FACULTAD DE CIENCIAS FISICAS Y MATEMATICAS
UNIVERSIDAD DE CHILE

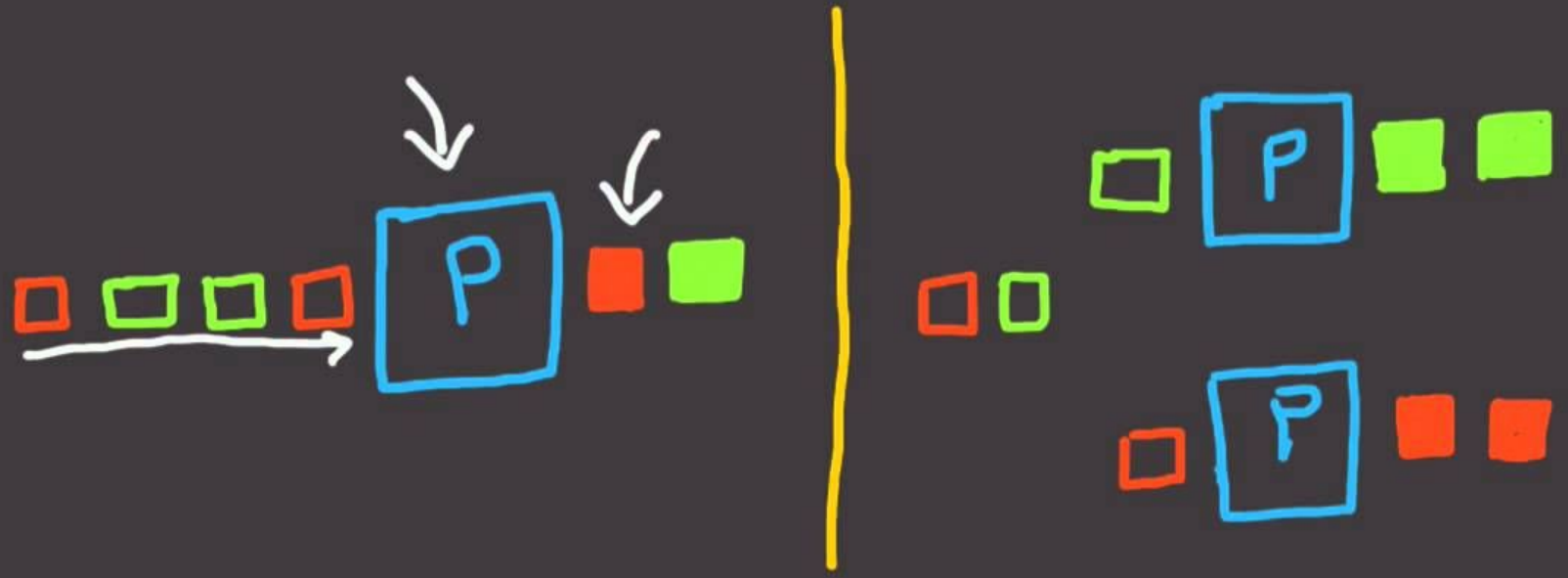
Programación en OpenCL

Joaquín Torres Paris

¿Qué es la programación paralela?

Programa que ejecuta una serie de operaciones en paralelo (al mismo tiempo)

Parallel Computing



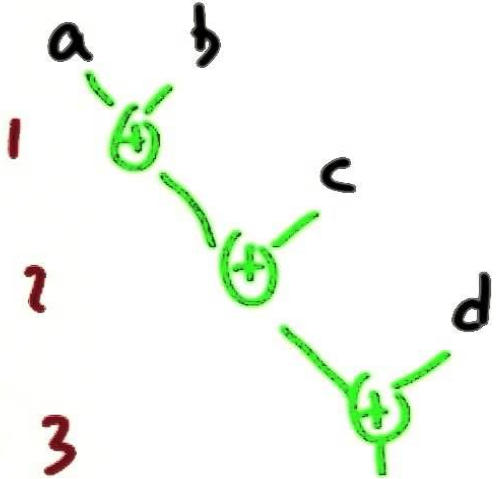
¿Qué es la programación paralela?

Puede convertir la ejecución a una no determinista.

Toma ventaja de las arquitecturas actuales.

Hay que mirar los problemas desde otro punto de vista.

Serial Reduce

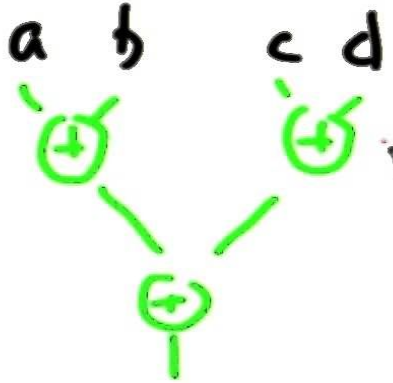


3 work

steps

$((a+b)+c)+d$

Parallel Reduce



3 work

steps

$(a+b)+(c+d)$

¿Qué ganamos con el paralelismo?

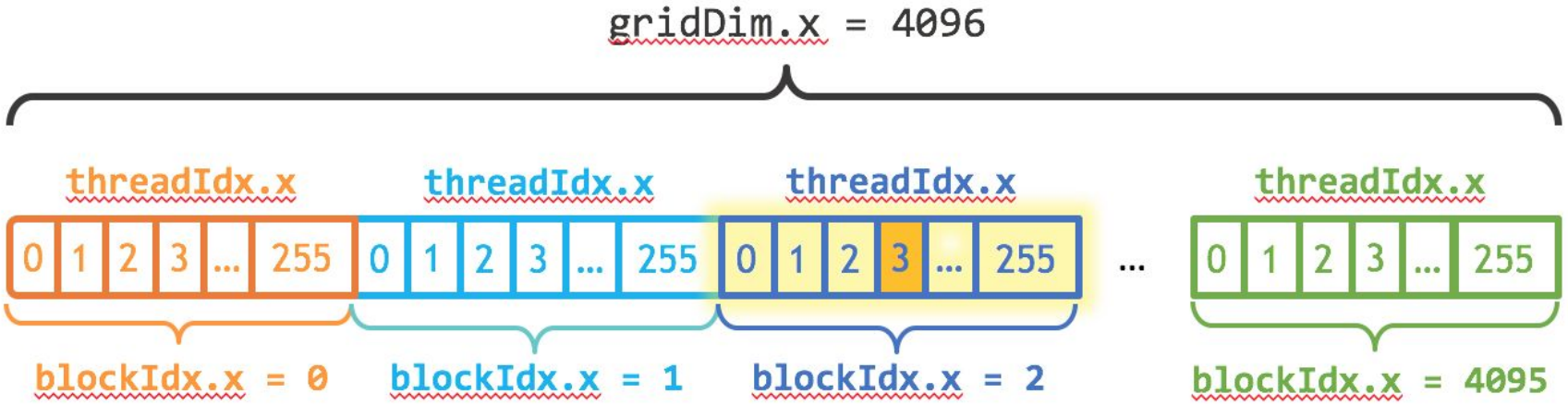
Una solución más rápida.

Resolver problemas más grandes.

Uso efectivo de los recursos del computador.

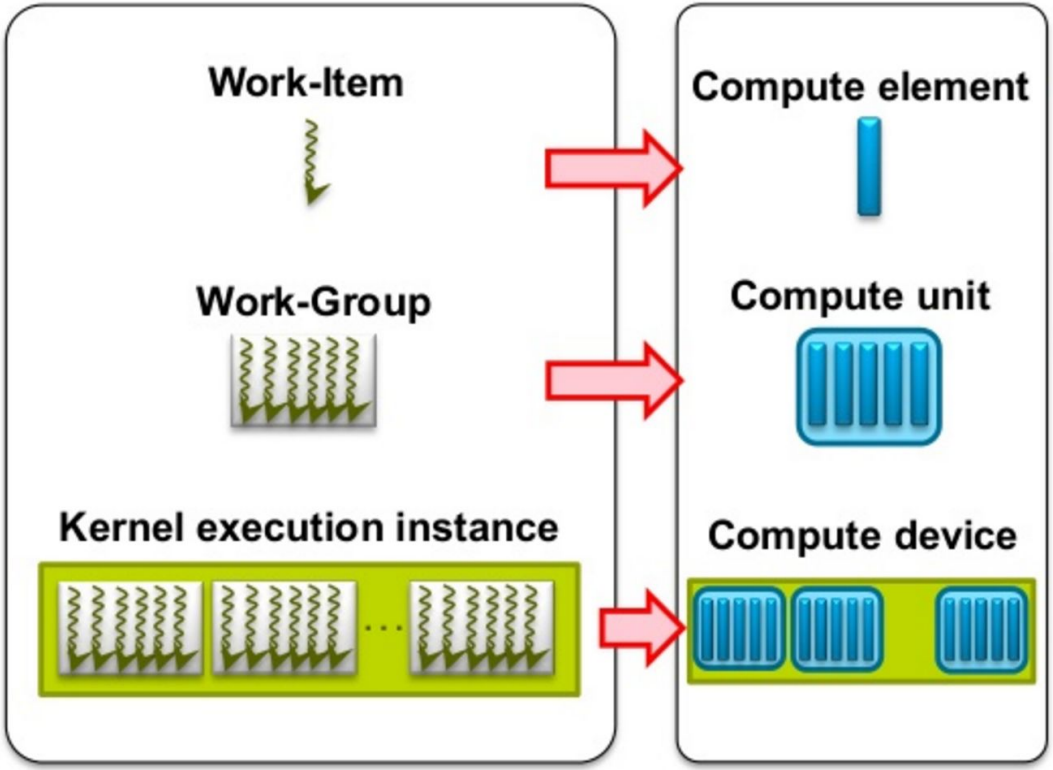
Programación en GPU

Threads & Blocks




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

index =    (2)      *   (256)    +     (3)      = 515
```

Threads & Blocks



- Each **work-item** is executed by a **compute element**
- Each **work-group** is executed on a **compute unit**
- Several concurrent **work-groups** can reside on one **compute unit** depending on work-group's memory requirements and compute unit's memory resources
- Each **kernel** is executed on a **compute device**

Keywords

__global

__kernel

__local

__read_only

__constant

https://www.khronos.org/registry/OpenCL/specs/opencl-2.0-openclc.pdf

Keywords

```
kernel void my_func(…)
{
      local float      a;    // A single float allocated
                             // in local address space

      local float      b[10];// An array of 10 floats
                             // allocated in local address space.

      if (…)
      {
          // example of variable in __local address space but not
          // declared at __kernel function scope.
          local float c;    ← not allowed.
      }
}
```

https://www.khronos.org/registry/OpenCL/specs/opencl-2.0-openclc.pdf

Funciones para work-items

`get_global_id`

`get_local_id`

`get_work_dim`

Funciones para work-items

```
kernel void bar(global int *g, local int *l)
{
    int *var;
    if (is_even(get_global_id(0))
        var = g;
    else
        var = l;
    *var = 42;
    …
}
```

```
local atomic_int local_guide;
if (get_local_id(0) == 0)
  atomic_init(&guide, 42);
```

Funciones atómicas

atomic_add

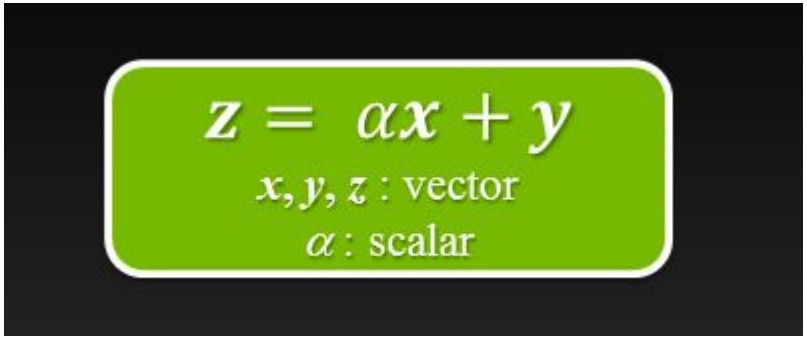
atomic_max

atomic_or

atomic_inc

```
__global uint *counter;
*counter = 0;          //initialize variable with zero
uint old_val = atomic_inc( counter ); //make atomic increment on it
```

Single Precision A□X Plus Y



Ejemplo OpenCL

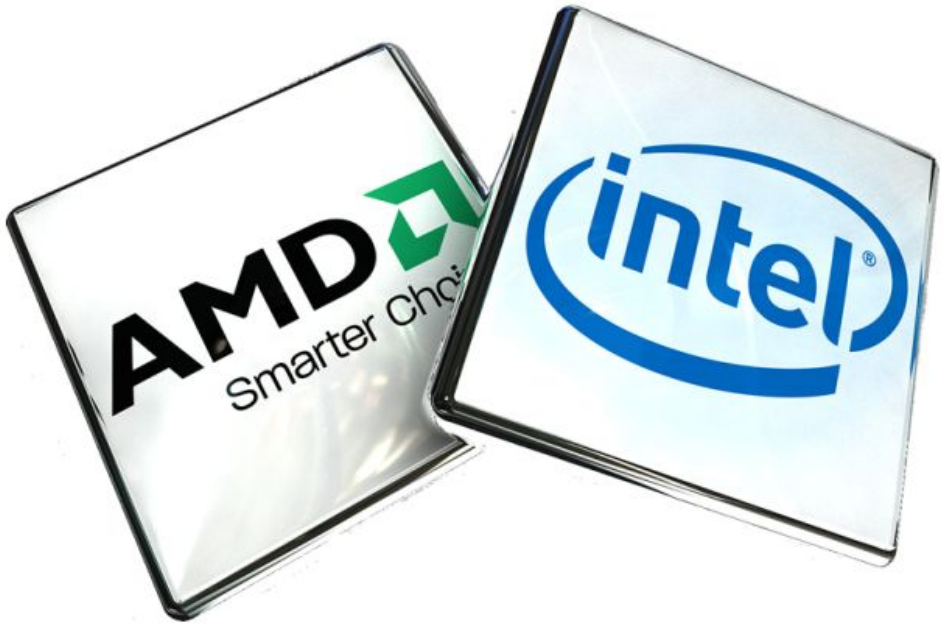
```
__kernel void SAXPY (__global float* x,
__global float* y, float a)
{
    const int i = get_global_id (0);

    y [i] = a * x [i] + y [i];
}
```

¿Cómo incializar OpenCL?

Headers

```
#ifdef __APPLE__
    #include "OpenCL/opencl.h"
#else
    #include "CL/cl.h"
#endif
```

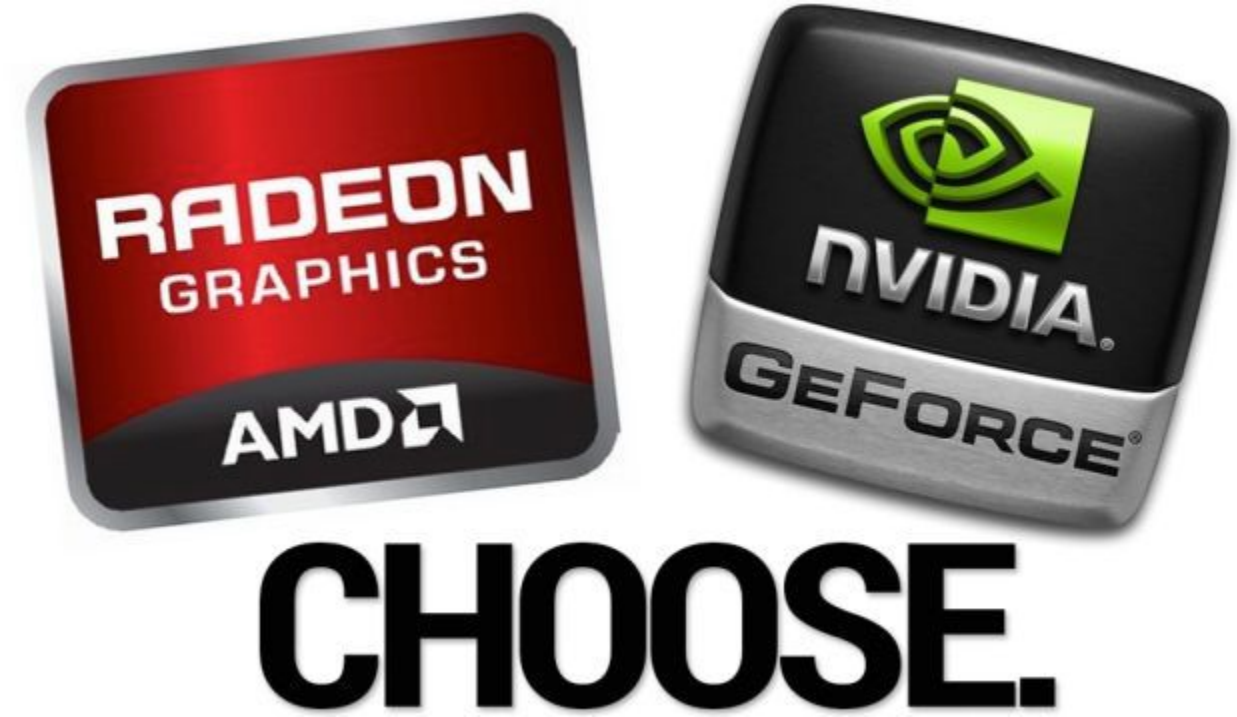
Inicializar Plataforma



Inicializar Plataforma

```
cl_uint platformIdCount = 0;
clGetPlatformIDs (0, nullptr, &platformIdCount);

std::vector<cl_platform_id> platformIds (platformIdCount);
clGetPlatformIDs (platformIdCount, platformIds.data (), nullptr);
```

Inicializar Dispositivo



Inicializar Dispositivo

```
cl_uint deviceIdCount = 0;
clGetDeviceIDs (platformIds [0], CL_DEVICE_TYPE_ALL, 0,
    nullptr, &deviceIdCount);
std::vector<cl_device_id> deviceIds (deviceIdCount);
clGetDeviceIDs (platformIds [0], CL_DEVICE_TYPE_ALL,
    deviceIdCount, deviceIds.data (), nullptr);
```

Inicializar Contexto

```
const cl_context_properties contextProperties [] =
{
    CL_CONTEXT_PLATFORM,
    reinterpret_cast<cl_context_properties> (platformIds [0]),
    0, 0
};

cl_context context = clCreateContext (
    contextProperties, deviceIdCount,
    deviceIds.data (), nullptr,
    nullptr, &error);
```

Crear Buffer para copiar los datos

```
cl_mem aBuffer = clCreateBuffer (context,
    CL_MEM_READ_ONLY |
CL_MEM_COPY_HOST_PTR,
    sizeof (float) * (testDataSize),
    a.data (), &error);
CheckError (error);
```

Compilar programa

```
CheckError (clBuildProgram (program,
deviceIdCount,
    deviceIds.data (), nullptr, nullptr,
nullptr));

cl_kernel kernel = clCreateKernel (program,
"SAXPY", &error);
CheckError (error);
```

Kernel

```
__kernel void SAXPY (__global float* x, __global float* y,
float a)
{
    const int i = get_global_id (0);

    y [i] += a * x [i];
}
```

Entregar Parámetros

```
clSetKernelArg (kernel, 0, sizeof (cl_mem),
&aBuffer);
clSetKernelArg (kernel, 1, sizeof (cl_mem),
&bBuffer);
static const float two = 2.0f;
clSetKernelArg (kernel, 2, sizeof (float), &two);
```

Encolarlos a la ejecución

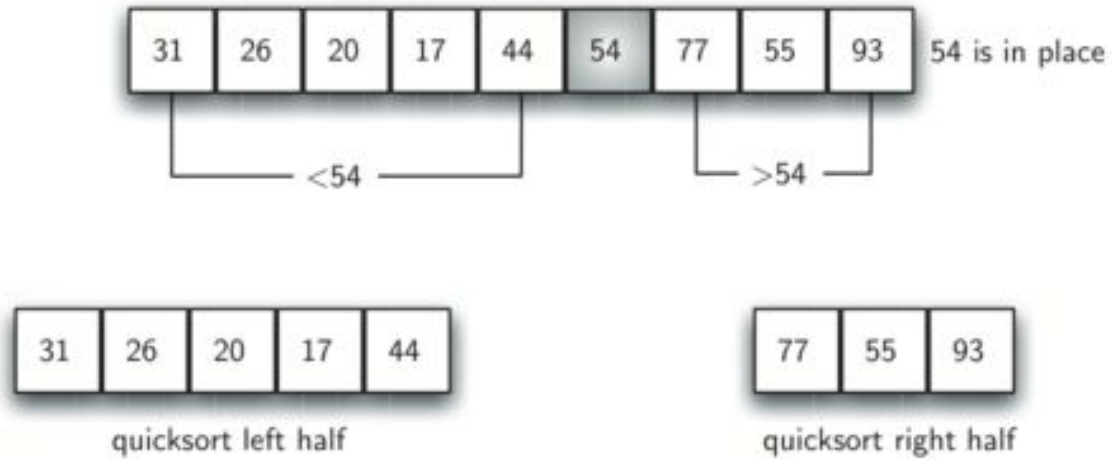
```
const size_t globalWorkSize [] = {
testDataSize, 0, 0 };
CheckError (clEnqueueNDRangeKernel (queue,
    kernel,
    1, // One dimension
    nullptr,
    globalWorkSize,
    nullptr,
    0, nullptr, nullptr));
```

Leer resultados

```
err = clEnqueueReadBuffer( queue,
    output,
    CL_TRUE,
    0,
    sizeof(float) * count,
    results,
    0, NULL, NULL );
```

GPU Quicksort

Quicksort


31
26
20
17
44
54
77
55
93
54 is in place
<54
>54
31
26
20
17
44
quicksort left half
77
55
93
quicksort right half

¿Cómo paralelizamos esto?

Quicksort: Aproximación 1

Por cada paso generado por quicksort paralelizar en dos threads cada arreglo resultante.

Quicksort: Aproximación 1

Por cada paso generado por quicksort paralelizar en dos threads cada arreglo resultante.

INEFICIENTE, perdemos mucho tiempo de cómputo en buscar elementos en la memoria.

Quicksort: Aproximación 2

Dividir el arreglo por la cantidad de bloques. Y generar quicksort paralelo por bloque.

Quicksort: Aproximación 2

Dividir el arreglo por la cantidad de bloques. Y generar quicksort paralelo por bloque.

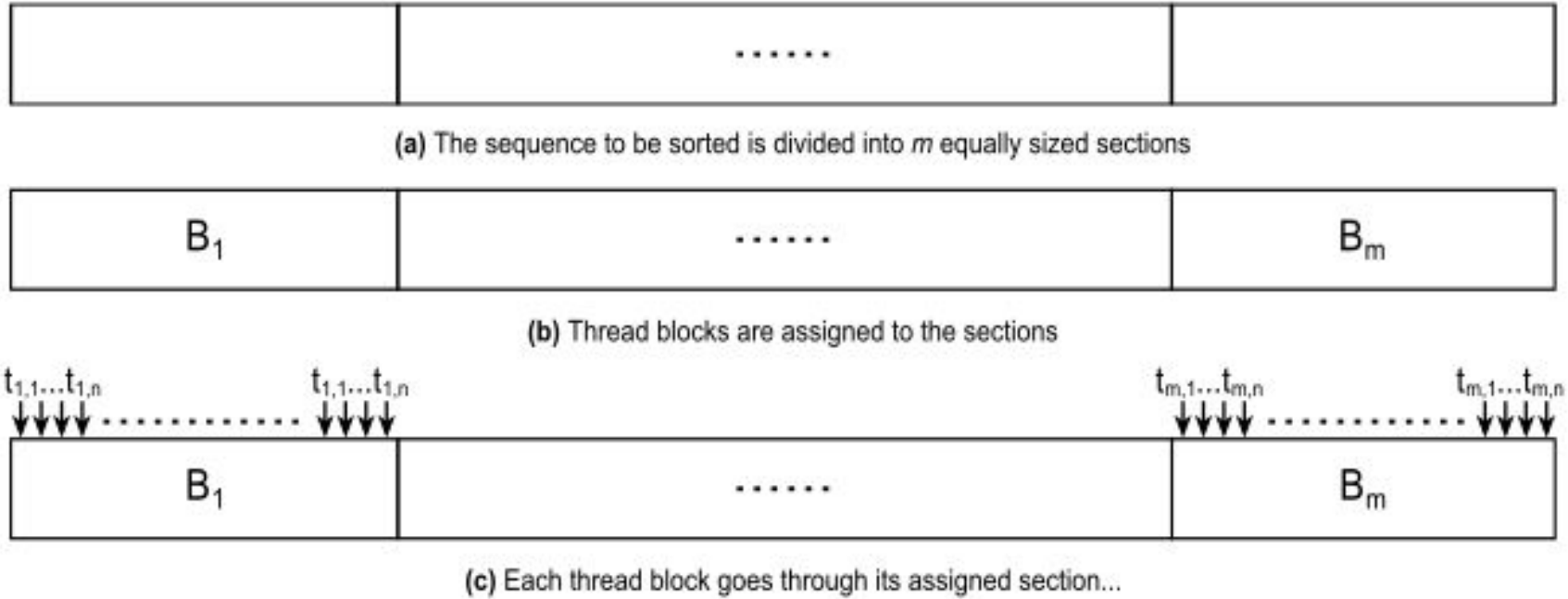
Este podría funcionar, pero como sincronizamos los threads dentro de un bloque.

GPU-Quicksort

Dividir el tamaño del problema al tamaño de la memoria de un bloque.

Cada bloque ejecuta quicksort en su segmento de arreglo.

GPU-Quicksort



(a) The sequence to be sorted is divided into *m* equally sized sections

(b) Thread blocks are assigned to the sections

(c) Each thread block goes through its assigned section...

GPU-Quicksort

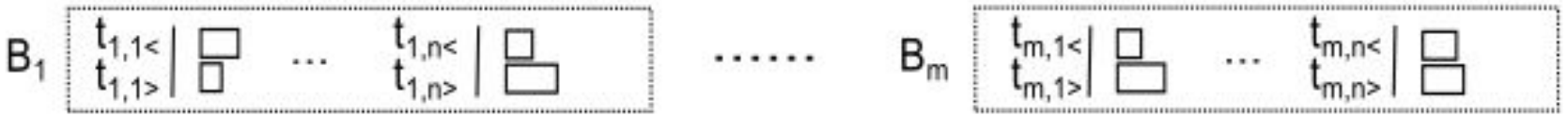
El problema viene a cómo sincronizar los threads dentro de un bloque para que no quede un arreglo en estado inconsistente.

GPU-Quicksort

Se suma la cantidad de threads que suman a un lado.

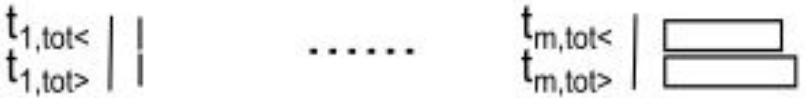
La posición del thread dentro de los de un lado.

GPU-Quicksort



...keeping track of the number of elements above and below the pivot

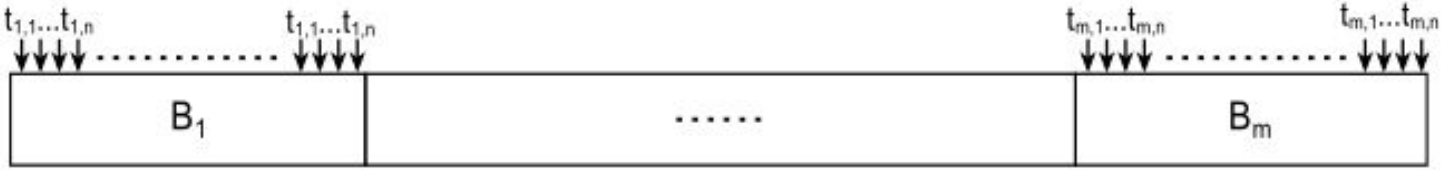
(d) Each thread block calculates the cumulative sum of the two types of elements seen



(e) A cumulative sum of each thread blocks total is calculated

GPU-Quicksort

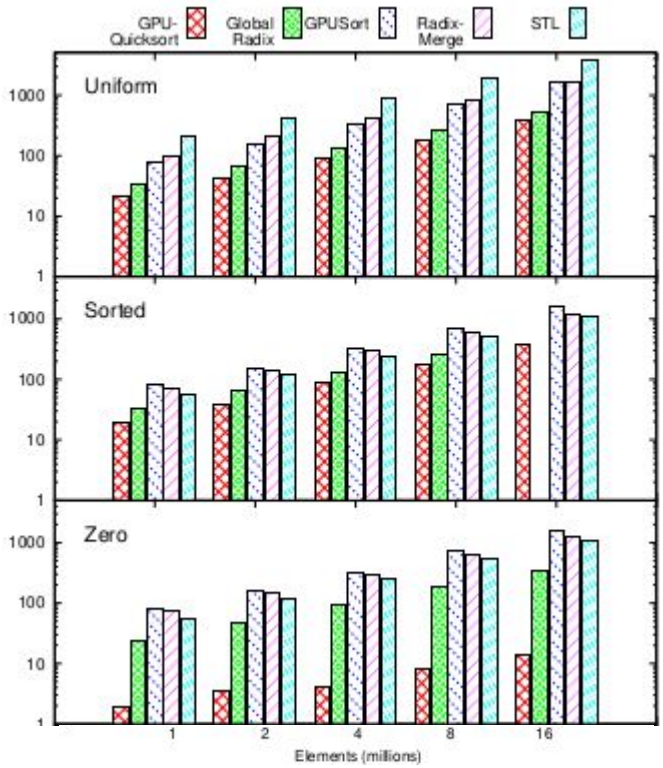
(f) Each thread uses the cumulative sums to find out where to write



(g) Each thread block goes through its assigned data, writing it to the auxiliary array

(h) Block B_m fills the gap between the left and right subsequence with the pivot value

GPU-Quicksort



GPU-Quicksort

Explicación del algoritmo:

http://www.cse.chalmers.se/~tsigas/papers/GPU-Quicksort-jea.pdf

Implementación de QuickSort en CUDA:

https://github.com/khaman1/GPU-QuickSort-Algorithm

FACULTAD DE CIENCIAS FISICAS Y MATEMATICAS
UNIVERSIDAD DE CHILE

Programación en OpenCL

Joaquín Torres Paris