UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament d'Arquitectura de Computadors

Tarjetas Gráficas y Aceleradores

CUDA – Sesion02 - Puzzles

Agustín Fernández

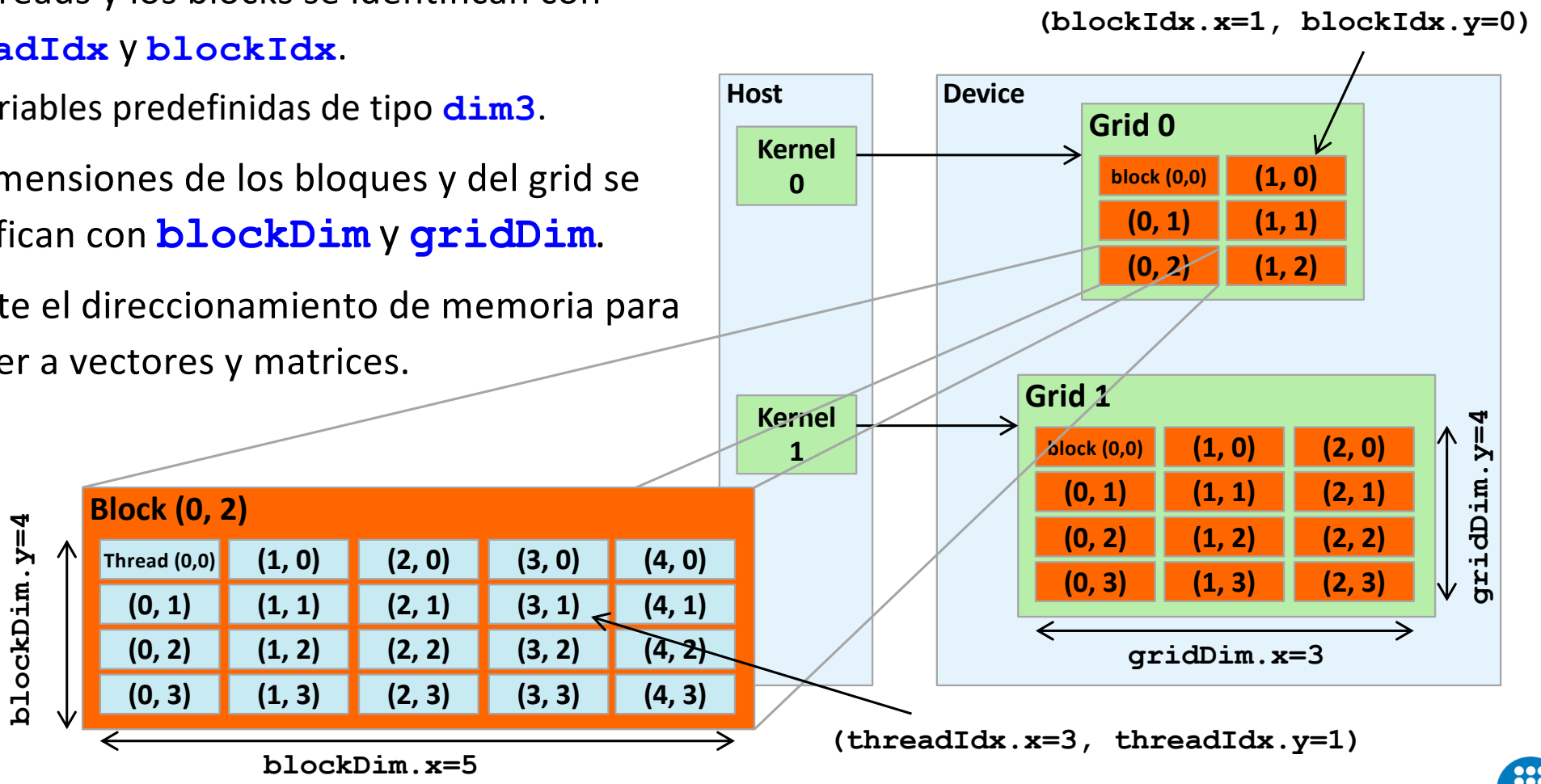
Departament d'Arquitectura de Computadors

Facultat d'Informàtica de Barcelona

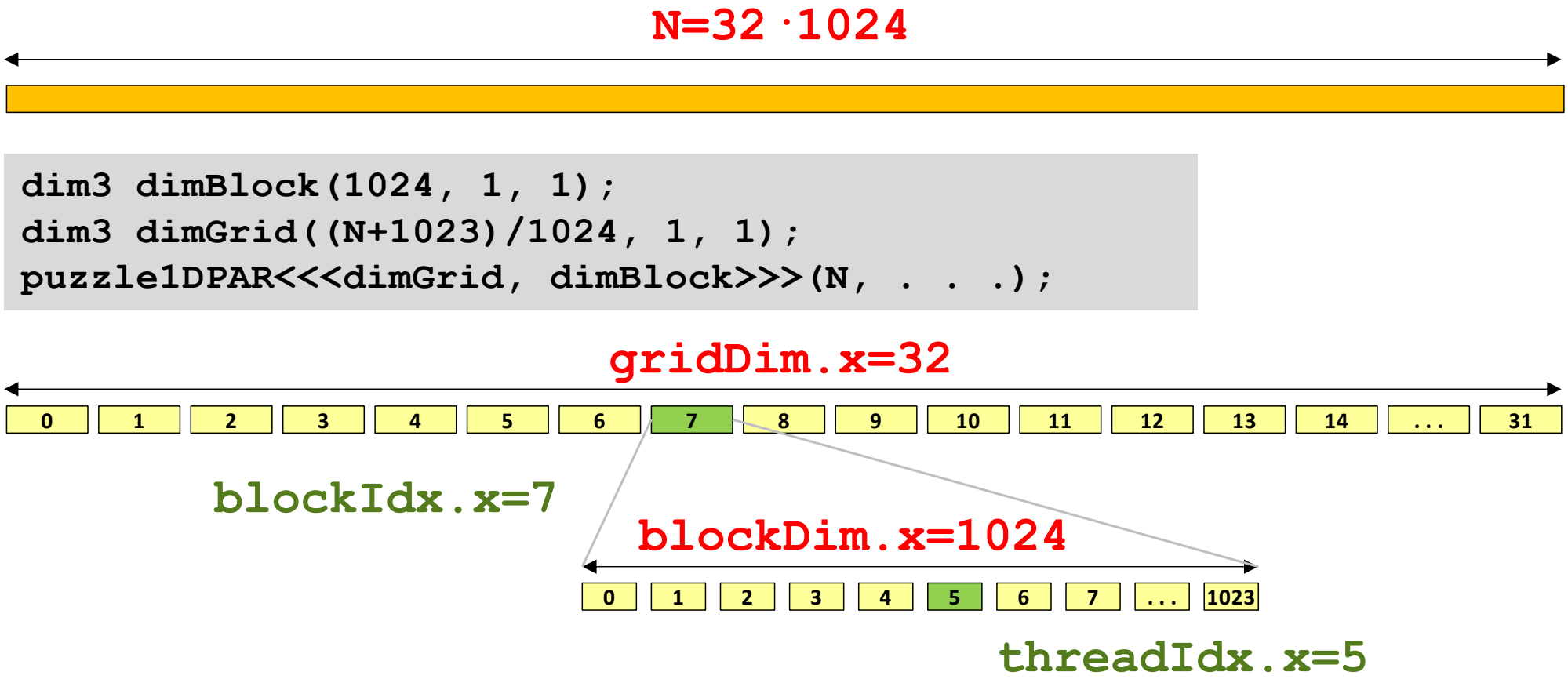
Universitat Politècnica de Catalunya

blockIdx & threadIdx

- Los threads y los blocks se identifican con **`threadIdx`** y **`blockIdx`**.
  - Variables predefinidas de tipo **`dim3`**.
- Las dimensiones de los bloques y del grid se identifican con **`blockDim`** y **`gridDim`**.
- Permite el direccionamiento de memoria para acceder a vectores y matrices.



Puzzle 1D



Id = blockIdx.x * blockDim.x + threadIdx.x = 7·1024+5 = 7173

Puzzle1D

```
void puzzle1DSeq(int N, float *z, float *x, float *y) {
  int i;
  for (i=0; i<N; i++)
    z[i] = 0.5*x[i] + 0.75*y[i] + x[i]*y[i];
}
```

```
__global__ void puzzle1DPAR(int N, float *z, float *x, float *y) {
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  z[i] = 0.5*x[i] + 0.75*y[i] + x[i]*y[i];
}

nThreads = 1024;
nBlocks = (N + nThreads - 1)/nThreads;
dim3 dimGrid(nBlocks, 1, 1);
dim3 dimBlock(nThreads, 1, 1);
puzzle1DPAR<<<dimGrid, dimBlock>>>(N, dZ, dX, dY);
```

Puzzle1D. Cuestiones

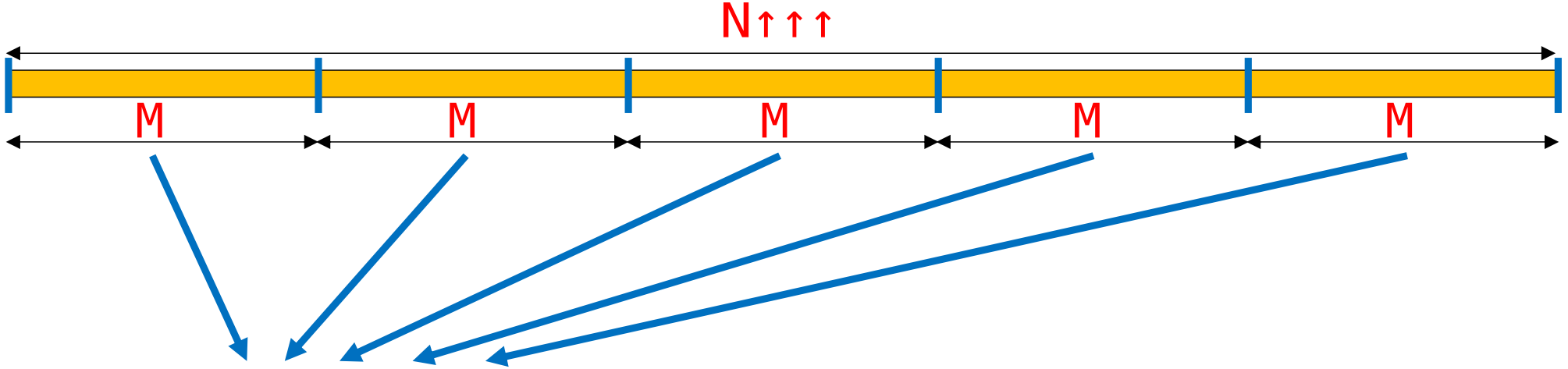
2. Cambiad el tamaño para que NO sea múltiplo del número de threads. Modificad el código, dónde sea necesario, para hacer que funcione correctamente.

```
__global__ void puzzle1DPAR(int N, float *z, float *x, float *y) {
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  if (i < N)
    z[i] = 0.5*x[i] + 0.75*y[i] + x[i]*y[i];
}

nThreads = 1024;
nBlocks = (N + nThreads - 1)/nThreads;
dim3 dimGrid(nBlocks, 1, 1);
dim3 dimBlock(nThreads, 1, 1);
puzzle1DPAR<<<dimGrid, dimBlock>>>(N, dZ, dX, dY);
```

Puzzle1D. Cuestiones

3. ¿Qué modificarías en la implementación para considerar ahora que el tamaño del problema no cabe en la memoria de la GPU?

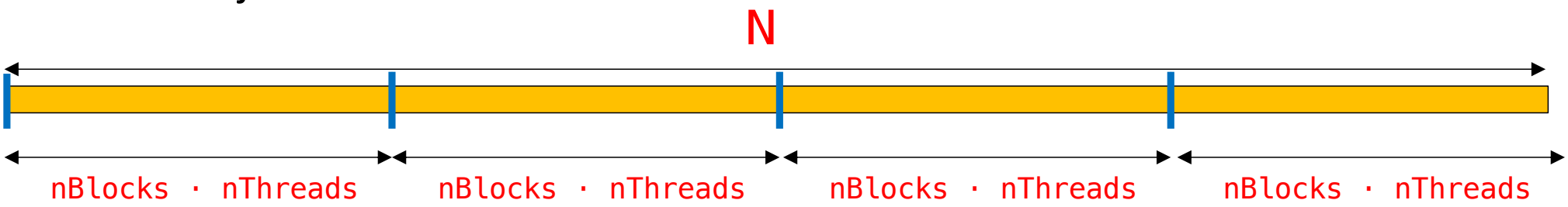



```
cudaMemcpy(…, M, HtD);
kernel<<<…>>>(M, …);
cudaMemcpy(…, M, DtH);
```

Hay que trocear el problema, para que los subproblemas quepan en memoria.
Sólo hay que manipular punteros.

Puzzle1D. Cuestiones

4. ¿Qué modificarías en la implementación para considerar que el número de Blocks es fijo?

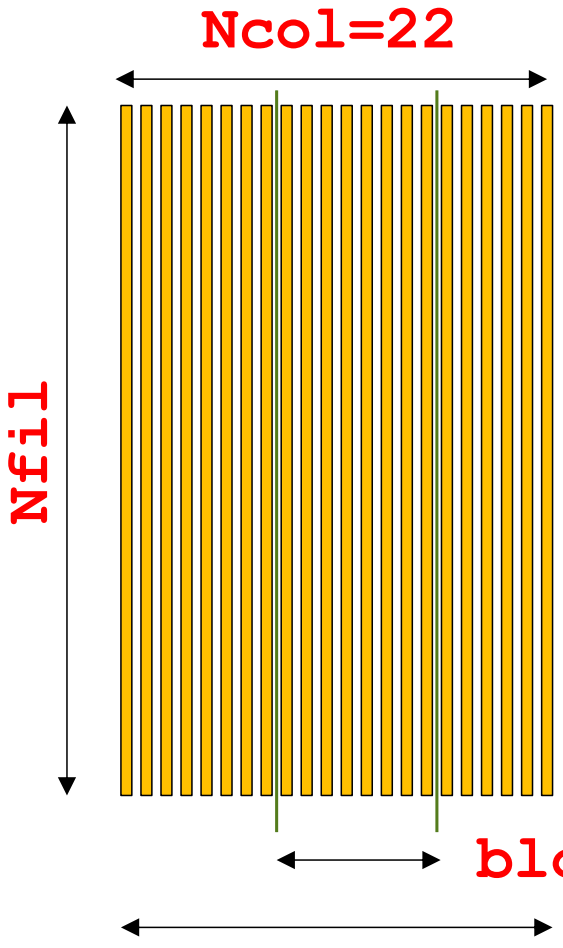



```
__global__ void puzzle1DPAR(int N, float *z, float *x, float *y) {
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  int stride = blockDim.x * gridDim.x;

  while (i < N) {
    z[i] = 0.5*x[i] + 0.75*y[i] + x[i]*y[i];
    i = i + stride;
  }
}

puzzle1DPAR<<<10000, 1024>>>(N, dZ, dX, dY);
```

Puzzle 2D por columnas




```
dim3 dimBlockC(8, 1, 1);
dim3 dimGridC((22+7)/8, 1, 1);
puzzle2DPAR<<<dimGridC, dimBlockC>>>(N, . . .);
```

```
col = blockIdx.x * blockDim.x + threadIdx.x
```

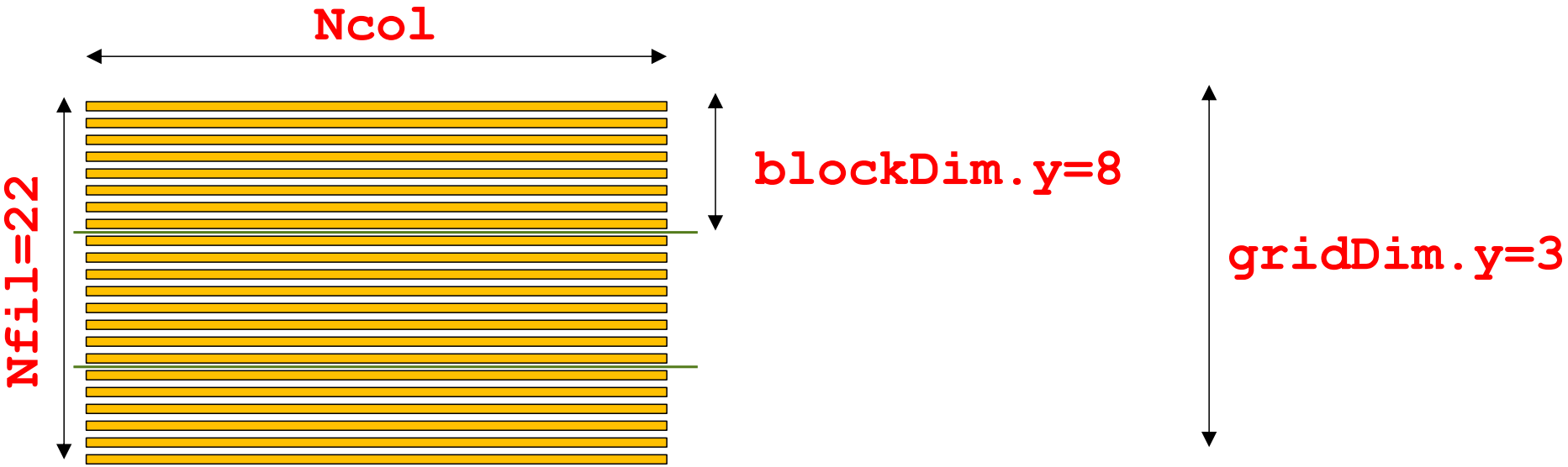
Puzzle2D por columnas

```
void puzzle2DSeq(int Nfil, int Ncol, float *z, float *x, float *y) {
  int i, j, ind;
  for (i=0; i<Nfil; i++)
    for (j=0; j<Ncol; j++){
      ind = i * Ncol + j;
      z[ind] = 0.5*x[ind] + 0.75*y[ind] + x[ind]*y[ind];
    }
}
```

```
__global__ void puzzle2DPARcol(int Nfil, int Ncol, float *z, float *x, float *y) {
  int j = blockIdx.x * blockDim.x + threadIdx.x;
  int ind = j;
  if (j < Ncol)
    for (int i=0; i<Nfil; i++, ind = ind + Ncol)
      z[ind] = 0.5*x[ind] + 0.75*y[ind] + x[ind]*y[ind];
}

nThreads = 256;
nBlocks = (Ncol + nThreads - 1)/nThreads;
dim3 dimGridC(nBlocks, 1, 1);
dim3 dimBlockC(nThreads, 1, 1);
puzzle2DPARcol<<<dimGridC, dimBlockC>>>(Nfil, Ncol, dZ, dX, dY);
```

Puzzle 2D por filas




```
dim3 dimBlockF(1, 8, 1);
dim3 dimGridF(1,(22+7)/8, 1);
puzzle2DPAR<<<dimGridF, dimBlockF>>>(N, . . .);
```

```
fil = blockIdx.y * blockDim.y + threadIdx.y
```

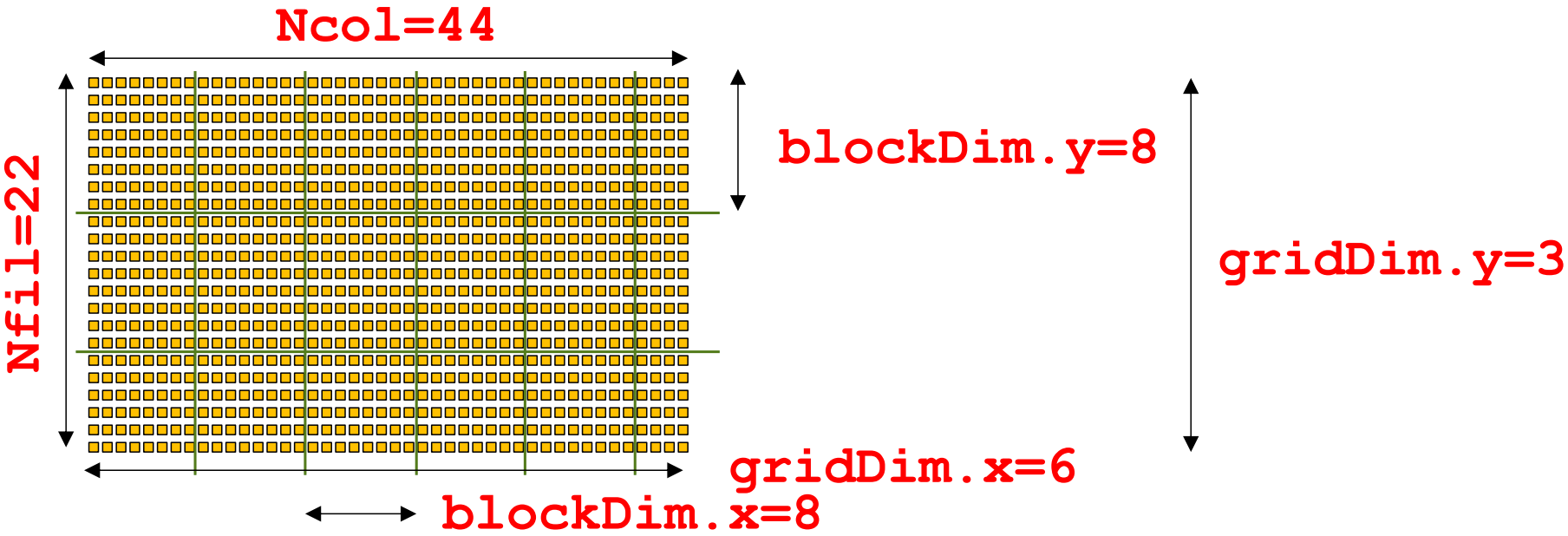
Puzzle2D por filas

```
void puzzle2DSeq(int Nfil, int Ncol, float *z, float *x, float *y) {
  int i, j, ind;
  for (i=0; i<Nfil; i++)
    for (j=0; j<Ncol; j++){
      ind = i * Ncol + j;
      z[ind] = 0.5*x[ind] + 0.75*y[ind] + x[ind]*y[ind];
    }
}
```

```
__global__ void puzzle2DPARfil(int Nfil, int Ncol, float *z, float *x, float *y) {
  int i = blockIdx.y * blockDim.y + threadIdx.y;
  int ind = i * Ncol;
  if (i < Nfil)
    for (int j=0; j<Ncol; j++, ind++)
      z[ind] = 0.5*x[ind] + 0.75*y[ind] + x[ind]*y[ind];
}

nThreads = 256;
nBlocks = (Nfil + nThreads - 1)/nThreads;
dim3 dimGridF(1, nBlocks, 1);
dim3 dimBlockF(1, nThreads, 1);
puzzle2DPARfil<<<dimGridF, dimBlockF>>>(Nfil, Ncol, dZ, dX, dY);
```

Puzzle 2D elemento a elemento




```
dim3 dimBlockE(8, 8, 1);
dim3 dimGridE((44+7)/8,(22+7)/8, 1);
puzzle2DPAR<<<dimGridE, dimBlockE>>>(N, . . .);
```

```
fil = blockIdx.y * blockDim.y + threadIdx.y
col = blockIdx.x * blockDim.x + threadIdx.x
```

Puzzle2D elemento a elemento

```
void puzzle2DSeq(int Nfil, int Ncol, float *z, float *x, float *y) {
  int i, j, ind;
  for (i=0; i<Nfil; i++)
    for (j=0; j<Ncol; j++){
      ind = i * Ncol + j;
      z[ind] = 0.5*x[ind] + 0.75*y[ind] + x[ind]*y[ind];
    }
}
```

```
__global__ void puzzle2DPAR1x1(int Nfil, int Ncol, float *z, float *x, float *y) {
  int i = blockIdx.y * blockDim.y + threadIdx.y;
  int j = blockIdx.x * blockDim.x + threadIdx.x;
  int ind = i * Ncol + j;
  if (i < Nfil && j < Ncol)
    z[ind] = 0.5*x[ind] + 0.75*y[ind] + x[ind]*y[ind];
}
nThreads = 16;
nBlocksCol = (Ncol + nThreads - 1)/nThreads;
nBlocksFil = (Nfil + nThreads - 1)/nThreads;
dim3 dimGridE(nBlocksCol, nBlocksFil, 1);
dim3 dimBlockE(nThreads, nThreads, 1);
puzzle2DPAR1x1<<<dimGridE, dimBlockE>>>(Nfil, Ncol, dZ, dX, dY);
```

Puzzle2D columnas vs filas vs elementos

```
__global__ void puzzle2DPARcol(DSeq(int Nfil, int Ncol, float *z, float *x, float *y) {
  int j = blockIdx.x * blockDim.x + threadIdx.x;
  int ind = j;
  if (j < Ncol)
    for (int i=0; i<Nfil; i++, ind = ind + Ncol)
      z[ind] = 0.5*x[ind] + 0.75*y[ind] + x[ind]*y[ind];
}
```

```
nTh = 256;
nBlC = (Ncol + nTh-1)/nTh;
dim3 dimGridC(nBlC, 1, 1);
dim3 dimBlockC(nTh, 1, 1);
```

```
__global__ void puzzle2DPARfil(DSeq(int Nfil, int Ncol, float *z, float *x, float *y) {
  int i = blockIdx.y * blockDim.y + threadIdx.y;
  int ind = i * Ncol;
  if (i < Nfil)
    for (int j=0; j<Ncol; j++, ind++)
      z[ind] = 0.5*x[ind] + 0.75*y[ind] + x[ind]*y[ind];
}
```

```
nTh = 256;
nBlF = (Nfil + nTh-1)/nTh;
dim3 dimGridF(1, nBlF, 1);
dim3 dimBlockF(1, nTh, 1);
```

```
__global__ void puzzle2DPAR1x1(int Nfil, int Ncol, float *z, float *x, float *y) {
  int i = blockIdx.y * blockDim.y + threadIdx.y;
  int j = blockIdx.x * blockDim.x + threadIdx.x;
  int ind = i * Ncol + j;
  if (i < Nfil && j < Ncol)
    z[ind] = 0.5*x[ind] + 0.75*y[ind] + x[ind]*y[ind];
}
```

```
nTh = 16;
nBlC = (Ncol + nTh-1)/nTh;
nBlF = (Nfil + nTh-1)/nTh;
dim3 dimGridE(nBlC, nBlF, 1);
dim3 dimBlockE(nTh, nTh, 1);
```

Puzzle2D columnas vs filas vs elementos

```
nThreads = 256;
nBlocks = (Ncol + nThreads - 1)/nThreads;
dim3 dimGridC(nBlocks, 1, 1);
dim3 dimBlockC(nThreads, 1, 1);
puzzle2DPARcol<<<dimGridC, dimBlockC>>>(Nfil, Ncol, dZ, dX, dY);
```

```
nThreads = 256;
nBlocks = (Nfil + nThreads - 1)/nThreads;
dim3 dimGridF(1, nBlocks, 1);
dim3 dimBlockF(1, nThreads, 1);
puzzle2DPARfil<<<dimGridF, dimBlockF>>>(Nfil, Ncol, dZ, dX, dY);
```

```
nThreads = 16;
nBlocksCol = (Ncol + nThreads - 1)/nThreads;
nBlocksFil = (Nfil + nThreads - 1)/nThreads;
dim3 dimGridE(nBlocksCol, nBlocksFil, 1);
dim3 dimBlockE(nThreads, nThreads, 1);
puzzle2DPAR1x1<<<dimGridE, dimBlockE>>>(Nfil, Ncol, dZ, dX, dY);
```

Puzzle2D Rendimiento

```
Kernel por Filas
Dimension problema: 1023 filas x 1023 columnas
Dimension Block: 1 x 256 x 1 (256) threads
Dimension Grid: 1 x 4 x 1 (4) blocks

Kernel por Columnas
Dimension problema: 1023 filas x 1023 columnas
Dimension Block: 256 x 1 x 1 (256) threads
Dimension Grid: 4 x 1 x 1 (4) blocks

Kernel Elemento a Elemento
Dimension problema: 1023 filas x 1023 columnas
Dimension Block: 16 x 16 x 1 (256) threads
Dimension Grid: 64 x 64 x 1 (4096) blocks

Resumen Rendimiento
Tiempo Paralelo Kernel filas: 2.729280 ms (1.92 GFLOPS)
Tiempo Paralelo Kernel columnas: 0.738496 ms (7.09 GFLOPS)
Tiempo Paralelo Kernel elemento a elemento: 0.117344 ms (44.59 GFLOPS)
Tiempo Secuencial: 0.805800 milseg (6.49 GFLOPS)
```

¿Explicación de los rendimientos?

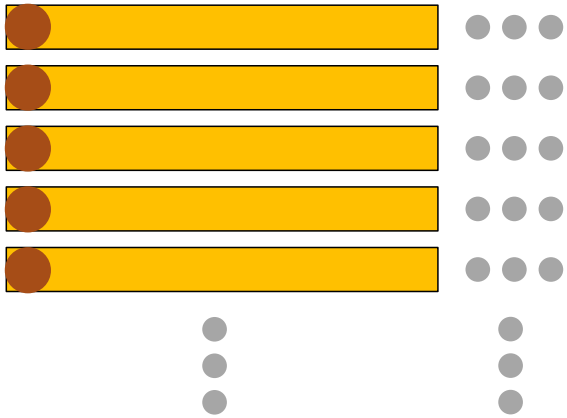
Número de Bloques

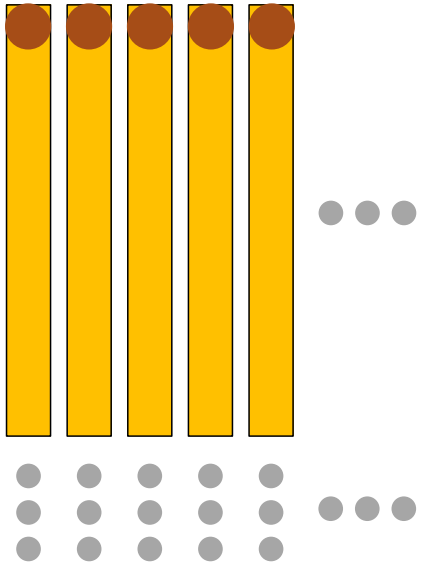
Patrones de acceso a memoria de los warps

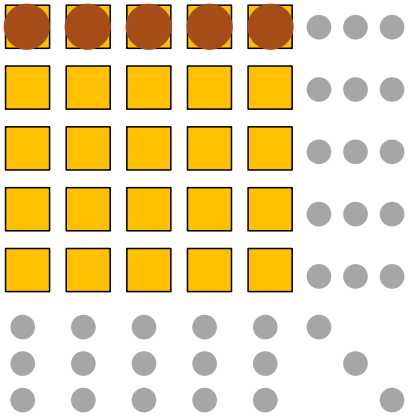
Puzzle2D Rendimiento

¿Explicación de los rendimientos?

Patrones de acceso a memoria de los warps







```
Kernel por Filas
1.92 GFLOPS
4 blocks (256 threads)
```

```
Kernel por Columnas
7.09 GFLOPS
4 blocks (256 threads)
```

```
Kernel Elemento a Elemento
44.59 GFLOPS
4096 blocks (16x16 threads)
```

Puzzle3D elemento a elemento

```
void puzzle3DSeq(int Ncar, int Nfil, int Ncol, float *z, float *x, float *y) {
  int i, j, t, ind;
  for (t=0; i<Ncar; t++)
    for (i=0; i<Nfil; i++)
      for (j=0; j<Ncol; j++){
        ind = t*Nfil*Ncol + i*Ncol + j;
        z[ind] = 0.5*x[ind] + 0.75*y[ind] + x[ind]*y[ind];
    }
}
```

```
__global__ void puzzle3D1x1x1(int Ncar, int Nfil, int Ncol, float *z, float *x, float *y) {
  int i = blockIdx.y * blockDim.y + threadIdx.y;
  int j = blockIdx.x * blockDim.x + threadIdx.x;
  int t = blockIdx.z * blockDim.z + threadIdx.z;
  int ind = t*Nfil*Ncol + i*Ncol + j;
  if (t<Ncar && i<Nfil && j<Ncol)
    z[ind] = 0.5*x[ind] + 0.75*y[ind] + x[ind]*y[ind];
}
```

Puzzle3D elemento a elemento

```
nThreads = 8;
nBlocksCol = (Ncol + nThreads - 1)/nThreads;
nBlocksFil = (Nfil + nThreads - 1)/nThreads;
nBlocksCar = (Ncar + nThreads - 1)/nThreads;
dim3 dimGridE(nBlocksCol, nBlocksFil, nBlocksCar);
dim3 dimBlockE(nThreads, nThreads, nThreads);
puzzle3D1x1x1<<<dimGridE, dimBlockE>>>(Ncar, Nfil, Ncol, dZ, dX, dY);
```

```
__global__ void puzzle3D1x1x1(int Ncar, int Nfil, int Ncol, float *z, float *x, float *y) {
  int i = blockIdx.y * blockDim.y + threadIdx.y;
  int j = blockIdx.x * blockDim.x + threadIdx.x;
  int t = blockIdx.z * blockDim.z + threadIdx.z;
  int ind = t*Nfil*Ncol + i*Ncol + j;
  if (t<Ncar && i<Nfil && j<Ncol)
    z[ind] = 0.5*x[ind] + 0.75*y[ind] + x[ind]*y[ind];
}
```

Opciones de nvprof

```
nvprof --print-gpu-summary ./puzzle2D.exe 1024 1024 Y
```

```
==78241== NVPROF is profiling process 78241, command: ./puzzle2D.exe 1024 1024 Y
==78241== Profiling application: ./puzzle2D.exe 1024 1024 Y
==78241== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 37.85%  4.1430ms         3  1.3810ms  752.17us  2.1809ms  [CUDA memcpy DtoH]
 31.20%  3.4150ms         2  1.7075ms  1.6854ms  1.7296ms  [CUDA memcpy HtoD]
 23.76%  2.6010ms         1  2.6010ms  2.6010ms  2.6010ms  puzzle2DPARfil(...)
  6.37%  696.96us         1  696.96us  696.96us  696.96us  puzzle2DPARcol(...)
  0.82%  89.568us         1  89.568us  89.568us  89.568us  puzzle2DPAR1x1(...)
```

Opciones de nvprof

```
nvprof --print-gpu-trace ./puzzle2D.exe 1024 1024 Y
```

```
==78527== Profiling result:
   Start  Duration  Grid Size  Block Size   Regs    SSMem  DSMem      Size  Throughput Name
10.9343s  1.2034ms          -           -     -       -      -  3.9922MB  3.2397GB/s [CUDA memcpy HtoD]
10.9358s  1.2115ms          -           -     -       -      -  3.9922MB  3.2181GB/s [CUDA memcpy HtoD]
10.9378s  2.6032ms    (1 4 1)   (1 256 1)    16      0B     0B         -           - puzzle2DPARfil()
10.9405s  2.1098ms          -           -     -       -      -  3.9922MB  1.8479GB/s [CUDA memcpy DtoH]
10.9498s  694.50us    (4 1 1)   (256 1 1)    20      0B     0B         -           - puzzle2DPARcol()
10.9505s  855.94us          -           -     -       -      -  3.9922MB  4.5548GB/s [CUDA memcpy DtoH]
10.9556s  89.857us  (64 64 1)   (16 16 1)    12      0B     0B         -           - puzzle2DPAR1x1()
10.9557s  753.51us          -           -     -       -      -  3.9922MB  5.1740GB/s [CUDA memcpy DtoH]
```

Opciones de nvprof

```
nvprof --metrics OP1,OP2,OP3,OP4,OP5 ./puzzle2D.exe 1024 1024 Y
```

```
Invocations                               Metric Name                         Metric Description         Min         Max         Avg
Device "Tesla K40c (0)"
Kernel: puzzle2DPARfil(int, int, float*, float*, float*)
1                                      sm_efficiency                    Multiprocessor Activity      26.46%      26.46%      26.46%
1                                  achieved_occupancy                         Achieved Occupancy    0.124689    0.124689    0.124689
1                            gld_requested_throughput          Requested Global Load Throughput  2.9858GB/s  2.9858GB/s  2.9858GB/s
1                            gst_requested_throughput         Requested Global Store Throughput  1.4929GB/s  1.4929GB/s  1.4929GB/s
1                                    dram_utilization                Device Memory Utilization     Low (1)     Low (1)     Low (1)
Kernel: puzzle2DPAR1x1(int, int, float*, float*, float*)
1                                      sm_efficiency                    Multiprocessor Activity      93.70%      93.70%      93.70%
1                                  achieved_occupancy                         Achieved Occupancy    0.810772    0.810772    0.810772
1                            gld_requested_throughput          Requested Global Load Throughput  86.601GB/s  86.601GB/s  86.601GB/s
1                            gst_requested_throughput         Requested Global Store Throughput  43.300GB/s  43.300GB/s  43.300GB/s
1                                    dram_utilization                Device Memory Utilization    High (7)    High (7)    High (7)
Kernel: puzzle2DPARcol(int, int, float*, float*, float*)
1                                      sm_efficiency                    Multiprocessor Activity      26.27%      26.27%      26.27%
1                                  achieved_occupancy                         Achieved Occupancy    0.124639    0.124639    0.124639
1                            gld_requested_throughput          Requested Global Load Throughput  11.185GB/s  11.185GB/s  11.185GB/s
1                            gst_requested_throughput         Requested Global Store Throughput  5.5924GB/s  5.5924GB/s  5.5924GB/s
1                                    dram_utilization                Device Memory Utilization     Low (1)     Low (1)     Low (1)
```

Distribuyendo el trabajo entre todas las GPUs

En Boada-5 hay 4 GPUs. Si no hacemos nada todos los Jobs se ejecutan en la misma GPU

En el job.sh

```
…
#SBATCH --gres=gpu:4
…
```

```
cudaGetDeviceCount(&count);

srand(time(NULL));
gpu = rand();

cudaSetDevice((gpu>>3) % count);
```

La rutina `cudaGetDeviceCount` nos dice cuantas GPUs hay en el sistema, el resultado estará en `count`.

Obtiene un número aleatorio. Nos aseguramos que sea un valor diferente cada vez que ejecutamos el programa.

La rutina `cudaSetDevice(X)` indica que todo lo que viene a continuación se ejecuta en la GPU `X`. `X` ha de ser un valor entre `0` y `count-1`.

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament d'Arquitectura de Computadors

Tarjetas Gráficas y Aceleradores

CUDA – Sesion02 - Puzzles

Agustín Fernández

Departament d'Arquitectura de Computadors

Facultat d'Informàtica de Barcelona

Universitat Politècnica de Catalunya