UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament d'Arquitectura de Computadors

Tarjetas Gráficas y Aceleradores

CUDA – Reducciones

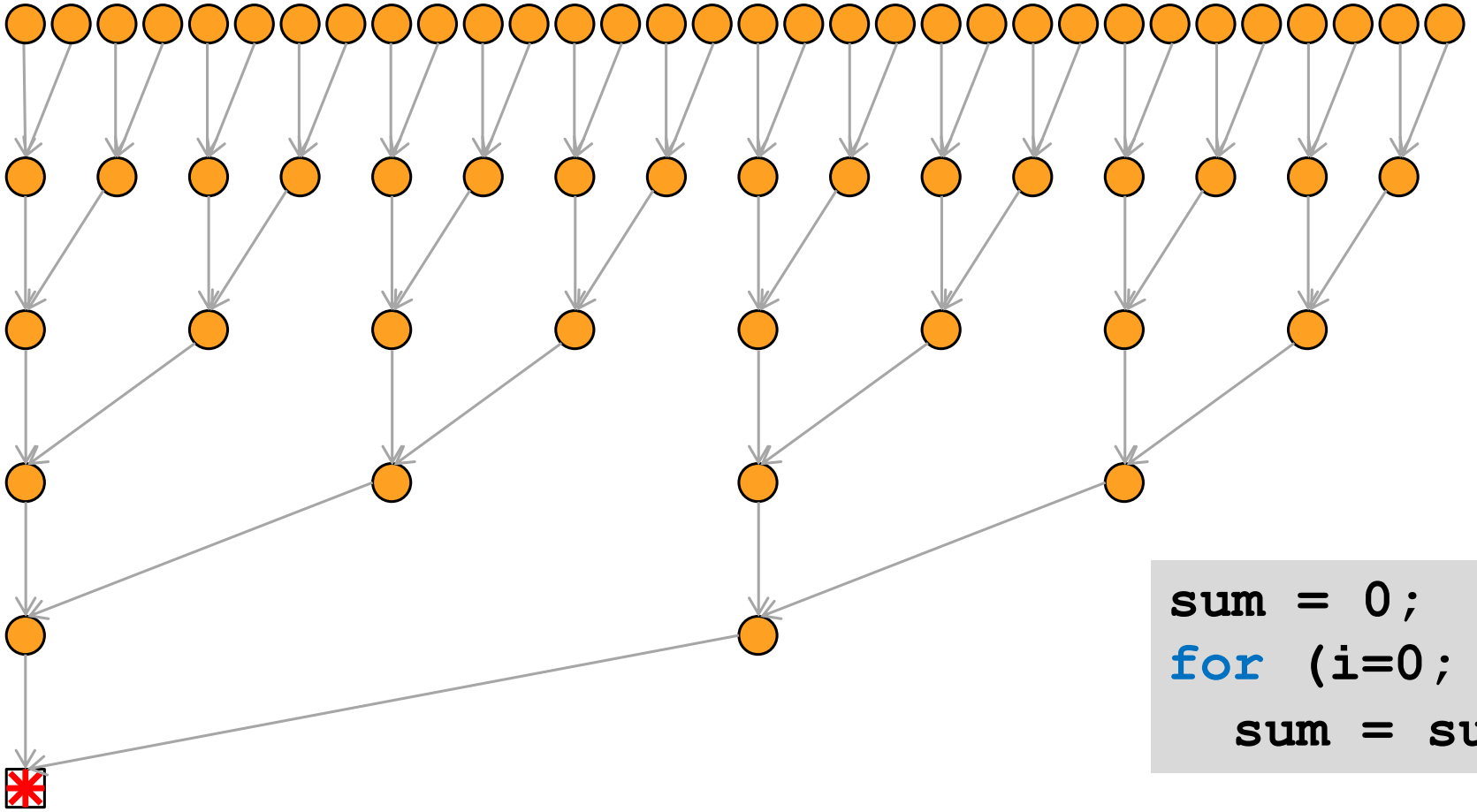
Agustín Fernández

Departament d'Arquitectura de Computadors

Facultat d'Informàtica de Barcelona

Universitat Politècnica de Catalunya

Kernel 01



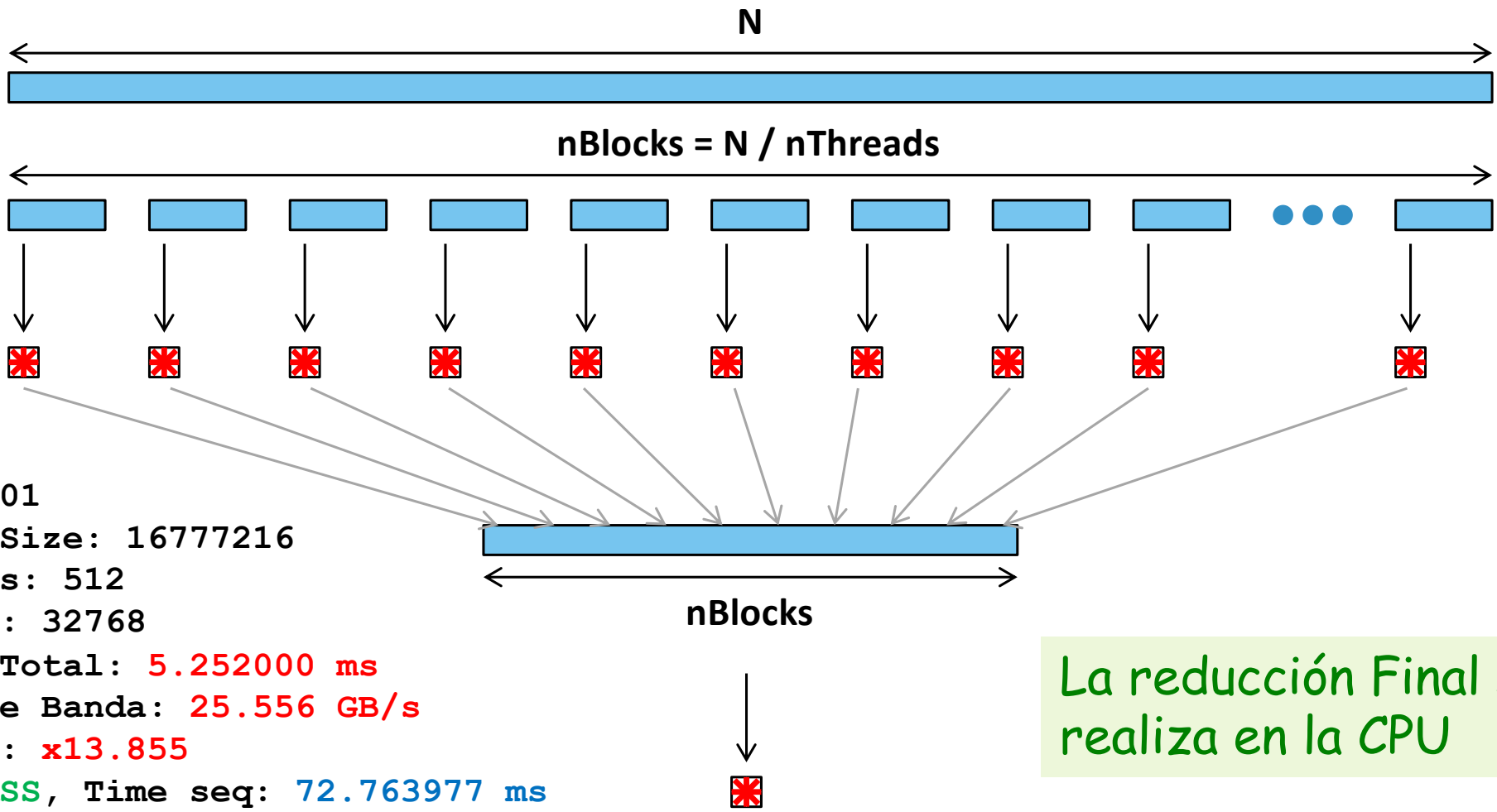
```
sum = 0;
for (i=0; i<N; i++)
  sum = sum + V[i];
```

Kernel 01

```
__global__ void Kernel01(double *g_idata, double *g_odata) {
  __shared__ double sdata[512];

  // Cada thread carga 1 elemento desde la memoria global
  unsigned int i = blockIdx.x*blockDim.x + threadIdx.x;
  sdata[threadIdx.x] = g_idata[i];
  __syncthreads();

  // Hacemos la reduccion en la memoria compartida
  for(unsigned int s=1; s < blockDim.x; s *= 2) {
    if (threadIdx.x % (2*s) == 0)
      sdata[threadIdx.x] += sdata[threadIdx.x + s];
    __syncthreads();
  }

  // El thread 0 escribe el resultado de este bloque en la memoria global
  if (threadIdx.x == 0) g_odata[blockIdx.x] = sdata[0];
}
```

nThreads = 512 = 2^9

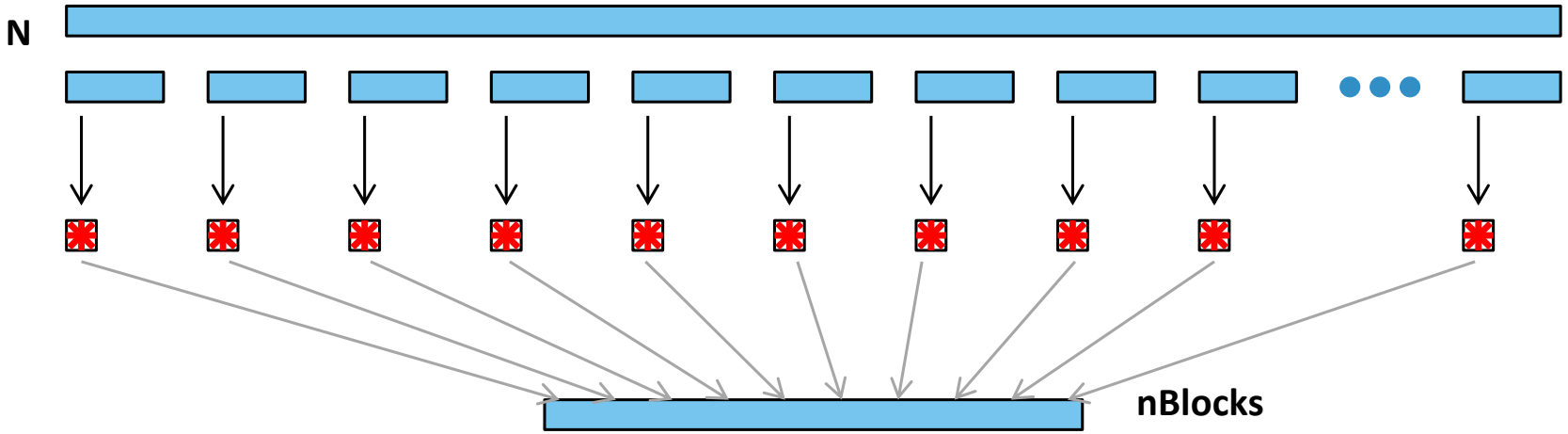
Kernel 01




```
KERNEL 01
Vector Size: 16777216
nThreads: 512
nBlocks: 32768
Tiempo Total: 5.252000 ms
Ancho de Banda: 25.556 GB/s
Speedup: x13.855
TEST PASS, Time seq: 72.763977 ms
```

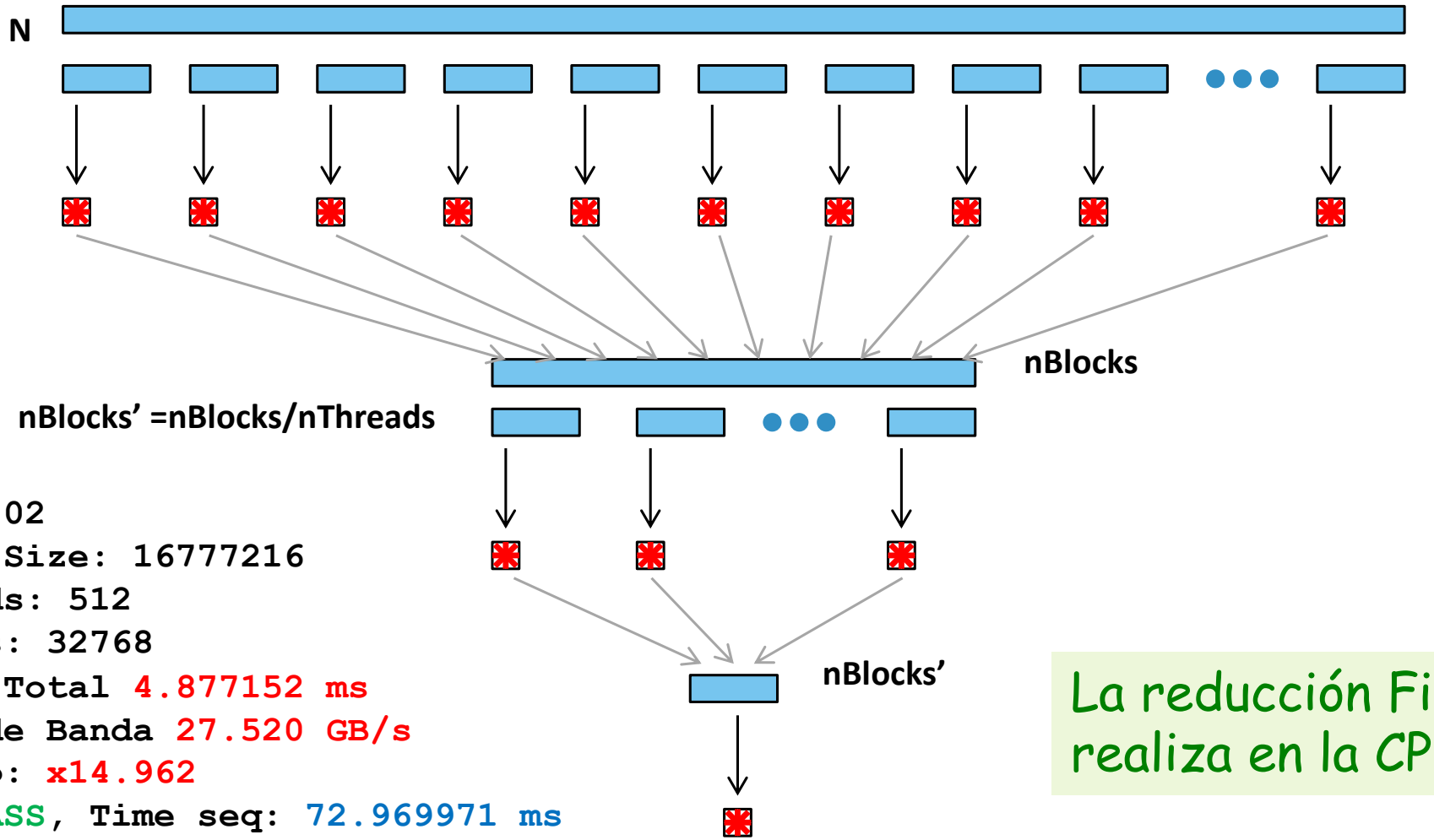
La reducción Final se realiza en la CPU

Kernel 02



¿Porqué no usar la GPU para hacer parte de la reducción final?

Kernel 02




```
KERNEL 02
Vector Size: 16777216
nThreads: 512
nBlocks: 32768
Tiempo Total 4.877152 ms
Ancho de Banda 27.520 GB/s
Speedup: x14.962
TEST PASS, Time seq: 72.969971 ms
```

La reducción Final se realiza en la CPU

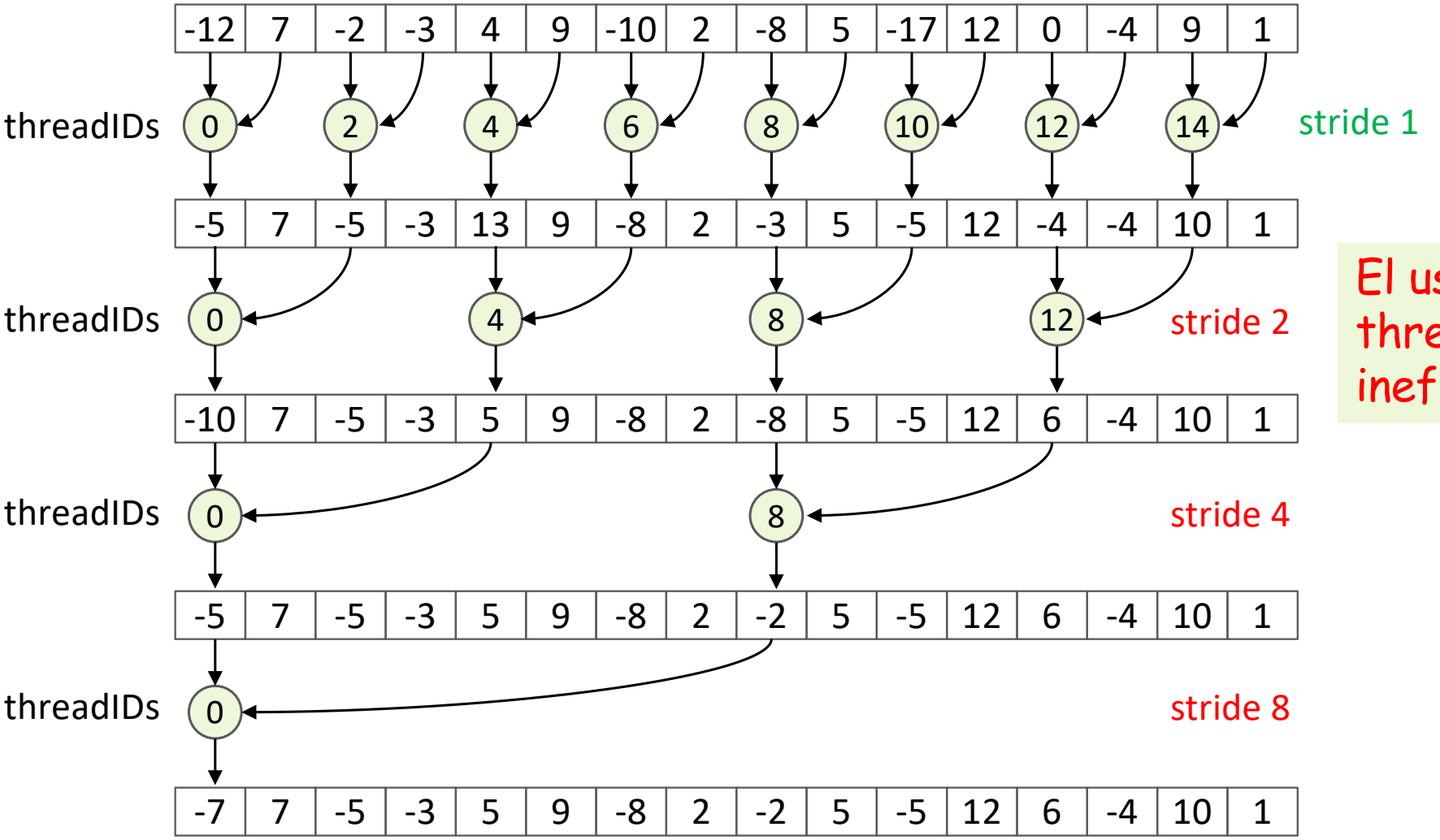
Kernel 01-02

```
__global__ void Kernel01(double *g_idata, double *g_odata) {
  __shared__ double sdata[512];

  // Cada thread carga 1 elemento desde la memoria global
  unsigned int i = blockIdx.x*blockDim.x + threadIdx.x;
  sdata[threadIdx.x] = g_idata[i];
  __syncthreads();

  // Hacemos la reduccion en la memoria compartida
  for(unsigned int s=1; s < blockDim.x; s *= 2) {
    if (threadIdx.x % (2*s) == 0)
      sdata[threadIdx.x] += sdata[threadIdx.x + s];
    __syncthreads();
  }

  // El thread 0 escribe el resultado de este bloque en la memoria global
  if (threadIdx.x == 0) g_odata[blockIdx.x] = sdata[0];
}
```

El uso de los threads es muy ineficiente

Esta operación es MUY COSTOSA

Kernel 01-02



El uso de los threads es muy ineficiente.

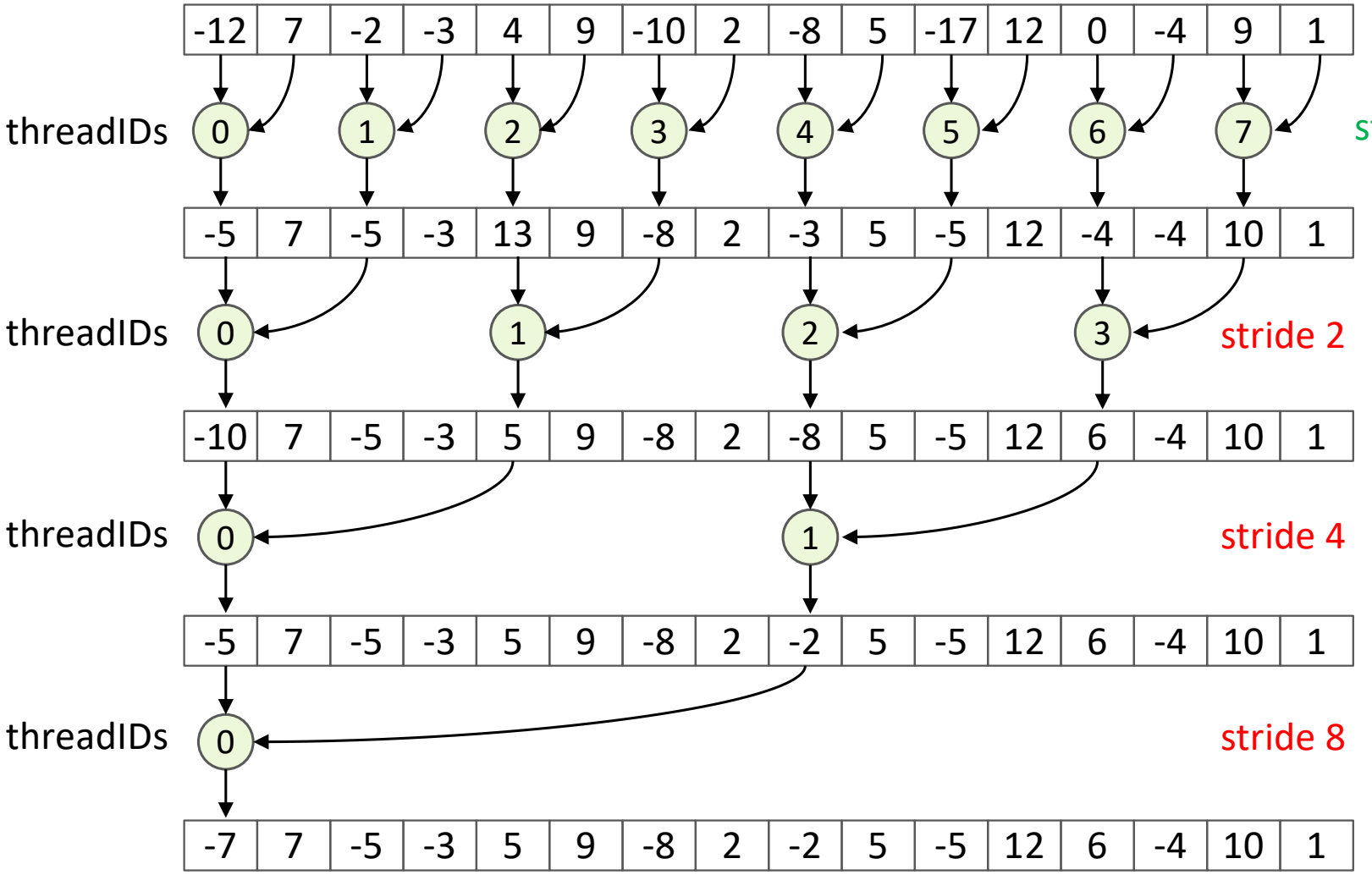
Kernel 03

```
__global__ void Kernel01(double *g_idata, double *g_odata) {
  __shared__ double sdata[512];

  // Cada thread carga 1 elemento desde la memoria global
  unsigned int i = blockIdx.x*blockDim.x + threadIdx.x;
  sdata[threadIdx.x] = g_idata[i];
  __syncthreads();

  // Hacemos la reduccion en la memoria compartida
  for(unsigned int s=1; s < blockDim.x; s *= 2) {
    int index = 2 * s * threadIdx.x;
    if (index < blockDim.x)
      sdata[index] += sdata[index + s];
    __syncthreads();
  }

  // El thread 0 escribe el resultado de es
  if (threadIdx.x == 0)
    g_odata[blockIdx.x] = sdata[0];
}
```

Este código es mucho más rápido.

```
KERNEL 03
Vector Size: 16777216
nThreads: 512
nBlocks: 32768
Tiempo Total 4.220544 ms
Ancho de Banda 31.801 GB/s
Speedup: x17.228
TEST PASS, Time seq: 72.710999 ms
```

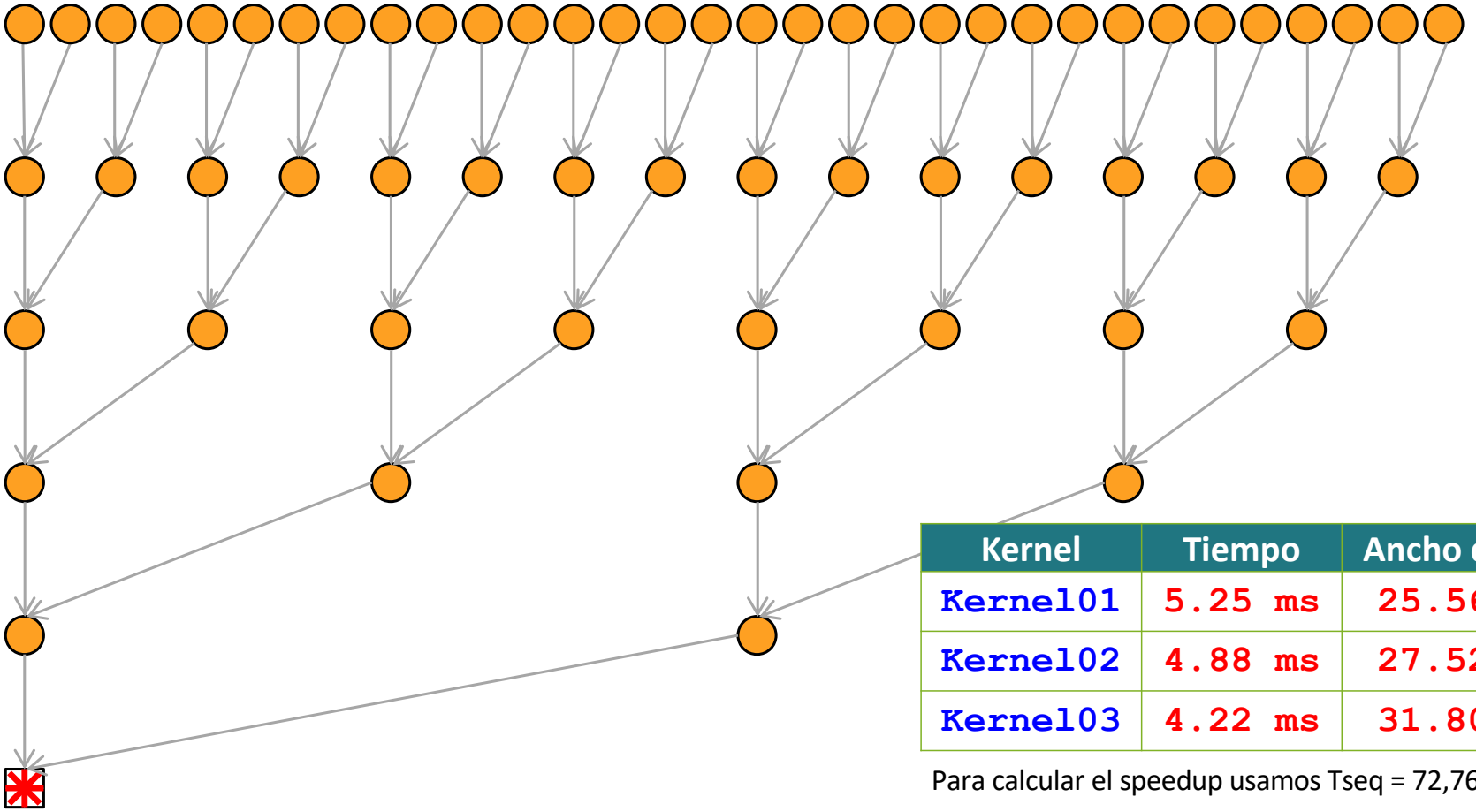
Kernel 03



El uso de los threads es ineficiente.

Conflicto en el acceso a los bancos de Memoria.

Kernel 01, 02 y 03



Kernel	Tiempo	Ancho de Banda	Speedup
Kernel01	5.25 ms	25.56 GB/s	X13.86
Kernel02	4.88 ms	27.52 GB/s	X14.91
Kernel03	4.22 ms	31.80 GB/s	X17.24

Para calcular el speedup usamos Tseq = 72,764ms

Nvprof: Kernel 01-03

	Kernel 01	Kernel 02	Kernel 03
Invocaciones	1	2	2
Grid Size	(32768, 1, 1)	(32768, 1, 1) (64 1 1)*	(32768, 1, 1) (64 1 1)*
Block Size	(512, 1, 1)	(512, 1, 1)	(512, 1, 1)
Reg	13	13	11
Shared Memory	4 KB	4 KB	4 KB
sm_efficiency	99.83%	99,85%	99,83%
Achieved_occupancy	0,971	0,971	0,967
gld_request_throughput	25,59 GB/s	25,83 GB/s	29,99 GB/s
gst_request_throughput	51,19 MB/s	51,67 MB/s	59,99 MB/s
Dram_utilization	Low (2)	Low (2)	Low (1)
RENDIMIENTO (GB/s)	25.56	27.52	31.80

(*) Tiempo despreciable del orden de µs

¿Ancho de Banda?

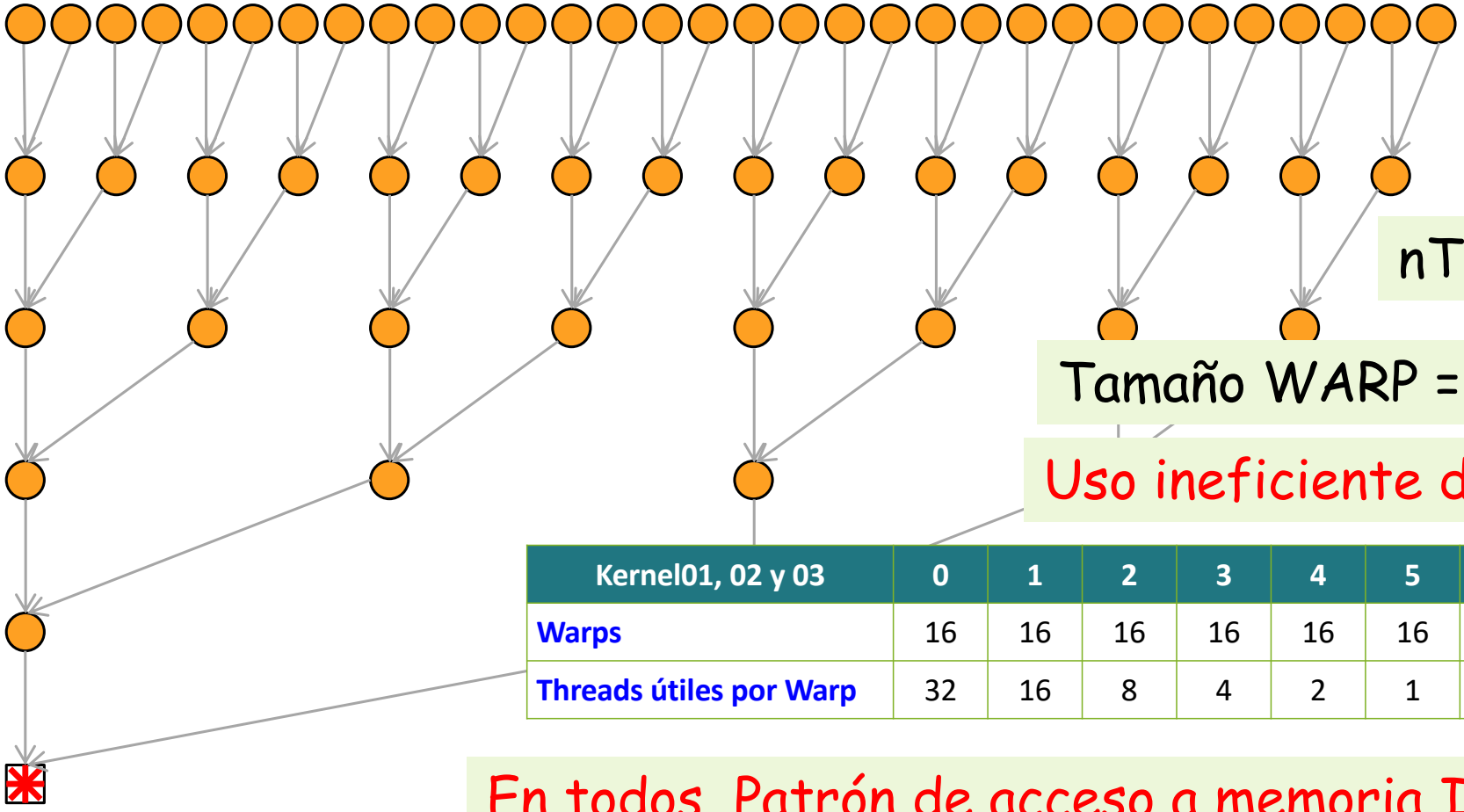
```
. . .
  cudaEventRecord(start, 0);

  // Ejecutar el kernel
  Kernel02<<<nBlocks, nThreads>>>(d_v, d_w);
  Kernel02<<<nBlocks/nThreads, nThreads>>>(d_w, d_x);
  // Obtener el resultado parcial desde el host
  cudaMemcpy(h_x, d_x, numBytesX, cudaMemcpyDeviceToHost);
  SUM = 0.0;
  for (i=0; i<(nBlocks/nThreads); i++)
    SUM = SUM + h_x[i];

  cudaEventRecord(stop, 0);
  cudaEventSynchronize(stop);
. . .
  cudaEventElapsedTime(&elapsedTime, start, stop);
  printf("Ancho de Banda %f GB/s\n", (N * sizeof(double))/(1000000 * elapsedTime));
```

Código a Evaluar

Ancho de Banda de la K40: **288 GB/s**

¡Tenemos mucho margen de MEJORA!

Kernel 01, 02 y 03



nThreads = 512

Tamaño WARP = 32 threads

Uso ineficiente de los WARPs

Kernel01, 02 y 03	0	1	2	3	4	5	6	7	8	9
Warps	16	16	16	16	16	16	8	4	2	1
Threads útiles por Warp	32	16	8	4	2	1	1	1	1	1

En todos, Patrón de acceso a memoria INEFICIENTE

Kernel04

Mejor USO de los warps

	0	1	2	3	4	5	6	7	8	9
Warps	16	8	4	2	1	1	1	1	1	1
Threads útiles por Warp	32	32	32	32	32	16	8	4	2	1

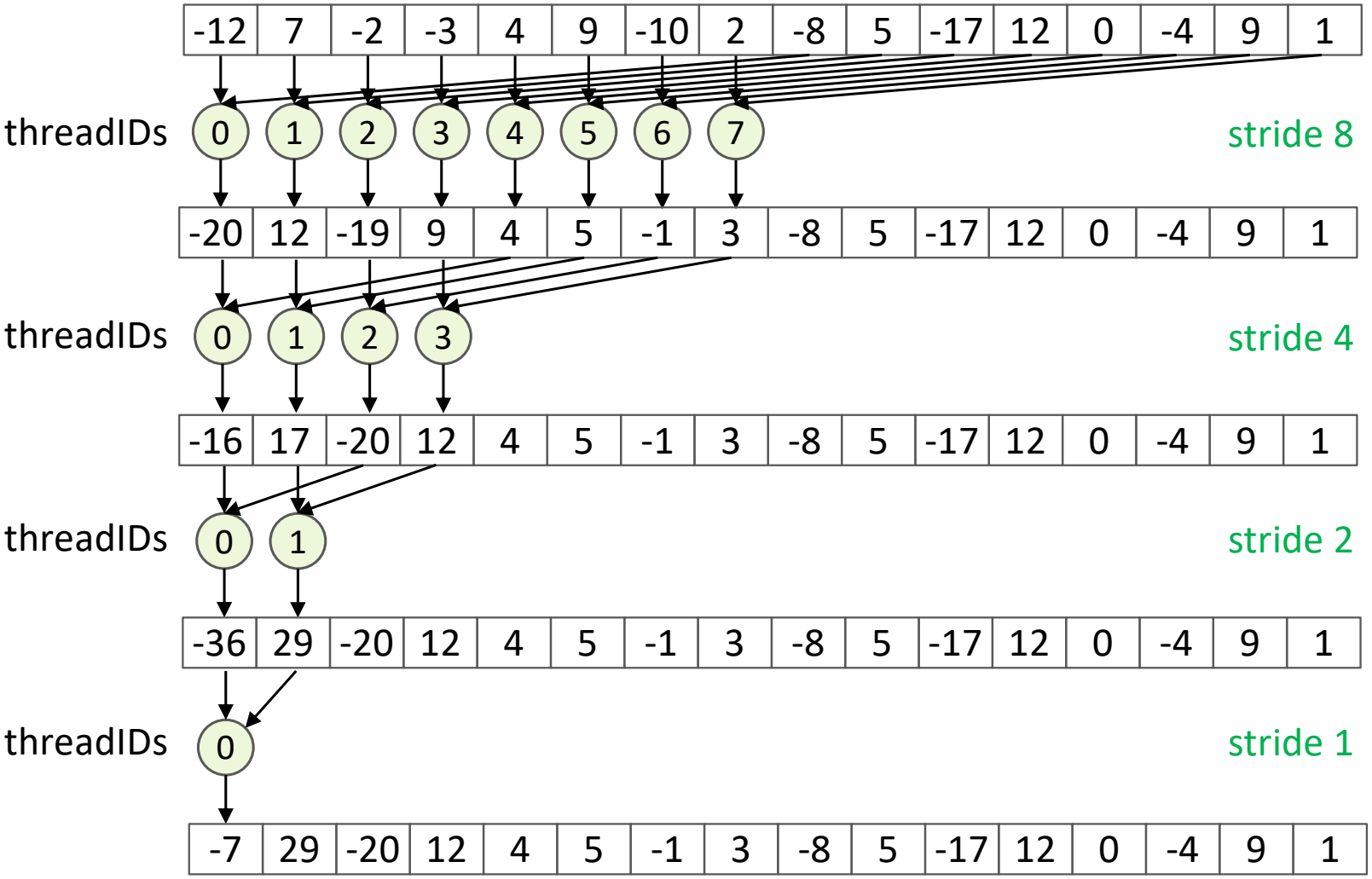
Acceso a POSICIONES CONSECUTIVAS de memoria

Kernel04

```
__global__ void Kernel04(double *g_idata, double *g_odata) {
  __shared__ double sdata[512];

  // Cada thread carga 1 elemento desde la memoria global
  unsigned int i = blockIdx.x*blockDim.x + threadIdx.x;
  sdata[threadIdx.x] = g_idata[i];
  __syncthreads();

  // Hacemos la reduccion en la memoria compartida
  for (unsigned int s=blockDim.x/2; s>0; s>>=1) {
    if (threadIdx.x < s)
      sdata[threadIdx.x] += sdata[threadIdx.x + s];
    __syncthreads();
  }

  // El thread 0 escribe el resultado de este bloque en la memoria global
  if (threadIdx.x == 0) g_odata[blockIdx.x] = sdata[0];
}
```

Buen uso de los WARPs
Buen patrón de acceso a Memoria

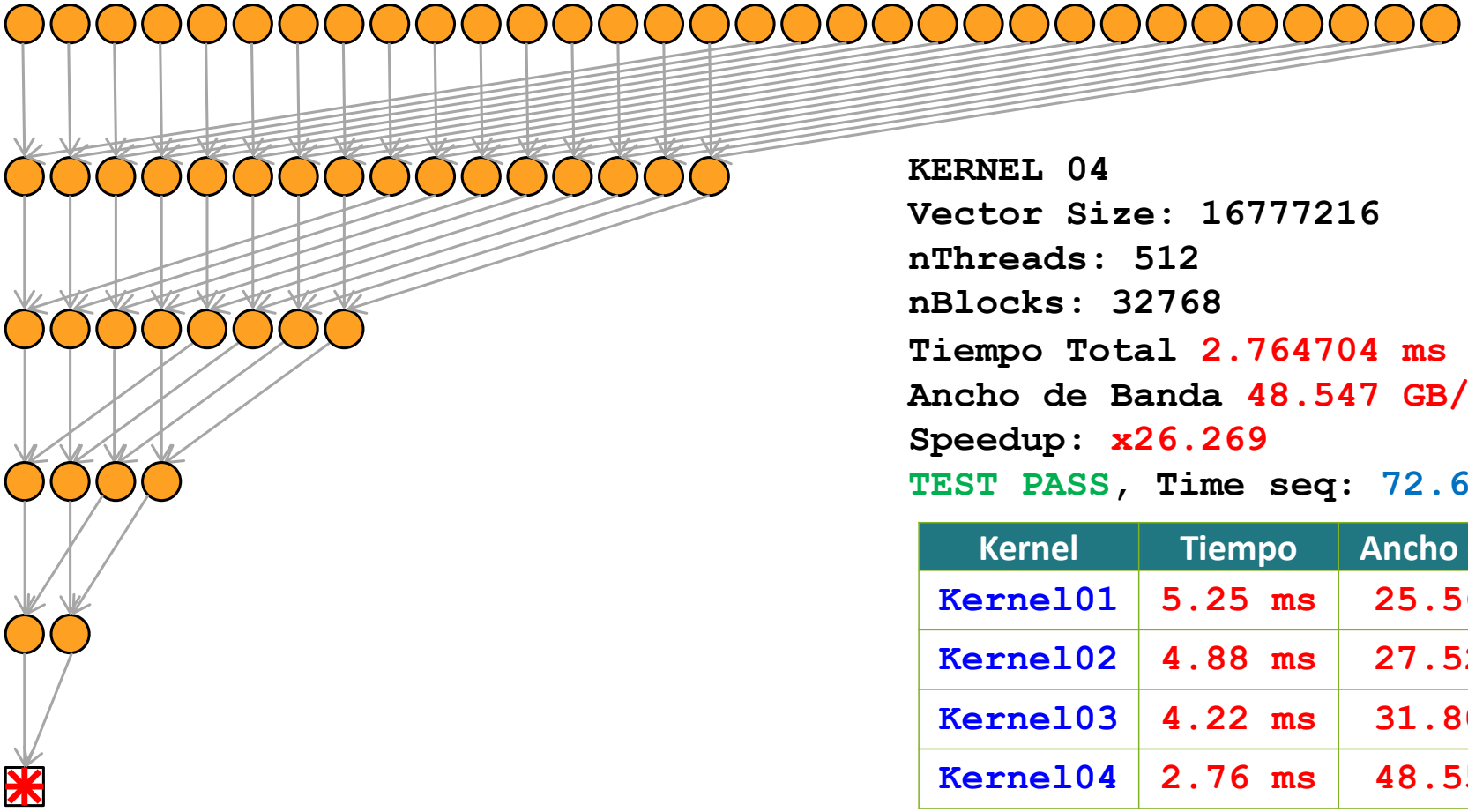
Kernel 04



Buen uso de los warps

Acceso Secuencial sin conflictos

Kernel04



```
KERNEL 04
Vector Size: 16777216
nThreads: 512
nBlocks: 32768
Tiempo Total 2.764704 ms
Ancho de Banda 48.547 GB/s
Speedup: x26.269
TEST PASS, Time seq: 72.625977 ms
```

Kernel	Tiempo	Ancho de Banda	Speedup
Kernel01	5.25 ms	25.56 GB/s	X13.86
Kernel02	4.88 ms	27.52 GB/s	X14.91
Kernel03	4.22 ms	31.80 GB/s	X17.24
Kernel04	2.76 ms	48.55 GB/s	X26.36

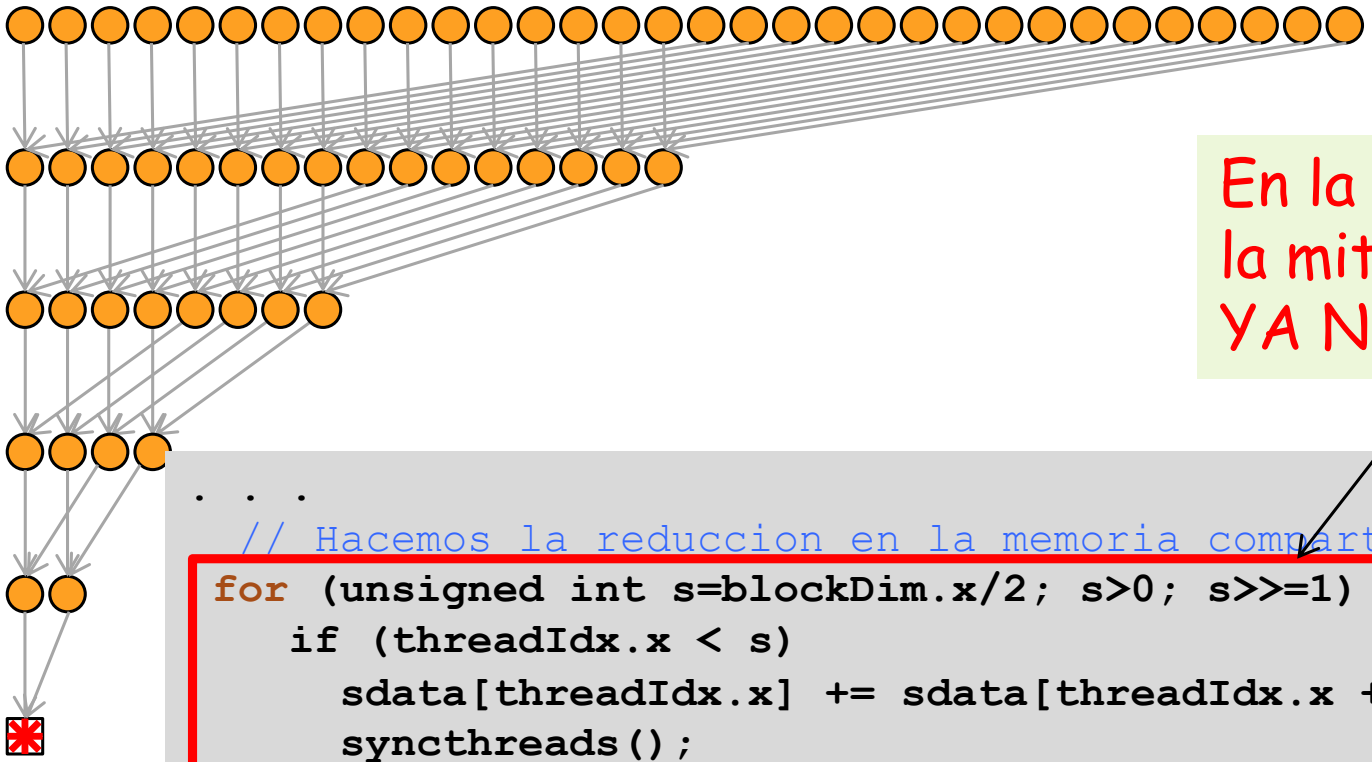
Para calcular el speedup usamos Tseq = 72,764ms

Nvprof: Kernel 01-04

	Kernel 01	Kernel 02	Kernel 03	Kernel 04
Invocaciones	1	2	2	2
Grid Size	(32768, 1, 1)	(32768, 1, 1) (64 1 1)*	(32768, 1, 1) (64 1 1)*	(32768, 1, 1) (64 1 1)*
Block Size	(512, 1, 1)	(512, 1, 1)	(512, 1, 1)	(512, 1, 1)
Reg	13	13	11	10
Shared Memory	4 KB	4 KB	4 KB	4 KB
sm_efficiency	99.83%	99,85%	99,83%	99,78%
Achieved_occupancy	0,971	0,971	0,967	0,952
gld_request_throughput	25,59 GB/s	25,83 GB/s	29,99 GB/s	45,14 GB/s
gst_request_throughput	51,19 MB/s	51,67 MB/s	59,99 MB/s	90,28 MB/s
Dram_utilization	Low (2)	Low (2)	Low (1)	Low (3)
RENDIMIENTO (GB/s)	25.56	27.52	31.80	48.55

(*) Tiempo despreciable del orden de µs

Kernel04



En la primera iteración, la mitad de los threads, YA NO HACEN NADA.

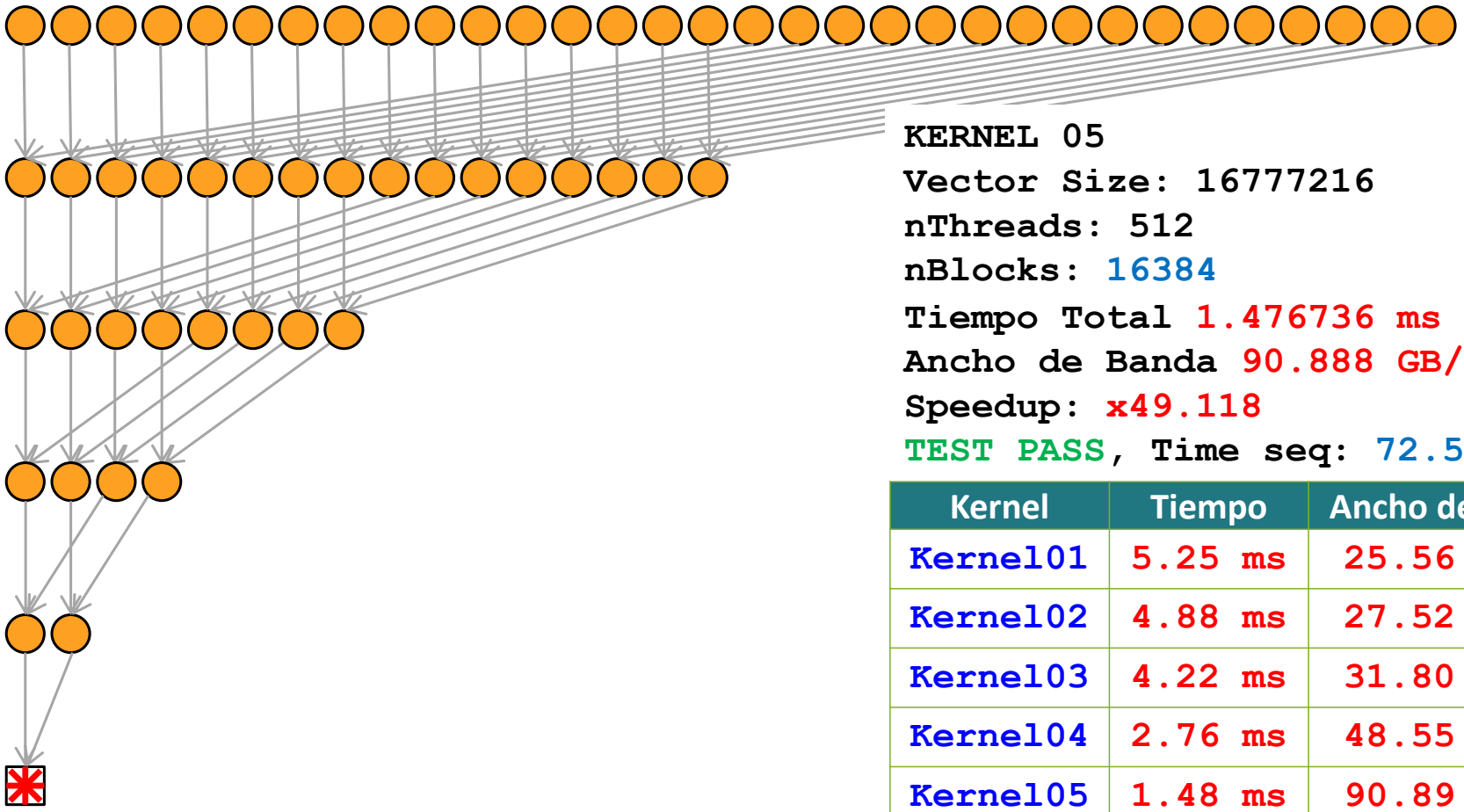
```
. . .
  // Hacemos la reduccion en la memoria compartida
  for (unsigned int s=blockDim.x/2; s>0; s>>=1) {
     if (threadIdx.x < s)
       sdata[threadIdx.x] += sdata[threadIdx.x + s];
     __syncthreads();
   }
. . .
```

Kernel05

```
__global__ void Kernel05(double *g_idata, double *g_odata) {
  __shared__ double sdata[512];

  // Cada thread carga 2 elementos desde la memoria global,
  // los suma y los deja en la memoria compartida
  unsigned int i = blockIdx.x*(blockDim.x*2) + threadIdx.x;
  sdata[threadIdx.x] = g_idata[i] + g_idata[i+blockDim.x];
  __syncthreads();

  // Hacemos la reduccion en la memoria compartida
  for (unsigned int s=blockDim.x/2; s>0; s>>=1) {
    if (threadIdx.x < s)
      sdata[threadIdx.x] += sdata[threadIdx.x + s];
    __syncthreads(
  }
  // El thread 0 e
  if (threadIdx.x =
}
```

Cada Block procesa 2·nThreads elementos.
Todos los threads hacen por lo menos 1 suma parcial.
Se ejecutan la mitad de thread Blocks.

Kernel05



```
KERNEL 05
Vector Size: 16777216
nThreads: 512
nBlocks: 16384
Tiempo Total 1.476736 ms
Ancho de Banda 90.888 GB/s
Speedup: x49.118
TEST PASS, Time seq: 72.533997 ms
```

Kernel	Tiempo	Ancho de Banda	Speedup
Kernel01	5.25 ms	25.56 GB/s	X13.86
Kernel02	4.88 ms	27.52 GB/s	X14.91
Kernel03	4.22 ms	31.80 GB/s	X17.24
Kernel04	2.76 ms	48.55 GB/s	X26.36
Kernel05	1.48 ms	90.89 GB/s	x49.16

Para calcular el speedup usamos Tseq = 72,764ms

Nvprof: Kernel 01-05

	Kernel 01	Kernel 02	Kernel 03	Kernel 04	Kernel 05
Invocaciones	1	2	2	2	2
Grid Size	(32768, 1, 1)	(32768, 1, 1) (64 1 1)*	(32768, 1, 1) (64 1 1)*	(32768, 1, 1) (64 1 1)*	(16384, 1, 1) (16 1 1)*
Block Size	(512, 1, 1)	(512, 1, 1)	(512, 1, 1)	(512, 1, 1)	(512, 1, 1)
Reg	13	13	11	10	13
Shared Memory	4 KB	4 KB	4 KB	4 KB	4 KB
sm_efficiency	99.83%	99,85%	99,83%	99,78%	99,55%
Achieved_occupancy	0,971	0,971	0,967	0,952	0,949
gld_request_throughput	25,59 GB/s	25,83 GB/s	29,99 GB/s	45,14 GB/s	88,77 GB/s
gst_request_throughput	51,19 MB/s	51,67 MB/s	59,99 MB/s	90,28 MB/s	88,77 MB/s
Dram_utilization	Low (2)	Low (2)	Low (1)	Low (3)	Mid (5)
RENDIMIENTO (GB/s)	25.56	27.52	31.80	48.55	90.89

(*) Tiempo despreciable del orden de µs

Kernel05

```
__global__ void Kernel05(double *g_idata, double *g_odata) {
  __shared__ double sdata[512];

  // Cada thread carga 2 elementos d
  // los suma y los deja en la memo
  unsigned int i = blockIdx.x*(bloc
  sdata[threadIdx.x] = g_idata[i] + g_idata[i+blockDim.x];
  __syncthreads();

  // Hacemos la reduccion en la memoria compartida
  for (unsigned int s=blockDim.x/2; s>0; s>>=1) {
    if (threadIdx.x < s)
      sdata[threadIdx.x] += sdata[threadIdx.x + s];
    __syncthreads();
  }

  // El thread 0 escribe el resultado de este bloque en la memoria global
  if (threadIdx.x == 0) g_odata[blockIdx.x] = sdata[0];
}
```

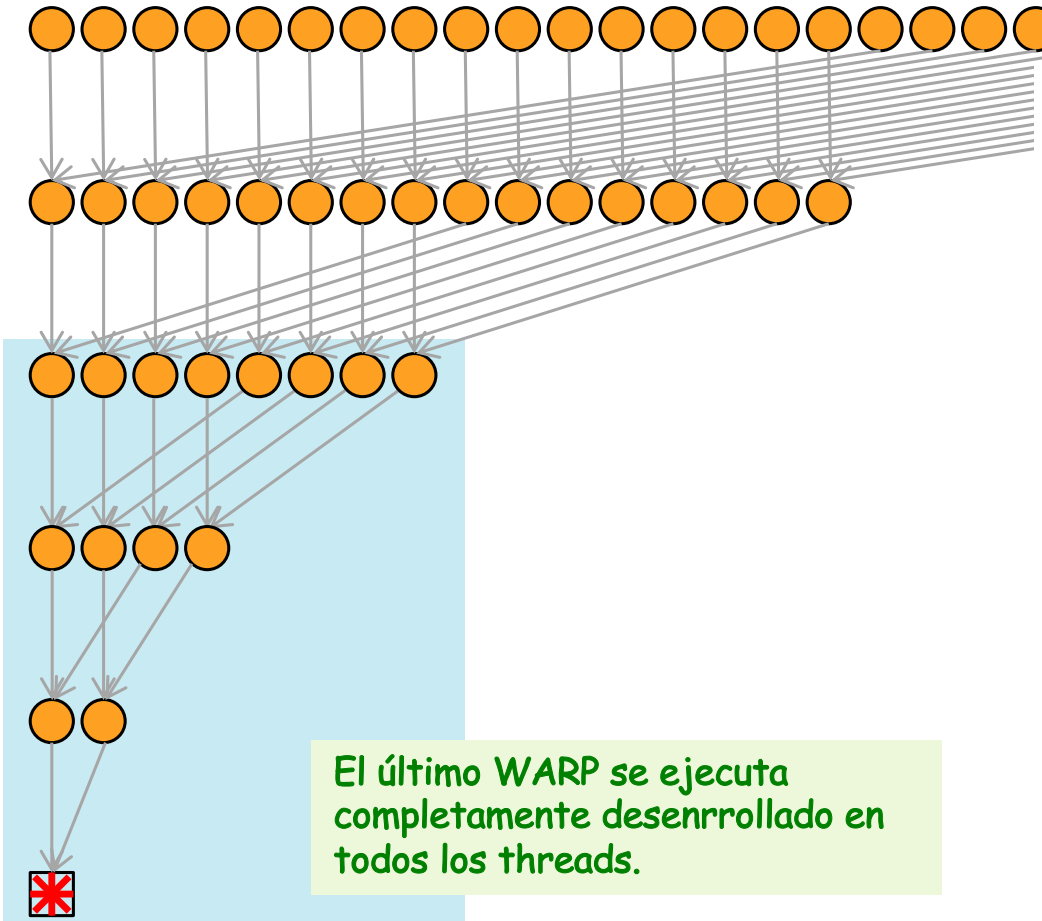
Imprescindible para sincronizar los threads de WARPS diferentes.
Inútil en el último WARP.

Kernel06

```
__global__ void Kernel06(double *g_idata, double *g_odata) {
  . . .
  // Hacemos la reduccion en la memoria compartida
  for (unsigned int s=blockDim.x/2; s>32; s>>=1) {
    if (threadIdx.x < s)
      sdata[threadIdx.x] += sdata[threadIdx.x + s];
    __syncthreads();
  }
// desenrrollamos el ultimo warp activo
  if (threadIdx.x < 32) {
    volatile double *smem = sdata;
    smem[threadIdx.x] += smem[threadIdx.x + 32];
    smem[threadIdx.x] += smem[threadIdx.x + 16];
    smem[threadIdx.x] += smem[threadIdx.x + 8];
    smem[threadIdx.x] += smem[threadIdx.x + 4];
    smem[threadIdx.x] += smem[threadIdx.x + 2];
    smem[threadIdx.x] += smem[threadIdx.x + 1];
  }
  // El thread 0 escribe el resultado de este bloque en la memoria global
  if (threadIdx.x == 0) g_odata[blockIdx.x] = sdata[0];
}
```

SÓLO los threads del último WARP ejecutan este código. Sólo nos interesa el resultado que queda en el thread 0.

Kernel06



El último WARP se ejecuta completamente desenrrollado en todos los threads.

```
KERNEL 06
Vector Size: 16777216
nThreads: 512
nBlocks: 16384
Tiempo Total 1.028416 ms
Ancho de Banda 130.509 GB/s
Speedup: x70.699
TEST PASS, Time seq: 72.708008 ms
```

Kernel	Tiempo	Ancho de Banda	Speedup
Kernel01	5.25 ms	25.56 GB/s	X13.86
Kernel02	4.88 ms	27.52 GB/s	X14.91
Kernel03	4.22 ms	31.80 GB/s	X17.24
Kernel04	2.76 ms	48.55 GB/s	X26.36
Kernel05	1.48 ms	90.89 GB/s	x49.16
Kernel06	1.03 ms	130.5 GB/s	X70.64

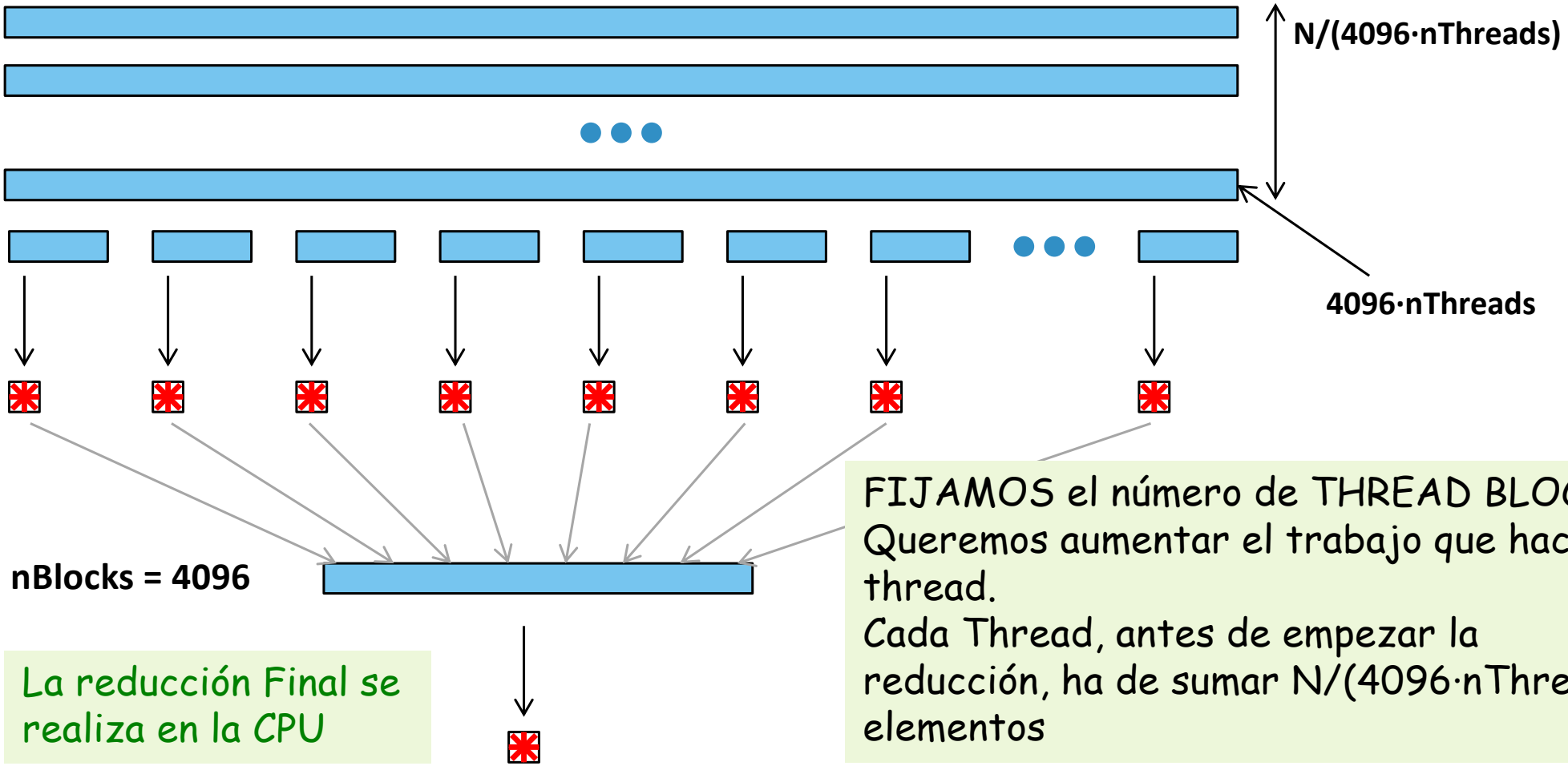
Para calcular el speedup usamos Tseq = 72,764ms

Nvprof: Kernel 01-06

	Kernel 01	Kernel 02	Kernel 03	Kernel 04	Kernel 05	Kernel 06
Invocaciones	1	2	2	2	2	2
Grid Size	(32768, 1, 1)	(32768, 1, 1) (64 1 1)*	(32768, 1, 1) (64 1 1)*	(32768, 1, 1) (64 1 1)*	(16384, 1, 1) (16 1 1)*	(32768, 1, 1) (64 1 1)*
Block Size	(512, 1, 1)	(512, 1, 1)	(512, 1, 1)	(512, 1, 1)	(512, 1, 1)	(512, 1, 1)
Reg	13	13	11	10	13	14
Shared Memory	4 KB	4 KB	4 KB	4 KB	4 KB	4 KB
sm_efficiency	99.83%	99,85%	99,83%	99,78%	99,55%	99,35%
Achieved_occupancy	0,971	0,971	0,967	0,952	0,949	0,754
gld_request_throughput	25,59 GB/s	25,83 GB/s	29,99 GB/s	45,14 GB/s	88,77 GB/s	127,9 GB/s
gst_request_throughput	51,19 MB/s	51,67 MB/s	59,99 MB/s	90,28 MB/s	88,77 MB/s	127,9 MB/s
Dram_utilization	Low (2)	Low (2)	Low (1)	Low (3)	Mid (5)	Mid (6)
RENDIMIENTO (GB/s)	25.56	27.52	31.80	48.55	90.89	130.5

(*) Tiempo despreciable del orden de µs

Kernel07



La reducción Final se realiza en la CPU

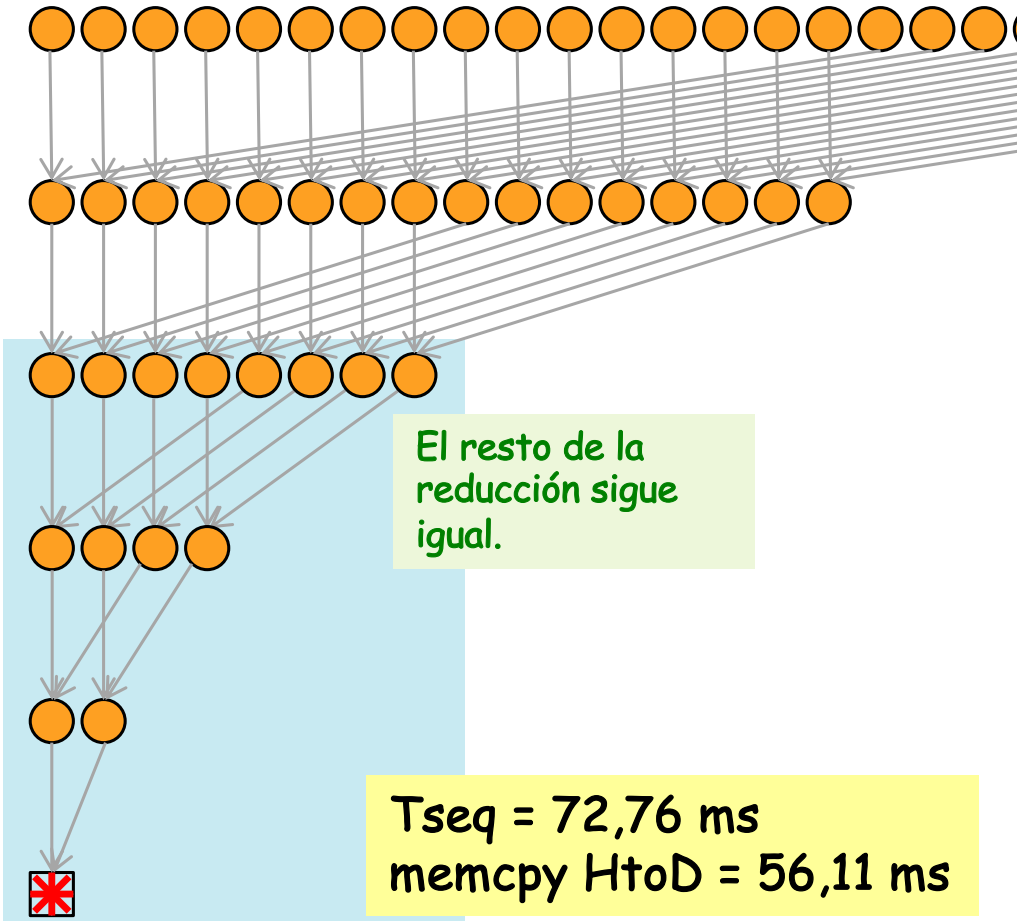
FIJAMOS el número de THREAD BLOCKS. Queremos aumentar el trabajo que hace 1 thread.
Cada Thread, antes de empezar la reducción, ha de sumar N/(4096·nThreads) elementos

Kernel 07

```
__global__ void Kernel07(double *g_idata, double *g_odata) {
  __shared__ double sdata[512];

  // Cada thread realiza la suma parcial de los datos que le
  // corresponden y la deja en la memoria compartida
  unsigned int i = blockIdx.x*(blockDim.x*2) + threadIdx.x;
  unsigned int gridSize = blockDim.x*2*gridDim.x;
  sdata[threadIdx.x] = 0;
  while (i < N) {
    sdata[threadIdx.x] += g_idata[i] + g_idata[i+blockDim.x];
    i += gridSize;
  }
  __syncthreads();

. . .
}
```

Todos los threads empiezan haciendo N/(4096·nThreads) sumas.

Kernel07



El resto de la reducción sigue igual.

Tseq = 72,76 ms
memcpy HtoD = 56,11 ms

```
KERNEL 07
Vector Size: 16777216
nThreads: 512
nBlocks: 4096
Tiempo Total 0.798464 ms
Ancho de Banda 168.095 GB/s
Speedup: x90.950
TEST PASS, Time seq: 72.619995 ms
```

Kernel	Tiempo	Ancho de Banda	Speedup
Kernel01	5.25 ms	25.56 GB/s	X13.86
Kernel02	4.88 ms	27.52 GB/s	X14.91
Kernel03	4.22 ms	31.80 GB/s	X17.24
Kernel04	2.76 ms	48.55 GB/s	X26.36
Kernel05	1.48 ms	90.89 GB/s	x49.16
Kernel06	1.03 ms	130.5 GB/s	X70.64
Kernel07	0.80 ms	168.1 GB/s	X90.96

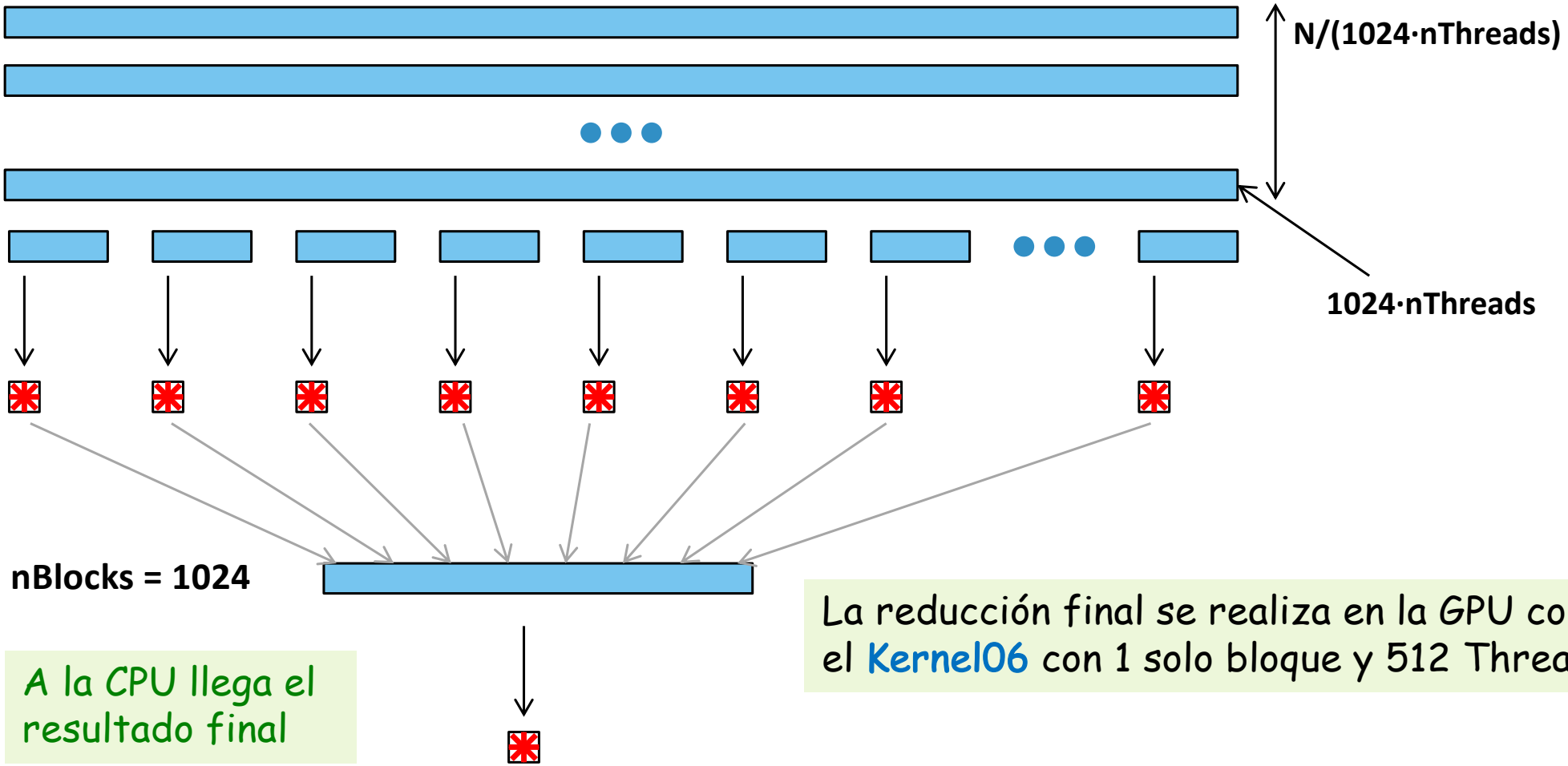
Para calcular el speedup usamos Tseq = 72,764ms

Nvprof: Todos los kernels

	Kernel 01	Kernel 02	Kernel 03	Kernel 04	Kernel 05	Kernel 06	Kernel 07
Invocaciones	1	2	2	2	2	2	1
Grid Size	(32768, 1, 1)	(32768, 1, 1) (64 1 1)*	(32768, 1, 1) (64 1 1)*	(32768, 1, 1) (64 1 1)*	(16384, 1, 1) (16 1 1)*	(32768, 1, 1) (64 1 1)*	(4096, 1, 1)
Block Size	(512, 1, 1)	(512, 1, 1)	(512, 1, 1)	(512, 1, 1)	(512, 1, 1)	(512, 1, 1)	(512, 1, 1)
Reg	13	13	11	10	13	14	16
Shared Memory	4 KB	4 KB	4 KB	4 KB	4 KB	4 KB	4 KB
sm_efficiency	99.83%	99,85%	99,83%	99,78%	99,55%	99,35%	98,98%
Achieved_occupancy	0,971	0,971	0,967	0,952	0,949	0,754	0,908
gld_request_throughput	25,59 GB/s	25,83 GB/s	29,99 GB/s	45,14 GB/s	88,77 GB/s	127,9 GB/s	176,3 GB/s
gst_request_throughput	51,19 MB/s	51,67 MB/s	59,99 MB/s	90,28 MB/s	88,77 MB/s	127,9 MB/s	44,08 MB/s
Dram_utilization	Low (2)	Low (2)	Low (1)	Low (3)	Mid (5)	Mid (6)	High (8)
RENDIMIENTO (GB/s)	25.56	27.52	31.80	48.55	90.89	130.5	168.1

(*) Tiempo despreciable del orden de µs

Kernel08



A la CPU llega el resultado final

La reducción final se realiza en la GPU con el Kernel06 con 1 solo bloque y 512 Threads.

Kernel08

```
KERNEL 08
Vector Size: 16777216
nThreads: 512
nBlocks: 1024
Tiempo Total 0.766368 ms
Ancho de Banda 175.135 GB/s
Speedup: x94.984
TEST PASS, Time seq: 72.793030 ms
```

Kernel	Tiempo	Ancho de Banda	Speedup
Kernel01	5.25 ms	25.56 GB/s	X13.86
Kernel02	4.88 ms	27.52 GB/s	X14.91
Kernel03	4.22 ms	31.80 GB/s	X17.24
Kernel04	2.76 ms	48.55 GB/s	X26.36
Kernel05	1.48 ms	90.89 GB/s	x49.16
Kernel06	1.03 ms	130.5 GB/s	X70.64
Kernel07	0.80 ms	168.1 GB/s	X90.96
Kernel08	0.77 ms	175.1 GB/s	X94.50

Para calcular el speedup usamos Tseq = 72,764ms

Kernel01 → Kernel02
Kernel07 → Kernel08

Es la misma optimización

Resultados en una NVIDIA GTX 1080Ti

	Tesla K40c (double)			NVIDIA GTX 1080Ti (float)		
Kernel	Tiempo	Ancho de Banda	Speedup	Tiempo	Ancho de Banda	Speedup
Kernel01	5.25 ms	25.56 GB/s	X13.86	1.86 ms	36.08 GB/s	x7.03
Kernel02	4.88 ms	27.52 GB/s	X14.91	1.69 ms	39.71 GB/s	x7.73
Kernel03	4.22 ms	31.80 GB/s	X17.24	0.87 ms	71.14 GB/s	x15.02
Kernel04	2.76 ms	48.55 GB/s	X26.36	0.76 ms	88.30 GB/s	x17.20
Kernel05	1.48 ms	90.89 GB/s	x49.16	0.38 ms	176.60 GB/s	x34.40
Kernel06	1.03 ms	130.5 GB/s	X70.64	0.26 ms	258.11 GB/s	x50.27
Kernel07	0.80 ms	168.1 GB/s	X90.96	0.23 ms	291.78 GB/s	x56.83
Kernel08	0.77 ms	175.1 GB/s	X94.50	0.21 ms	319.57 GB/s	x62.24

Para calcular el speedup usamos Tseq = 72,764ms

Para calcular el speedup usamos Tseq = 13,071ms

Ancho de Banda de la K40: **288 GB/s**

Ancho de Banda de la GTX 1080Ti: **484,3 GB/s**

Resultados en una NVIDIA GeForce RTX 2060 SUPER

	Tesla K40c (double)			NVIDIA GeForce RTX 2060 SUPER (float)		
Kernel	Tiempo	Ancho de Banda	Speedup	Tiempo	Ancho de Banda	Speedup
Kernel01	5.25 ms	25.56 GB/s	X13.86	1.52 ms	44.15 GB/s	x8.60
Kernel02	4.88 ms	27.52 GB/s	X14.91	1.50 ms	44.74 GB/s	x8.71
Kernel03	4.22 ms	31.80 GB/s	X17.24	1.16 ms	57.85 GB/s	x11.27
Kernel04	2.76 ms	48.55 GB/s	X26.36	0.96 ms	69.91 GB/s	X13.62
Kernel05	1.48 ms	90.89 GB/s	x49.16	0.44 ms	152.52 GB/s	x29.71
Kernel06	1.03 ms	130.5 GB/s	X70.64	0.31 ms	216.48 GB/s	x42.16
Kernel07	0.80 ms	168.1 GB/s	X90.96	0.20 ms	335.54 GB/s	x65.36

Para calcular el speedup usamos Tseq = 72,764ms

Para calcular el speedup usamos Tseq = 13,071ms

Ancho de Banda de la K40: **288 GB/s**

Ancho de Banda de la RTX 2060 SUPER: **448 GB/s**

Datos obtenidos por el grupo tga0019 (17/Mar/2020)

Algún comentario sobre CF (float vs double)

Esta práctica estaba diseñada para trabajar con float.

```
Resultado GPU: 8389059.0
Resultado CPU: 8389376.0
Diferencia   :     317.0
Error        :   0.00004

Resultado GPU: 8388595.5
Resultado CPU: 8388490.0
Diferencia   :     105.5
Error        :   0.00001

Resultado GPU: 8387599.5
Resultado CPU: 8386772.0
Diferencia   :     827.5
Error        :   0.00010
```

Kernel01 con **float**

```
Resultado GPU: 8388336.1
Resultado CPU: 8388336.1
Diferencia   :       0.0
Error        :   0.00000

Resultado GPU: 8390269.9
Resultado CPU: 8390269.9
Diferencia   :       0.0
Error        :   0.00000

Resultado GPU: 8388889.0
Resultado CPU: 8388889.0
Diferencia   :       0.0
Error        :   0.00000
```

Kernel01 con **double**

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament d'Arquitectura de Computadors

Tarjetas Gráficas y Aceleradores

CUDA – Reducciones

Agustín Fernández

Departament d'Arquitectura de Computadors

Facultat d'Informàtica de Barcelona

Universitat Politècnica de Catalunya