Parallel Computing with GPUs

Sorting and Libraries
Part 2 - Libraries

The University Of Sheffield.

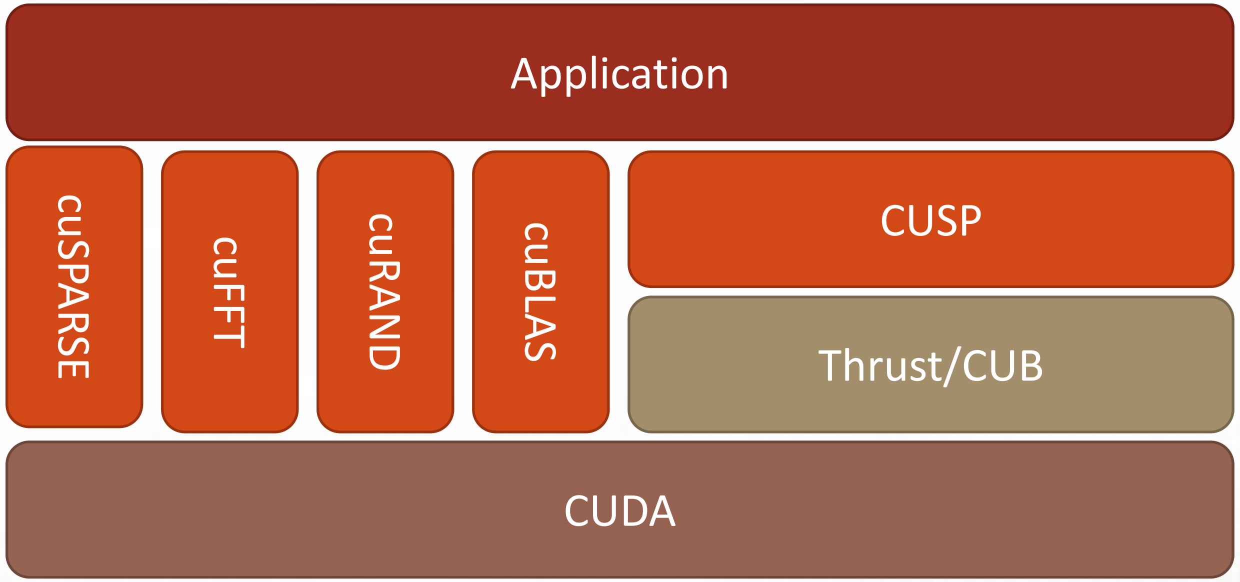
Dr Paul Richmond
http://paulrichmond.shef.ac.uk/teaching/COM4521/

This Lecture (learning objectives)

- ❑Libraries and Thrust
  - ❑Describe the purpose of CUDA libraries
  - ❑Demonstrate Thrust containers for data storage
  - ❑Explain the relationship between raw pointers and Thrust iterators
  - ❑Give example of Thrust algorithms

CUDA libraries

- ❑Abstract CUDA model away from programmer
- ❑Highly optimised implementations of common tools
  - ❑Mainly focused on linear algebra



Thrust

- Template Library for CUDA
  - Implements many parallel primitives (scan, sort, reduction etc.)
  - Part of standard CUDA release
  - Level of Abstraction which hides kernels, mallocs and memcpy’s
- Designed for C++ programmers
  - Similar in design and operation as the C++ Standard Template Library (STL)
  - Only a small amount of C++ required..

Thrust containers

- ❑Thrust uses only high level *vector* containers
  - ❑`host_vector`: on host
  - ❑`device_vector`: on GPU
- ❑Other STL containers include
  - ❑queue
  - ❑list
  - ❑tack
  - ❑queue
  - ❑priority_queue
  - ❑set
  - ❑multiset
  - ❑map
  - ❑multimap
  - ❑bitset
- ❑STL containers can be used to initialise a Thrust vector

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

int main()
{
  //create a vector on the host
  thrust::host_vector<int> h_vec(10);

  //create a vector on the device
  thrust::device_vector<int> d_vec = h_vec;

  //device data manipulated directly from host
  for (int i = 0; i < 10; i++)
    d_vec[i] = i;

  //vector memory automatically released
  return 0;
}
```

Thrust Iterators

- ❑They point to regions of a vector
- ❑Can be used like pointers
  - ❑Explicit cast when dereferencing very important

```
thrust::device_vector<int>::iterator begin = d_vec.begin();
thrust::device_vector<int>::iterator end = d_vec.end();
printf("d_vec at begin=%d", (int)*begin);
begin++;//move on a single position
printf("d_vec at begin++=%d", (int)*begin);
*end = 88;
printf("d_vec at end=%d", (int)*end);
```

```
d_vec at begin=0
d_vec at begin++=1
d_vec at end=88
```

Thrust Iterators

- Can be converted to a raw pointer

```
int * d_ptr = thrust::raw_pointer_cast(begin);
int * d_ptr = thrust::raw_pointer_cast(begin[0]);

kernel<BLOCKS, TPB>(d_ptr);
```

- Raw pointers can be used in Thrust
  - BUT not exactly the same as a vector

```
int* d_ptr;
cudaMalloc((void**)&d_ptr, N);

thrust::device_ptr<int> d_vec =  thrust::device_pointer_cast(d_ptr);
//or
thrust::device_ptr<int> d_vec = thrust::device_ptr<int>(d_ptr)
```

Thrust Algorithms

- Transformations
  - Application of a function to each element within the range of a vector
- Reduction
  - Reduction of a set of values to a single value using binary associative operator
  - Can also be used to count occurrences of a value
- Prefix Sum
  - Both inclusive and exclusive scans
- Sort
  - Can sort keys or key value pairs
- Binary Search
  - Position of a target value

Thrust Transformations

❑Some examples of the many transformations

```
//copy a vector (or part of a vector) to another vector
thrust:: copy(d_vec.begin(), d_vec.begin() + 10, d_vec_cpy.begin());

//fill a vector with a value
thrust::fill(d_vec.begin(), d_vec.begin() + 10, 0);

//rand is a predefined Thrust generator
thrust::generate(d_vec.begin(), d_vec.begin() + 10, rand);

// fill d_vec with {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
thrust::sequence(d_vec.begin(), d_vec.begin() + 10);

//all occurrences of the value 1 are replaced with the value 10
thrust::replace(d_vec.begin(), d_vec.end(), 1, 10);
```

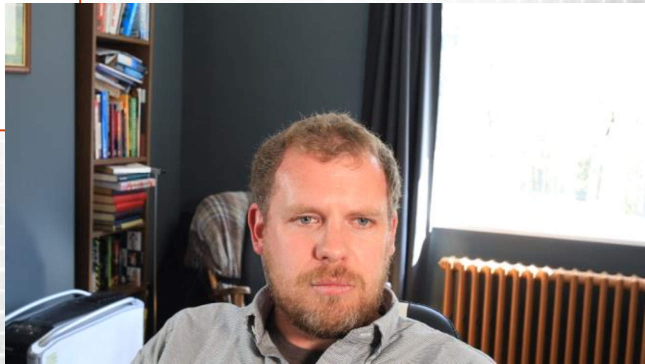
Thrust Algorithms

❑ Either in-place or to output vector

```
thrust::device_vector<int> d_vec(10);
thrust::device_vector<int> d_vec_out(10);

//fill d_vec with {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
thrust::sequence(d_vec.begin(), d_vec.begin() + 10);

//inclusive scan to output vector
thrust::inclusive_scan(d_vec.begin(), d_vec.end(), d_vec_out.begin());

//inclusive scan in place
thrust::inclusive_scan(d_vec.begin(), d_vec.end(), d_vec.begin());

//generate random data (actually a transformation)
thrust::generate(d_vec.begin(), d_vec.end(), rand);

//sort in place
thrust::sort(d_vec.begin(), d_vec.end());

//sort data from a raw pointer (N is number of elements)
thrust::device_ptr<int> dt_ptr =  thrust::device_pointer_cast(d_a_ptr);
thrust::sort(dt_ptr, dt_ptr+N);
```

Custom Transformations

```
thrust::device_vector<int> d_vec(10);
thrust::device_vector<int> d_vec_out(10);

//fill d_vec with {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
d_vec = thrust::sequence(d_vec.begin(), d_vec.begin() + 10);


//declare a custom operator
struct add_5{
  __host__ __device__ int operator()(int a){
     return a + 5;
  }
};

add_5 func;

//apply custom transformation
thrust::transform(d_vec.begin(), d_vec.end(), d_vec_out.begin(), func);

//d_vec is now {5, 6, 7, 8, 9, 10, 11, 12, 13, 14}
```



Thrust Fusion

- For best performance it is necessary to fuse operations

```
struct absolute{
  __host__ __device__ int operator()(int a){
    return a < 0 ? -a : a ;
  }
};
absolute func;

//custom transformation to calculate absolute value
thrust::transform(d_vec.begin(), d_vec.end(), d_vec.begin(), func);
//apply reduction, maximum binary associate operator
int result = thrust::reduce(d_vec.begin(), d_vec.end(), 0, thrust::maximum<int>());
```

```
struct absolute{
  __host__ __device__ int operator()(int a){
    return a < 0 ? -a : a ;
  }
};
absolute func;

//apply transform reduction maximum binary associate operator
int result = thrust::transform_reduce(d_vec.begin(), d_vec.end(), func, 0, thrust::maximum<int>());
```

Summary

- ❑Libraries and Thrust
  - ❑Describe the purpose of CUDA libraries
  - ❑Demonstrate Thrust containers for data storage
  - ❑Explain the relationship between raw pointers and Thrust iterators
  - ❑Give example of Thrust algorithms

- ❑Next: Applications of Sort