Thrust

What is Thrust

- Thrust is a C++ template library for CUDA based on the Standard Template Library (STL). Thrust allows you to implement high performance parallel applications with minimal programming effort through a high-level interface that is fully interoperable with CUDA C.

Thrust Vectors

- Thrust provides 2 vector containers, they both operate like std::vector in C++ STL
  - host_vector: stored in host memory
  - device_vector: stored in GPU device memory

- Just like std::vector, Thrust vectors are generic containers that can be resized dynamically.

Thrust Vectors

- Thrust provides implementations for a set of standard vector operations, such as:
  - Constructor

    thrust::device_vector<int> D(10, 1);
  - Fill: sets the specified range with that value

    thrust::fill (D.begin(), D.begin() + 7, 9);
  - Sequence: sets the specified range with sequence (0, 1, 2, …)

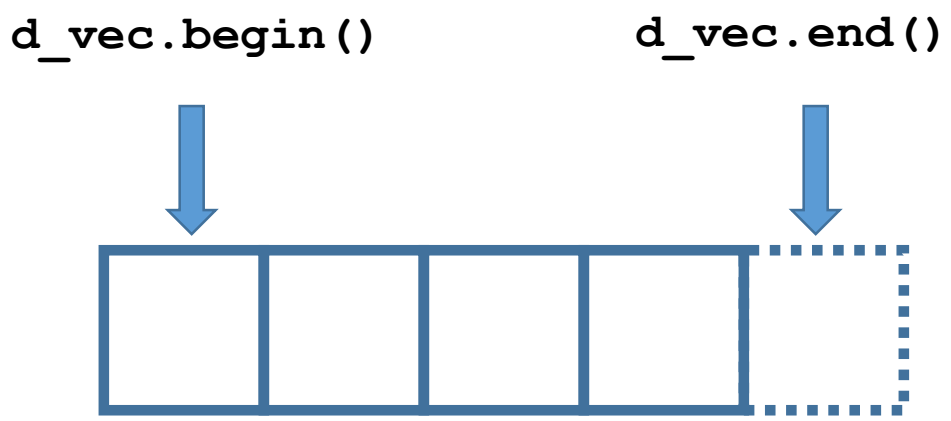
    thrust::sequence (D.begin(), D.end());
  - Copy: copies the content of a vector to another one

    thrust::copy (H.begin(), H.end(), D.begin()); // copies all of H to the beginning of D

Iterators

- Iterators can be thought of as pointers to array elements, however they carry more information than a regular pointer.

- Thrust vector has 2 members that return first and one past the last element of the vector



Iterators

- Notice we didn’t have to tell thrust::fill whether it is operating on a device_vector or a host_vector, that information is stored in the iterator.
- When a Thrust function is called, it inspects the type of the iterator to determine whether to use a host or a device implementation. This process is known as *static dispatching* since the host/device dispatch is resolved at compile time. Note that this implies that there is *no runtime overhead* to the dispatch process.

Wrapping Raw Pointers

- To use thrust vector functions on raw device pointers, you have to wrap them first with a thrust::device_ptr object.

```
size_t N = 10;
// raw pointer to device memory
int * raw_ptr; cudaMalloc((void **) &raw_ptr, N * sizeof(int));
// wrap raw pointer with a device_ptr
thrust::device_ptr<int> dev_ptr(raw_ptr);
// use device_ptr in thrust algorithms
thrust::fill(dev_ptr, dev_ptr + N, (int) 0);
```

Algorithms

Thrust provides a large number of common parallel algorithms. Many of these algorithms have direct analogs in the STL, and when an equivalent STL function exists, we choose the name (e.g. `thrust::sort` and `std::sort`).

Sorting

Thrust offers several function to sort data or rearrange data accorind to a given criterion.

```
#include <thrust/sort.h>
const int N = 6;
int A[N] = {1, 4, 2, 8, 5, 7};
thrust::sort(A, A + N);
// A is now {1, 2, 4, 5, 7, 8}
```

Sorting

Just like STL, thrust sorting functions accept user-defined comparison operators, and offers a group of pre-defined comparators

```
#include <thrust/sort.h>
#include <thrust/functional.h>
const int N = 6;
int A[N] = {1, 4, 2, 8, 5, 7};
thrust::stable_sort(A, A + N, thrust::greater<int>());
// A is now {8, 7, 5, 4, 2, 1}
```