ICNPG 2023

Clase 5: librería Thrust

nvprof

<<<G, B,M>>>

CudaGetDeviceProp

__device__

reduction

grillas

kernels

hilos

blockDim

Paralelismo

syncthreads

Host

CudaMemCopy

punteros

Nvidia

__global__

¿Preguntas de la clase 1,2,3?

Device

Shared-memory

bloques

CUDA

performances

blockId

dim3

threadId

__host__

nvcc

CudaMalloc

gridDim

cudaFree

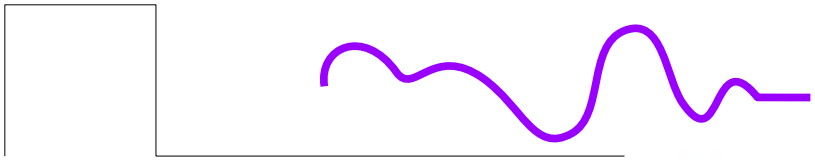
CPU-RAM

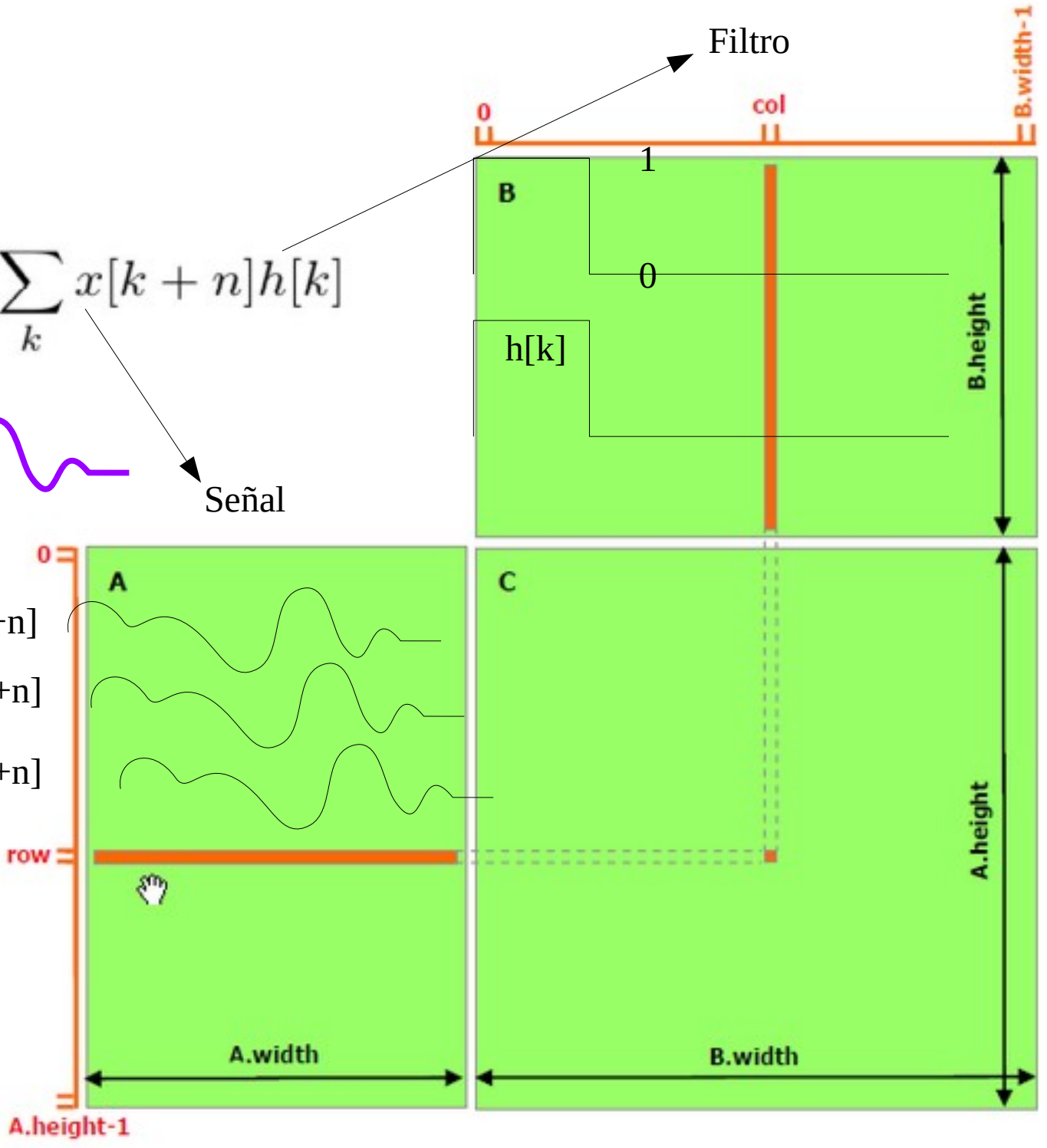
GPU-RAM

cudaDeviceSynchronize

Convolución

$$y[n] = [x * h][n] = \sum_k x[k+n]h[k]$$



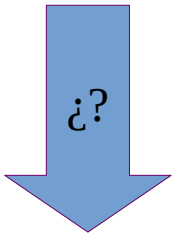


Tarea de precalentamiento

- Escribir el kernel correspondiente a esta convolución
  - ¡Y que ande para cualquier grilla!

$$y[n] = [x * h][n] = \sum_k x[k+n]h[k]$$

```
/* convolucion en la cpu: requiere dos loops */
void conv_sec(FLOAT* input, FLOAT* output, FLOAT * filter)
{
        FLOAT temp;
        for(int j=0;j<N;j++){
                temp=0.0;
                for(int i=0;i<M;i++){
                        temp += filter[i]*input[i+j];
                }
                output[j] = temp;
        }
}
```

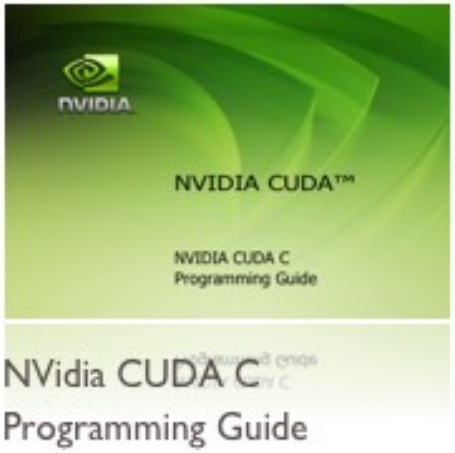


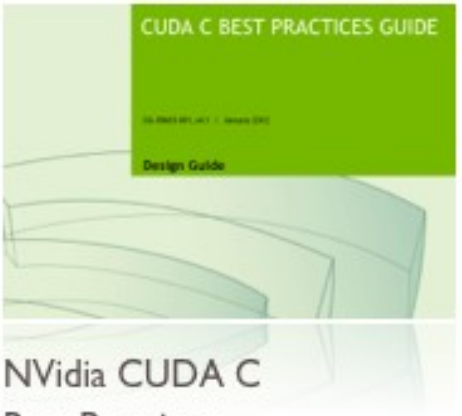

```
// kernel
__global__ void conv_par(FLOAT* input, FLOAT* output, FLOAT* filter)
{
        //TODO: completar el kernel de convolucion
}
```

Hasta aquí...

CUDA C/C++

- Control de “bajo” nivel del mapeo al hardware
- Desarrollo de algoritmos de alta performance.
- cálculos → ejecución en hilos en el device

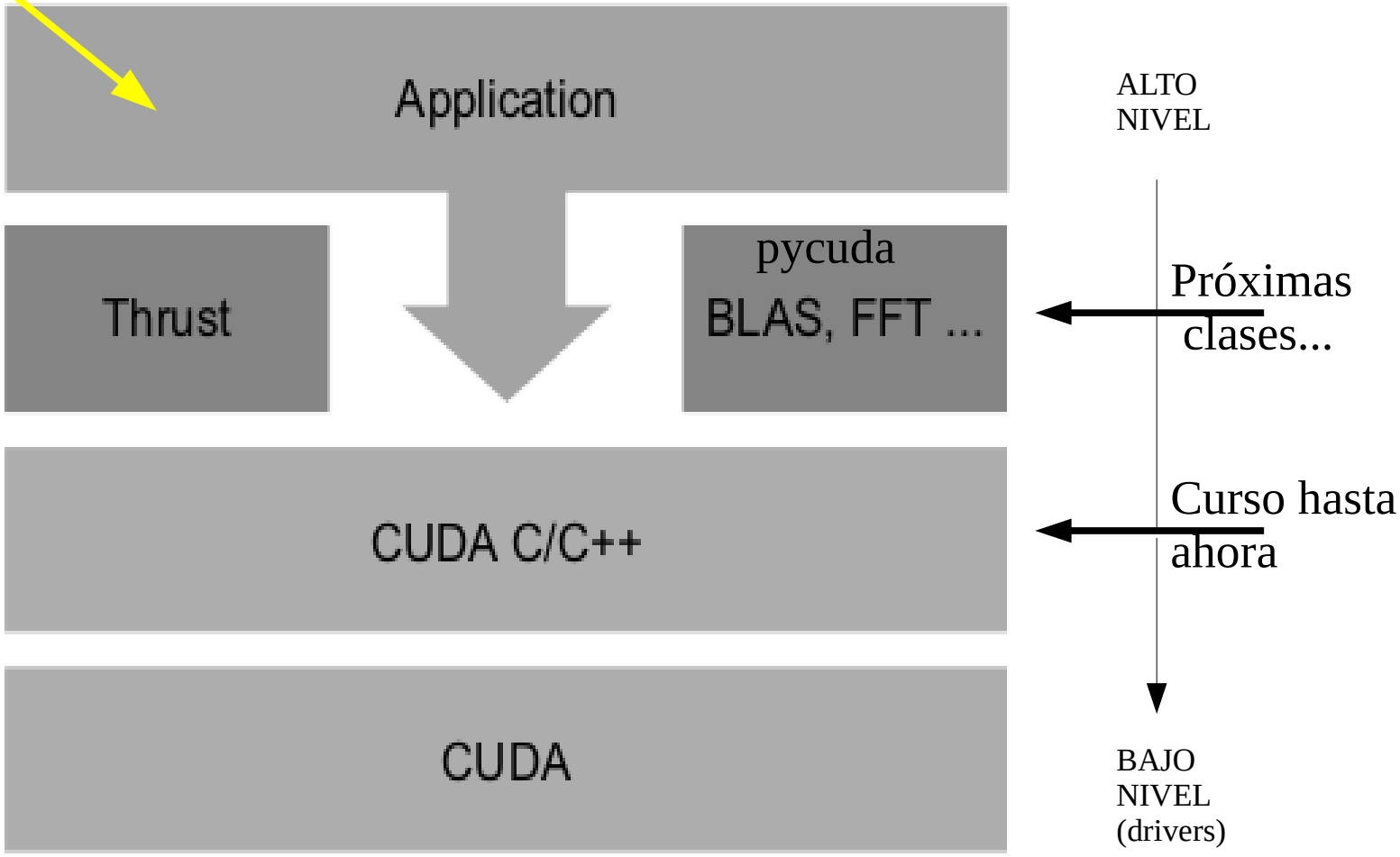




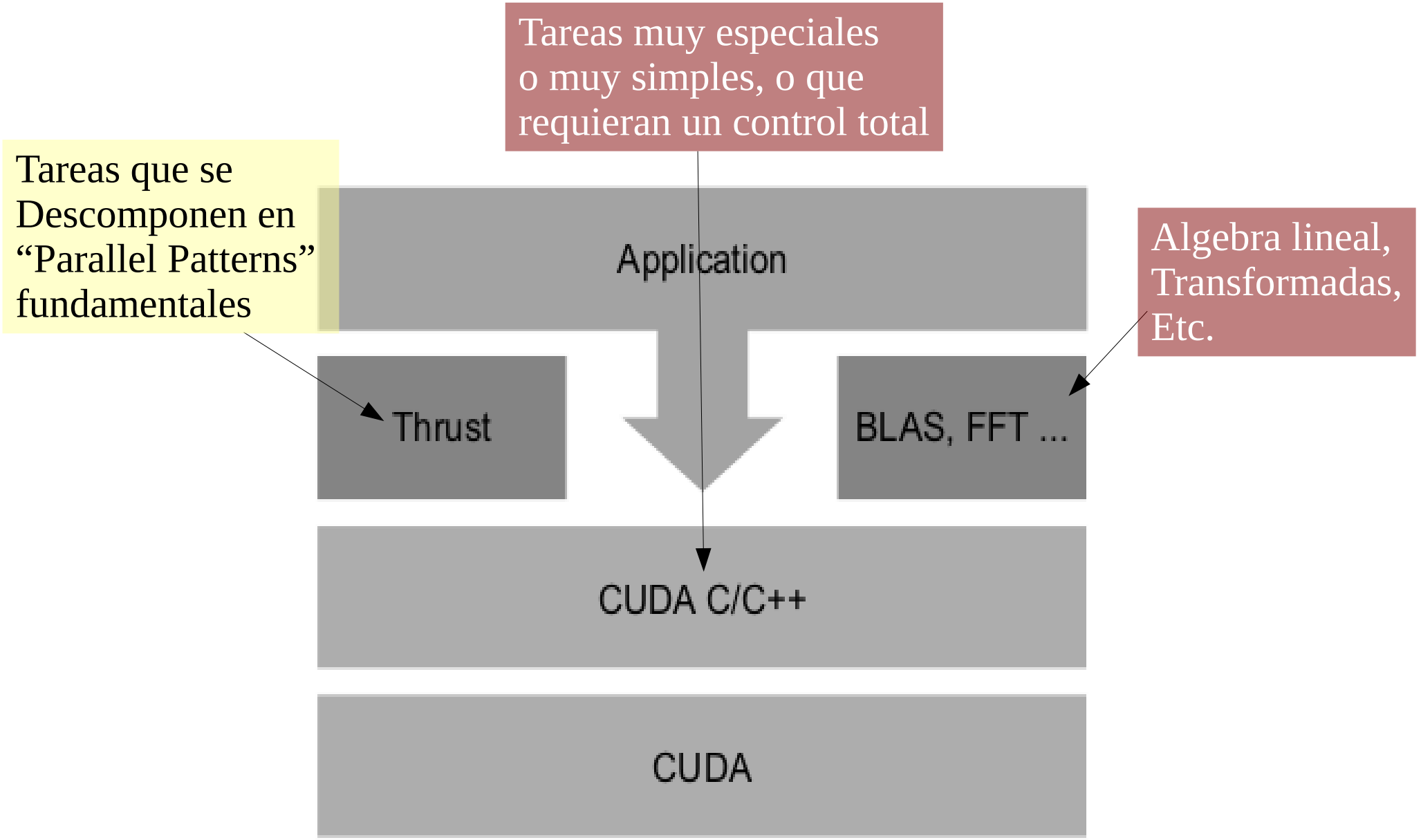

```
__global__ void mi_kernel(...)
{
  int i = ???
  // y aqui que hago???
}

int main(){
// alocacion de memoria, etc...
dim3 G(???); dim3 B(???);
mi_kernel<<<G,B>>>(....);
}
```

Problema a Resolver Numerica y concurrentemente

Librerías para Cuda



Podemos “delegar” ciertas implementaciones...



***“Productividad de codigo”*:**

- “No reinventar la rueda”, delegar ciertas implementaciones de *tareas comunes* a libs optimizadas.
- Dedicar nuestro tiempo/esfuerzo/creatividad un poco menos al “*como*” *y mas al “que”* calcular ... (reducir el tiempo de programacion)

¿ Que es Thrust ?

http://thrust.github.io/

excelente documentación, muchos ejemplos

¿Necesito Thrust?

¿ Para que/quien es ?

https://developer.nvidia.com/content/expressive-algorithmic-programming-thrust

Parte del actual toolkit de cuda

https://developer.nvidia.com/thrust

Thrust facilita la escritura de código en CUDA C/C++
(ver “CUDA Application Design and Development” de Rob Farber)

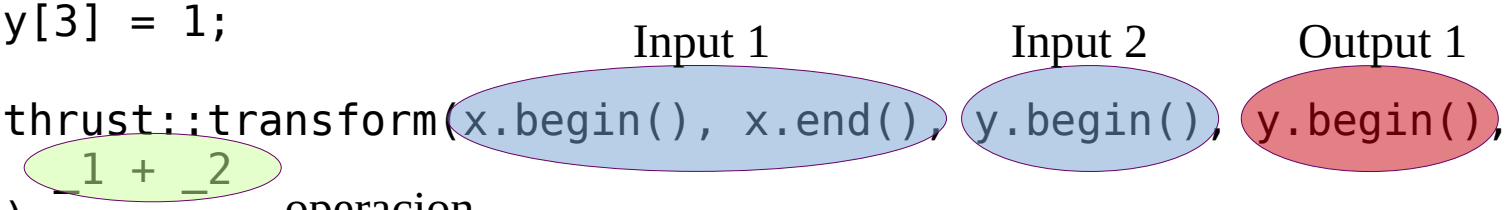
Buena noticia para los que usan C++ y la Standard Template Library
→ Ya saben Thrust

Suma de arrays paralela en la GPU

suma.cu

```
#include <thrust/device_vector.h>
#include <thrust/transform.h>
using namespace thrust::placeholders;
int main()
{
  thrust::device_vector<float> x(4), y(4);
  x[0] = 1;
  x[1] = 2;
  x[2] = 3;
  x[3] = 4;

  y[0] = 4;
  y[1] = 3;
  y[2] = 2;
  y[3] = 1;

  thrust::transform(x.begin(), x.end(), y.begin(), y.begin(),
    _1 + _2
  );
  // y[] es ahora {5, 5, 5, 5} --> HANDS-ON: comprobar!
}
```

- CudaMalloc?, CudaFree?
- CudaMemCpy?
- Kernel?



Combinación lineal de arrays

saxpy.cu

```
#include <thrust/device_vector.h>
#include <thrust/transform.h>
using namespace thrust::placeholders;

int main()
{
  thrust::device_vector<float> x(4), y(4);
  x[0] = 2;
  x[1] = 4;
  x[2] = 6;
  x[3] = 8;

  y[0] = 4;
  y[1] = 3;
  y[2] = 2;
  y[3] = 1;
  float a=0.5;


  thrust::transform(x.begin(), x.end(), y.begin(), y.begin(),
    a*_1 + _2
  );
  // y[] ahora es {5, 5, 5, 5} --> HANDS-ON: comprobar!
}
```

Transformación arbitraria de dos arrays

transforma.cu

```
#include <thrust/device_vector.h>
#include <thrust/transform.h>

struct mi_operacion
{
  __device__
  float operator()(float a, float b)
  {
    return sqrt(a+b);
  }
};
int main()
{

  thrust::device_vector<float> x(4), y(4);
  x[0] = 1; y[0] = 4; // → sqrt(1+4)=sqrt(5)=2.23606797749979
  X[1] = 2; y[1] = 3;
  x[2] = 3; y[2] = 2;
  X[3] = 4; y[3] = 1;

  thrust::transform(x.begin(), x.end(), y.begin(), y.begin(),mi_operacion());
  // HANDS-ON: da bien?
}
```

Transformacion binaria:
2 arrays → 1 array
(generalizable a N→ M arrays)

Transformación arbitraria de dos arrays

transforma_lambda.cu

```
#include <thrust/device_vector.h>
#include <thrust/transform.h>
#include <cmath>
#include <iostream>

int main()
{
  thrust::device_vector<float> x(4), y(4);
  x[0] = 1;
  x[1] = 2;
  x[2] = 3;
  x[3] = 4;

  y[0] = 4;
  y[1] = 3;
  y[2] = 2;
  y[3] = 1;

  thrust::transform(x.begin(), x.end(), y.begin(), y.begin(),
        [=]  __device__ (float x1,float x2)
        {
                return sqrt(x1+x2);
        }
  );

  for(int i=0;i<4;i++)
  std::cout << y[i] << std::endl;
}
```

In C++, lambda functions are a way to define a small anonymous function inline, without giving it a name. They are essentially a shorthand way of defining a function object, which can be used like any other object in C++.

Función lambda
[=]__device__(){….}

[capture list] (parameter list) -> return type { function body }

nvcc -O2 --expt-extended-lambda -std=c++11 -o transforma_lambda transforma_lambda.cu

Normalización de un array

normaliza.cu

```
#include <thrust/device_vector.h>
#include <thrust/transform.h>
#include <thrust/reduce.h>

struct mi_operacion // normaliza por N
{
  float Norm;
  mi_operacion(float suma){Norm=suma;};
  __device__ float operator()(float a)
  {
  return a/Norm;
  }
};

int main()
{
  thrust::device_vector<float> x(4), y(4);
  x[0] = 2;
  x[1] = 4;
  x[2] = 3;
  x[3] = 1;

  float suma = thrust::reduce(x.begin(), x.end());
  thrust::transform(x.begin(), x.end(), x.begin(),mi_operacion(suma));
  // HANDS-ON: chequear que este normalizado...
}
```

Reduccion seguida por una Transformacion unaria:
1 array → 1 array

Normalización de un array

normaliza2.cu

```
#include <thrust/device_vector.h>
#include <thrust/transform.h>
#include <thrust/reduce.h>
using namespace thrust::placeholders;

int main()
{
  thrust::device_vector<float> x(4), y(4);
  x[0] = 2;
  x[1] = 4;
  x[2] = 3;
  x[3] = 1;

  float suma = thrust::reduce(x.begin(), x.end());

  thrust::transform(x.begin(), x.end(), x.begin(),_1/suma);
  // HANDS-ON: chequear que este bien normalizado...
}
```

Reduccion seguida por una Transformacion unaria:
1 array → 1 array
Uso de placeholders...

Normalización de un array

normaliza3.cu

```
#include <thrust/device_vector.h>
#include <thrust/transform.h>
#include <thrust/reduce.h>

__global__
void kernel_normaliza(float *x, float suma, int dim)
{
    int id = threadIdx.x + (blockIdx.x * blockDim.x);
    if (id < dim){
         x[id] = x[id]/suma;
    }
}
int main()
{
  thrust::device_vector<float> x(4);
  x[0] = 2;
  x[1] = 4;
  x[2] = 3;
  x[3] = 1;
  float suma = thrust::reduce(x.begin(), x.end());

  float * x_ptr = thrust::raw_pointer_cast(&x[0]);
  kernel_normaliza<<<1,4>>>(x_ptr,suma,4);
  // HANDS-ON: chequear que este normalizado...
}
```

Interoperabilidad

Normalización de un array

normaliza4.cu

```
#include <thrust/device_ptr.h>
#include <thrust/device_free.h>
#include <thrust/transform.h>
#include <thrust/reduce.h>
using namespace thrust::placeholders;
int main(void)
{
  float *raw_ptr;
  cudaMalloc((void **)&raw_ptr, 4*sizeof(float));
  thrust::device_ptr<float> x(raw_ptr);

  x[0] = 2;  x[1] = 4;  x[2] = 3;  x[3] = 1;

  float suma = thrust::reduce(x, x+4);
  thrust::transform(x,x+4,x,_1/suma);// en CPU o GPU?

  // HANDS-ON: chequear que este bien normalizado...

  thrust::device_free(x);
  return 0;
}
```

Normalización de un array

(por su norma, no por la suma...)

normaliza5.cu

```
#include <thrust/device_vector.h>
#include <thrust/transform.h>
#include <thrust/transform_reduce.h>
#include <thrust/reduce.h>

using namespace thrust::placeholders;

int main()
{
  thrust::device_vector<float> x(4);
  X[0] = 2; x[1] = 4; x[2] = 3; x[3] = 1;

  // vamos a normalizar un vector por su norma

  #ifdef INEFICIENTE
  //forma 1:
  thrust::device_vector<float> xx(4);
  thrust::transform(x.begin(), x.end(),xx.begin(),_1*_1);
  float norma = sqrt(thrust::reduce(xx.begin(),xx.end()));
  thrust::transform(x.begin(), x.end(), x.begin(),_1/norma);

  #else

  //forma 2:
  float norma = sqrt(thrust::transform_reduce(x.begin(),
x.end(),_1*_1,0,thrust::plus<float>()));
  thrust::transform(x.begin(), x.end(), x.begin(),_1/norma);

  #endif
}
```

norma vive en GPU o CPU?

Si es en CPU se explicaría por qué la forma 2 es más rápida, no?

Interoperabilidad

Pantallazo de thrust

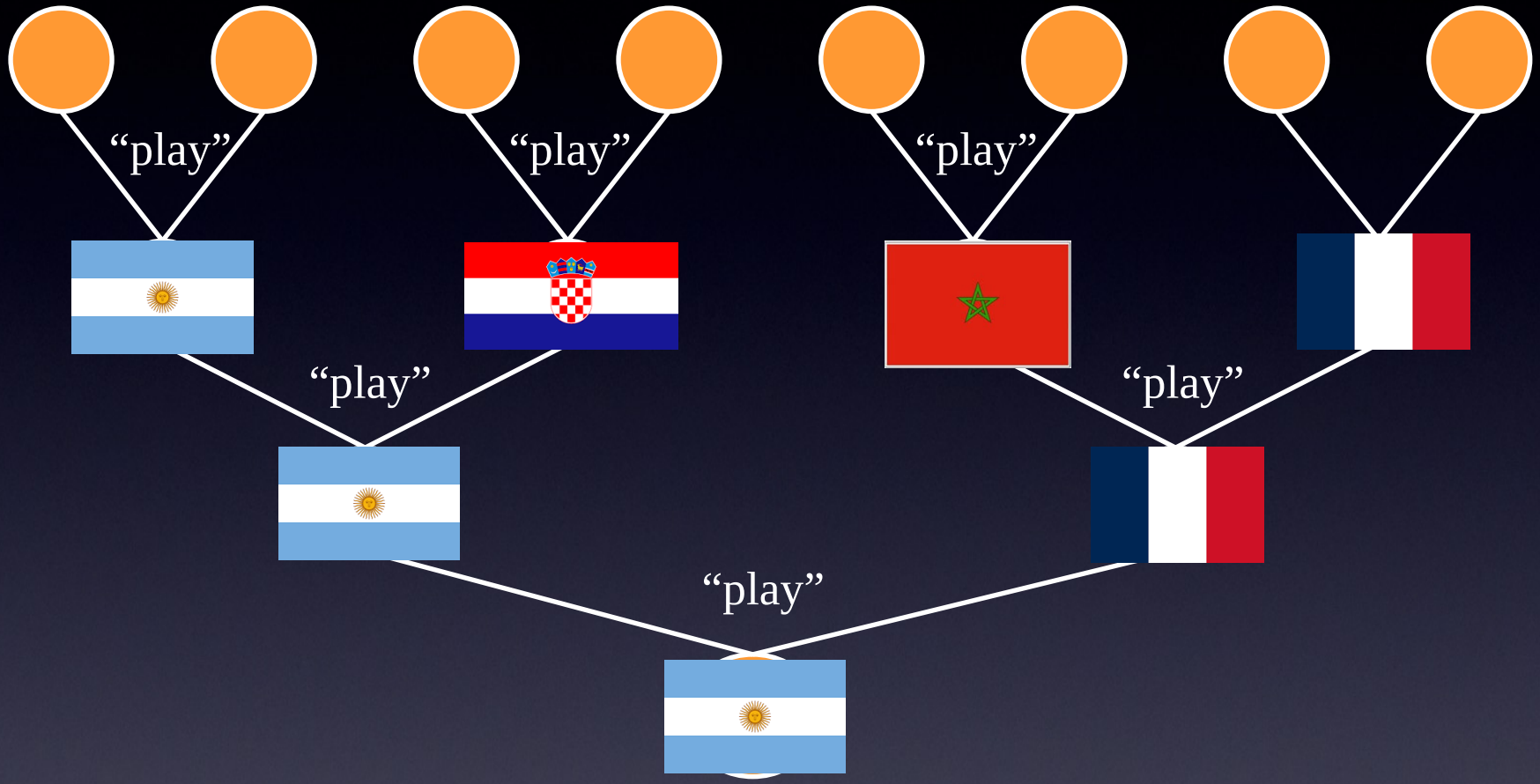
http://thrust.github.io/

```
//#includes...
int main(void)
{
  // generate 32M random numbers serially
  thrust::host_vector<int> h_vec(32 << 20);
  std::generate(h_vec.begin(), h_vec.end(), rand);

  // transfer data to the device
  thrust::device_vector<int> d_vec = h_vec;

  // sort data on the device (846M keys per second on GeForce GTX 480)
  thrust::sort(d_vec.begin(), d_vec.end());

  // transfer data back to host
  thrust::copy(d_vec.begin(), d_vec.end(), h_vec.begin());

  return 0;
}
```

¿ Que tipo de algoritmos tiene Thrust ?

“Parallel building blocks”

Copa del Mundo de Futbol: 32 equipos → 16 a octavos de final



Calcula el *campeón*

Vale para int, float, double,o cualquier tipo de dato con la operación “play” bien definida...

Reducción genérica en Thrust

```
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>
#include <thrust/generate.h>
#include <thrust/reduce.h>
#include <thrust/functional.h>
#include <cstdlib>

int main(void)
{
  // generate random data on the host
  thrust::host_vector<int> h_vec(100);
  thrust::generate(h_vec.begin(), h_vec.end(), rand);

  // transfer to device and compute sum
  thrust::device_vector<int> d_vec = h_vec;
  int x = thrust::reduce(d_vec.begin(), d_vec.end(), 0, thrust::plus<int>());
  return 0;
}
```

Cualquier operacion binaria asoc. y conmutativa (+,-,*, max, min, etc, o user-defined)

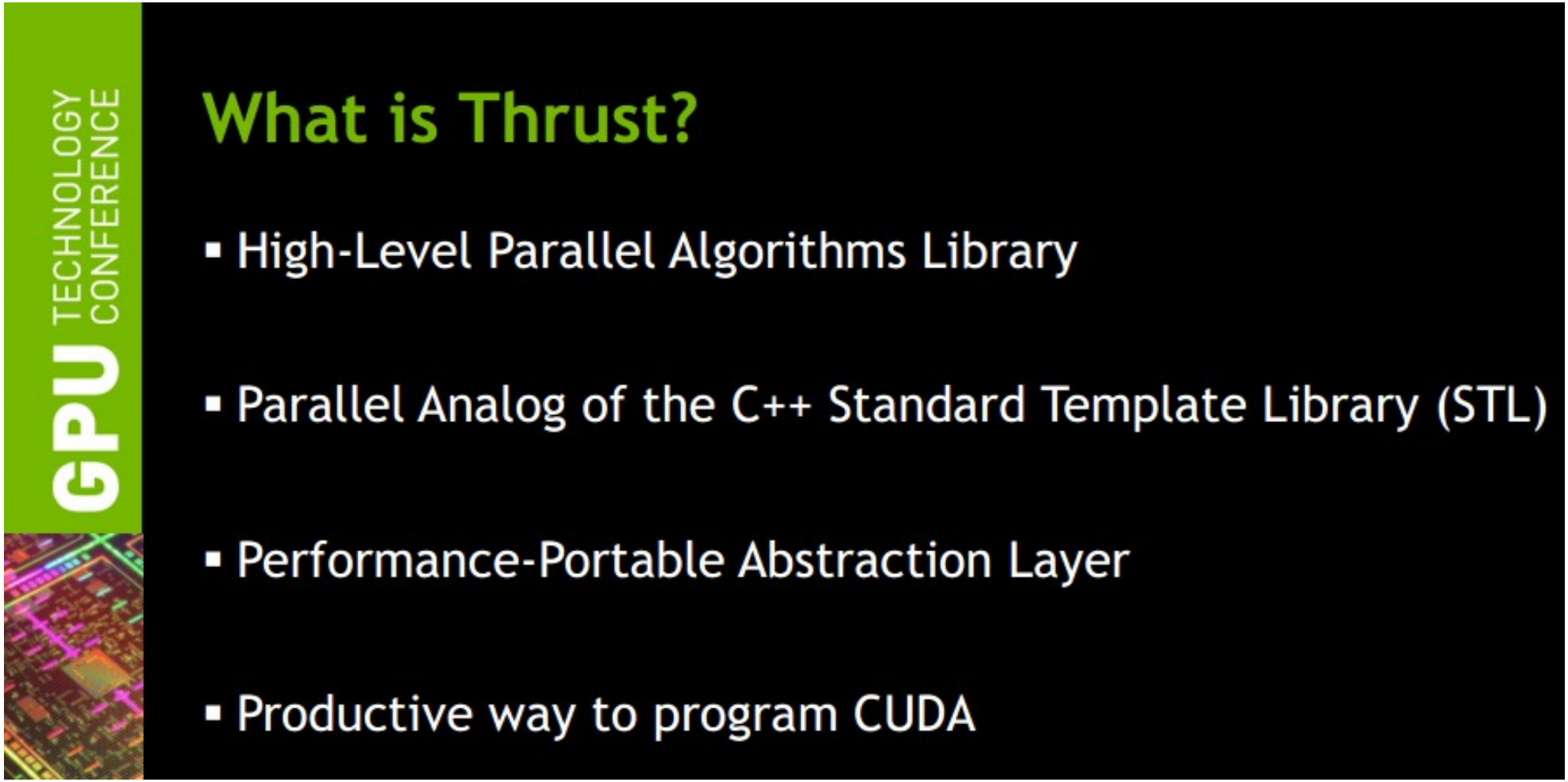
definida sobre **cualquier e**structura de datos (int, float, etc, o user-defined),

Viva en el HOST o en el DEVICE (GPU)

Generic algorithms

Thrust

Thrust Content from GTC 2012: Nathan Bell



Principales desarroladores: Jared Hoberock y Nathan Bell

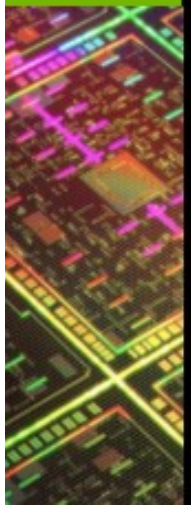
Thrust

Thrust Content from GTC 2012: Nathan Bell

Thrust

Thrust Content from GTC 2012: Nathan Bell

GPU TECHNOLOGY CONFERENCE



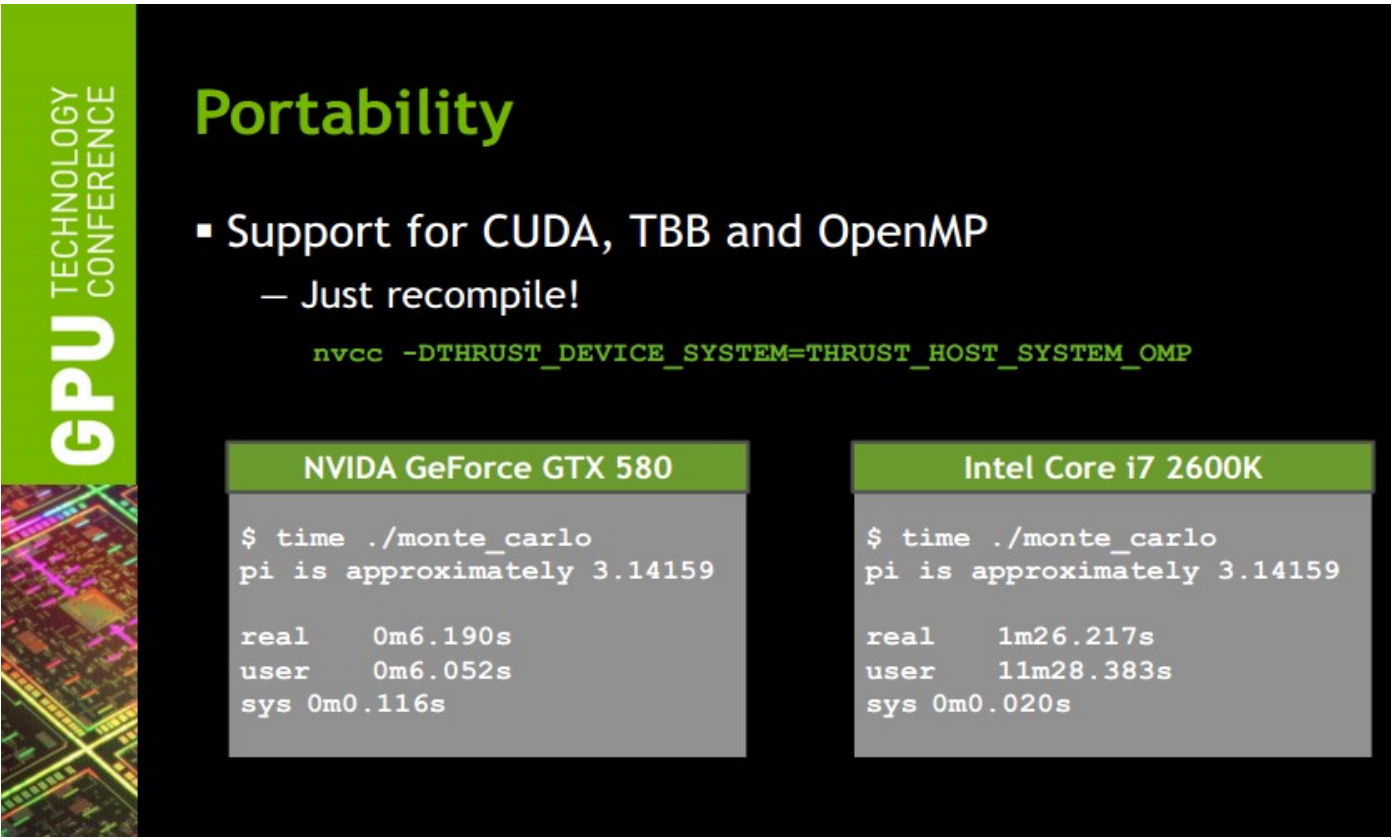
Productivity

- Containers
  - host_vector
  - device_vector
- Memory Mangement
  - Allocation
  - Transfers
- Algorithm Selection
  - Location is implicit

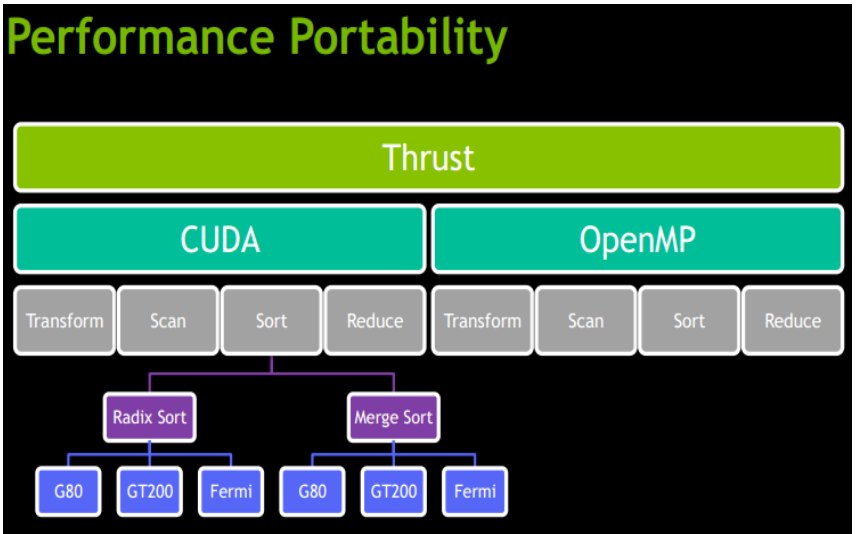
```
// allocate host vector with two elements
thrust::host_vector<int> h_vec(2);

// copy host data to device memory
thrust::device_vector<int> d_vec = h_vec;

// write device values from the host
d_vec[0] = 27;
d_vec[1] = 13;

// read device values from the host
int sum = d_vec[0] + d_vec[1];

// invoke algorithm on device
thrust::sort(d_vec.begin(), d_vec.end());

// memory automatically released
```

¿Qué significa esto?

Thrust



MISMO código
Corre en CPU Multicore!!



Thrust Content from GTC 2012: Nathan Bell

Delegamos la implementación
de bajo nivel o “mapeo al hardware”
a la librería

¡Thrust Portability!

Un solo codigo fuente → tres ejecutables

CPU secuencial usando g++

```
g++ -O2 miprog.cpp -
DTHRUST_DEVICE_SYSTEM=THRUST_
DEVICE_SYSTEM_CPP
-I/usr/local/cuda/include/ -o
ejecutable_cpp
```

GPU paralelo

```
nvcc -O2 -o ejecutable_gpu miprog.cu
```

CPU secuencial usando nvcc

```
nvcc -O2 miprog.cu -
DTHRUST_DEVICE_SYSTEM=THRUST
_DEVICE_SYSTEM_CPP
ejecutable_cpp
```

CPU paralelo

```
g++ -O2 miprog.cpp -fopenmp -
DTHRUST_DEVICE_SYSTEM=THRUST_
DEVICE_SYSTEM_OMP -lgomp
-I/usr/local/cuda/include/ -o
ejecutable_omp
```

Thrust Portability

```
cp miprog.cu miprog.cpp   //extension para g++

g++ -O2 -o miprogcpu miprog.cpp -fopenmp
-DTHRUST_DEVICE_SYSTEM=THRUST_DEVICE_SYSTEM_OMP
-lgomp -I<path-to-thrust-headers>
```

- No necesitan el cuda toolkit ni una GPU para desarrollar, compilar y correr sus codigos (claro, si estos usan solo thrust). Solo copiar los headers de thrust.
- Pueden comparar aceleración del cálculo paralelo en distintas CPU multicore, y cambiar la variable OMP_NUM_THREADS.
- Cuando tengan acceso a una placa, pueden comparar la aceleración simplemente recompilando con nvcc.
- Todo con exactamente el mismo código…

Thrust portability

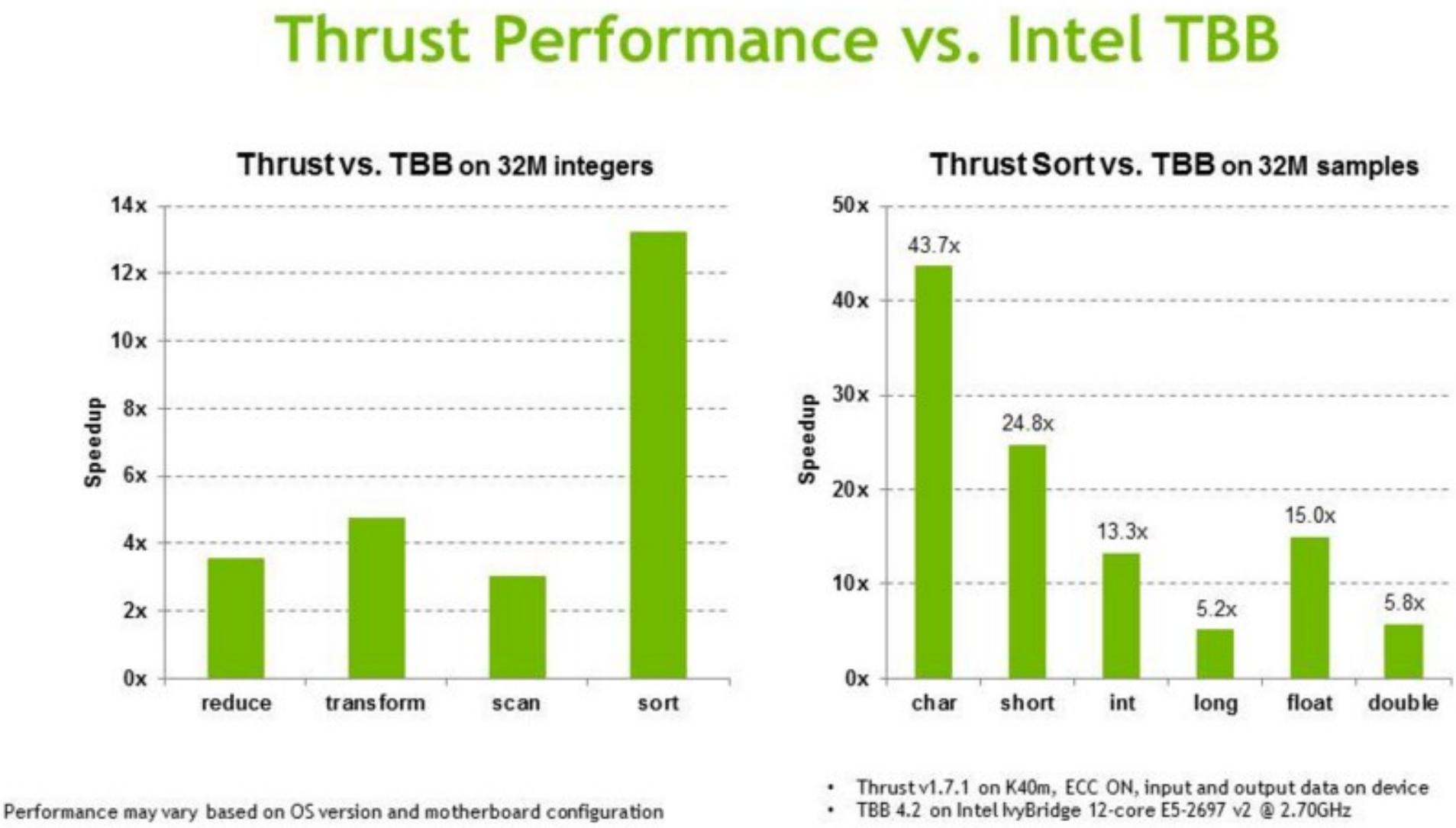
FILES: jobGPU3backends ompvsgpu.cu openompbackend.sh

```
int main(int argc, char **argv)
{
// …
thrust::host_vector<float> hx( N );
thrust::generate(hx.begin(),hx.end(),rand);
thrust::transform(hx.begin(),hx.end(),hx.begin(),_1/RAND_MAX);

thrust::device_vector<float> dx=hx;

float suma = thrust::reduce(dx.begin(),dx.end());
// …
}
```

Ejercicios

- Simplificar todos los códigos de CUDA C usando thrust:
  - Manejo de memoria
  - Kernels
- Consultar http://thrust.github.io/doc/modules.html, pensar que herramientas vienen bien para el proyecto.
- Consultar los ejemplos: https://github.com/thrust/thrust/tree/master/examples

Thrust Performance GPU vs CPU



Thrust

Thrust Content from GTC 2012: Nathan Bell

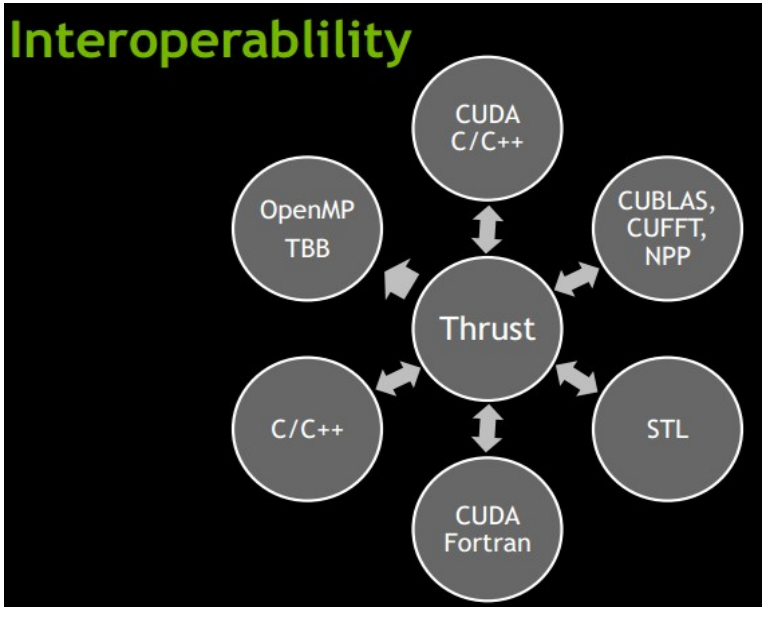
Productividad

- Large set of algorithms
  - ~75 functions
  - ~125 variations
- Flexible
  - User-defined types
  - User-defined operators

Algorithm	Description
reduce	Sum of a sequence
find	First position of a value in a sequence
mismatch	First position where two sequences differ
inner_product	Dot product of two sequences
equal	Whether two sequences are equal
min_element	Position of the smallest value
count	Number of instances of a value
is_sorted	Whether sequence is in sorted order
transform_reduce	Sum of transformed sequence

La mayoría son “Parallel primitives”

Interoperablility



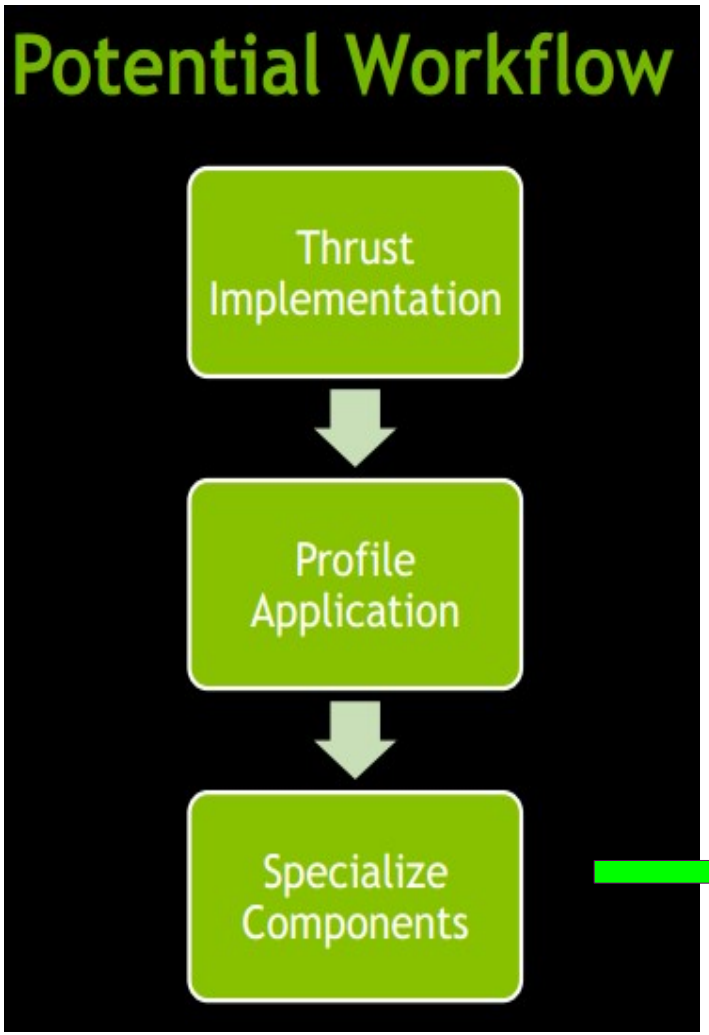
Thrust se puede usar para alivianar la escritura de gran parte del código:

Alocación de memoria, copias, y composición de patrones paralelas.

Limitaciones de Thrust

Review de M. Harris en:
https://developer.nvidia.com/content/expressive-algorithmic-programming-thrust

GTC 2012 Nathan Bell



No entiendo cuál es la limitación de Thrust

Usualmente en Cuda C/C++ o librerias nuevas de mas bajo nivel que Thrust como CUB

Resumen

Containers

- Manage host and device memory
- Simplify data transfers

Iterators

- Act like pointers
- Keep track of memory spaces

Algorithms

- Applied to Containers

Resumen

Review de M. Harris en:
https://developer.nvidia.com/content/expressive-algorithmic-programming-thrust