeval tv;
          unsigned long long start;

    float *x = new float[N];
    float *y = new float[N];

    for (int i =0; i < N; i++ ){
        x[i]= 1.0f;
        y[i]= 2.0f;
    }

                                                .tv_usec;
```

```
    add(N, x, y);

    gettimeofday(&tv, NULL);
          printf("Tiempo del cálculo : %lf ms\n", (tv.tv_sec*1000000+tv.tv_usec -

   float maxError = 0.0f;
   int contError = 0;

   for (int i=0; i <N; i++){
       maxError=fmax(maxError,fabs(y[i]-3.0f));
       if (y[i] != 3.0) contError++;
   }
   std::cout << "suma de " << N << " Elementos" << std::endl;
   std::cout << "Número de Errores: " <<contError << std::endl;
   std::cout << "Max error: " <<maxError << std::endl;


   delete [] x;
   delete [] y;
   return 0;
}
```

Compilamos el archivo suma0.cu

```
!/usr/local/cuda/bin/nvcc -arch=sm_52 -rdc=true suma0.cu -o suma0 -lcudadevrt
```

```
!ls
```

```
suma0  suma0.cu
```

Corremos el fichero ejecutable

```
!./suma0
```

```
Tiempo del cálculo : 3.147000 ms
suma de 1048576 Elementos
Número de Errores: 0
Max error: 0
```

Suma en la GPU con un solo thread (serie)

Escribir en el directorio actual el fichero suma1.cu

```
%%writefile suma1.cu
```



```
#include <time.h>
```

```
#include <sys/time.h>

__global__ void add(int n, float *x, float *y) {

    for (int i =0; i < n; i++ ){
        y[i]=x[i]+y[i];
    }
}


int main(void) {

    int N = 1 <<20;  // N = 2^20 = 1024*1024= 1.048.576
    struct timeval tv;
      unsigned long long start;
    float *x; // = new float[N];
    float *y; // = new float[N];

    cudaMallocManaged(&x, N*sizeof(float));
    cudaMallocManaged(&y, N*sizeof(float));

    for (int i =0; i < N; i++ ){
        x[i]= 1.0f;
        y[i]= 2.0f;
    }

    gettimeofday(&tv, NULL);
    start=tv.tv_sec*1000000+tv.tv_usec;

    add<<<1,1>>>(N, x, y);

    gettimeofday(&tv, NULL);
      printf("Tiempo del cálculo : %lf ms\n", (tv.tv_sec*1000000+tv.tv_usec - start)/1000.

   cudaDeviceSynchronize();
    gettimeofday(&tv, NULL);
      printf("Tiempo con el synchronize  : %lf ms\n", (tv.tv_sec*1000000+tv.tv_usec - star

   float maxError = 0.0f;
   int contError = 0;

   for (int i=0; i <N; i++){
       maxError=fmax(maxError,fabs(y[i]-3.0f));
       if (y[i] != 3.0) contError++;
   }
   std::cout << "suma de " << N << " Elementos" << std::endl;
   std::cout << "Número de Errores: " <<contError << std::endl;
   std::cout << "Max error: " <<maxError << std::endl;

   cudaFree (x);
   cudaFree (y);
```

Se ha guardado correctamente

```
Writing suma1.cu
```

Compilamos el archivo suma1.cu

```
!ls -la
```

```
total 696
drwxr-xr-x 2 root root   4096 Nov 12 17:46 .
drwxr-xr-x 1 root root   4096 Nov 12 17:46 ..
-rwxr-xr-x 1 root root 692320 Nov 12 17:46 suma0
-rw-r--r-- 1 root root   1091 Nov 12 17:46 suma0.cu
-rw-r--r-- 1 root root   1370 Nov 12 17:46 suma1.cu
```

```
!/usr/local/cuda/bin/nvcc -arch=sm_75 -rdc=true suma1.cu -o suma1 -lcudadevrt
```

```
!/usr/local/cuda/bin/nvcc -rdc=true suma1.cu -o suma1sinarch -lcudadevrt
```

```
!ls
```

```
suma0  suma0.cu  suma1  suma1.cu  suma1sinarch
```

Corremos el fichero ejecutable compilado sin flag -arch (Modelo de computación de la GPU)

```
!./suma1sinarch
```

```
Tiempo del cálculo : 0.049000 ms
Tiempo con el synchronize  : 108.860000 ms
suma de 1048576 Elementos
Número de Errores: 0
Max error: 0
```

Dependiendo de la GPU puede encontrar funcionamiento no correcto en este caso (sin flag arch) ¿Qué estaría pasando?

Ejecutamos el compilado con flag -arch (Modelo de computación de la GPU)

```
!./suma1
```

```
Tiempo del cálculo : 0.031000 ms
Tiempo con el synchronize  : 100.214000 ms
suma de 1048576 Elementos
Número de Errores: 0
Max error: 0
```

P1 ¿Qué característica del modelo de computación esta siendo necesaria para que el ejemplo funcione correctamente?

Respuesta: Lo imprescindible es que se está seleccionando la arquitectura que sigue la GPU para poder optimizar recursos en tiempo de ejecución. Si no se marca ningun argumento en -arch, la arquitectura sm_52 es la que se marca por defecto. En la que podemos ver en suma1sinarch que el tiempo en synchronize es de unos 110ms. Por otra parte, de ahora en

Hacemos el profile de ejecución

```
!nvprof ./suma1
```

```
==250== NVPROF is profiling process 250, command: ./suma1
Tiempo del cálculo : 0.065000 ms
Tiempo con el synchronize  : 80.100000 ms
suma de 1048576 Elementos
Número de Errores: 0
Max error: 0
==250== Profiling application: ./suma1
==250== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:  100.00%  80.021ms         1  80.021ms  80.021ms  80.021ms  add(int
      API calls:   79.54%  317.27ms         2  158.64ms  48.048us  317.22ms  cudaMal
                   20.06%  80.000ms         1  80.000ms  80.000ms  80.000ms  cudaDev
                    0.16%  655.93us         2  327.97us  320.74us  335.20us  cudaFre
                    0.16%  629.33us         1  629.33us  629.33us  629.33us  cuDevic
                    0.06%  247.74us       101  2.4520us     182ns  96.783us  cuDevic
                    0.01%  49.763us         1  49.763us  49.763us  49.763us  cudaLau
                    0.01%  32.934us         1  32.934us  32.934us  32.934us  cuDevic
                    0.00%  6.2530us         1  6.2530us  6.2530us  6.2530us  cuDevic
                    0.00%  2.1440us         3     714ns     324ns  1.3190us  cuDevic
                    0.00%  1.3640us         2     682ns     269ns  1.0950us  cuDevic
                    0.00%     478ns         1     478ns     478ns     478ns  cuDevic

==250== Unified Memory profiling result:
Device "Tesla T4 (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
      48  170.67KB  4.0000KB  0.9961MB  8.000000MB  806.6710us  Host To Device
      24  170.67KB  4.0000KB  0.9961MB  4.000000MB  359.8630us  Device To Host
      12         -         -         -           -  2.284593ms  Gpu page fault group
Total CPU Page faults: 36
```

P2 ¿La suma ha tardado más o menos que en la CPU? Justifique como está funcionando.

Respuesta: Lo que se puede ver mediante el profiler es que la suma en la GPU está tardando bastante más que ejecutandose en CPU, esto es porque estamos ejecutando con 1 bloque y un [illegible] es el equivalente a una ejecución en serie. De ahí que la diferencia entre ejecuciones sea de en torno a 30 veces más lenta.

Suma en la GPU con mas threads <<<1,256>>>. Paralelismo sin aprovechar y posible Data Race

NOTA: Con este ejercicio se pretende comprobar un funcionamiento **no deseado** y por tanto no debe usarse como ejemplo a seguir.

Escribir en el directorio actual el fichero suma2.cu

```
%%writefile suma2.cu

#include <iostream>
#include <math.h>
#include <time.h>
#include <sys/time.h>

__global__ void add(int n, float *x, float *y) {

    for (int i =0; i < n; i++ ){
        y[i]=x[i]+y[i];
    }
}


int main(void) {

    int N = 1 <<20;  // N = 2^20 = 1024*1024= 1.048.576
    struct timeval tv;
      unsigned long long start;
    float *x; // = new float[N];
    float *y; // = new float[N];

    cudaMallocManaged(&x, N*sizeof(float));
    cudaMallocManaged(&y, N*sizeof(float));

    for (int i =0; i < N; i++ ){
        x[i]= 1.0f;
        y[i]= 2.0f;
    }

    gettimeofday(&tv, NULL);
    start=tv.tv_sec*1000000+tv.tv_usec;

    add<<<1,256>>>(N, x, y);

    gettimeofday(&tv, NULL);
                                    ma2 : %lf ms\n", (tv.tv_sec*1000000+tv.tv_usec - start)

    gettimeofday(&tv, NULL);
```

Se ha guardado correctamente

```
        printf("Tiempo con el synchronize  : %lf ms\n", (tv.tv_sec*1000000+tv.tv_usec - star


   float maxError = 0.0f;
   int contError = 0;

   for (int i=0; i <N; i++){
       maxError=fmax(maxError,fabs(y[i]-3.0f));
       if (y[i] != 3.0) contError++;
   }
   std::cout << "Suma de " << N << " elementos" << std::endl;
   std::cout << "Número de errores: " <<contError << std::endl;
   std::cout << "Max error: " <<maxError << std::endl;



   cudaFree (x);
   cudaFree (y);

   return 0;
}
```

```
    Writing suma2.cu
```

```
%ls
```

```
    suma0*  suma0.cu  suma1*  suma1.cu  suma1sinarch*  suma2.cu
```

Compilamos el archivo suma2.cu

```
!/usr/local/cuda/bin/nvcc -arch=sm_75 -rdc=true suma2.cu -o suma2 -lcudadevrt
```

```
!ls
```

```
    suma0  suma0.cu  suma1  suma1.cu  suma1sinarch  suma2  suma2.cu
```

Corremos el fichero ejecutable

```
!./suma2
```

```
    Tiempo del cálculo suma2 : 0.028000 ms
    Tiempo con el synchronize  : 117.423000 ms
    Suma de 1048576 elementos
    Número de errores: 1046925
    Max error: 7
```



```
!nvprof ./suma2
```

```
==299== NVPROF is profiling process 299, command: ./suma2
Tiempo del cálculo suma2 : 0.067000 ms
Tiempo con el synchronize  : 117.069000 ms
Suma de 1048576 elementos
Número de errores: 1048258
Max error: 7
==299== Profiling application: ./suma2
==299== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:  100.00%  116.98ms         1  116.98ms  116.98ms  116.98ms  add(int
      API calls:   72.65%  314.13ms         2  157.06ms  54.192us  314.07ms  cudaMal
                   27.04%  116.91ms         1  116.91ms  116.91ms  116.91ms  cudaDev
                    0.16%  709.16us         2  354.58us  353.59us  355.57us  cudaFre
                    0.09%  367.92us         1  367.92us  367.92us  367.92us  cuDevic
                    0.04%  155.03us       101  1.5340us     132ns  73.603us  cuDevic
                    0.01%  48.611us         1  48.611us  48.611us  48.611us  cudaLau
                    0.01%  29.190us         1  29.190us  29.190us  29.190us  cuDevic
                    0.00%  5.7870us         1  5.7870us  5.7870us  5.7870us  cuDevic
                    0.00%  1.6150us         2     807ns     285ns  1.3300us  cuDevic
                    0.00%  1.5800us         3     526ns     201ns     990ns  cuDevic
                    0.00%     260ns         1     260ns     260ns     260ns  cuDevic

==299== Unified Memory profiling result:
Device "Tesla T4 (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
      48  170.67KB  4.0000KB  0.9961MB  8.000000MB  809.9990us  Host To Device
      24  170.67KB  4.0000KB  0.9961MB  4.000000MB  367.4470us  Device To Host
      12         -         -         -           -  2.295952ms  Gpu page fault group
Total CPU Page faults: 36
```

P3 ¿La suma ha funcionado? ¿Ha tardado más o menos que en el caso anterior? Explique como está funcionando y si podrían generarse situaciones de funcionamiento incorrecto.

Respuesta: En este caso está tardando ligeramente más que antes. Además de que se debe a que el paralelismo no se está aprovechando del todo bien, también se puede explicar porque el acceso a memoria es muy ineficiente. Esto lo podemos ver además en que hay una cantidad muy grande de errores en la suma, casi con total seguridad porque se está produciendo un Data Race. Esto implica que el acceso a los elementos de los vectores no se está haciendo de forma correcta, porque cada hilo puede estar escogiendo más de una vez el mismo recurso, de ahí que se de que a lo mejor el tiempo de ejecución sea un poco más alto.

Suma en la GPU con mas threads <<<256,1>> sin paralelismo y Data Race

Se ha guardado correctamente

NOTA: Con este ejercicio se pretende comprobar un funcionamiento no deseado

Escribir en el directorio actual el fichero suma3.cu

```
%%writefile suma3.cu

#include <iostream>
#include <math.h>
#include <time.h>
#include <sys/time.h>


__global__ void add(int n, float *x, float *y) {

    for (int i =0; i < n; i++ ){
        y[i]=x[i]+y[i];
    }
}


int main(void) {

    int N = 1 <<20;  // N = 2^20 = 1024*1024= 1.048.576
    struct timeval tv;
      unsigned long long start;
    float *x; // = new float[N];
    float *y; // = new float[N];

    cudaMallocManaged(&x, N*sizeof(float));
    cudaMallocManaged(&y, N*sizeof(float));

    for (int i =0; i < N; i++ ){
        x[i]= 1.0f;
        y[i]= 2.0f;
    }

    gettimeofday(&tv, NULL);
    start=tv.tv_sec*1000000+tv.tv_usec;

    add<<<256,1>>>(N, x, y);

    gettimeofday(&tv, NULL);
      printf("Tiempo del cálculo suma3 : %lf ms\n", (tv.tv_sec*1000000+tv.tv_usec - start)

   cudaDeviceSynchronize();
    gettimeofday(&tv, NULL);
      printf("Tiempo con el synchronize  : %lf ms\n", (tv.tv_sec*1000000+tv.tv_usec - star

   float maxError = 0.0f;
   int contError = 0;


                                            y[i]-3.0f));
                                            ;
   }
```

```
   std::cout << "suma de " << N << " Elementos" << std::endl;
   std::cout << "Número de Errores: " <<contError << std::endl;
   std::cout << "Max error: " <<maxError << std::endl;

   cudaFree (x);
   cudaFree (y);

   return 0;
}
```

```
Writing suma3.cu
```

```
!ls
```

```
suma0  suma0.cu  suma1  suma1.cu  suma1sinarch  suma2  suma2.cu  suma3.cu
```

Compilamos el archivo suma3.cu

```
!/usr/local/cuda/bin/nvcc -arch=sm_37 -rdc=true suma3.cu -o suma3 -lcudadevrt
```

```
nvcc warning : The 'compute_35', 'compute_37', 'compute_50', 'sm_35', 'sm_37' and 's
```

```
!ls
```

```
suma0     suma1     suma1sinarch  suma2.cu  suma3.cu
suma0.cu  suma1.cu  suma2         suma3
```

Corremos el fichero ejecutable

```
!./suma3
```

```
Tiempo del cálculo suma3 : 0.037000 ms
Tiempo con el synchronize  : 136.031000 ms
suma de 1048576 Elementos
Número de Errores: 1048569
Max error: 255
```

Ejecutamos varias veces

```
!./suma3
```

```
Tiempo del cálculo suma3 : 0.032000 ms
Tiempo con el synchronize  : 135.283000 ms
suma de 1048576 Elementos
Número de Errores: 1048572
Max error: 255
```



```
!./suma3
```

```
Tiempo del cálculo suma3 : 0.030000 ms
Tiempo con el synchronize  : 139.586000 ms
suma de 1048576 Elementos
Número de Errores: 1048569
Max error: 255
```

Hacemos el profile de ejecución

```
!nvprof ./suma3
```

```
==354== NVPROF is profiling process 354, command: ./suma3
Tiempo del cálculo suma3 : 0.062000 ms
Tiempo con el synchronize  : 82.873000 ms
suma de 1048576 Elementos
Número de Errores: 1048565
Max error: 255
==354== Profiling application: ./suma3
==354== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:  100.00%  82.795ms         1  82.795ms  82.795ms  82.795ms  add(int
      API calls:   78.91%  314.29ms         2  157.14ms  49.784us  314.24ms  cudaMal
                   20.78%  82.779ms         1  82.779ms  82.779ms  82.779ms  cudaDev
                    0.16%  625.47us         2  312.74us  305.85us  319.62us  cudaFre
                    0.10%  383.85us         1  383.85us  383.85us  383.85us  cuDevic
                    0.04%  139.93us       101  1.3850us     130ns  59.789us  cuDevic
                    0.01%  46.761us         1  46.761us  46.761us  46.761us  cudaLau
                    0.01%  27.665us         1  27.665us  27.665us  27.665us  cuDevic
                    0.00%  5.2640us         1  5.2640us  5.2640us  5.2640us  cuDevic
                    0.00%  1.3030us         2     651ns     306ns     997ns  cuDevic
                    0.00%  1.2580us         3     419ns     207ns     668ns  cuDevic
                    0.00%     291ns         1     291ns     291ns     291ns  cuDevic

==354== Unified Memory profiling result:
Device "Tesla T4 (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
      48  170.67KB  4.0000KB  0.9961MB  8.000000MB  811.0240us  Host To Device
      24  170.67KB  4.0000KB  0.9961MB  4.000000MB  348.2160us  Device To Host
      23         -         -         -           -  3.368830ms  Gpu page fault group
Total CPU Page faults: 36
```

P4 La suma ha tardado más tiempo y con errores ¿Qué está pasando?

Respuesta: Para empezar, el aprovechamiento de la paralelización de CUDA no está siendo correcto, debido a que se está ejecutando un solo hilo por bloque. Esto es lo que hace que esté tardando más. El número de errores es debido a que, si antes era una posibilidad, ahora el Data Race es inevitable debido a que no hay especificada ninguna lógica de sincronización entre [illegible] incipales culpables del alto tiempo de ejecución que

Se ha guardado correctamente

Suma en la GPU con paralelismo <<<1,256>>

Escribir en el directorio actual el fichero suma4.cu

```
%%writefile suma4.cu

#include <iostream>
#include <math.h>
#include <time.h>
#include <sys/time.h>

__global__
void add(int n, float *x, float *y)
{
  int index = threadIdx.x;
  int stride = blockDim.x;
  for (int i = index; i < n; i += stride)
      y[i] = x[i] + y[i];
}

//__global__ void add(int n, float *x, float *y) {
//    for (int i =0; i < n; i++ ){
//        y[i]=x[i]+y[i];
//    }
//}

int main(void) {

    int N = 1 <<20;  // N = 2^20 = 1024*1024= 1.048.576
    struct timeval tv;
      unsigned long long start;
    float *x; // = new float[N];
    float *y; // = new float[N];

  // Allocate Unified Memory -- accessible from CPU or GPU
    cudaMallocManaged(&x, N*sizeof(float));
    cudaMallocManaged(&y, N*sizeof(float));

    for (int i =0; i < N; i++ ){
        x[i]= 1.0f;
        y[i]= 2.0f;
    }


    gettimeofday(&tv, NULL);
    start=tv.tv_sec*1000000+tv.tv_usec;

    add<<<1,256>>>(N, x, y);

      printf("Tiempo del cálculo suma 4 : %lf ms\n", (tv.tv_sec*1000000+tv.tv_usec - start
```

Se ha guardado correctamente

```
   cudaDeviceSynchronize();
    gettimeofday(&tv, NULL);
      printf("Tiempo con el synchronize  : %lf ms\n", (tv.tv_sec*1000000+tv.tv_usec - star

    cudaDeviceSynchronize();

   float maxError = 0.0f;
  int contError = 0;

   for (int i=0; i <N; i++){
       maxError=fmax(maxError,fabs(y[i]-3.0f));
       if (y[i] != 3.0) contError++;
   }
   std::cout << "Suma de " << N << " elementos" << std::endl;
   std::cout << "Número de errores: " <<contError << std::endl;
   std::cout << "Max error: " <<maxError << std::endl;


  // Free memory
   cudaFree (x);
   cudaFree (y);

   return 0;
}
```

```
    Writing suma4.cu
```

```
!ls
```

```
    suma0     suma1     suma1sinarch  suma2.cu  suma3.cu
    suma0.cu  suma1.cu  suma2         suma3     suma4.cu
```

Compilamos el archivo suma4.cu

```
!/usr/local/cuda/bin/nvcc -arch=sm_75 -rdc=true suma4.cu -o suma4 -lcudadevrt
```

```
!ls
```

```
    suma0     suma1     suma1sinarch  suma2.cu  suma3.cu  suma4.cu
    suma0.cu  suma1.cu  suma2         suma3     suma4
```

Corremos el fichero ejecutable

```
!./suma4
```

```
    Tiempo del cálculo suma 4 : 0.032000 ms
                              .880000 ms
```



```
    Max error: 0
```

Hacemos el profile de ejecución

```
!nvprof ./suma4
```

```
==403== NVPROF is profiling process 403, command: ./suma4
Tiempo del cálculo suma 4 : 0.070000 ms
Tiempo con el synchronize  : 3.959000 ms
Suma de 1048576 elementos
Número de errores: 0
Max error: 0
==403== Profiling application: ./suma4
==403== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:  100.00%  3.8779ms         1  3.8779ms  3.8779ms  3.8779ms  add(int
      API calls:   98.41%  317.69ms         2  158.84ms  57.109us  317.63ms  cudaMal
                    1.20%  3.8588ms         2  1.9294ms  7.1630us  3.8516ms  cudaDev
                    0.21%  689.43us         2  344.72us  311.65us  377.78us  cudaFre
                    0.11%  342.83us         1  342.83us  342.83us  342.83us  cuDevic
                    0.04%  142.74us       101  1.4130us     130ns  62.229us  cuDevic
                    0.02%  51.639us         1  51.639us  51.639us  51.639us  cudaLau
                    0.01%  27.782us         1  27.782us  27.782us  27.782us  cuDevic
                    0.00%  5.9120us         1  5.9120us  5.9120us  5.9120us  cuDevic
                    0.00%  1.6240us         3     541ns     198ns  1.0240us  cuDevic
                    0.00%  1.4320us         2     716ns     432ns  1.0000us  cuDevic
                    0.00%     263ns         1     263ns     263ns     263ns  cuDevic

==403== Unified Memory profiling result:
Device "Tesla T4 (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
      48  170.67KB  4.0000KB  0.9961MB  8.000000MB  812.2680us  Host To Device
      24  170.67KB  4.0000KB  0.9961MB  4.000000MB  360.6980us  Device To Host
      12         -         -         -           -  2.276465ms  Gpu page fault group
Total CPU Page faults: 36
```

P5 Compare con el tiempo de ejecución de un solo Thread y con la ejecución en CPU.¿Que puede deducir de estos comportamientos?

Respuesta: Lo que se puede ver es que el tiempo en comparación con un Thread es mucho mayor, porque por fin estamos haciendo un uso eficiente de los accesos a memoria de cada uno de los threads. Cuando lo comparamos con CPU, podemos ver que son relativamente similares en tiempo de ejecución. Uno podría asumir que por qué una ejecución en serie de una CPU es similar a una ligera paralelización en GPU. Sin embargo, hay que recordar que los cores de una CPU son mucho más potentes que lo puedes tener en una GPU, por tanto la comparación a nivel single thread es absurda entre las dos, y solo tiene sentido un uso de GPU para computación en el momento en el que lo conceptualizas con una tarea donde es factible hacer una paralelización masiva.

Se ha guardado correctamente

P6 ¿Se puede seguir optimizando?

Respuesta: Se puede hacer algo más eficiente con una paralelización no solo por threads, sino también a nivel de bloque. Si el acceso a la memoria está también pensado para esa configuración de kernel, es posible hacerlo superior en tiempo de ejecución al equivalente en una CPU.

P7 Realice el paralelismo correspondiente a la Suma en la GPU con paralelismo <<<256,1>> y estudie el comportamiento de los tiempos de ejecución.

```
%%writefile suma4_p7.cu

#include <iostream>
#include <math.h>
#include <time.h>
#include <sys/time.h>

__global__
void add(int n, float *x, float *y)
{
  int index = blockIdx.x;
  int stride = gridDim.x;
  for (int i = index; i < n; i += stride)
      y[i] = x[i] + y[i];
}

//__global__ void add(int n, float *x, float *y) {
//    for (int i =0; i < n; i++ ){
//        y[i]=x[i]+y[i];
//    }
//}

int main(void) {

    int N = (1 <<20);  // N = 2^20 = 1024*1024= 1.048.576
    struct timeval tv;
      unsigned long long start;
    float *x; // = new float[N];
    float *y; // = new float[N];

  // Allocate Unified Memory -- accessible from CPU or GPU
    cudaMallocManaged(&x, N*sizeof(float));
    cudaMallocManaged(&y, N*sizeof(float));

    for (int i =0; i < N; i++ ){
        x[i]= 1.0f;
```

```
    gettimeofday(&tv, NULL);
    start=tv.tv_sec*1000000+tv.tv_usec;

    add<<<256,1>>>(N, x, y);

    gettimeofday(&tv, NULL);
      printf("Tiempo del cálculo suma 4 : %lf ms\n", (tv.tv_sec*1000000+tv.tv_usec - start

   cudaDeviceSynchronize();
    gettimeofday(&tv, NULL);
      printf("Tiempo con el synchronize  : %lf ms\n", (tv.tv_sec*1000000+tv.tv_usec - star

    cudaDeviceSynchronize();

   float maxError = 0.0f;
  int contError = 0;

   for (int i=0; i <N; i++){
       maxError=fmax(maxError,fabs(y[i]-3.0f));
       if (y[i] != 3.0) contError++;
   }
   std::cout << "Suma de " << N << " elementos" << std::endl;
   std::cout << "Número de errores: " <<contError << std::endl;
   std::cout << "Max error: " <<maxError << std::endl;


  // Free memory
   cudaFree (x);
   cudaFree (y);

   return 0;
}
```

```
    Writing suma4_p7.cu
```

```
!ls
```

```
    suma0     suma1     suma1sinarch  suma2.cu  suma3.cu  suma4.cu
    suma0.cu  suma1.cu  suma2         suma3     suma4     suma4_p7.cu
```

Compilamos el archivo suma4.cu

```
!/usr/local/cuda/bin/nvcc -arch=sm_75 -rdc=true suma4_p7.cu -o suma4_p7 -lcudadevrt
```

```
!ls
```

```
    suma0     suma1     suma1sinarch  suma2.cu  suma3.cu  suma4.cu  suma4_p7.cu
    suma0.cu  suma1.cu  suma2         suma3     suma4     suma4_p7
```

Se ha guardado correctamente

Corremos el fichero ejecutable

```
!./suma4_p7
```

```
Tiempo del cálculo suma 4 : 0.030000 ms
Tiempo con el synchronize  : 3.681000 ms
Suma de 1048576 elementos
Número de errores: 0
Max error: 0
```

Hacemos el profile de ejecución

```
!nvprof ./suma4_p7
```

```
==452== NVPROF is profiling process 452, command: ./suma4_p7
Tiempo del cálculo suma 4 : 0.064000 ms
Tiempo con el synchronize  : 4.058000 ms
Suma de 1048576 elementos
Número de errores: 0
Max error: 0
==452== Profiling application: ./suma4_p7
==452== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:  100.00%  3.9841ms         1  3.9841ms  3.9841ms  3.9841ms  add(int
      API calls:   98.36%  312.04ms         2  156.02ms  47.993us  311.99ms  cudaMal
                    1.25%  3.9697ms         2  1.9848ms  5.7640us  3.9639ms  cudaDev
                    0.20%  625.41us         2  312.70us  294.64us  330.77us  cudaFre
                    0.12%  365.93us         1  365.93us  365.93us  365.93us  cuDevic
                    0.05%  149.69us       101  1.4820us     142ns  64.541us  cuDevic
                    0.02%  48.496us         1  48.496us  48.496us  48.496us  cudaLau
                    0.01%  29.034us         1  29.034us  29.034us  29.034us  cuDevic
                    0.00%  6.1410us         1  6.1410us  6.1410us  6.1410us  cuDevic
                    0.00%  2.0610us         3     687ns     181ns  1.5100us  cuDevic
                    0.00%  1.3110us         2     655ns     304ns  1.0070us  cuDevic
                    0.00%     339ns         1     339ns     339ns     339ns  cuDevic

==452== Unified Memory profiling result:
Device "Tesla T4 (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
      48  170.67KB  4.0000KB  0.9961MB  8.000000MB  803.1840us  Host To Device
      24  170.67KB  4.0000KB  0.9961MB  4.000000MB  361.1760us  Device To Host
      12         -         -         -           -  2.513872ms  Gpu page fault group
Total CPU Page faults: 36
```

Respuesta: En este caso podemos ver que el comportamiento de este tipo de paralelismo es prácticamente equivalente a hacerlo mediante hilos, solamente cambiando las variables con las que se coordinan los bloques para no establecer una condición de Data Race

Suma en la GPU con paralelismo de bloques

https://developer.nvidia.com/blog/even-easier-introduction-cuda/

Escribir en el directorio actual el fichero suma5.cu

```
%%writefile suma5.cu

#include <iostream>
#include <math.h>
#include <time.h>
#include <sys/time.h>

__global__
void add(int n, float *x, float *y)
{
  int index = blockIdx.x * blockDim.x + threadIdx.x;
  int stride = blockDim.x * gridDim.x;
  for (int i = index; i < n; i += stride)
    y[i] = x[i] + y[i];
}

int main(void) {

    int N = 1 <<20;  // N = 2^20 = 1024*1024= 1.048.576
    struct timeval tv;
      unsigned long long start;
    float *x; // = new float[N];
    float *y; // = new float[N];

  // Allocate Unified Memory -- accessible from CPU or GPU
    cudaMallocManaged(&x, N*sizeof(float));
    cudaMallocManaged(&y, N*sizeof(float));

    for (int i =0; i < N; i++ ){
        x[i]= 1.0f;
        y[i]= 2.0f;
    }

    int blockSize = 256;
    int numBlocks = (N + blockSize - 1) / blockSize;

    gettimeofday(&tv, NULL);
    start=tv.tv_sec*1000000+tv.tv_usec;

   add<<<numBlocks, blockSize>>>(N, x, y);

    gettimeofday(&tv, NULL);
      printf("Tiempo del cálculo suma 5 : %lf ms\n", (tv.tv_sec*1000000+tv.tv_usec - start

   cudaDeviceSynchronize();
    gettimeofday(&tv, NULL);
      printf("Tiempo con el synchronize  : %lf ms\n", (tv.tv_sec*1000000+tv.tv_usec - star
```

Se ha guardado correctamente

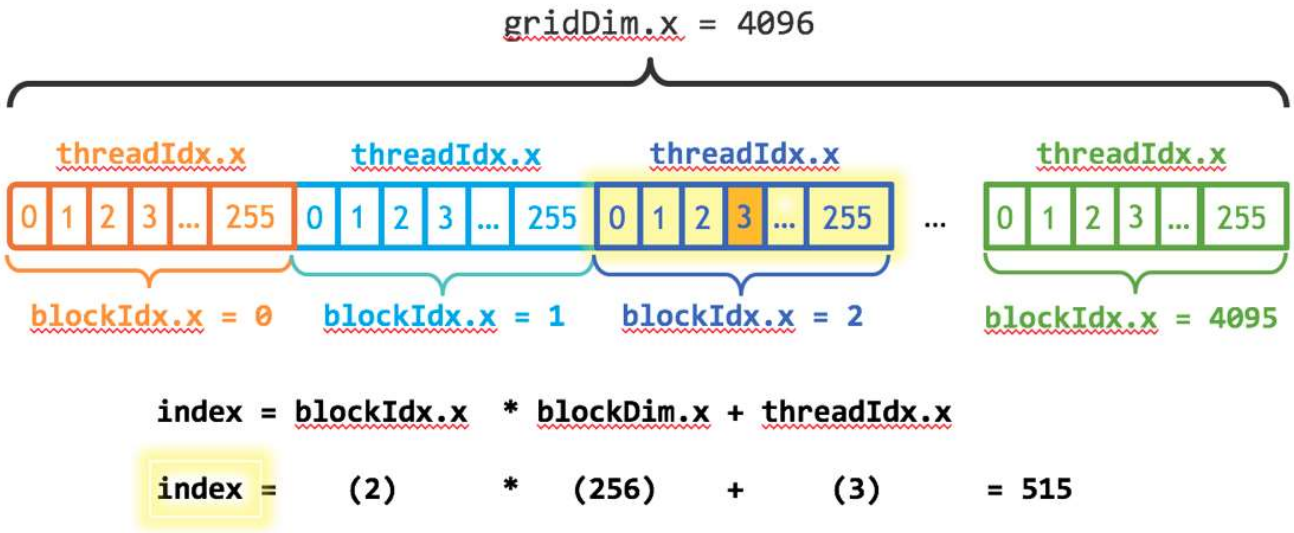
```
    for (int i=0; i <N; i++){
        maxError=fmax(maxError,fabs(y[i]-3.0f));
        if (y[i] != 3.0) contError++;
    }
    std::cout << "Suma de " << N << " elementos" << std::endl;
    std::cout << "Número de errores: " <<contError << std::endl;
    std::cout << "Max error: " <<maxError << std::endl;


  // Free memory
    cudaFree (x);
    cudaFree (y);

    return 0;
}
```

```
Writing suma5.cu
```

Para entender los bloques y threads por bloque



update the kernel code to take into account the entire grid of thread blocks. CUDA provides gridDim.x, which contains the number of blocks in the grid, and blockIdx.x, which contains the index of the current thread block in the grid. Figure 1 illustrates the the approach to indexing into an array (one-dimensional) in CUDA using blockDim.x, gridDim.x, and threadIdx.x. The idea is that each thread gets its index by computing the offset to the beginning of its block (the block index times the block size: blockIdx.x * blockDim.x) and adding the thread's index within the block (threadIdx.x). The code blockIdx.x * blockDim.x + threadIdx.x is idiomatic CUDA.

Compilamos el archivo suma5.cu

```
!/usr/local/cuda/bin/nvcc -arch=sm_75 -rdc=true suma5.cu -o suma5 -lcudadevrt
```

```
!ls
```

Se ha guardado correctamente

```
suma0     suma1.cu      suma2.cu  suma4     suma4_p7.cu
suma0.cu  suma1sinarch  suma3     suma4.cu  suma5
suma1     suma2         suma3.cu  suma4_p7  suma5.cu
```

Corremos el fichero ejecutable

```
!./suma5
```

```
Tiempo del cálculo suma 5 : 0.039000 ms
Tiempo con el synchronize  : 2.254000 ms
Suma de 1048576 elementos
Número de errores: 0
Max error: 0
```

Hacemos el profile de ejecución

```
!nvprof ./suma5
```

```
==500== NVPROF is profiling process 500, command: ./suma5
Tiempo del cálculo suma 5 : 0.066000 ms
Tiempo con el synchronize  : 2.526000 ms
Suma de 1048576 elementos
Número de errores: 0
Max error: 0
==500== Profiling application: ./suma5
==500== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:  100.00%  2.4493ms         1  2.4493ms  2.4493ms  2.4493ms  add(int
      API calls:   98.85%  310.34ms         2  155.17ms  62.929us  310.28ms  cudaMal
                    0.77%  2.4244ms         1  2.4244ms  2.4244ms  2.4244ms  cudaDev
                    0.19%  596.52us         2  298.26us  290.47us  306.05us  cudaFre
                    0.11%  354.34us         1  354.34us  354.34us  354.34us  cuDevic
                    0.05%  142.91us       101  1.4140us     129ns  62.352us  cuDevic
                    0.02%  49.845us         1  49.845us  49.845us  49.845us  cudaLau
                    0.01%  28.194us         1  28.194us  28.194us  28.194us  cuDevic
                    0.00%  6.0570us         1  6.0570us  6.0570us  6.0570us  cuDevic
                    0.00%  1.5290us         3     509ns     179ns     998ns  cuDevic
                    0.00%  1.0420us         2     521ns     217ns     825ns  cuDevic
                    0.00%     258ns         1     258ns     258ns     258ns  cuDevic

==500== Unified Memory profiling result:
Device "Tesla T4 (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
     104  78.769KB  4.0000KB  980.00KB  8.000000MB  947.3720us  Host To Device
      24  170.67KB  4.0000KB  0.9961MB  4.000000MB  360.4380us  Device To Host
      11         -         -         -           -  2.378806ms  Gpu page fault group
Total CPU Page faults: 36
```

P8 Compare con el tiempo de ejecución con el resto de ejemplos ... lucir de estos comportamientos?

Se ha guardado correctamente

Respuesta: Lo que se puede deducir es que ahora sí que se está haciendo un uso adecuado del paralelismo por GPU, con un número elevado de bloques e hilos se está realizando una paralelización masiva, que es lo que por fin permite obtener un rendimiento superior a lo que obtendríamos con una CPU.

P9 Pruebe el último ejercicio con más threads por bloque y explique los resultados obtenidos.

Respuesta: Tras probar con 512 y 1024 threads por bloque, podemos concluir que el resultado es totalmente equivalente en términos de tiempo. La razón de esto es que, como se puede leer en el código, el número de bloques se calcula en función de cuantos threads por bloque se especifican, por tanto, no estamos optimizando nada porque aumentar el número de threads por bloque lo único que hace es disminuir el número de bloques, pero el número de ejecuciones en paralelo es esencialmente el mismo.

P10 Modifique el último ejercicio variando N para tamaños que no sean múltiplos del número de threads por bloque que se esté usando.

Para ello, vamos a reducir en 255 el número de operaciones que hace, para que utilice el mismo número de bloques, pero no esté utilizando el 100% de las capacidades del kernel que hemos definido

```
%%writefile suma5_bl.cu

#include <iostream>
#include <math.h>
#include <time.h>
#include <sys/time.h>

__global__
void add(int n, float *x, float *y)
{
  int index = blockIdx.x * blockDim.x + threadIdx.x;
  int stride = blockDim.x * gridDim.x;
  for (int i = index; i < n; i += stride)
    y[i] = x[i] + y[i];
}

int main(void) {
```

Se ha guardado correctamente

```
2^20 = 1024*1024= 1.048.576
  struct timeval tv;
```

```
      unsigned long long start;
    float *x; // = new float[N];
    float *y; // = new float[N];

  // Allocate Unified Memory -- accessible from CPU or GPU
    cudaMallocManaged(&x, N*sizeof(float));
    cudaMallocManaged(&y, N*sizeof(float));

    for (int i =0; i < N; i++ ){
        x[i]= 1.0f;
        y[i]= 2.0f;
    }

    int blockSize = 256;
    int numBlocks = (N + blockSize - 1) / blockSize;

    gettimeofday(&tv, NULL);
    start=tv.tv_sec*1000000+tv.tv_usec;

   add<<<numBlocks, blockSize>>>(N, x, y);

    gettimeofday(&tv, NULL);
      printf("Tiempo del cálculo suma 5 : %lf ms\n", (tv.tv_sec*1000000+tv.tv_usec - start

   cudaDeviceSynchronize();
    gettimeofday(&tv, NULL);
      printf("Tiempo con el synchronize  : %lf ms\n", (tv.tv_sec*1000000+tv.tv_usec - star

   float maxError = 0.0f;
   int contError = 0;

   for (int i=0; i <N; i++){
       maxError=fmax(maxError,fabs(y[i]-3.0f));
       if (y[i] != 3.0) contError++;
   }
   std::cout << "Suma de " << N << " elementos" << std::endl;
   std::cout << "Número de errores: " <<contError << std::endl;
   std::cout << "Max error: " <<maxError << std::endl;


  // Free memory
   cudaFree (x);
   cudaFree (y);

   return 0;
}
```

```
    Writing suma5_bl.cu
```

```
!/usr/local/cuda/bin/nvcc -arch=sm_75 -rdc=true suma5_bl.cu -o suma5_bl -lcudadevrt
```

```
suma0     suma1.cu      suma2.cu  suma4     suma4_p7.cu  suma5_bl.cu
suma0.cu  suma1sinarch  suma3     suma4.cu  suma5        suma5.cu
suma1     suma2         suma3.cu  suma4_p7  suma5_bl
```

Corremos el fichero ejecutable

```
!./suma5_bl
```

```
Tiempo del cálculo suma 5 : 0.028000 ms
Tiempo con el synchronize  : 2.131000 ms
Suma de 1048321 elementos
Número de errores: 0
Max error: 0
```

Hacemos el profile de ejecución

```
!nvprof ./suma5_bl
```

```
==548== NVPROF is profiling process 548, command: ./suma5_bl
Tiempo del cálculo suma 5 : 0.063000 ms
Tiempo con el synchronize  : 2.484000 ms
Suma de 1048321 elementos
Número de errores: 0
Max error: 0
==548== Profiling application: ./suma5_bl
==548== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:  100.00%  2.4124ms         1  2.4124ms  2.4124ms  2.4124ms  add(int
      API calls:   98.76%  314.77ms         2  157.38ms  48.284us  314.72ms  cudaMal
                    0.75%  2.3871ms         1  2.3871ms  2.3871ms  2.3871ms  cudaDev
                    0.26%  836.46us         2  418.23us  408.08us  428.38us  cudaFre
                    0.15%  465.02us         1  465.02us  465.02us  465.02us  cuDevic
                    0.05%  174.52us       101  1.7270us     128ns  79.195us  cuDevic
                    0.01%  46.172us         1  46.172us  46.172us  46.172us  cudaLau
                    0.01%  26.677us         1  26.677us  26.677us  26.677us  cuDevic
                    0.00%  6.7090us         1  6.7090us  6.7090us  6.7090us  cuDevic
                    0.00%  1.4710us         3     490ns     179ns     986ns  cuDevic
                    0.00%  1.2480us         2     624ns     298ns     950ns  cuDevic
                    0.00%     241ns         1     241ns     241ns     241ns  cuDevic

==548== Unified Memory profiling result:
Device "Tesla T4 (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
      98  83.592KB  4.0000KB  988.00KB  8.000000MB  929.9990us  Host To Device
      24  170.67KB  4.0000KB  0.9961MB  4.000000MB  360.7240us  Device To Host
      11         -         -         -           -  2.345138ms  Gpu page fault group
Total CPU Page faults: 36
```

Respuesta: Lo que podemos concluir es que es preferible utilizar el kernel en toda su capacidad definida, porque debido a que se ejecuta en paralelo, tarda lo mismo que si estamos utilizando [illegible]s por ello, que aunque pongamos algunas operaciones menos (cierto es que en magnitud, las operaciones que hemos quitado son muy pocas, aunque

Se ha guardado correctamente

se podría aumentar la magnitud y pasaría lo mismo) el tiempo de ejecución no termina de variar

Suma de matrices cuadradas

Vamos a realizar una implementación del algoritmo de suma de matrices, donde hacemos uso de memoria unificada, definiendo además la forma del kernel de forma que estemos utilizando lo mejor posible las capacidades del mismo. Las matrices que definimos son idénticas, por tanto, la matriz suma resultante siempre tendrá valdrá en cada elemento el doble del valor de las matrices A y B en esa misma posición

```
%%writefile sumaMatrices.cu

#include <iostream>
#include <math.h>
#include <time.h>
#include <sys/time.h>

#define N 1024
#define MAX 8

__global__
void add(int n, float *a, float *b, float *c)
{
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  int j = blockIdx.y * blockDim.y + threadIdx.y;

  if (i < n && j < n){
    c[n * i + j] = a[n * i + j] + b[n * i + j];
  }
}

int main(void) {

    int size = N*N;
    float *a; // = new float[size];
    float *b; // = new float[size];
    float *c;

  // Allocate Unified Memory -- accessible from CPU or GPU
    cudaMallocManaged(&a, size*sizeof(float));
    cudaMallocManaged(&b, size*sizeof(float));
    cudaMallocManaged(&c, size*sizeof(float));

    for (int i =0; i < N; i++ ){
      for (int j = 0; j < N; j++) {
                                    j);
                                    j);
      }
    }
```

```
    dim3 dimGrid((N + MAX - 1) / MAX, (N + MAX - 1) / MAX, 1);
    dim3 dimBlock(MAX, MAX, 1);

    add<<<dimGrid, dimBlock>>>(N, a, b, c);
    cudaDeviceSynchronize();

    std::cout << "Resultado en 10: " << a[10] << " + " << b[10] << " = " << c[10] << std::
    std::cout << "Resultado en 15: " << a[15] << " + " << b[15] << " = " << c[15] << std::
    std::cout << "Resultado en 20: " << a[20] << " + " << b[20] << " = " << c[20] << std::

  // Free memory
   cudaFree (a);
   cudaFree (b);
   cudaFree (c);

   return 0;
}
```

```
     Writing sumaMatrices.cu
```

```
!/usr/local/cuda/bin/nvcc -arch=sm_75 -rdc=true sumaMatrices.cu -o sumaMatrices -lcudadevr
```

Corremos el fichero ejecutable, y comprobamos que algunos elementos estén mostrando el resultado correcto.

```
!./sumaMatrices
```

```
     Resultado en 10: 10 + 10 = 20
     Resultado en 15: 15 + 15 = 30
     Resultado en 20: 20 + 20 = 40
```

Hacemos el profile de ejecución

```
!nvprof ./sumaMatrices
```

```
     ==595== NVPROF is profiling process 595, command: ./sumaMatrices
     Resultado en 10: 10 + 10 = 20
     Resultado en 15: 15 + 15 = 30
     Resultado en 20: 20 + 20 = 40
     ==595== Profiling application: ./sumaMatrices
     ==595== Profiling result:
                 Type  Time(%)      Time     Calls       Avg       Min       Max  Name
      GPU activities:  100.00%  3.1713ms         1  3.1713ms  3.1713ms  3.1713ms  add(int
           API calls:   98.53%  309.06ms         3  103.02ms  10.473us  309.00ms  cudaMal
                        1.03%  3.2233ms         1  3.2233ms  3.2233ms  3.2233ms  cudaDev
                        0.25%  774.95us         3  258.32us  179.21us  357.88us  cudaFre
                        0.12%  377.29us         1  377.29us  377.29us  377.29us  cuDevic
                               .23us       101  1.4370us     136ns  61.637us  cuDevic
                               697us         1  52.697us  52.697us  52.697us  cudaLau
                               114us         1  32.114us  32.114us  32.114us  cuDevic
                        0.00%  6.4870us         1  6.4870us  6.4870us  6.4870us  cuDevic
```

Se ha guardado correctamente

```
                    0.00%  2.0160us         3     672ns     191ns  1.4390us  cuDevic
                    0.00%  1.4500us         2     725ns     310ns  1.1400us  cuDevic
                    0.00%     295ns         1     295ns     295ns     295ns  cuDevic

    ==595== Unified Memory profiling result:
    Device "Tesla T4 (0)"
       Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
         352  23.273KB  4.0000KB  288.00KB  8.000000MB  1.541314ms  Host To Device
           6  32.000KB  4.0000KB  60.000KB  192.0000KB  25.15100us  Device To Host
           8         -         -         -           -  4.569318ms  Gpu page fault group
    Total CPU Page faults: 27
```

Stencil1D

En esta parte, vamos a hacer un estudio de las diferentes versiones donde, iremos sofisticando la paralelización que aplica la GPU en el algoritmo, y debatiendo sus resultados.

Stencil1D sin uso de memoria compartida

```
%%writefile "stencil_1d.cu"

#include <stdio.h>

#define RADIUS        3
#define BLOCK_SIZE    256
#define NUM_ELEMENTS  (4096*2)

// CUDA API error checking macro
#define cudaCheck(error) \
  if (error != cudaSuccess) { \
    printf("Fatal error: %s at %s:%d\n", \
      cudaGetErrorString(error), \
      __FILE__, __LINE__); \
    exit(1); \
  }

__global__ void stencil_1d(int *in, int *out)
{
    // Just one global index
    int index = threadIdx.x + (blockIdx.x * blockDim.x) + RADIUS;

    // Apply the stencil
    int result = 0;
    for (int offset = -RADIUS ; offset <= RADIUS ; offset++)
        result += in[index + offset];

    // Store the result
```

```
int main()
{
  unsigned int i;
  int h_in[NUM_ELEMENTS + 2 * RADIUS], h_out[NUM_ELEMENTS];
  int *d_in, *d_out;

  // Initialize host data
  for( i = 0; i < (NUM_ELEMENTS + 2*RADIUS); ++i )
    h_in[i] = 1; // With a value of 1 and RADIUS of 3, all output values should be 7

  // Allocate space on the device
  cudaCheck( cudaMalloc( &d_in, (NUM_ELEMENTS + 2*RADIUS) * sizeof(int)) );
  cudaCheck( cudaMalloc( &d_out, NUM_ELEMENTS * sizeof(int)) );

  // Copy input data to device
  cudaCheck( cudaMemcpy( d_in, h_in, (NUM_ELEMENTS + 2*RADIUS) * sizeof(int), cudaMemcpyHc

  stencil_1d<<< (NUM_ELEMENTS + BLOCK_SIZE - 1)/BLOCK_SIZE, BLOCK_SIZE >>> (d_in, d_out);

  cudaCheck( cudaMemcpy( h_out, d_out, NUM_ELEMENTS * sizeof(int), cudaMemcpyDeviceToHost)

  // Verify every out value is 7
  for( i = 0; i < NUM_ELEMENTS; ++i )
    if (h_out[i] != 7)
    {
      printf("Element h_out[%d] == %d != 7\n", i, h_out[i]);
      break;
    }

  if (i == NUM_ELEMENTS)
    printf("SUCCESS!\n");

  // Free out memory
  cudaFree(d_in);
  cudaFree(d_out);

  return 0;
}
```

```
    Overwriting stencil_1d.cu
```

```
!/usr/local/cuda/bin/nvcc -arch=sm_75 -rdc=true stencil_1d.cu -o stencil_1d -lcudadevrt
```

Corremos el fichero ejecutable

```
!./stencil_1d
```

```
    SUCCESS!
```

```
!nvprof ./stencil_1d
```

```
==4692== NVPROF is profiling process 4692, command: ./stencil_1d
SUCCESS!
==4692== Profiling application: ./stencil_1d
==4692== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   35.86%  5.1520us         1  5.1520us  5.1520us  5.1520us  [CUDA m
                   33.41%  4.8000us         1  4.8000us  4.8000us  4.8000us  stencil
                   30.73%  4.4150us         1  4.4150us  4.4150us  4.4150us  [CUDA m
      API calls:   99.73%  299.03ms         2  149.52ms  3.7440us  299.03ms  cudaMal
                    0.11%  341.09us         1  341.09us  341.09us  341.09us  cuDevic
                    0.06%  169.24us         2  84.621us  49.904us  119.34us  cudaFre
                    0.05%  149.81us       101  1.4830us     126ns  64.049us  cuDevic
                    0.03%  77.854us         2  38.927us  31.149us  46.705us  cudaMem
                    0.01%  29.296us         1  29.296us  29.296us  29.296us  cuDevic
                    0.01%  24.861us         1  24.861us  24.861us  24.861us  cudaLau
                    0.00%  5.8390us         1  5.8390us  5.8390us  5.8390us  cuDevic
                    0.00%  1.6670us         3     555ns     188ns  1.0750us  cuDevic
                    0.00%  1.3070us         2     653ns     190ns  1.1170us  cuDevic
                    0.00%     250ns         1     250ns     250ns     250ns  cuDevic
```

En esta versión vemos que el algoritmo está funcionando bien, pero sin sus inconveneintes. El principal problema que vemos en el perfilador es que la GPU está invirtiendo mucho más tiempo en copiar memoria de Host a Drive y viceversa que en ejecutar el algoritmo. Por tanto, un uso de memoria compartida podría solventar esta pérdida en tiempo de ejecución.

Stencil1D con memoria compartida sin sincronización

```
%%writefile stencil1d_shared.cu

#include <stdio.h>

#define RADIUS        3
#define BLOCK_SIZE    256
#define NUM_ELEMENTS  (4096*2)

// CUDA API error checking macro
#define cudaCheck(error) \
  if (error != cudaSuccess) { \
    printf("Fatal error: %s at %s:%d\n", \
      cudaGetErrorString(error), \
      __FILE__, __LINE__); \
    exit(1); \
  }

__global__ void stencil_1d(int *in, int *out)
{
                                        + 2 * RADIUS];
                                    ckIdx.x * blockDim.x) + RADIUS;
    int lindex = threadIdx.x + RADIUS;
```

Se ha guardado correctamente

```
    // Read input elements into shared memory
    temp[lindex] = in[gindex];
    if (threadIdx.x < RADIUS)
    {
        temp[lindex - RADIUS] = in[gindex - RADIUS];
        temp[lindex + BLOCK_SIZE] = in[gindex + BLOCK_SIZE];
    }

    // Apply the stencil
    int result = 0;
    for (int offset = -RADIUS ; offset <= RADIUS ; offset++)
        result += temp[lindex + offset];

    // Store the result
    out[gindex-RADIUS] = result;
}

int main()
{
  unsigned int i;
  int h_in[NUM_ELEMENTS + 2 * RADIUS], h_out[NUM_ELEMENTS];
  int *d_in, *d_out;

  // Initialize host data
  for( i = 0; i < (NUM_ELEMENTS + 2*RADIUS); ++i )
    h_in[i] = 1; // With a value of 1 and RADIUS of 3, all output values should be 7

  // Allocate space on the device
  cudaCheck( cudaMalloc( &d_in, (NUM_ELEMENTS + 2*RADIUS) * sizeof(int)) );
  cudaCheck( cudaMalloc( &d_out, NUM_ELEMENTS * sizeof(int)) );

  // Copy input data to device
  cudaCheck( cudaMemcpy( d_in, h_in, (NUM_ELEMENTS + 2*RADIUS) * sizeof(int), cudaMemcpyHc

  stencil_1d<<< (NUM_ELEMENTS + BLOCK_SIZE - 1)/BLOCK_SIZE, BLOCK_SIZE >>> (d_in, d_out);

  cudaCheck( cudaMemcpy( h_out, d_out, NUM_ELEMENTS * sizeof(int), cudaMemcpyDeviceToHost)

  // Verify every out value is 7
  for( i = 0; i < NUM_ELEMENTS; ++i )
    if (h_out[i] != 7)
    {
      printf("Element h_out[%d] == %d != 7\n", i, h_out[i]);
      break;
    }

  if (i == NUM_ELEMENTS)
    printf("SUCCESS!\n");

  // Free out memory
```



```
  return 0;
```

```
}
```

```
Writing stencil1d_shared.cu
```

```
!/usr/local/cuda/bin/nvcc -arch=sm_75 -rdc=true stencil1d_shared.cu -o stencil1d_shared -l
```

Realizamos 100 ejecuciones para comprobar la consistencia de los resultados

```
for _ in range(100):
  !./stencil1d_shared
```

```
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
```

Se ha guardado correctamente

```
Element h_out[0] == 2 != 7
```

```
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
Element h_out[0] == 2 != 7
```

```
!nvprof ./stencil1d_shared
```

```
==5042== NVPROF is profiling process 5042, command: ./stencil1d_shared
Element h_out[0] == 2 != 7
==5042== Profiling application: ./stencil1d_shared
==5042== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   36.47%  5.2160us         1  5.2160us  5.2160us  5.2160us  [CUDA m
                   32.66%  4.6720us         1  4.6720us  4.6720us  4.6720us  stencil
                   30.87%  4.4160us         1  4.4160us  4.4160us  4.4160us  [CUDA m
      API calls:   99.69%  280.14ms         2  140.07ms  4.7570us  280.13ms  cudaMal
                    0.14%  406.24us         1  406.24us  406.24us  406.24us  cuDevic
                    0.06%  165.87us         2  82.933us  22.450us  143.42us  cudaFre
                    0.05%  142.87us       101  1.4140us     133ns  60.292us  cuDevic
                    0.03%  84.133us         2  42.066us  34.211us  49.922us  cudaMem
                    0.01%  26.855us         1  26.855us  26.855us  26.855us  cuDevic
                    0.01%  25.657us         1  25.657us  25.657us  25.657us  cudaLau
                    0.00%  6.1640us         1  6.1640us  6.1640us  6.1640us  cuDevic
                    0.00%  1.5060us         3     502ns     223ns     993ns  cuDevic
                    0.00%  1.3710us         2     685ns     382ns     989ns  cuDevic
                    0.00%     249ns         1     249ns     249ns     249ns  cuDevic
```

Resolver el problema con el uso de la memoria compartidas trae nuevos problemas. Debido a que físicamente los hilos no se pueden ejecutar estrictamente a la vez, estamos creando una condición de carrera, y por tanto hay hilos compitiendo por los mismos recursos de memoria. Para solventar esto, ahora veremos como podemos forzar un bloqueo, en el que todos los hilos tienen que ejecutar una llamada a la función __syncthreads() antes de que todos prosigan a la vez con la ejecución.

Stencil1D con memoria compartida y sincronización

```
%%writefile stencil1d_shared_sync.cu
```



```
#define RADIUS        3
```

```
#define BLOCK_SIZE    256
#define NUM_ELEMENTS  (4096*2)

// CUDA API error checking macro
#define cudaCheck(error) \
  if (error != cudaSuccess) { \
    printf("Fatal error: %s at %s:%d\n", \
      cudaGetErrorString(error), \
      __FILE__, __LINE__); \
    exit(1); \
  }

__global__ void stencil_1d(int *in, int *out)
{
    __shared__ int temp[BLOCK_SIZE + 2 * RADIUS];
    int gindex = threadIdx.x + (blockIdx.x * blockDim.x) + RADIUS;
    int lindex = threadIdx.x + RADIUS;

    // Read input elements into shared memory
    temp[lindex] = in[gindex];
    if (threadIdx.x < RADIUS)
    {
        temp[lindex - RADIUS] = in[gindex - RADIUS];
        temp[lindex + BLOCK_SIZE] = in[gindex + BLOCK_SIZE];
    }

    __syncthreads();

    // Apply the stencil
    int result = 0;
    for (int offset = -RADIUS ; offset <= RADIUS ; offset++)
        result += temp[lindex + offset];

    // Store the result
    out[gindex-RADIUS] = result;
}

int main()
{
  unsigned int i;
  int h_in[NUM_ELEMENTS + 2 * RADIUS], h_out[NUM_ELEMENTS];
  int *d_in, *d_out;

  // Initialize host data
  for( i = 0; i < (NUM_ELEMENTS + 2*RADIUS); ++i )
    h_in[i] = 1; // With a value of 1 and RADIUS of 3, all output values should be 7

  // Allocate space on the device
  cudaCheck( cudaMalloc( &d_in, (NUM_ELEMENTS + 2*RADIUS) * sizeof(int)) );
  cudaCheck( cudaMalloc( &d_out, NUM_ELEMENTS * sizeof(int)) );

  , (NUM_ELEMENTS + 2*RADIUS) * sizeof(int), cudaMemcpyHc

  stencil_1d<<< (NUM_ELEMENTS + BLOCK_SIZE - 1)/BLOCK_SIZE, BLOCK_SIZE >>> (d_in, d_out);
```

```
  cudaCheck( cudaMemcpy( h_out, d_out, NUM_ELEMENTS * sizeof(int), cudaMemcpyDeviceToHost)

  // Verify every out value is 7
  for( i = 0; i < NUM_ELEMENTS; ++i )
    if (h_out[i] != 7)
    {
      printf("Element h_out[%d] == %d != 7\n", i, h_out[i]);
      break;
    }

  if (i == NUM_ELEMENTS)
    printf("SUCCESS!\n");

  // Free out memory
  cudaFree(d_in);
  cudaFree(d_out);

  return 0;
}
```

```
Writing stencil1d_shared_sync.cu
```

```
!/usr/local/cuda/bin/nvcc -arch=sm_75 -rdc=true stencil1d_shared_sync.cu -o stencil1d_shar
```

Realizamos 100 ejecuciones para comprobar la consistencia de los resultados

```
for _ in range(100):

  !./stencil1d_shared_sync
```

```
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
```



```
SUCCESS!
```

```
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
SUCCESS!
```

Hacemos el profile de ejecución

```
!nvprof ./stencil1d_shared_sync
```

```
==5152== NVPROF is profiling process 5152, command: ./stencil1d_shared_sync
SUCCESS!
==5152== Profiling application: ./stencil1d_shared_sync
==5152== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   34.73%  5.0560us         1  5.0560us  5.0560us  5.0560us  [CUDA m
                   33.85%  4.9280us         1  4.9280us  4.9280us  4.9280us  stencil
                   31.43%  4.5760us         1  4.5760us  4.5760us  4.5760us  [CUDA m
      API calls:   99.75%  297.99ms         2  148.99ms  3.9540us  297.98ms  cudaMal
                    0.11%  336.57us         1  336.57us  336.57us  336.57us  cuDevic
                    0.05%  159.18us       101  1.5760us     128ns  65.739us  cuDevic
                    0.04%  123.28us         2  61.638us  10.045us  113.23us  cudaFre
                    0.03%  82.442us         2  41.221us  32.899us  49.543us  cudaMem
                    0.01%  27.411us         1  27.411us  27.411us  27.411us  cuDevic
                [illegible]970us         1  22.970us  22.970us  22.970us  cudaLau
                [illegible]820us         1  5.7820us  5.7820us  5.7820us  cuDevic
                    0.00%  1.5140us         3     504ns     201ns  1.0040us  cuDevic
```

Se ha guardado correctamente

```
 0.00%  1.3230us         2     661ns     371ns     952ns  cuDevic
 0.00%     233ns         1     233ns     233ns     233ns  cuDevic
```

Ahora hemos resuelto ambas cosas, el tiempo de copia es ligeramente más pequeño y el problema creado por el uso de memoria compartida está resuelto mediante sincronización. Esta sería una versión muchísimo más interesante para utilizar para tamaños mayores, donde estamos además haciendo un mejor uso de GPU.

PARTE OPCIONAL

Multiplicación de matrices

```
%%writefile matrix_mul.cu

#include <stdio.h>
#include <iostream>
#define N 16
#define MAX 8
void matrixMultCPU(int a[N][N], int b[N][N], int c[N][N]) {
  int n,m;
  for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
      int sum = 0;
      for (int k = 0; k < N; k++) {
        m = a[i][k];
        n = b[k][j];
        sum += m * n;
      }
      c[i][j] = sum;
    }
  }
}
__global__ void matrixMultGPU(int *a, int *b, int *c) {
  int k, sum = 0;
  int col = threadIdx.x + blockDim.x * blockIdx.x;
  int fil = threadIdx.y + blockDim.y * blockIdx.y;
  if (col < N && fil < N) {
    for (k = 0; k < N; k++) {
      sum += a[fil * N + k] * b[k * N + col];
    }
    c[fil * N + col] = sum;
  }
}
int main() {

  int cont,i,j;
```

```
cudaEvent_t start;
cudaEventCreate(&start);
cudaEvent_t stop;
cudaEventCreate(&stop);


/* inicializando variables con datos*/
for (i = 0; i < N; i++) {
  cont = 0;
  for (j = 0; j < N; j++) {
    a[i][j] = cont;
    b[i][j] = cont;
    cont++;
  }
}

int size = N * N * sizeof(int);

cudaMalloc((void **) &dev_a, size);
cudaMalloc((void **) &dev_b, size);
cudaMalloc((void **) &dev_c, size);

cudaMemcpy(dev_a, a, size, cudaMemcpyHostToDevice);
cudaMemcpy(dev_b, b, size, cudaMemcpyHostToDevice);

dim3 dimGrid((N + MAX - 1)/MAX, (N + MAX - 1)/MAX);
dim3 dimBlock(MAX, MAX);

int nIter = 1000;
cudaEventRecord(start, NULL);

 for (int j = 0; j < nIter; j++)
  matrixMultGPU<<<dimGrid, dimBlock>>>(dev_a, dev_b, dev_c);

cudaEventRecord(stop, NULL);
cudaEventSynchronize(stop);

float msecTotal = 0.0f;
cudaEventElapsedTime(&msecTotal, start, stop);

cudaMemcpy(c, dev_c, size, cudaMemcpyDeviceToHost);

cudaFree(dev_a);
cudaFree(dev_b);
cudaFree(dev_c);

for (int y = 0; y < N; y++) {
  for (int x = 0; x < N; x++) {
    printf("[%d][%d]=%d ", y, x, c[y][x]);
  }
```



```
float msecPerKernelExecution = msecTotal / nIter;
```

```
  double flopsPerMMul = 2.0 * N * N * N;
  double gigaFlops = (flopsPerMMul * 1.0e-9f) /
                     (msecPerKernelExecution / 1000.0f);

  std::cout << "gigaFlops ==> "<< gigaFlops << std::endl;

  return 0;
}
```

```
Overwriting matrix_mul.cu
```

```
!/usr/local/cuda/bin/nvcc -arch=sm_75 -rdc=true matrix_mul.cu -o matrix_mul -lcudadevrt
!nvprof ./matrix_mul
```

```
==3410== NVPROF is profiling process 3410, command: ./matrix_mul
[0][0]=0 [0][1]=120 [0][2]=240 [0][3]=360 [0][4]=480 [0][5]=600 [0][6]=720 [0][7]=84
[1][0]=0 [1][1]=120 [1][2]=240 [1][3]=360 [1][4]=480 [1][5]=600 [1][6]=720 [1][7]=84
[2][0]=0 [2][1]=120 [2][2]=240 [2][3]=360 [2][4]=480 [2][5]=600 [2][6]=720 [2][7]=84
[3][0]=0 [3][1]=120 [3][2]=240 [3][3]=360 [3][4]=480 [3][5]=600 [3][6]=720 [3][7]=84
[4][0]=0 [4][1]=120 [4][2]=240 [4][3]=360 [4][4]=480 [4][5]=600 [4][6]=720 [4][7]=84
[5][0]=0 [5][1]=120 [5][2]=240 [5][3]=360 [5][4]=480 [5][5]=600 [5][6]=720 [5][7]=84
[6][0]=0 [6][1]=120 [6][2]=240 [6][3]=360 [6][4]=480 [6][5]=600 [6][6]=720 [6][7]=84
[7][0]=0 [7][1]=120 [7][2]=240 [7][3]=360 [7][4]=480 [7][5]=600 [7][6]=720 [7][7]=84
[8][0]=0 [8][1]=120 [8][2]=240 [8][3]=360 [8][4]=480 [8][5]=600 [8][6]=720 [8][7]=84
[9][0]=0 [9][1]=120 [9][2]=240 [9][3]=360 [9][4]=480 [9][5]=600 [9][6]=720 [9][7]=84
[10][0]=0 [10][1]=120 [10][2]=240 [10][3]=360 [10][4]=480 [10][5]=600 [10][6]=720 [1
[11][0]=0 [11][1]=120 [11][2]=240 [11][3]=360 [11][4]=480 [11][5]=600 [11][6]=720 [1
[12][0]=0 [12][1]=120 [12][2]=240 [12][3]=360 [12][4]=480 [12][5]=600 [12][6]=720 [1
[13][0]=0 [13][1]=120 [13][2]=240 [13][3]=360 [13][4]=480 [13][5]=600 [13][6]=720 [1
[14][0]=0 [14][1]=120 [14][2]=240 [14][3]=360 [14][4]=480 [14][5]=600 [14][6]=720 [1
[15][0]=0 [15][1]=120 [15][2]=240 [15][3]=360 [15][4]=480 [15][5]=600 [15][6]=720 [1
gigaFlops ==> 1.45164
==3410== Profiling application: ./matrix_mul
==3410== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   99.88%  4.3555ms      1000  4.3550us  4.2560us  14.080us  matrixM
                    0.08%  3.2960us         2  1.6480us  1.4080us  1.8880us  [CUDA m
                    0.05%  2.0800us         1  2.0800us  2.0800us  2.0800us  [CUDA m
      API calls:   98.03%  313.05ms         2  156.52ms     803ns  313.05ms  cudaEve
                    1.23%  3.9356ms      1000  3.9350us  2.9840us  25.564us  cudaLau
                    0.45%  1.4242ms         1  1.4242ms  1.4242ms  1.4242ms  cudaEve
                    0.12%  397.84us         1  397.84us  397.84us  397.84us  cuDevic
                    0.05%  156.53us       101  1.5490us     129ns  63.158us  cuDevic
                    0.05%  146.01us         3  48.671us  2.2470us  140.52us  cudaMal
                    0.04%  125.97us         3  41.988us  2.4760us  113.55us  cudaFre
                    0.02%  56.893us         3  18.964us  8.2360us  29.866us  cudaMem
                    0.01%  28.785us         1  28.785us  28.785us  28.785us  cuDevic
                    0.00%  8.2840us         2  4.1420us  3.8820us  4.4020us  cudaEve
                    0.00%  6.1240us         1  6.1240us  6.1240us  6.1240us  cuDevic
                    0.00%  1.8270us         1  1.8270us  1.8270us  1.8270us  cudaEve
                    0.00%  1.4150us         3     471ns     205ns     931ns  cuDevic
                    0.00%  1.0230us         2     511ns     268ns     755ns  cuDevic
                    0.00%     298ns         1     298ns     298ns     298ns  cuDevic
```

Se ha guardado correctamente

```
#include <stdio.h>
#include <iostream>
#define N 16
#define MAX 8

void matrixMultCPU(int a[N][N], int b[N][N], int c[N][N]) {
  int n,m;
  for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
      int sum = 0;
      for (int k = 0; k < N; k++) {
        m = a[i][k];
        n = b[k][j];
        sum += m * n;
      }
      c[i][j] = sum;
    }
  }
}
__global__ void matrixMultGPU(int *a, int *b, int *c) {
  int sum = 0;

  int thr_x = threadIdx.x;
  int thr_y = threadIdx.y;
  int col = threadIdx.x + blockDim.x * blockIdx.x;
  int fil = threadIdx.y + blockDim.y * blockIdx.y;

  __shared__ float A[MAX][MAX];
  __shared__ float B[MAX][MAX];

  for(int i = 0; i < (N + MAX - 1)/MAX; i++){

    if(thr_y + (i*MAX) < N && col < N)
      A[thr_y][thr_x] = a[(col * N) + (thr_y + i*MAX)];
    else
      A[thr_y][thr_x] = 0.0f;

    if(thr_x + (i*MAX) < N && fil < N)
      B[thr_y][thr_x] = b[((thr_x + (i*MAX))*N) + fil];
    else
      B[thr_y][thr_x] = 0.0f;

    __syncthreads();

    for (int j = 0; j < MAX; j++){
            sum += A[j][thr_x] * B[thr_y][j];
    }

    __syncthreads();
  }
  if (col < N && fil < N) {
```

Se ha guardado correctamente

```
int main() {
  int a[N][N], b[N][N], c[N][N];
  int *dev_a, *dev_b, *dev_c;
  int cont,i,j;

  cudaEvent_t start;
  cudaEventCreate(&start);
  cudaEvent_t stop;
  cudaEventCreate(&stop);


  /* inicializando variables con datos*/
  for (i = 0; i < N; i++) {
    cont = 0;
    for (j = 0; j < N; j++) {
      a[i][j] = cont;
      b[i][j] = cont;
      cont++;
    }
  }

  int size = N * N * sizeof(int);

  cudaMalloc((void **) &dev_a, size);
  cudaMalloc((void **) &dev_b, size);
  cudaMalloc((void **) &dev_c, size);

  cudaMemcpy(dev_a, a, size, cudaMemcpyHostToDevice);
  cudaMemcpy(dev_b, b, size, cudaMemcpyHostToDevice);

  dim3 dimGrid((N + MAX - 1)/MAX, (N + MAX - 1)/MAX);
  dim3 dimBlock(MAX, MAX);

  int nIter = 1000;
  cudaEventRecord(start, NULL);

   for (int j = 0; j < nIter; j++)
    matrixMultGPU<<<dimGrid, dimBlock>>>(dev_a, dev_b, dev_c);

  cudaEventRecord(stop, NULL);
  cudaEventSynchronize(stop);

  float msecTotal = 0.0f;
  cudaEventElapsedTime(&msecTotal, start, stop);

  cudaMemcpy(c, dev_c, size, cudaMemcpyDeviceToHost);

  cudaFree(dev_a);
  cudaFree(dev_b);
  cudaFree(dev_c);

                    ecTotal / nIter;
                          * N;
  double gigaFlops = (flopsPerMMul * 1.0e-9f) /
```

```
                    (msecPerKernelExecution / 1000.0f);

  std::cout << "gigaFlops ==> "<< gigaFlops << std::endl;

  return 0;
}
```

```
Overwriting matrix_mul_shared.cu
```

```
%%shell
for N in 16 32 64 128 256 512
do
  sed -i "/#define N/c\#define N ${N}" matrix_mul.cu
  /usr/local/cuda/bin/nvcc -arch=sm_75 -rdc=true matrix_mul.cu -o matrix_mul -lcudadevrt
  nvprof ./matrix_mul
done
```

```
==3121== NVPROF is profiling process 3121, command: ./matrix_mul
gigaFlops ==> 1.45686
==3121== Profiling application: ./matrix_mul
==3121== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   99.88%  4.3447ms      1000  4.3440us  4.2560us  10.912us  matri
                    0.07%  3.2320us         2  1.6160us  1.4080us  1.8240us  [CUDA
                    0.05%  2.1120us         1  2.1120us  2.1120us  2.1120us  [CUDA
      API calls:   97.68%  311.90ms         2  155.95ms     919ns  311.90ms  cudaE
                    1.29%  4.1124ms      1000  4.1120us  3.2270us  25.177us  cudaL
                    0.56%  1.7886ms         1  1.7886ms  1.7886ms  1.7886ms  cudaE
                    0.24%  751.74us         3  250.58us  2.5920us  738.79us  cudaF
                    0.11%  349.23us         1  349.23us  349.23us  349.23us  cuDev
                    0.05%  152.53us         3  50.841us  2.5720us  146.64us  cudaM
                    0.05%  150.58us       101  1.4900us     131ns  63.916us  cuDev
                    0.02%  56.584us         3  18.861us  8.1340us  30.787us  cudaM
                    0.01%  29.647us         1  29.647us  29.647us  29.647us  cuDev
                    0.00%  11.218us         2  5.6090us  3.3650us  7.8530us  cudaE
                    0.00%  6.1370us         1  6.1370us  6.1370us  6.1370us  cuDev
                    0.00%  2.3460us         1  2.3460us  2.3460us  2.3460us  cudaE
                    0.00%  1.7510us         3     583ns     204ns  1.2520us  cuDev
                    0.00%  1.4530us         2     726ns     211ns  1.2420us  cuDev
                    0.00%     256ns         1     256ns     256ns     256ns  cuDev
==3168== NVPROF is profiling process 3168, command: ./matrix_mul
gigaFlops ==> 10.5287
==3168== Profiling application: ./matrix_mul
==3168== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   99.88%  4.9180ms      1000  4.9180us  4.8310us  14.016us  matri
                    0.07%  3.6800us         2  1.8400us  1.6640us  2.0160us  [CUDA
                    0.05%  2.2400us         1  2.2400us  2.2400us  2.2400us  [CUDA
      API calls:   97.64%  280.96ms         2  140.48ms     853ns  280.96ms  cudaE
                    1.47%  4.2392ms      1000  4.2390us  3.1820us  40.381us  cudaL
                    0.56%  1.6163ms         1  1.6163ms  1.6163ms  1.6163ms  cudaE
                    0.13%  369.16us         1  369.16us  369.16us  369.16us  cuDev
                    0.06%  171.66us       101  1.6990us     145ns  86.574us  cuDev
                    0.05%  134.96us         3  44.987us  2.6680us  129.07us  cudaM
                    0.04%  133.84us         3  44.614us  2.8470us  121.83us  cudaF
                    0.03%  81.261us         3  27.087us  5.9590us  44.396us  cudaM
                    0.01%  26.780us         1  26.780us  26.780us  26.780us  cuDev
```

Se ha guardado correctamente

```
                    0.00%  8.5500us         2  4.2750us  4.2250us  4.3250us  cudaE
                    0.00%  6.2660us         1  6.2660us  6.2660us  6.2660us  cuDev
                    0.00%  2.2650us         1  2.2650us  2.2650us  2.2650us  cudaE
                    0.00%  1.7670us         3     589ns     238ns  1.2580us  cuDev
                    0.00%  1.4500us         2     725ns     446ns  1.0040us  cuDev
                    0.00%     311ns         1     311ns     311ns     311ns  cuDev
==3213== NVPROF is profiling process 3213, command: ./matrix_mul
gigaFlops ==> 52.7781
==3213== Profiling application: ./matrix_mul
==3213== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   99.89%  8.6967ms      1000  8.6960us  8.6070us  13.247us  matri
                    0.07%  6.4000us         2  3.2000us  3.1680us  3.2320us  [CUDA
                    0.04%  3.1040us         1  3.1040us  3.1040us  3.1040us  [CUDA
      API calls:   96.63%  304.63ms         2  152.31ms     887ns  304.63ms  cudaE
                    1.79%  5.6327ms         1  5.6327ms  5.6327ms  5.6327ms  cudaE
                    1.27%  4.0184ms      1000  4.0180us  3.0470us  24.000us  cudaL
```

SIN MEMORIA COMPARTIDA

Tamaño de la matriz	CPU -> GPU(msec)	GPU -> CPU(msec)	Ejecución(msec)	Ratio vs 128x128	GFLOPs
16x16	0.003	0.002	4.345	0.123	1.46
32x32	0.004	0.002	4.918	0.139	10.52
64x64	0.006	0.003	8.697	0,24	52.78
128x128	0.019	0.006	35.394	1	114.46
256x256	0.048	0.021	213.79	6.04	156.09
512x512	0.177	0.081	813.98	23	329.47

```
%%shell
for N in 16 32 64 128 256 512
do
  sed -i "/#define N/c\#define N ${N}" matrix_mul_shared.cu
  /usr/local/cuda/bin/nvcc -arch=sm_75 -rdc=true matrix_mul_shared.cu -o matrix_mul_sharec
  nvprof ./matrix_mul_shared
done
```

```
==3769== NVPROF is profiling process 3769, command: ./matrix_mul_shared
gigaFlops ==> 1.22003
==3769== Profiling application: ./matrix_mul_shared
==3769== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   99.90%  5.4144ms      1000  5.4140us  5.3430us  9.1520us  matri
                    0.06%  3.2000us         2  1.6000us  1.4080us  1.7920us  [CUDA
                    0.04%  2.0800us         1  2.0800us  2.0800us  2.0800us  [CUDA
      API calls:   97.73%  318.70ms         2  159.35ms     948ns  318.70ms  cudaE
                    1.26%  4.0970ms      1000  4.0970us  3.0310us  92.854us  cudaL
                    0.71%  2.3142ms         1  2.3142ms  2.3142ms  2.3142ms  cudaE
                    0.11%  369.14us         1  369.14us  369.14us  369.14us  cuDev
                    0.06%  185.12us       101  1.8320us     128ns  84.954us  cuDev
                    0.05%  150.67us         3  50.221us  2.7230us  144.47us  cudaM
                    0.04%  120.00us         3  40.000us  2.4810us  107.94us  cudaF
                    0.02%  77.802us         3  25.934us  7.9190us  48.381us  cudaM
                    0.01%  43.927us         2  21.963us  4.1350us  39.792us  cudaE
                    0.01%  30.896us         1  30.896us  30.896us  30.896us  cuDev
                    0.00%  6.9460us         1  6.9460us  6.9460us  6.9460us  cuDev
```

Se ha guardado correctamente

```
                    0.00%  2.4130us         1  2.4130us  2.4130us  2.4130us  cudaE
                    0.00%  2.1450us         3     715ns     198ns  1.5870us  cuDev
                    0.00%  1.3120us         2     656ns     312ns  1.0000us  cuDev
                    0.00%     295ns         1     295ns     295ns     295ns  cuDev
==3814== NVPROF is profiling process 3814, command: ./matrix_mul_shared
gigaFlops ==> 8.06954
==3814== Profiling application: ./matrix_mul_shared
==3814== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   99.91%  6.7758ms      1000  6.7750us  6.6880us  12.127us  matri
                    0.05%  3.6160us         2  1.8080us  1.6320us  1.9840us  [CUDA
                    0.03%  2.2720us         1  2.2720us  2.2720us  2.2720us  [CUDA
      API calls:   97.03%  280.39ms         2  140.19ms     836ns  280.39ms  cudaE
                    1.80%  5.2077ms      1000  5.2070us  3.0750us  42.384us  cudaL
                    0.86%  2.4805ms         1  2.4805ms  2.4805ms  2.4805ms  cudaE
                    0.12%  340.74us         1  340.74us  340.74us  340.74us  cuDev
                    0.06%  160.39us         3  53.463us  3.1840us  152.16us  cudaM
                    0.06%  159.39us       101  1.5780us     130ns  78.041us  cuDev
                    0.04%  121.06us         3  40.354us  2.4530us  109.78us  cudaF
                    0.02%  68.382us         3  22.794us  9.4250us  34.477us  cudaM
                    0.01%  30.405us         1  30.405us  30.405us  30.405us  cuDev
                    0.00%  10.665us         2  5.3320us  5.0290us  5.6360us  cudaE
                    0.00%  7.1910us         1  7.1910us  7.1910us  7.1910us  cuDev
                    0.00%  3.5510us         1  3.5510us  3.5510us  3.5510us  cudaE
                    0.00%  1.7130us         3     571ns     218ns  1.1610us  cuDev
                    0.00%  1.2260us         2     613ns     284ns     942ns  cuDev
                    0.00%     260ns         1     260ns     260ns     260ns  cuDev
==3859== NVPROF is profiling process 3859, command: ./matrix_mul_shared
gigaFlops ==> 41.2411
==3859== Profiling application: ./matrix_mul_shared
==3859== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   99.92%  11.482ms      1000  11.482us  11.392us  12.384us  matri
                    0.06%  6.4640us         2  3.2320us  2.8800us  3.5840us  [CUDA
                    0.03%  3.0720us         1  3.0720us  3.0720us  3.0720us  [CUDA
      API calls:   95.82%  305.92ms         2  152.96ms     853ns  305.92ms  cudaE
                    2.60%  8.3129ms         1  8.3129ms  8.3129ms  8.3129ms  cudaE
```

CON MEMORIA COMPARTIDA

Tamaño de la matriz	CPU -> GPU(msec)	GPU -> CPU(msec)	Ejecución(msec)	Ratio vs 128x128	GFLOPs
16x16	0.003	0.002	5.414	0.146	1.22
32x32	0.004	0.002	6.775	0.183	8.06
64x64	0.006	0.003	11.482	0.311	41.24
128x128	0.018	0.007	36.935	1	109.85
256x256	0.049	0.022	209.52	5.67	159.21
512x512	0.175	0.09	746.66	20.215	359.15

Respuesta: Lo que podemos comprobar a la hora de realizar estas pruebas es que la memoria compartida puede ser un recurso muy útil para poder optimizar el rendimiento de algoritmos. Las tablas, y sobre todo los ratios, indican que hacer un uso de memoria compartida no tiene [illegible] usando poca memoria, pero a medida que el tamaño de la matriz aumenta (que además recordemos que aumenta los tiempos de ejecución en gran

medida debido a la complejidad del problema) el ratio de tiempos de ejecución respecto a 128x128 es menor que sin memoria compartida. Sin embargo también vemos un ligero aumento en la versión de memoria compartida para tamaños menores a 128x128, de ahí la conclusión que hemos sacado

Ejemplos de CUDA

```
%cd /usr/local/cuda/samples/
!ls
```

```
/usr/local/cuda-11.2/samples
```

```
%cd  0_Simple/matrixMul/
%ls
```

```
/usr/local/cuda-11.2/samples/0_Simple/matrixMul
Makefile  matrixMul.cu  NsightEclipse.xml  readme.txt
```

```
!cat matrixMul.cu
```

```
/**
 * Copyright 1993-2015 NVIDIA Corporation.  All rights reserved.
 *
 * Please refer to the NVIDIA end user license agreement (EULA) associated
 * with this source code for terms and conditions that govern your use of
 * this software. Any use, reproduction, disclosure, or distribution of
 * this software and related documentation outside the terms of the EULA
 * is strictly prohibited.
 *
 */

/**
 * Matrix multiplication: C = A * B.
 * Host code.
 *
 * This sample implements matrix multiplication which makes use of shared memory
 * to ensure data reuse, the matrix multiplication is done using tiling
 * approach. It has been written for clarity of exposition to illustrate various
 * CUDA programming principles, not with the goal of providing the most
 * performant generic kernel for matrix multiplication. See also: V. Volkov and
 * J. Demmel, "Benchmarking GPUs to tune dense linear algebra," in Proc. 2008
 * ACM/IEEE Conf. on Supercomputing (SC '08), Piscataway, NJ: IEEE Press, 2008,
 * pp. Art. 31:1-11.
 */

// System includes
#include <assert.h>
#include <stdio.h>
```

```
// Helper functions and utilities to work with CUDA
#include <helper_cuda.h>
#include <helper_functions.h>

/**
 * Matrix multiplication (CUDA Kernel) on the device: C = A * B
 * wA is A's width and wB is B's width
 */
template <int BLOCK_SIZE>
__global__ void MatrixMulCUDA(float *C, float *A, float *B, int wA, int wB) {
  // Block index
  int bx = blockIdx.x;
  int by = blockIdx.y;

  // Thread index
  int tx = threadIdx.x;
  int ty = threadIdx.y;

  // Index of the first sub-matrix of A processed by the block
  int aBegin = wA * BLOCK_SIZE * by;

  // Index of the last sub-matrix of A processed by the block
  int aEnd = aBegin + wA - 1;

  // Step size used to iterate through the sub-matrices of A
```

```
!/usr/local/cuda/bin/nvcc -I /usr/local/cuda/samples/common/inc -arch=sm_75 -rdc=true matr
```

```
!nvprof ./matrixMul
```

```
[Matrix Multiply Using CUDA] - Starting...
==4364== NVPROF is profiling process 4364, command: ./matrixMul
GPU Device 0: "Turing" with compute capability 7.5

MatrixA(320,320), MatrixB(640,320)
Computing result using CUDA Kernel...
done
Performance= 328.79 GFlop/s, Time= 0.399 msec, Size= 131072000 Ops, WorkgroupSize= 1
Checking computed result for correctness: Result = PASS

NOTE: The CUDA Samples are not meant for performancemeasurements. Results may vary w
==4364== Profiling application: ./matrixMul
==4364== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   99.86%  119.63ms       301  397.45us  322.30us  412.09us  void Ma
                    0.09%  107.42us         2  53.711us  38.687us  68.735us  [CUDA m
                    0.05%  63.551us         1  63.551us  63.551us  63.551us  [CUDA m
      API calls:   71.88%  310.35ms         3  103.45ms  4.8070us  310.34ms  cudaHos
                   27.38%  118.21ms         1  118.21ms  118.21ms  118.21ms  cudaEve
                    0.30%  1.3063ms       301  4.3390us  3.4460us  36.752us  cudaLau
                    0.12%  535.62us         2  267.81us  68.696us  466.92us  cudaStr
                    0.10%  430.60us         3  143.53us  5.0620us  405.93us  cudaFre
                    0.08%  341.76us         1  341.76us  341.76us  341.76us  cuDevic
                    0.04%  177.84us       101  1.7600us     128ns  76.676us  cuDevic
                                .65us         3  44.217us  1.9670us  127.43us  cudaMal
                                833us         3  29.277us  2.3090us  79.348us  cudaFre
                                509us         7  8.3720us     301ns  56.249us  cudaDev
                    0.01%  46.555us         3  15.518us  4.3550us  22.177us  cudaMem
```

Se ha guardado correctamente

```
 0.01%  36.399us         2  18.199us     621ns  35.778us  cudaEve
 0.01%  29.277us         1  29.277us  29.277us  29.277us  cuDevic
 0.00%  18.713us         1  18.713us  18.713us  18.713us  cudaStr
 0.00%  7.5090us         2  3.7540us  2.0850us  5.4240us  cudaEve
 0.00%  6.6920us         1  6.6920us  6.6920us  6.6920us  cuDevic
 0.00%  5.1740us         1  5.1740us  5.1740us  5.1740us  cudaEve
 0.00%  2.5720us         1  2.5720us  2.5720us  2.5720us  cudaSet
 0.00%  2.0880us         2  1.0440us     451ns  1.6370us  cudaEve
 0.00%  1.3850us         3     461ns     177ns     924ns  cuDevic
 0.00%  1.0930us         2     546ns     197ns     896ns  cuDevic
 0.00%     923ns         1     923ns     923ns     923ns  cudaGet
 0.00%     298ns         1     298ns     298ns     298ns  cuDevic
```

Respuesta: Como podemos ver en el perfilado, y sin mucha sorpresa, el rendimiento (medido en GFLOPs) que tiene esta implementación es bastante superior al que obtenemos en nuestra implementación. Esto se debe a que la elección de kernel de ejecución es un poco más cuidadosa que nuestra implementación, consiguiendo optimizar el uso de recursos de la GPU.

Estudio de reducciones

```
%%writefile labsolReduction0.cu
// includes, kernels
#include <stdio.h>
#include <assert.h>
#define NUM_ELEMENTS 512
// **===-----------------------------------------------------------------===**
//! @param g_idata  input data in global memory
//                  result is expected in index 0 of g_idata
//! @param n        input number of elements to scan from input data
// **===-----------------------------------------------------------------===**
__global__ void reduction(float *g_data, int n)
{
  int stride;
  // Define shared memory
  __shared__ float scratch[NUM_ELEMENTS];
  // Load the shared memory
  scratch[threadIdx.x ] = g_data[threadIdx.x];
  if(threadIdx.x + blockDim.x < n)
    scratch[threadIdx.x + blockDim.x] = g_data[threadIdx.x + blockDim.x];
  __syncthreads();
  // Do sum reduction from shared memory
  for(stride = 1 ; stride < blockDim.x; stride *= 2)
  {
    __syncthreads();
    if(threadIdx.x % (2*stride) == 0)
        scratch[threadIdx.x] += scratch[threadIdx.x + stride];
  }
                                                    emory

    g_data[0] = scratch[0];
```

Se ha guardado correctamente

```
  return;
}


void runTest( int argc, char** argv);
float computeOnDevice(float* h_data, int array_mem_size);
extern "C" void computeGold( float* reference, float* idata, const unsigned int len);
int main( int argc, char** argv)
{
    runTest( argc, argv);
    return EXIT_SUCCESS;
}

////////////////////////////////////////////////////////////////////////////////
//! Run naive scan test
////////////////////////////////////////////////////////////////////////////////
void runTest( int argc, char** argv)
{
    int num_elements = NUM_ELEMENTS;
    const unsigned int array_mem_size = sizeof( float) * num_elements;
    // allocate host memory to store the input data
    float* h_data = (float*) malloc( array_mem_size);
    // * No arguments: Randomly generate input data and compare against the host's

            // initialize the input data on the host to be integer values
            // between 0 and 1000
            for( unsigned int i = 0; i < num_elements; ++i)
            {
                //h_data[i] = floorf(1000*(rand()/(float)RAND_MAX));
                h_data[i] = i*1.0;
            }

        // compute reference solution
    float reference = 0.0f;
    computeGold(&reference , h_data, num_elements);


    float result = computeOnDevice(h_data, num_elements);
    // We can use an epsilon of 0 since values are integral and in a range
    // that can be exactly represented
    float epsilon = 0.0f;
    unsigned int result_regtest = (abs(result - reference) <= epsilon);
    printf( "Test %s\n", (1 == result_regtest) ? "PASSED" : "FAILED");
    printf( "device: %f  host: %f\n", result, reference);
    // cleanup memory
    free( h_data);
}
////////////////////////////////////////////////////////////////////
// Take h_data from host, copies it to device, setup grid and thread
// dimentions, excutes kernel function, and copy result of scan back
// to h_data.
                                    input and the output of this function.
                                    ////////////////////////////////////
                                    int num_elements)
{
```

Se ha guardado correctamente

```
  float* d_data = NULL;
  float result;
  // Memory allocation on device side
  cudaMalloc((void**)&d_data, num_elements*sizeof(float));
  // Copy from host memory to device memory
  cudaMemcpy(d_data, h_data, num_elements*sizeof(float), cudaMemcpyHostToDevice);
  //int threads = (num_elements/2) + num_elements%2;
  int threads = num_elements;
  // Invoke the kernel
  reduction<<<1,threads>>>(d_data,num_elements);
  // Copy from device memory back to host memory
  cudaMemcpy(&result, d_data, sizeof(float), cudaMemcpyDeviceToHost);
  cudaFree(d_data);
  return result;
}

////////////////////////////////////////////////////////////////////
void computeGold( float* reference, float* idata, const unsigned int len)
{
  reference[0] = 0;
  double total_sum = 0;
  unsigned int i;
  for( i = 0; i < len; ++i)
  {
      total_sum += idata[i];
  }
  reference[0] = total_sum;
}
////////////////////////////////////////////////////////////////////
```

```
Writing labsolReduction0.cu
```

```
!/usr/local/cuda/bin/nvcc -I /usr/local/cuda/samples/common/inc -arch=sm_75 -rdc=true labs
```

```
!nvprof ./labsolReduction0
```

```
==179== NVPROF is profiling process 179, command: ./labsolReduction0
Test PASSED
device: 130816.000000  host: 130816.000000
==179== Profiling application: ./labsolReduction0
==179== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   70.88%  10.048us         1  10.048us  10.048us  10.048us  reducti
                   14.90%  2.1120us         1  2.1120us  2.1120us  2.1120us  [CUDA m
                   14.22%  2.0160us         1  2.0160us  2.0160us  2.0160us  [CUDA m
      API calls:   99.67%  387.80ms         1  387.80ms  387.80ms  387.80ms  cudaMal
                    0.14%  547.59us         1  547.59us  547.59us  547.59us  cuDevic
                    0.09%  363.99us         1  363.99us  363.99us  363.99us  cuDevic
                    0.04%  162.37us       101  1.6070us     131ns  69.511us  cuDevic
                    0.03%  101.59us         1  101.59us  101.59us  101.59us  cudaFre
                    0.01%  50.020us         2  25.010us  21.045us  28.975us  cudaMem
                           .099us         1  33.099us  33.099us  33.099us  cuDevic
                           .815us         1  27.815us  27.815us  27.815us  cudaLau
                           .070us         3     502ns     200ns  1.0050us  cuDevic
```

```
                    0.00%  1.3890us         2     694ns     298ns  1.0910us  cuDevic
                    0.00%     298ns         1     298ns     298ns     298ns  cuDevic
```

Caso 1

```
%%writefile labsolReduction1.cu
// includes, kernels
#include <stdio.h>
#include <assert.h>
#define NUM_ELEMENTS 512
// **===-----------------------------------------------------------------------===**
//! @param g_idata  input data in global memory
//                  result is expected in index 0 of g_idata
//! @param n        input number of elements to scan from input data
// **===-----------------------------------------------------------------------===**
__global__ void reduction(float *g_data, int n)
{
  int stride;
  // Define shared memory
  __shared__ float scratch[NUM_ELEMENTS];
  // Load the shared memory
  scratch[threadIdx.x ] = g_data[threadIdx.x];
  if(threadIdx.x + blockDim.x < n)
    scratch[threadIdx.x + blockDim.x] = g_data[threadIdx.x + blockDim.x];
  __syncthreads();
  // Do sum reduction from shared memory
  for(stride = NUM_ELEMENTS/2 ; stride >= 1; stride >>= 1) {
      if(threadIdx.x < stride)
         scratch[threadIdx.x] += scratch[threadIdx.x + stride];
      __syncthreads();
  }
  // Store results back to global memory
  if(threadIdx.x == 0)
    g_data[0] = scratch[0];
  return;
}
////////////////////////////////////////////////////////////////////////////////
// Program main
////////////////////////////////////////////////////////////////////////////////
void runTest( int argc, char** argv);
float computeOnDevice(float* h_data, int array_mem_size);
extern "C" void computeGold( float* reference, float* idata, const unsigned int len);
int main( int argc, char** argv)
{
    runTest( argc, argv);
    return EXIT_SUCCESS;
}
////////////////////////////////////////////////////////////////////////////////
//! Run naive scan test
////////////////////////////////////////////////////////////////////////////////
                                                    )
{
```

Se ha guardado correctamente

```
    int num_elements = NUM_ELEMENTS;
    const unsigned int array_mem_size = sizeof( float) * num_elements;
    // allocate host memory to store the input data
    float* h_data = (float*) malloc( array_mem_size);
    // * No arguments: Randomly generate input data and compare against the host's

            // initialize the input data on the host to be integer values
            // between 0 and 1000
            for( unsigned int i = 0; i < num_elements; ++i)
            {
                //h_data[i] = floorf(1000*(rand()/(float)RAND_MAX));
                h_data[i] = i*1.0;
            }

        // compute reference solution
    float reference = 0.0f;
    computeGold(&reference , h_data, num_elements);
    float result;
    for (int i=0; i<100; i++)
      result = computeOnDevice(h_data, num_elements);
    // We can use an epsilon of 0 since values are integral and in a range
    // that can be exactly represented
    float epsilon = 0.0f;
    unsigned int result_regtest = (abs(result - reference) <= epsilon);
    printf( "Test %s\n", (1 == result_regtest) ? "PASSED" : "FAILED");
    printf( "device: %f  host: %f\n", result, reference);
    // cleanup memory
    free( h_data);
}
////////////////////////////////////////////////////////////////////////////
// Take h_data from host, copies it to device, setup grid and thread
// dimentions, excutes kernel function, and copy result of scan back
// to h_data.
// Note: float* h_data is both the input and the output of this function.
////////////////////////////////////////////////////////////////////////////
float computeOnDevice(float* h_data, int num_elements) {
  float* d_data = NULL;
  float result;
  // Memory allocation on device side
  cudaMalloc((void**)&d_data, num_elements*sizeof(float));
  // Copy from host memory to device memory
  cudaMemcpy(d_data, h_data, num_elements*sizeof(float), cudaMemcpyHostToDevice);
  int threads = (num_elements/2) + num_elements%2;
  // Invoke the kernel
  reduction<<<1,threads>>>(d_data,num_elements);
  // Copy from device memory back to host memory
  cudaMemcpy(&result, d_data, sizeof(float), cudaMemcpyDeviceToHost);
  cudaFree(d_data);
  return result;
}

                                    ////////////////////////////////////
                                    float* idata, const unsigned int len)
{
  reference[0] = 0;
```

Se ha guardado correctamente

```
  double total_sum = 0;
  unsigned int i;
  for( i = 0; i < len; ++i)
  {
      total_sum += idata[i];
  }
  reference[0] = total_sum;
}
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
```

```
Overwriting labsolReduction1.cu
```

```
!/usr/local/cuda/bin/nvcc -I /usr/local/cuda/samples/common/inc -arch=sm_35 -rdc=true labs
```

```
nvcc warning : The 'compute_35', 'compute_37', 'compute_50', 'sm_35', 'sm_37' and 's
```

```
!nvprof ./labsolReduction1
```

```
==387== NVPROF is profiling process 387, command: ./labsolReduction1
Test PASSED
device: 130816.000000  host: 130816.000000
==387== Profiling application: ./labsolReduction1
==387== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   64.18%  599.67us       100  5.9960us  5.7910us  6.6550us  reducti
                   18.62%  173.98us       100  1.7390us  1.6960us  2.0480us  [CUDA m
                   17.19%  160.64us       100  1.6060us  1.5360us  2.1120us  [CUDA m
      API calls:   97.39%  384.93ms       100  3.8493ms  86.992us  376.04ms  cudaMal
                    1.59%  6.2747ms       100  62.747us  55.281us  178.27us  cudaFre
                    0.65%  2.5789ms       200  12.894us  5.4650us  120.75us  cudaMem
                    0.21%  832.63us       100  8.3260us  6.5480us  39.238us  cudaLau
                    0.10%  376.19us         1  376.19us  376.19us  376.19us  cuDevic
                    0.05%  185.77us       101  1.8390us     135ns  80.561us  cuDevic
                    0.01%  42.589us         1  42.589us  42.589us  42.589us  cuDevic
                    0.00%  6.6440us         1  6.6440us  6.6440us  6.6440us  cuDevic
                    0.00%  1.7620us         3     587ns     213ns  1.2310us  cuDevic
                    0.00%  1.3200us         2     660ns     250ns  1.0700us  cuDevic
                    0.00%     256ns         1     256ns     256ns     256ns  cuDevic
```

En el primer caso, lo primero que estamos viendo es que hace un enfoque en el se van realizando sucesivas reducciones, sumando elemento a elemento los de la mitad inferior con sus correspondientes posiciones en la mitad superior. Este método también mantiene la memoria agrupada además tratando los posible conflictos en el banco de memoria debido a que las sucesivas ejecuciones se guardan en memoria en posiciones secuenciales.

Se ha guardado correctamente

```
%%writefile labsolReduction2.cu
// includes, kernels
#include <stdio.h>
#include <assert.h>
#define NUM_ELEMENTS 512

// **===-----------------------------------------------------------------===**
//! @param g_idata  input data in global memory
//                  result is expected in index 0 of g_idata
//! @param n        input number of elements to scan from input data
// **===-----------------------------------------------------------------===**
__global__ void reduction(float *g_data, int n)
{
  int stride;
  // Define shared memory
  __shared__ float scratch[NUM_ELEMENTS];
  // Load the shared memory
  scratch[threadIdx.x ] = g_data[threadIdx.x];
  if(threadIdx.x + blockDim.x < n)
    scratch[threadIdx.x + blockDim.x] = g_data[threadIdx.x + blockDim.x];
  __syncthreads();
  // Do sum reduction from shared memory
  //Reduction scheme 2
  for (stride = NUM_ELEMENTS; stride > 1; stride >>= 1) {
     __syncthreads();
     if (threadIdx.x < (stride>>1)) {
        scratch[threadIdx.x] += scratch[stride - threadIdx.x - 1];
    }
  }
   __syncthreads();
     // Store results back to global memory
  if(threadIdx.x == 0)
    g_data[0] = scratch[0];
  return;
}

////////////////////////////////////////////////////////////////////////////////
// Program main
////////////////////////////////////////////////////////////////////////////////
void runTest( int argc, char** argv);
float computeOnDevice(float* h_data, int array_mem_size);
extern "C" void computeGold( float* reference, float* idata, const unsigned int len);
int main( int argc, char** argv)
{
    runTest( argc, argv);
    return EXIT_SUCCESS;
}
////////////////////////////////////////////////////////////////////////////////
//! Run naive scan test
////////////////////////////////////////////////////////////////////////////////
void runTest( int argc, char** argv)
{
    unsigned int num_elements = NUM_ELEMENTS;
    const unsigned int mem_size = sizeof( float) * num_elements;
    // allocate host memory to store the input data
```

```
    float* h_data = (float*) malloc( array_mem_size);
    // * No arguments: Randomly generate input data and compare against the host's

            // initialize the input data on the host to be integer values
            // between 0 and 1000
            for( unsigned int i = 0; i < num_elements; ++i)
            {
                //h_data[i] = floorf(1000*(rand()/(float)RAND_MAX));
                h_data[i] = i*1.0;
            }

        // compute reference solution
    float reference = 0.0f;
    computeGold(&reference , h_data, num_elements);
    float result;
    for (int i=0; i<100; i++)
      result = computeOnDevice(h_data, num_elements);
    // We can use an epsilon of 0 since values are integral and in a range
    // that can be exactly represented
    float epsilon = 0.0f;
    unsigned int result_regtest = (abs(result - reference) <= epsilon);
    printf( "Test %s\n", (1 == result_regtest) ? "PASSED" : "FAILED");
    printf( "device: %f  host: %f\n", result, reference);
    // cleanup memory
    free( h_data);
}
////////////////////////////////////////////////////////////////////////////
// Take h_data from host, copies it to device, setup grid and thread
// dimentions, excutes kernel function, and copy result of scan back
// to h_data.
// Note: float* h_data is both the input and the output of this function.
////////////////////////////////////////////////////////////////////////////
float computeOnDevice(float* h_data, int num_elements)
{
  float* d_data = NULL;
  float result;
  // Memory allocation on device side
  cudaMalloc((void**)&d_data, num_elements*sizeof(float));
  // Copy from host memory to device memory
  cudaMemcpy(d_data, h_data, num_elements*sizeof(float), cudaMemcpyHostToDevice);
  int threads = (num_elements/2) + num_elements%2;
  // Invoke the kernel
    reduction<<<1,threads>>>(d_data,num_elements);
    // Copy from device memory back to host memory
  cudaMemcpy(&result, d_data, sizeof(float), cudaMemcpyDeviceToHost);
  cudaFree(d_data);
  return result;
}

////////////////////////////////////////////////////////////////////
void computeGold( float* reference, float* idata, const unsigned int len)
```

Se ha guardado correctamente

```
  unsigned int i;
```

```
  for( i = 0; i < len; ++i)
  {
      total_sum += idata[i];
  }
  reference[0] = total_sum;
}
////////////////////////////////////////////////////////////////////////////
```

```
Writing labsolReduction2.cu
```

```
!/usr/local/cuda/bin/nvcc -I /usr/local/cuda/samples/common/inc -arch=sm_35 -rdc=true labs
```

```
nvcc warning : The 'compute_35', 'compute_37', 'compute_50', 'sm_35', 'sm_37' and 's
```

```
!nvprof ./labsolReduction2
```

```
==435== NVPROF is profiling process 435, command: ./labsolReduction2
Test PASSED
device: 130816.000000  host: 130816.000000
==435== Profiling application: ./labsolReduction2
==435== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   64.90%  618.08us       100  6.1800us  5.8560us  6.6240us  reducti
                   18.27%  174.05us       100  1.7400us  1.6950us  2.0480us  [CUDA m
                   16.83%  160.29us       100  1.6020us  1.5680us  2.0480us  [CUDA m
      API calls:   97.34%  370.68ms       100  3.7068ms  82.964us  361.88ms  cudaMal
                    1.58%  6.0052ms       100  60.052us  55.100us  120.26us  cudaFre
                    0.71%  2.7213ms       200  13.606us  5.4740us  31.185us  cudaMem
                    0.21%  792.02us       100  7.9200us  6.3300us  28.363us  cudaLau
                    0.10%  364.76us         1  364.76us  364.76us  364.76us  cuDevic
                    0.05%  199.84us       101  1.9780us     147ns  109.18us  cuDevic
                    0.01%  29.398us         1  29.398us  29.398us  29.398us  cuDevic
                    0.00%  6.7720us         1  6.7720us  6.7720us  6.7720us  cuDevic
                    0.00%  1.9330us         3     644ns     215ns  1.3530us  cuDevic
                    0.00%  1.6200us         2     810ns     307ns  1.3130us  cuDevic
                    0.00%     323ns         1     323ns     323ns     323ns  cuDevic
```

Este caso es diferente al anterior, puesto que la memoria no está colocada de forma agrupada, sino en otras posiciones. Lo que lo hace más propenso a divergencias en los threads. Lo único positivo que ofrece esto es que no tenemos el problema de los conflictos en el banco de memoria de ninguna manera, que sí podría darse en el ejemplo anterior.

Caso 3

```
%%writefile labsolReduction3.cu
// includes, kernels
```



```
// **===------------------------------------------------------------------===**
```

```
//! @param g_idata  input data in global memory
//                  result is expected in index 0 of g_idata
//! @param n        input number of elements to scan from input data
// **===-----------------------------------------------------------------===**
__global__ void reduction(float *g_data, int n)
{
  int stride;
  // Define shared memory
  __shared__ float scratch[NUM_ELEMENTS];
  // Load the shared memory
  scratch[threadIdx.x ] = g_data[threadIdx.x];
  if(threadIdx.x + blockDim.x < n)
    scratch[threadIdx.x + blockDim.x] = g_data[threadIdx.x + blockDim.x];
  __syncthreads();
  // Do sum reduction from shared memory
  ///////  Reduction scheme 3 //////
  int stride2;
  for (stride = 1, stride2=1; stride <= 9; stride++, stride2<<=1)
  {
      int t = threadIdx.x << stride;
      if (t  + stride2 < n)
            scratch[ t ] = scratch[ t ]  + scratch[ t + stride2];
      __syncthreads();
  }
  // Store results back to global memory
  if(threadIdx.x == 0)
    g_data[0] = scratch[0];
  return;
}
////////////////////////////////////////////////////////////////////////////////
// Program main
////////////////////////////////////////////////////////////////////////////////
void runTest( int argc, char** argv);
float computeOnDevice(float* h_data, int array_mem_size);
extern "C" void computeGold( float* reference, float* idata, const unsigned int len);
int main( int argc, char** argv)
{
    runTest( argc, argv);
    return EXIT_SUCCESS;
}
////////////////////////////////////////////////////////////////////////////////
//! Run naive scan test
////////////////////////////////////////////////////////////////////////////////
void runTest( int argc, char** argv)
{
    int num_elements = NUM_ELEMENTS;
    const unsigned int array_mem_size = sizeof( float) * num_elements;
    // allocate host memory to store the input data
    float* h_data = (float*) malloc( array_mem_size);
    // * No arguments: Randomly generate input data and compare against the host's

                                          data on the host to be integer values

                                          ; i < num_elements; ++i)
                {
```

```
                //h_data[i] = floorf(1000*(rand()/(float)RAND_MAX));
                h_data[i] = i*1.0;
            }

        // compute reference solution
    float reference = 0.0f;
    computeGold(&reference , h_data, num_elements);

    float result;
    for (int i=0; i<100; i++)
      result = computeOnDevice(h_data, num_elements);
    // We can use an epsilon of 0 since values are integral and in a range
    // that can be exactly represented
    float epsilon = 0.0f;
    unsigned int result_regtest = (abs(result - reference) <= epsilon);
        printf( "Test %s\n", (1 == result_regtest) ? "PASSED" : "FAILED");
    printf( "device: %f  host: %f\n", result, reference);
    // cleanup memory
    free( h_data);
}
////////////////////////////////////////////////////////////////////////////
// Take h_data from host, copies it to device, setup grid and thread
// dimentions, excutes kernel function, and copy result of scan back
// to h_data.
// Note: float* h_data is both the input and the output of this function.
////////////////////////////////////////////////////////////////////////////
float computeOnDevice(float* h_data, int num_elements)
{
  float* d_data = NULL;
  float result;
  // Memory allocation on device side
  cudaMalloc((void**)&d_data, num_elements*sizeof(float));
  // Copy from host memory to device memory
  cudaMemcpy(d_data, h_data, num_elements*sizeof(float), cudaMemcpyHostToDevice);
  int threads = (num_elements/2) + num_elements%2;
  // Invoke the kernel
  reduction<<<1,threads>>>(d_data,num_elements);
  // Copy from device memory back to host memory
  cudaMemcpy(&result, d_data, sizeof(float), cudaMemcpyDeviceToHost);
  cudaFree(d_data);
  return result;
}

////////////////////////////////////////////////////////////////////////////
void computeGold( float* reference, float* idata, const unsigned int len)
{
  reference[0] = 0;
  double total_sum = 0;
  unsigned int i;
  for( i = 0; i < len; ++i)
  {
```

Se ha guardado correctamente

```
}
/////
```

```
Overwriting labsolReduction3.cu
```

```
!/usr/local/cuda/bin/nvcc -I /usr/local/cuda/samples/common/inc -arch=sm_35 -rdc=true labs
```

```
nvcc warning : The 'compute_35', 'compute_37', 'compute_50', 'sm_35', 'sm_37' and 's
```

```
!nvprof ./labsolReduction3
```

```
==487== NVPROF is profiling process 487, command: ./labsolReduction3
Test PASSED
device: 130816.000000  host: 130816.000000
==487== Profiling application: ./labsolReduction3
==487== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   65.67%  640.15us       100  6.4010us  6.0160us  7.1360us  reducti
                   17.89%  174.37us       100  1.7430us  1.6960us  1.9520us  [CUDA m
                   16.44%  160.22us       100  1.6020us  1.5680us  2.0160us  [CUDA m
      API calls:   96.95%  325.38ms       100  3.2538ms  82.899us  316.40ms  cudaMal
                    1.82%  6.0971ms       100  60.971us  54.335us  98.392us  cudaFre
                    0.81%  2.7237ms       200  13.618us  5.4100us  26.104us  cudaMem
                    0.24%  814.25us       100  8.1420us  6.5240us  25.483us  cudaLau
                    0.11%  371.47us         1  371.47us  371.47us  371.47us  cuDevic
                    0.06%  189.10us       101  1.8720us     134ns  94.960us  cuDevic
                    0.01%  29.337us         1  29.337us  29.337us  29.337us  cuDevic
                    0.00%  5.5490us         1  5.5490us  5.5490us  5.5490us  cuDevic
                    0.00%  1.6340us         3     544ns     187ns  1.0360us  cuDevic
                    0.00%  1.2710us         2     635ns     409ns     862ns  cuDevic
                    0.00%     261ns         1     261ns     261ns     261ns  cuDevic
```

Este caso es esencialmente idéntico al caso 1, solo que en vez de escogerse el elemento de la otra mitad que ocupe la misma posición, se escoge el opuesto. No presenta ninguna ventaja con respecto los anteriores métodos.

Generalización caso 1

Vamos a optar por realizar una generalizacion del caso 1, que es el que a lo largo de 100 ejecuciones ha dado los mejores resultados.

```
%%writefile labsolReduction4.cu
// includes, kernels
#include <stdio.h>
#include <assert.h>
#define THREADS_PER_BLOCK 512
#define N 3000000
```



```
__global__ void reduction(float *g_data, int n)
```

```
{

  __shared__ float scratch[THREADS_PER_BLOCK*2];

  int index = blockIdx.x * 2 * blockDim.x + threadIdx.x;

  scratch[threadIdx.x] = (index < n) ? g_data[index] : 0.0;
  scratch[threadIdx.x + blockDim.x] = (index + blockDim.x < n) ? g_data[index + blockDim.x

  // Do sum reduction from shared memory
  for(int stride=2*blockDim.x ; stride > 1; stride >>= 1) {
      __syncthreads();
      if (threadIdx.x < (stride>>1))
         scratch[threadIdx.x] += scratch[stride - threadIdx.x - 1];
  }
  __syncthreads();
  // Store results back to global memory
  if(threadIdx.x == 0)
    g_data[blockIdx.x] = scratch[0];
  return;
}
////////////////////////////////////////////////////////////////////////////////
// Program main
////////////////////////////////////////////////////////////////////////////////
void runTest( int argc, char** argv);
float computeOnDevice(float* h_data, int array_mem_size);
extern "C" void computeGold( float* reference, float* idata, const unsigned int len);
int main( int argc, char** argv)
{
    runTest( argc, argv);
    return EXIT_SUCCESS;
}
////////////////////////////////////////////////////////////////////////////////
//! Run naive scan test
////////////////////////////////////////////////////////////////////////////////
void runTest( int argc, char** argv)
{
    int num_elements = N;
    const unsigned int array_mem_size = sizeof( float) * num_elements;
    // allocate host memory to store the input data
    float* h_data = (float*) malloc( array_mem_size);
    // * No arguments: Randomly generate input data and compare against the host's

    for( unsigned int i = 0; i < num_elements; ++i)
      h_data[i] = i*1.0;

    float reference = 0.0f;
    computeGold(&reference , h_data, num_elements);
    float result = computeOnDevice(h_data, num_elements);
    float epsilon = 0.0f;
    unsigned int result_regtest = (abs(result - reference) <= epsilon);
                                  lt_regtest) ? "PASSED" : "FAILED");
                                  ", result, reference);
}
```

```
////////////////////////////////////////////////////////////////////////
// Take h_data from host, copies it to device, setup grid and thread
// dimentions, excutes kernel function, and copy result of scan back
// to h_data.
// Note: float* h_data is both the input and the output of this function.
////////////////////////////////////////////////////////////////////////
float computeOnDevice(float* h_data, int num_elements) {

  float* d_data = NULL;
  // Memory allocation on device side
  cudaMalloc((void**)&d_data, num_elements*sizeof(float));
  while(true){

    int blocks = (num_elements + 2*THREADS_PER_BLOCK - 1)/(2*THREADS_PER_BLOCK);
  // Copy from host memory to device memory
    cudaMemcpy(d_data, h_data, num_elements*sizeof(float), cudaMemcpyHostToDevice);
  // Invoke the kernel
    reduction<<<blocks,THREADS_PER_BLOCK>>>(d_data,num_elements);

    cudaMemcpy(h_data, d_data, blocks*sizeof(float), cudaMemcpyDeviceToHost);
    if (blocks==1)
      break;

    num_elements = blocks;
  }
  cudaFree(d_data);
  return h_data[0];
}

////////////////////////////////////////////////////////////////////////
void computeGold( float* reference, float* idata, const unsigned int len)
{
  reference[0] = 0;
  double total_sum = 0;
  unsigned int i;
  for( i = 0; i < len; ++i)
  {
      total_sum += idata[i];
  }
  reference[0] = total_sum;
}
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
```

```
Overwriting labsolReduction4.cu
```

```
!/usr/local/cuda/bin/nvcc -I /usr/local/cuda/samples/common/inc -arch=sm_75 -rdc=true labs
```

```
!nvprof ./labsolReduction4
```

```
==1024== NVPROF is profiling process 1024, command: ./labsolReduction4
device: 4499998507008.000000  host: 4499998507008.000000
```

Se ha guardado correctamente

```
==1024== Profiling application: ./labsolReduction4
==1024== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   91.40%  2.3789ms         3  792.97us  1.3440us  2.3748ms  [CUDA m
                    8.34%  217.05us         3  72.350us  7.3280us  202.40us  reducti
                    0.26%  6.7200us         3  2.2400us  1.6320us  3.0080us  [CUDA m
      API calls:   98.28%  216.08ms         1  216.08ms  216.08ms  216.08ms  cudaMal
                    1.32%  2.8980ms         6  483.00us  4.0710us  2.6054ms  cudaMem
                    0.18%  393.01us         1  393.01us  393.01us  393.01us  cuDevic
                    0.10%  228.82us         1  228.82us  228.82us  228.82us  cudaFre
                    0.07%  160.91us       101  1.5930us     139ns  68.346us  cuDevic
                    0.02%  49.570us         3  16.523us  5.7850us  36.937us  cudaLau
                    0.02%  40.154us         1  40.154us  40.154us  40.154us  cuDevic
                    0.00%  5.5450us         1  5.5450us  5.5450us  5.5450us  cuDevic
                    0.00%  1.5170us         3     505ns     203ns  1.0380us  cuDevic
                    0.00%  1.3130us         2     656ns     222ns  1.0910us  cuDevic
                    0.00%     299ns         1     299ns     299ns     299ns  cuDevic
```