MODELO DE PROCESAMIENTO

DISPOSITIVOS E INFRAESTRUCTURAS PARA SISTEMAS MULTIMEDIA

GRADO EN INGENIERÍA MULTIMEDIA 2018-2019

Albert García-García < agarcia@dtic.ua.es >

Jose García-Rodríguez < jgarcia@dtic.ua.es>

CONTENIDO

SINTAXIS DEL LANZAMIENTO DE KERNELS
LÍMITES DEL LANZAMIENTO DE KERNELS
MALLAS, BLOQUES, HILOS Y WARPS
ESCALABILIDAD TRANSPARENTE
PLANIFICACIÓN DE HILOS
SINCRONIZACIÓN Y CONTROL DE FLUJO
STREAMS
MEDICIÓN DE TIEMPOS Y SINCRONIZACIÓN

SINTAXIS DEL LANZAMIENTO DE KERNELS

SINTAXIS (CUDA RT API)

```
kernel <<< malla, bloque, memoria_compartida, stream >>> (parámetro1,
                                                          parámetro2, ...,
                                                          parámetroN);
```

SINTAXIS DEL LANZAMIENTO DE KERNELS

SINTAXIS (CUDA DRIVER API)

```
CUResult cuLaunchKernel (
        CUfunction kernel,
        unsigned int mallaDimX,
        unsigned int mallaDimY,
        unsigned int mallaDimZ,
        unsigned int bloqueDimX,
        unsigned int bloqueDimY,
        unsigned int bloqueDimZ,
        unsigned int memoriaCompartidaBytes,
        CUstream stream,
        void ** parametros,
        void ** extra );
```

LÍMITES DEL LANZAMIENTO DE KERNELS

LIMITACIONES ESPACIALES

Recursos reservados al inicio

Registros limitados por hilo

Memoria local limitada por hilo

Memoria global no es infinita

No hay memoria virtual NI swapping

OOM

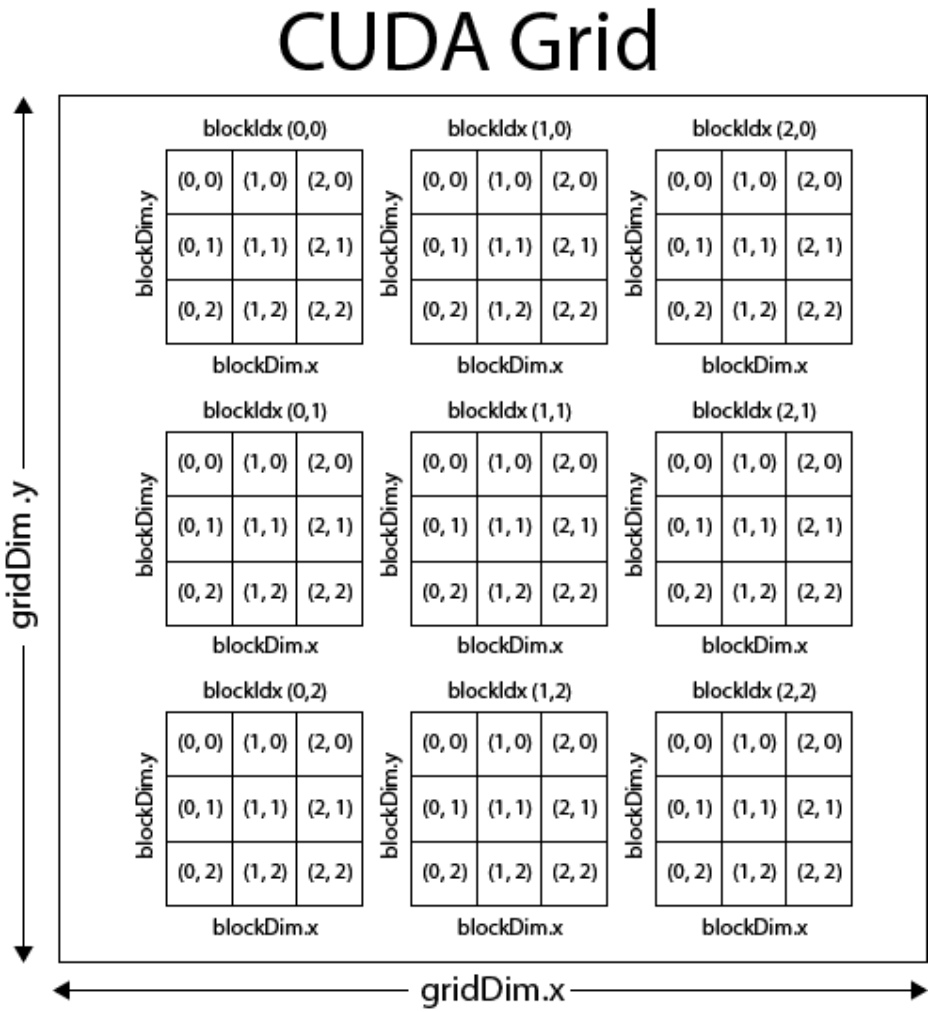
LÍMITES DEL LANZAMIENTO DE KERNELS

LIMITACIONES TEMPORALES

Quechua

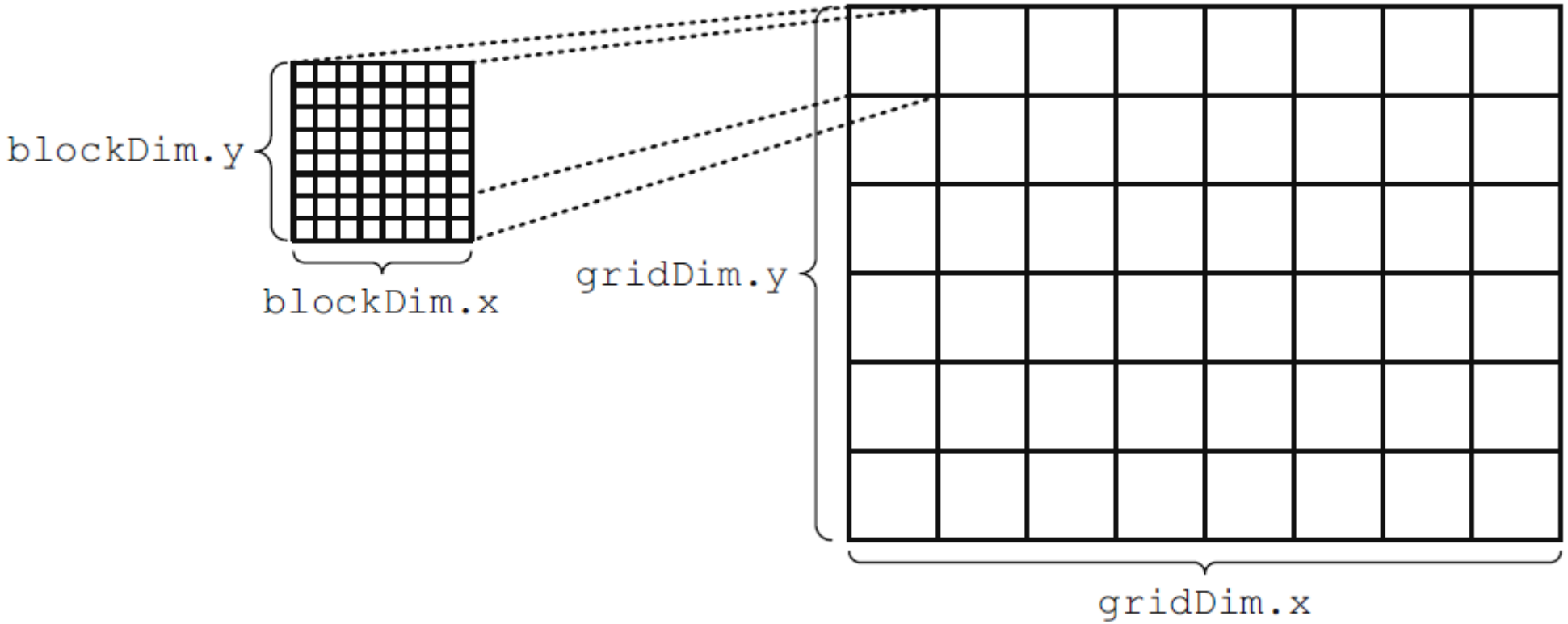
MALLAS, BLOQUES, HILOS Y WARPS

MALLAS Y BLOQUES



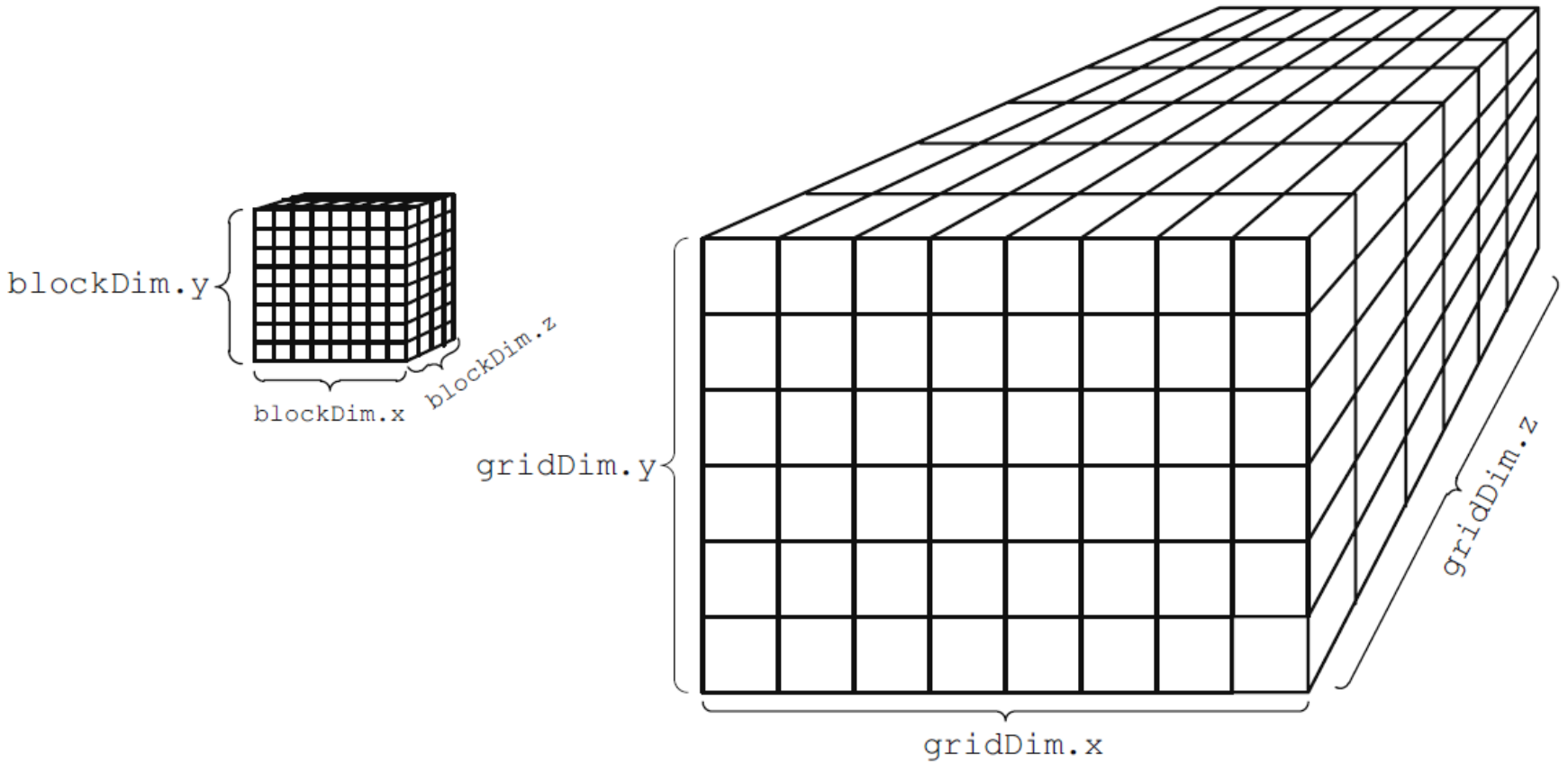
MALLAS, BLOQUES, HILOS Y WARPS

MALLAS Y BLOQUES (2D)



MALLAS, BLOQUES, HILOS Y WARPS

MALLAS Y BLOQUES (3D)



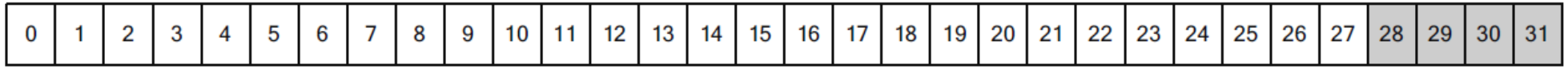
MALLAS, BLOQUES, HILOS Y WARPS

HILOS, WARPS Y LANES

32

MALLAS, BLOQUES, HILOS Y WARPS

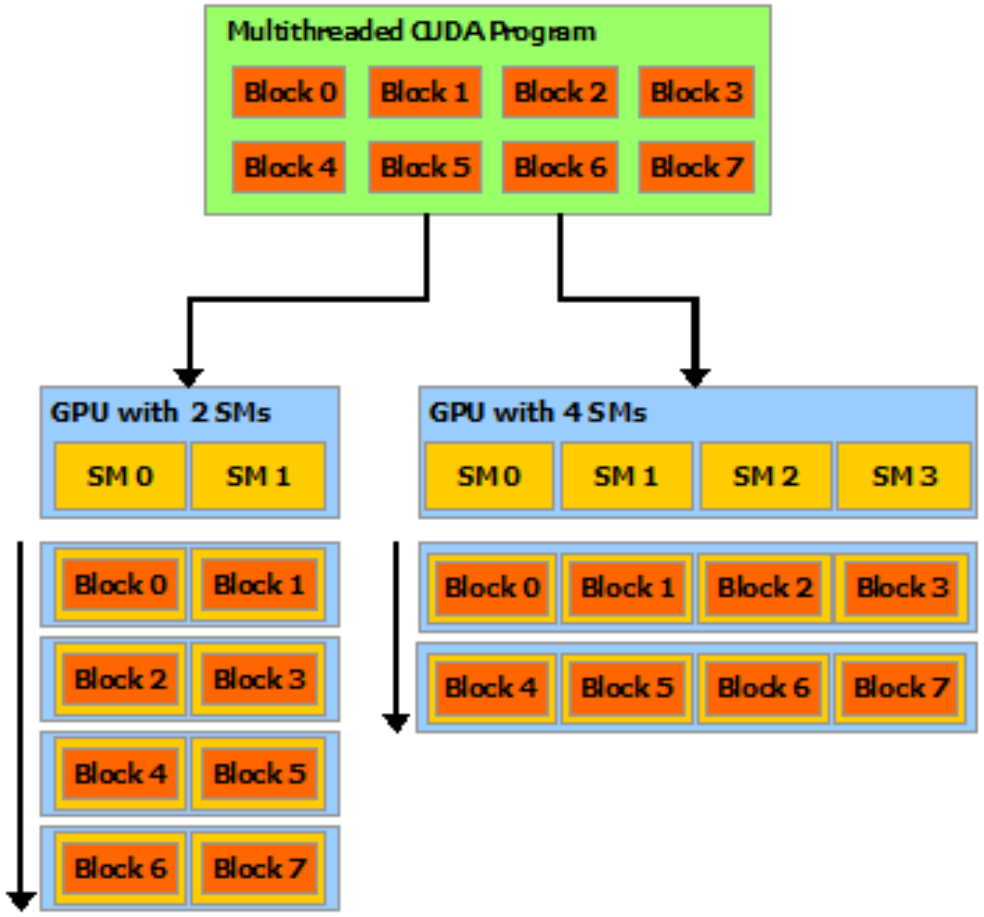
HILOS, WARPS Y LANES (SI EL BLOQUE NO ES MÚLTIPLO DE 32)



blockDim = (28,1,1)

ESCALABILIDAD TRANSPARENTE

LOS BLOQUES SE ASIGNAN A LO SMs PARA SU EJECUCIÓN



PLANIFICACIÓN DE HILOS

AGRUPACIÓN DE HILOS

Las lanes de un warp se ejecutan físicamente en paralelo

Todos los hilos de un warp ejecutan la misma instrucción

Elegidos según una cola de prioridad para ejecución

LOS WARPS SON LA UNIDAD DE EJECUCIÓN DE CUDA

SINCRONIZACIÓN Y CONTROL DE FLUJO EN GPU

MÉTODOS DE SINCRONIZACIÓN ENTRE HILOS DE UN MISMO BLOQUE

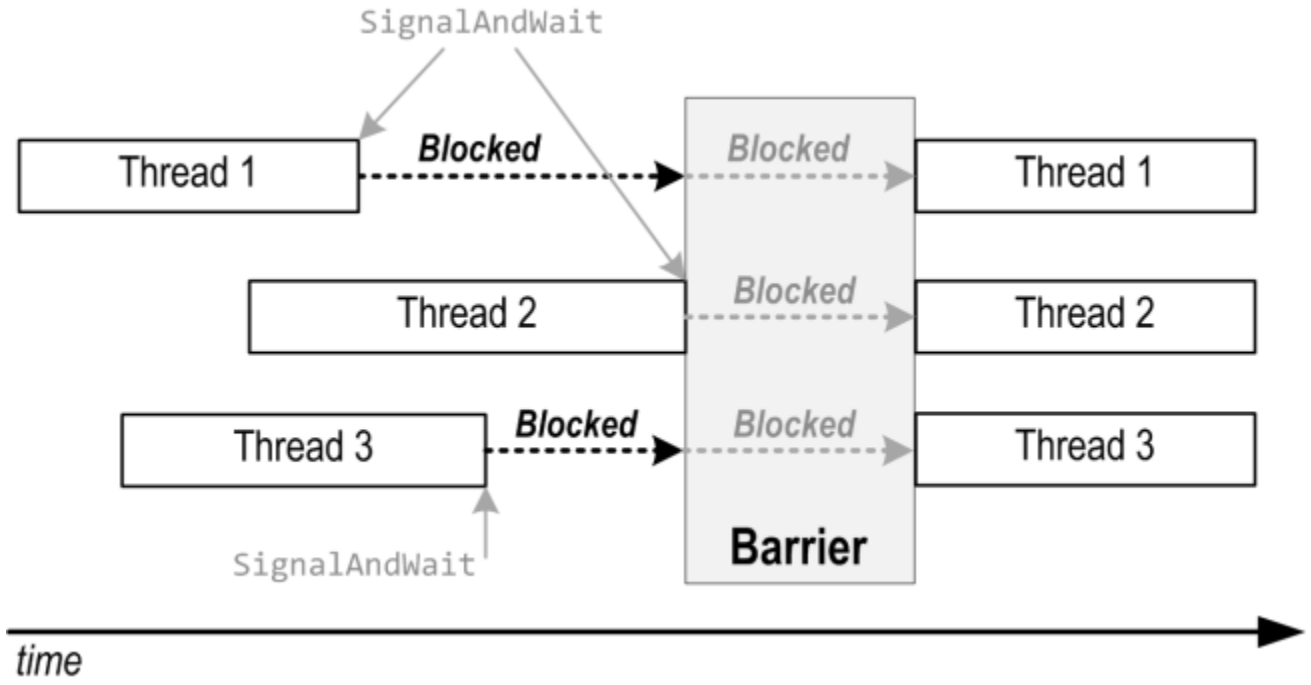
BARRERAS DE SINCRONIZACIÓN

OPERACIONES ATÓMICAS

MEMORIA COMPARTIDA

SINCRONIZACIÓN Y CONTROL DE FLUJO EN GPU

BARRERAS DE SINCRONIZACIÓN



SINCRONIZACIÓN Y CONTROL DE FLUJO EN GPU

CONTROL DE FLUJO

```
if (x < 0.0)
  z = x - 2.0;
else
  z = sqrt(x);
```

SINCRONIZACIÓN Y CONTROL DE FLUJO EN GPU

CONTROL DE FLUJO (PREDICADO)

```
cond: p = (x < 0.0)
p: z = x - 2.0;
!p: z = sqrt(x);
```

SINCRONIZACIÓN Y CONTROL DE FLUJO EN GPU

DIVERGENCIA (GRANULARIDAD INFERIOR AL TAMAÑO DEL WARP)

```
if (threadIdx.x > 2)
  dosomething;
else
  dootherthing;
```

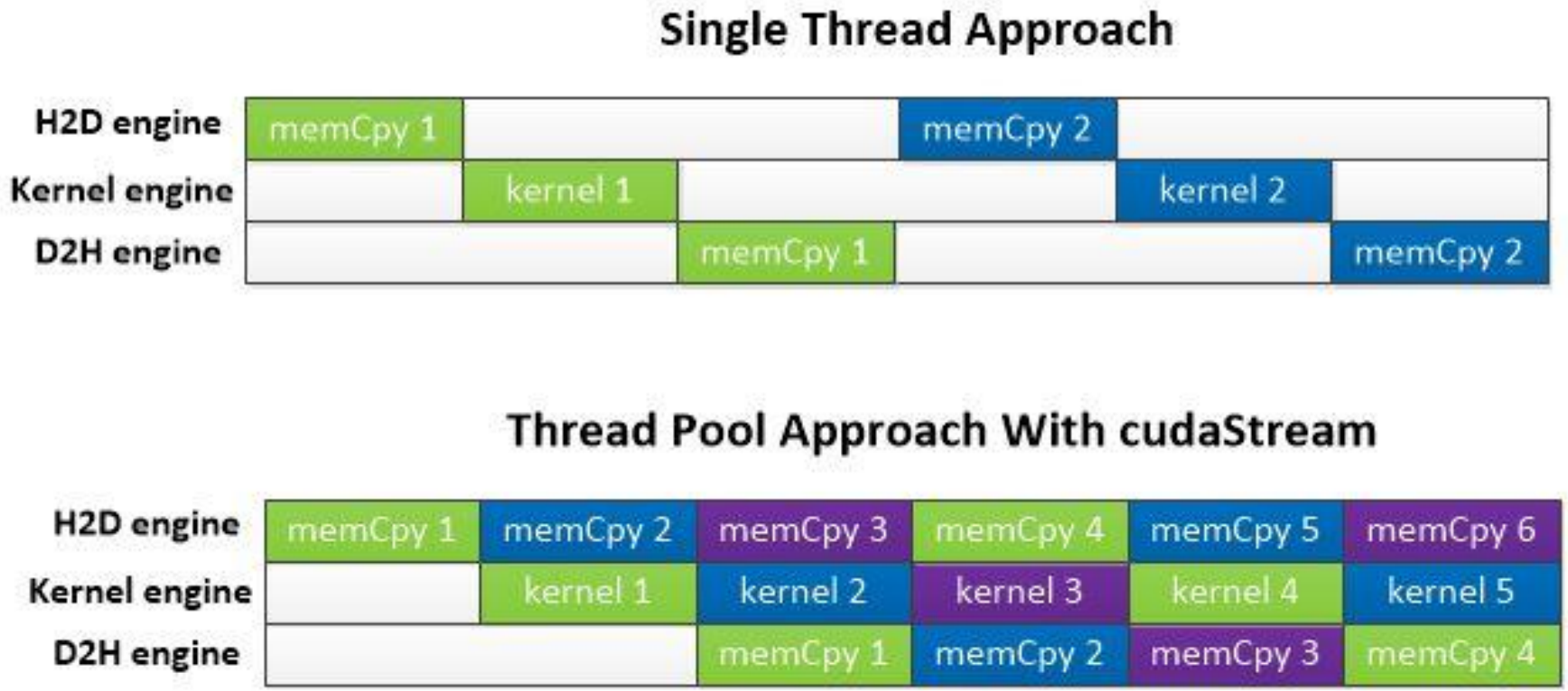
SINCRONIZACIÓN Y CONTROL DE FLUJO EN GPU

NO DIVERGENCIA (GRANULARIDAD MÚLTIPLO DEL TAMAÑO DEL WARP)

```
if (threadIdx.x / WARP_SIZE > 2)
  dosomething;
else
  dootherthing;
```

STREAMS

CONCURRENCIA DE GRANO GRUESO



MEDICIÓN DE TIEMPOS, SINCRONIZACIÓN Y EVENTOS

LANZAMIENTO DE KERNEL ASÍNCRONO Y TRANSFERENCIAS BLOQUEANTES

```
cudaMemcpy(d_x_, h_x_, vector_size_bytes_, cudaMemcpyHostToDevice);
// Host bloqueado hasta finalizar copia
cudaMemcpy(d_y_, h_y_, vector_size_bytes_, cudaMemcpyHostToDevice);
// Host bloqueado hasta finalizer copia

kernel<<<grid, block>>>(d_x_, d_y_, N);
// Host desbloqueado, continua ejecución nada más emitir la orden!

cudaMemcpy(h_y_, d_y_, vector_size_bytes_, cudaMemcpyDeviceToHost);
// Host bloqueado hasta finalizer copia
```

MEDICIÓN DE TIEMPOS, SINCRONIZACIÓN Y EVENTOS

MEDICIÓN DE TIEMPO DE LLAMADA, NO DE EJECUCIÓN

```
cudaMemcpy(d_x_, h_x_, vector_size_bytes_, cudaMemcpyHostToDevice);
// Host bloqueado hasta finalizar copia
cudaMemcpy(d_y_, h_y_, vector_size_bytes_, cudaMemcpyHostToDevice);
// Host bloqueado hasta finalizer copia

unsigned long t_start_ = cpu_timer();
kernel<<<grid, block>>>(d_x_, d_y_, N);
// Host desbloqueado, continua ejecución nada más emitir la orden!
unsigned long t_end_ = cpu_timer();

Unsigned long elapsed_ = t_end_ - t_start_;

cudaMemcpy(h_y_, d_y_, vector_size_bytes_, cudaMemcpyDeviceToHost);
// Host bloqueado hasta finalizar copia
```

MEDICIÓN DE TIEMPOS, SINCRONIZACIÓN Y EVENTOS

MEDICIÓN DE TIEMPO DE EJECUCIÓN FORZANDO SINCRONIZACIÓN CPU-GPU

```
cudaMemcpy(d_x_, h_x_, vector_size_bytes_, cudaMemcpyHostToDevice);
// Host bloqueado hasta finalizar copia
cudaMemcpy(d_y_, h_y_, vector_size_bytes_, cudaMemcpyHostToDevice);
// Host bloqueado hasta finalizer copia

unsigned long t_start_ = cpu_timer();
kernel<<<grid, block>>>(d_x_, d_y_, N);
// Host desbloqueado, continua ejecución nada más emitir la orden!
cudaDeviceSynchronize();
// Host bloqueado hasta que la GPU termine el último comando
unsigned long t_end_ = cpu_timer();

Unsigned long elapsed_ = t_end_ - t_start_;

cudaMemcpy(h_y_, d_y_, vector_size_bytes_, cudaMemcpyDeviceToHost);
// Host bloqueado hasta finalizar copia
```

MEDICIÓN DE TIEMPOS, SINCRONIZACIÓN Y EVENTOS

MEDICIÓN DE TIEMPO DE EJECUCIÓN SIN BLOQUEAR NI CPU NI GPU

```
cudaEvent_t t_start_, t_stop_;
cudaEventCreate(&t_start_);
cudaEventCreate(&t_stop_);

cudaMemcpy(d_x_, h_x_, vector_size_bytes_, cudaMemcpyHostToDevice);
cudaMemcpy(d_y_, h_y_, vector_size_bytes_, cudaMemcpyHostToDevice);

cudaEventRecord(t_start_);
kernel<<<grid, block>>>(d_x_, d_y_, N);
cudaEventRecord(t_stop_);

cudaMemcpy(h_y_, d_y_, vector_size_bytes_, cudaMemcpyDeviceToHost);
cudaEventSynchronize(t_stop_);
float milliseconds_ = 0.0f;
cudaEventElapsedTime(&milliseconds_, t_start_, t_stop_);
```

MODELO DE PROCESAMIENTO

DISPOSITIVOS E INFRAESTRUCTURAS PARA SISTEMAS MULTIMEDIA

GRADO EN INGENIERÍA MULTIMEDIA 2018-2019

Albert García-García < agarcia@dtic.ua.es >

Jose García-Rodríguez < jgarcia@dtic.ua.es>