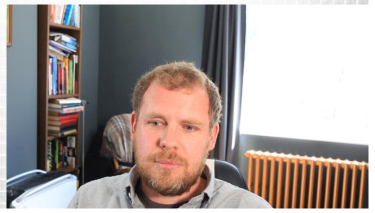
# Parallel Computing with GPUs

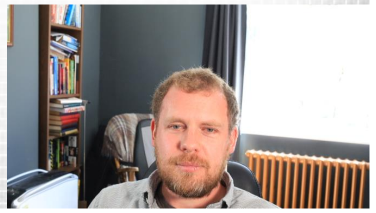
## Sorting and Libraries
## Part 1 - Sorting

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



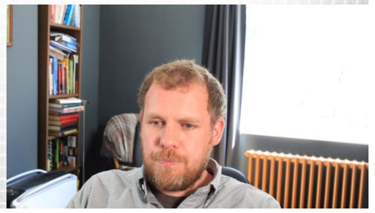
## This Lecture (learning objectives)

- Sorting Networks
  - Demonstrate the use of a sorting network to achieve parallel sorting
  - Compare sorting networks with serial sorting approaches
- Merge and Bitonic Sort
  - Present the merge sort and considers its performance implications
  - Identify performacne features of bitonic sorting



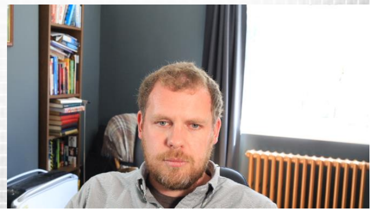
## Serial Sorting Examples

- Insertion Sort
  - Insert a new element into a sorted list.
    - E.g. [ 1 6 3 4 2 5 ]
      - [1] -> [1 6] -> [1 3 6] -> [1 3 4 6] -> [1 2 3 4 6] -> [1 2 3 4 5 6]
- Bubble Sort
  - Exchange and Sweep to compare each pair of adjacent elements
  - *$O(n^2)$* worst-case and average case, *$O(n)$* best case.
    - E.g. [ 1 6 3 4 2 5 ]
      - [1 6 3 4 2 5] -> [1 3 6 4 2 5] -> [1 3 4 6 2 5] -> [1 3 4 2 6 5] -> [1 3 4 2 5 6]
      - [1 3 2 4 5 6]
      - [1 2 3 4 5 6]



## Serial Sorting Examples

- Insertion Sort
  - Insert a new element into a sorted list.
    - E.g. [ 1 6 3 4 2 5 ]
      - [1] -> [1 6] -> [1 3 6] -> [1 3 4 6] -> [1 2 3 4 6] -> [1 2 3 4 5 6]
- Bubble Sort
  - Exchange and Sweep to compare each pair of adjacent elements
  - *$O(n^2)$* worst-case and average case, *$O(n)$* best case.
    - E.g. [ 1 6 3 4 2 5 ]
      - [1 6 3 4 2 5] -> [1 3 6 4 2 5] -> [1 3 4 6 2 5] -> [1 3 4 2 6 5] -> [1 3 4 2 5 6]
      - [1 3 2 4 5 6]
      - [1 2 3 4 5 6]

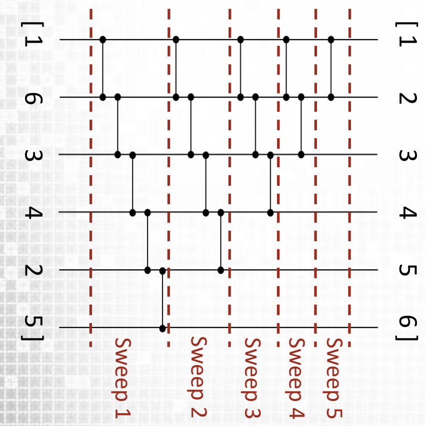


## Classifying Sort Techniques/Implementations

- Data driven
  - Each step of the algorithm depends on the previous step version
  - Highly serial
- Data independent
  - The algorithms performs fixed steps and does not change its processing based on data
  - Well suited to parallel implementations
  - Can be expressed as a sorting network…

## Sorting Networks

- A sorting network is a comparator network that sorts all input sequences
  - Following the same execution of stages
- Consider the previous Bubble Sort [ 1 6 3 4 2 5 ]



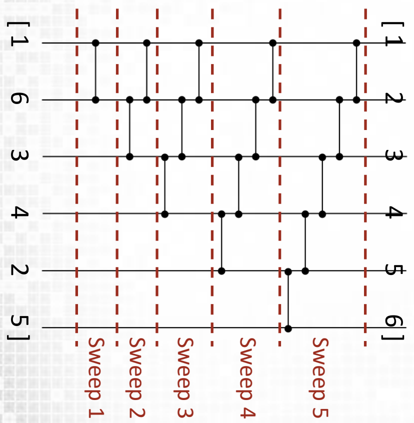

[1 6 3 4 2 5] -> [1 3 6 4 2 5] -> [1 3 4 6 2 5] -> [1 3 4 2 6 5] -> [1 3 4 2 5 6]
[1 3 4 2 5 6] -> [1 3 4 2 5 6] -> [1 3 2 4 5 6] -> [1 3 2 4 5 6]
[1 3 2 4 5 6] -> [1 2 3 4 5 6] -> [1 2 3 4 5 6]
[1 3 2 4 5 6] -> [1 2 3 4 5 6]
[1 2 3 4 5 6]

Sweeps

Not considered
Compared not swapped
**Compared and swapped**

## Sorting Networks

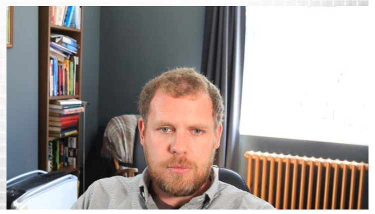
- And Insertion Sort…



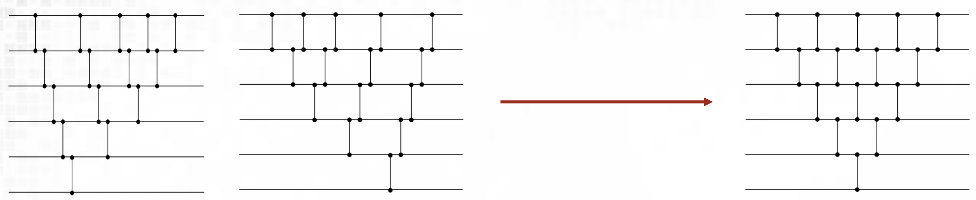

[1 6 3 4 2 5]
[1 3 6 4 2 5] -> [1 3 6 4 2 5]
[1 3 4 6 2 5] -> [1 3 4 6 2 5] -> [1 3 4 6 2 5]
[1 3 4 2 6 5] -> [1 3 2 4 6 5] -> [1 2 3 4 6 5] -> [1 2 3 4 6 5]
[1 2 3 4 5 6] -> [1 2 3 4 5 6] -> [1 2 3 4 5 6] -> [1 2 3 4 5 6] -> [1 2 3 4 5 6]

Sweeps

Not considered
Compared not swapped
**Compared and swapped**



## Parallel Sorting Networks



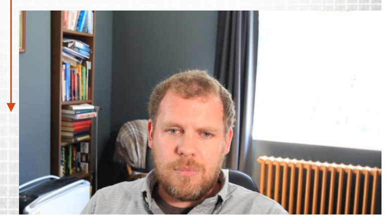
Bubble

Insertion

[1 6 3 4 2 5]
[1 3 6 4 2 5]
[1 3 4 6 2 5]
[1 3 4 2 6 5]
[1 3 2 4 5 6]
[1 2 3 4 5 6]
[1 2 3 4 5 6]
[1 2 3 4 5 6]
[1 2 3 4 5 6]

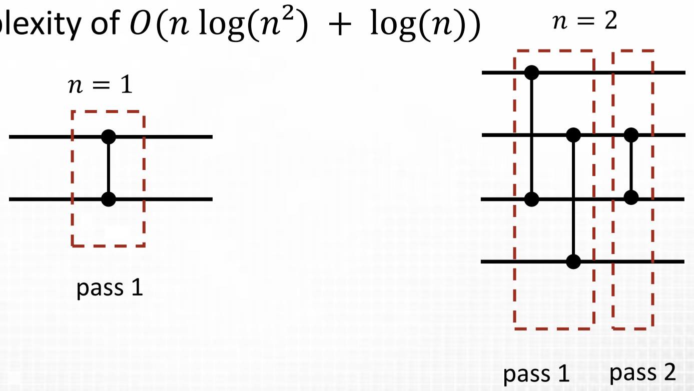
Sweeps = 9

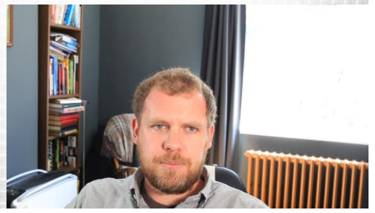
- Parallel Bubble and Insertion sorting network is still not very efficient
  - $2n - 3$ sweeps
  - $n(n-1)/2$ comparisons - $O(n^2)$ complexity



## Merge Sort

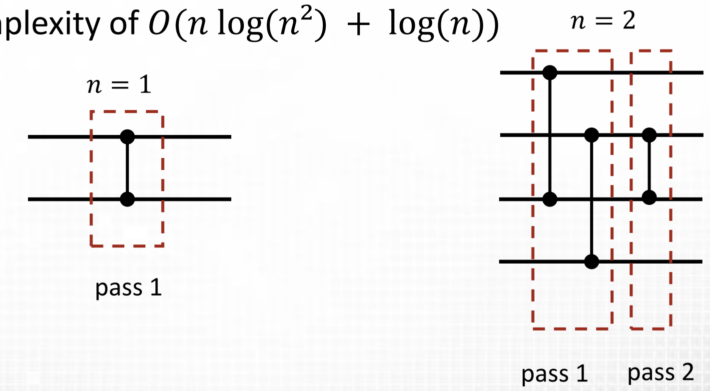
- To reduce the $O(n^2)$ overhead we need a better sorting network
- The odd-even merge sort network (for power of 2 $n$)
  - Sort all odd and even keys separately and then merge $m$ values of a stage
  - Merge a sorted sequence of elements on lines $< a_1, \ldots, an >$ with those on lines $< a_{n+1}, \ldots, a_{2n} >$
  - Each merge requires $\log(n)$ passes
  - Total complexity of $O(n \log(n^2) + \log(n))$

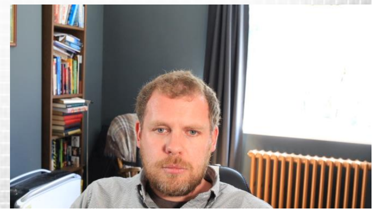


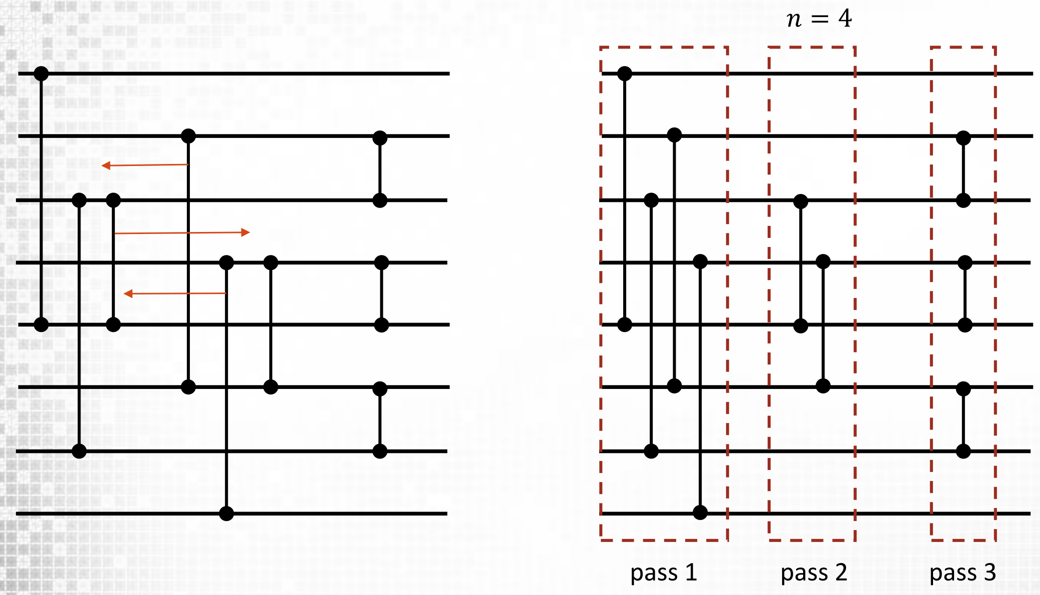
## Merge Sort

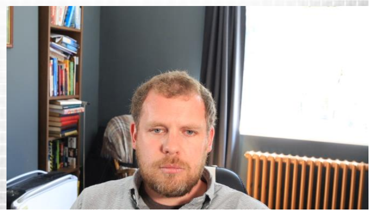
- To reduce the $O(n^2)$ overhead we need a better sorting network
- The odd-even merge sort network (for power of 2 $n$)
  - Sort all odd and even keys separately and then merge $m$ values of a stage
  - Merge a sorted sequence of elements on lines $< a_1, \ldots, an >$ with those on lines $< a_{n+1}, \ldots, a_{2n} >$
  - Each merge requires $\log(n)$ passes
  - Total complexity of $O(n \log(n^2) + \log(n))$



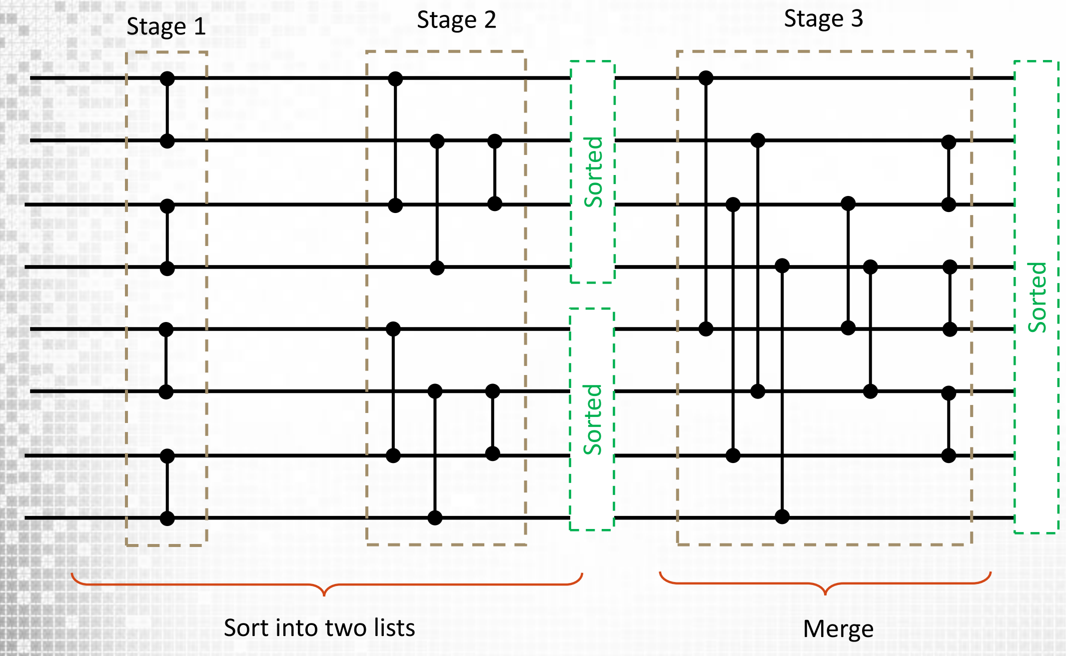



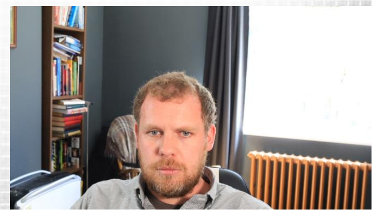
## Merging of two sorted sequences (n=4)



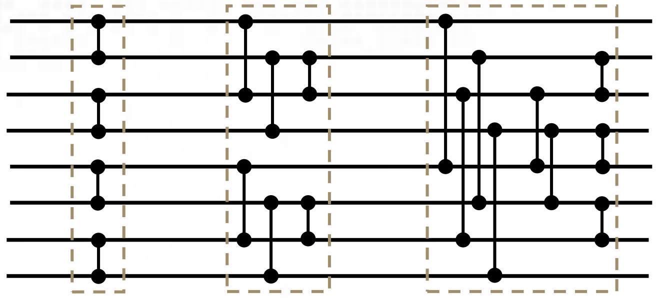



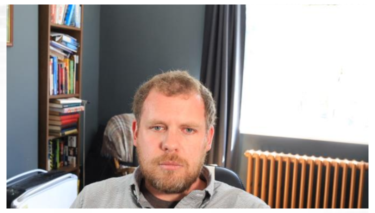
## Full Merge Sorting (n=4)



## Full Merge Sorting (n=4) example

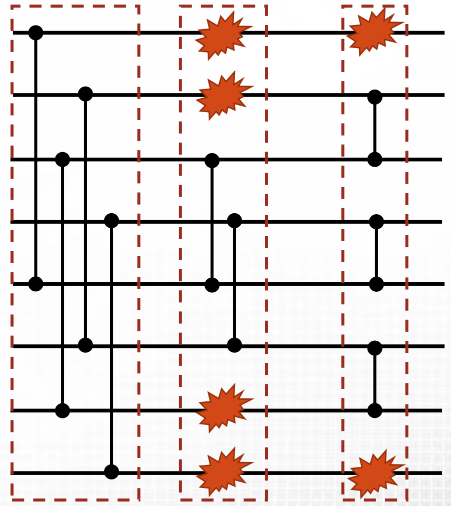


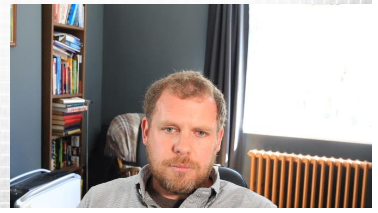


| Input | Stage 1 | Stage 2 | | | Stage 3 | | | | | | | Output |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 8 | 1 | 1 | | | 1 | | | | | | | 1 |
| 1 | 8 | | 5 | 3 | | | 3 | | | | 2 | 2 |
| 5 | 3 | 3 | | 5 | | 5 | | | 2 | | 3 | 3 |
| 3 | 5 | | 8 | | | | | 8 | | 4 | 4 | 4 |
| 6 | 2 | 2 | | | 2 | | | | 5 | | 5 | 5 |
| 2 | 6 | | 6 | 4 | | | 4 | | | 8 | 6 | 6 |
| 4 | 4 | 4 | | 6 | | 6 | | | | | 8 | 8 |
| 9 | 9 | | 9 | | | | | 9 | | | | 9 |

## Limitations of Merge Sort

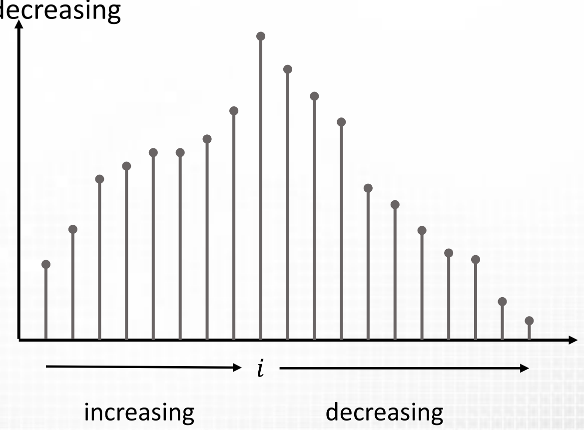
- What is potentially wrong with a merge sort GPU implementation?
  - Irregular memory accesses
  - Not all values are compared in each pass (uneven workload per thread)

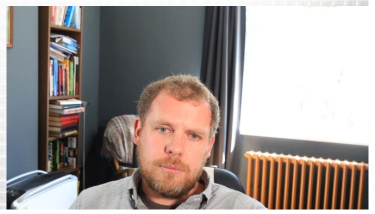




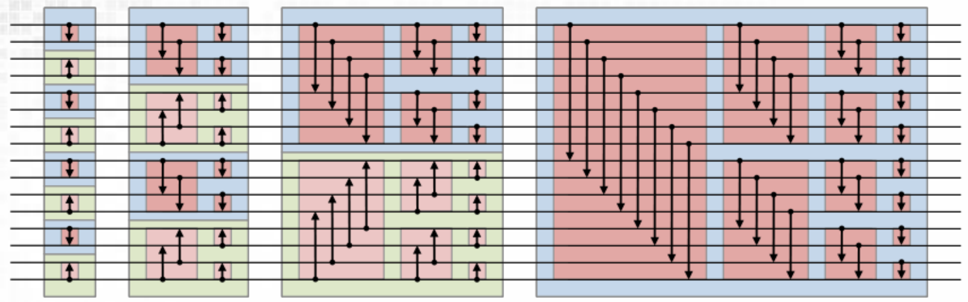
## Solution: Bitonic Sort

- Bitonic sorting network
  - Iterative splitting and merging of inputs into increasing large bionic sequences
  - A sequence is bitonic if
    - There is an $i$, such that , $a_0\ \dots, a_i$ is monotonically increasing and $a_i\ \dots, a_n$ is monotonically decreasing

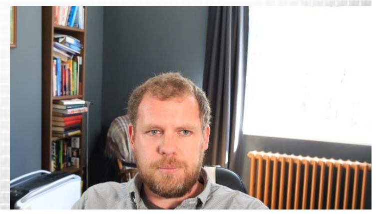


## Bitonic Sorting Network



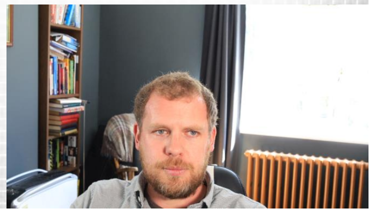
- Sorting and Merging increasing large bionic sequences
  - When $n = 2^k$ there are $k$ levels with $\frac{n}{2}$ comparisons each
- GPU Implementation
  - Regular access strides :-)
  - Efficiently balanced workload :-)
  - Requires multiple kernel launches to merge over $n$ > block size



## Summary

- ❑Sorting Networks
  - ❑Demonstrate the use of a sorting network to achieve parallel sorting
  - ❑Compare sorting networks with serial sorting approaches
- ❑Merge and Bitonic Sort
  - ❑Present the merge sort and considers its performance implications
  - ❑Identify performance features of bitonic sorting

- ❑Next: Libraries and Thrust

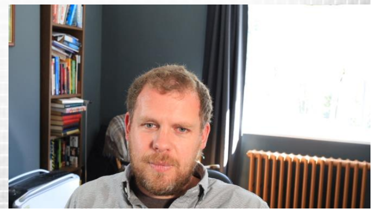


# Parallel Computing with GPUs

# Sorting and Libraries Part 2 - Libraries

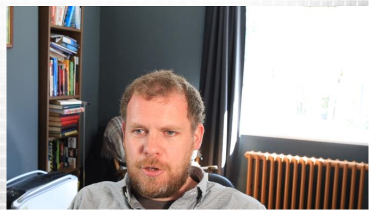
The University Of Sheffield.

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



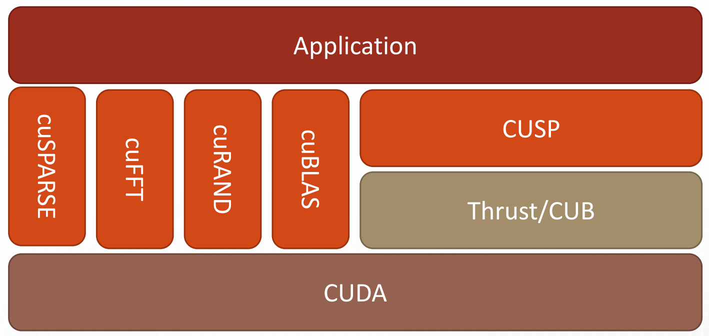
## This Lecture (learning objectives)

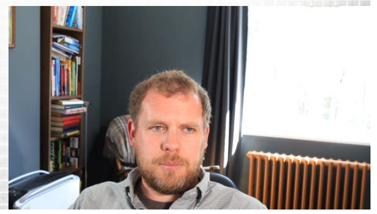
- ❑Libraries and Thrust
  - ❑Describe the purpose of CUDA libraries
  - ❑Demonstrate Thrust containers for data storage
  - ❑Explain the relationship between raw pointers and Thrust iterators
  - ❑Give example of Thrust algorithms



## CUDA libraries

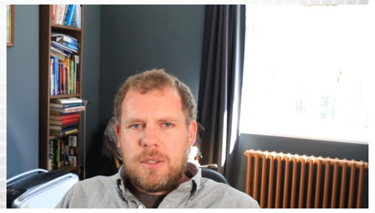
- ❑Abstract CUDA model away from programmer
- ❑Highly optimised implementations of common tools
  - ❑Mainly focused on linear algebra



# Thrust

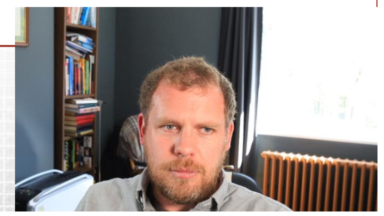
- Template Library for CUDA
  - Implements many parallel primitives (scan, sort, reduction etc.)
  - Part of standard CUDA release
  - Level of Abstraction which hides kernels, mallocs and memcpy's
- Designed for C++ programmers
  - Similar in design and operation as the C++ Standard Template Library (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;
}
```

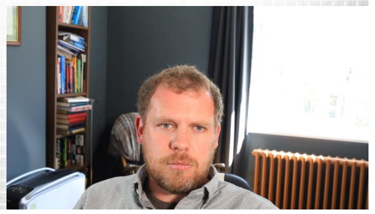


# Thrust Iterators

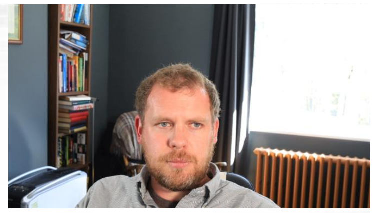
- They point to regions of a vector
- Can be used like pointers
  - Explicit cast when dereferencing very important

```
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

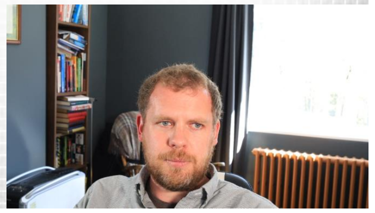
```
int * d_ptr = thrust::raw_pointer_cast(begin);
int * d_ptr = thrust::raw_pointer_cast(begin[0]);

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

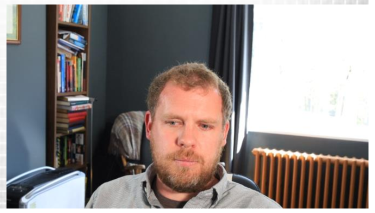
- Transformations
  - Application of a function to each element within the range of a vector
- Reduction
  - Reduction of a set of values to a single value using binary associative operator
  - Can also be used to count occurrences of a value
- Prefix Sum
  - Both inclusive and exclusive scans
- Sort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



## Thrust Transformations

- Some examples of the many transformations

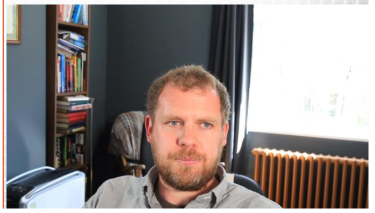
```
//copy a vector (or part of a vector) to another vector
thrust:: copy(d_vec.begin(), d_vec.begin() + 10, d_vec_cpy.begin());

//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

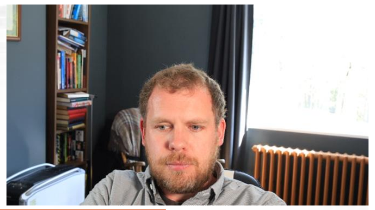
```
thrust::device_vector<int> d_vec(10);
thrust::device_vector<int> d_vec_out(10);

//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



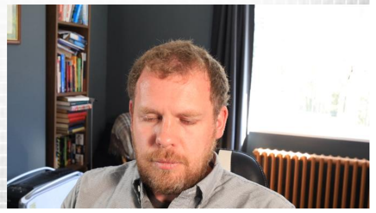
- For best performance it is necessary to fuse operations

```
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

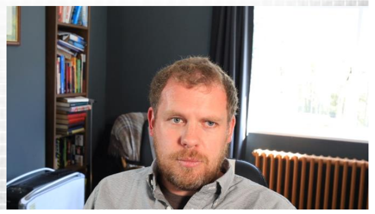


# Parallel Computing with GPUs

# Sorting and Libraries Part 3 - Applications of Sorting

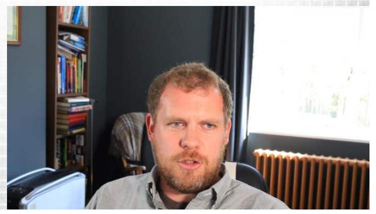
The University Of Sheffield.

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



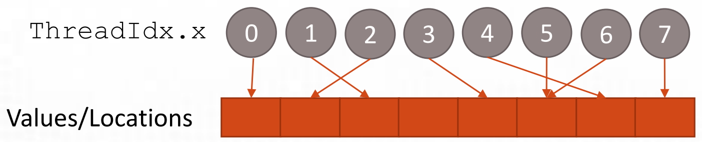
## This Lecture (learning objectives)

- Applications of Sorting (binning)
  - Present the concept of spatial binning
  - Demonstrate the use of spatial binning for particle interactions

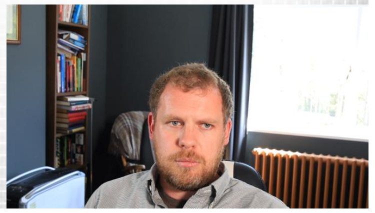


## Sorting and parallel primitives

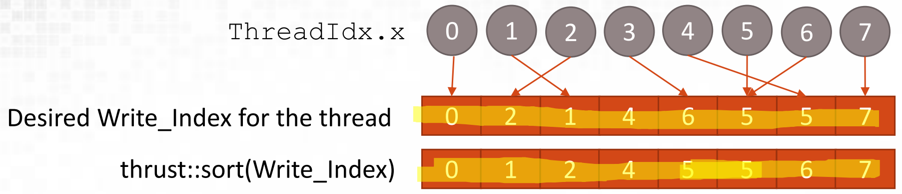
- Can be very useful for building data structures
  - We can use prefix sum for writing multiple values per element
- Remember Gather vs Scatter
  - What if our outputs are scattered to output
  - Very common in particle simulations etc.
    - Outputs might represent spatial bins




Scatter operation
- Write to a number of locations
- Random access write?
- How to read multiple values afterwards?



## Binning and Sorting

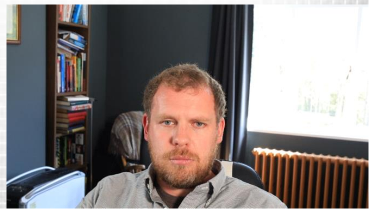



*Build a data structure*

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Unique write indices | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Count(Write_Index) | 1 | 1 | 1 | 0 | 1 | 2 | 1 | 1 |
| thrust::inclusive_scan(count) | 0 | 1 | 2 | 3 | 3 | 4 | 6 | 7 |

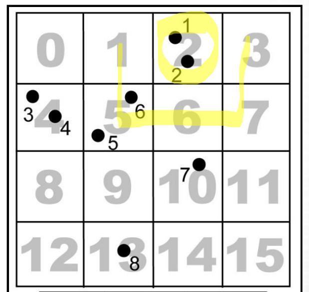
i.e. how many threads want to write to this index

- We can now read varying values from each bin
  - E.g. for location 5
    - inclusive_scan gives starting index of 4
    - Iterate from index 4 for a count of 2 to find all values of write_index 5



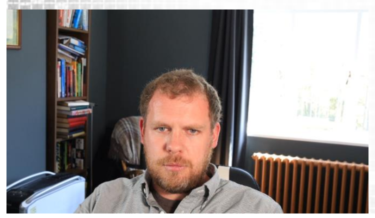
## Particle interaction example

- As with previous slide use sorting
  - Divide the environment according to some interaction radius
  - Output particle key value pairs (keys are location determined through some hash function)
  - Sort Keys
  - Reorder particles based on key pairs
  - Generate a partition boundary table
    - Histogram count and prefix sum
  - Each particle needs to read all particles in its own location and any neighbouring location
    - Guarantees particle interactions within the interaction radius

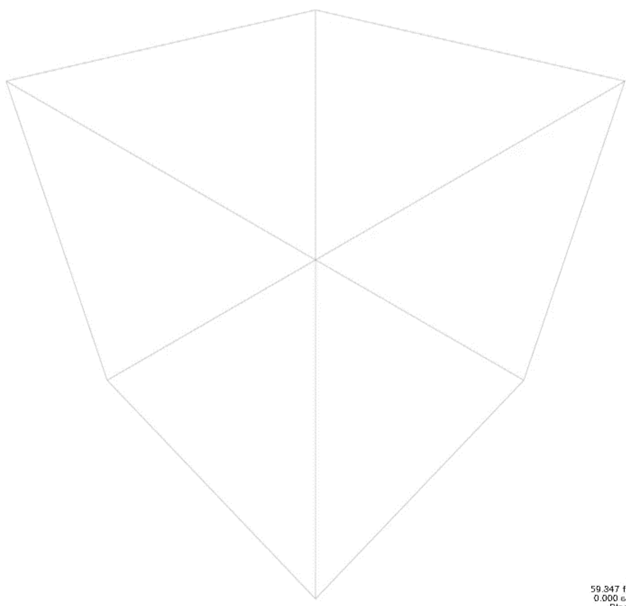



| Partition | First agent | Last agent |
|---|---|---|
| 0 | | |
| 1 | | |
| 2 | 1 | 2 |
| 3 | | |
| 4 | 3 | 4 |
| 5 | 5 | 6 |
| 6 | | |
| 7 | | |
| 8 | | |
| 9 | | |
| 10 | 7 | 7 |
| 11 | | |
| 12 | | |
| 13 | 8 | 8 |
| 14 | | |
| 15 | | |

The University Of Sheffield.



## Example: FLAME GPU



FLAME GPU

## Summary

- ❑Applications of Sorting (binning)
  - ❑Present the concept of spatial binning
  - ❑Demonstrate the use of spatial binning for particle interactions