The Thrust Library

Will Landau
Prof. Jarad Niemi

Getting Started

Iterators
Basic iterators
constant_iterator
counting_iterator
transform_iterator
permutation_iterator
zip_iterator

Containers

Algorithms
Transformations
Reductions
Scans
Reordering
Sorting

The Thrust Library

Will Landau
Prof. Jarad Niemi

November 14, 2012

Outline

The Thrust Library

Will Landau
Prof. Jarad Niemi

Getting Started

Iterators
Basic iterators
constant_iterator
counting_iterator
transform_iterator
permutation_iterator
zip_iterator

Containers

Algorithms
Transformations
Reductions
Scans
Reordering
Sorting

The Thrust Library

Will Landau
Prof. Jarad Niemi

Getting Started

Iterators
Basic iterators
constant_iterator
counting_iterator
transform_iterator
permutation_iterator
zip_iterator

Containers

Algorithms
Transformations
Reductions
Scans
Reordering
Sorting

Thrust is the CUDA analog of the Standard Template Library (STL) of C++. It comes with any installation of CUDA 4.2 and above and features:

- Dynamic data structures
- An encapsulation of GPU/CPU communication, memory management, and other low-level tasks.
- High-performance GPU-accelerated algorithms such as sorting and reduction

Brief history:

- Emerged from Komrade (deprecated) in 2009
- Maintained primarily by Jared Hoberock and Nathan Bell of NVIDIA.

vector1.cu I

The Thrust Library

Will Landau
Prof. Jarad Niemi

Getting Started

Iterators

Basic iterators
constant_iterator
counting_iterator
transform_iterator
permutation_iterator
zip_iterator

Containers

Algorithms

Transformations
Reductions
Scans
Reordering
Sorting

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

int main(void){

  // H has storage for 4 integers
  thrust::host_vector<int> H(4);

  // initialize individual elements
  H[0] = 14;
  H[1] = 20;
  H[2] = 38;
  H[3] = 46;

  // H.size() returns the size of vector H
  std::cout << "H has size " << H.size() << std::endl;

  // print contents of H
  for(int i = 0; i < H.size(); i++)
    std::cout << "H[" << i << "] = " << H[i] << std::endl;

  // resize H
  H.resize(2);
  std::cout << "H now has size " << H.size() << std::endl;

  // Copy host_vector H to device_vector D
  thrust::device_vector<int> D = H;

  // elements of D can be modified
```

vector1.cu II

```
D[0] = 99;
D[1] = 88;

// print contents of D
for(int i = 0; i < D.size(); i++)
  std::cout << "D[" << i << "] = " << D[i] << std::endl;

// H and D are automatically deleted when the function returns
return 0;
}
```

Output of vector1.cu I

The Thrust Library

Will Landau
Prof. Jarad Niemi

Getting Started

Iterators
Basic iterators
constant_iterator
counting_iterator
transform_iterator
permutation_iterator
zip_iterator

Containers

Algorithms
Transformations
Reductions
Scans
Reordering
Sorting

```
[landau@impact1 vector1]$ make
nvcc vector1.cu -o vector1
[landau@impact1 vector1]$ ./vector1
H has size 4
H[0] = 14
H[1] = 20
H[2] = 38
H[3] = 46
H now has size 2
D[0] = 99
D[1] = 88
[landau@impact1 vector1]$
```

The Thrust Library

Will Landau
Prof. Jarad Niemi

Getting Started

Iterators
Basic iterators
constant_iterator
counting_iterator
transform_iterator
permutation_iterator
zip_iterator

Containers

Algorithms
Transformations
Reductions
Scans
Reordering
Sorting

- ▶ Thrust takes care of malloc(), cudaMalloc(), free(), and cudaFree() for you without sacrificing performance.
- ▶ The "=" operator does a cudaMemcpy() if one vector is on the host and one is on the device.
- ▶ thrust:: and std:: clarify the *namespace* of the function after the double colon. For example, we need to distinguish between thrust::copy() and std::copy().
- ▶ The "<<" operator sends a value to an output stream, an alternative to printf().

vector2.cu I

```cpp
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>
#include <thrust/copy.h>
#include <thrust/fill.h>
#include <thrust/sequence.h>
#include <iostream>

int main(void){
  // initialize all ten integers of a device_vector to 1
  thrust::device_vector <int> D(10, 1);

  // set the first seven elements of a vector to 9
  thrust::fill(D.begin(), D.begin() + 7, 9);

  // initialize a host_vector with the first five elements of D
  thrust::host_vector<int> H(D.begin(), D.begin() + 5);

  // set the elements of H to 0, 1, 2, 3, ...
  thrust::sequence(H.begin(), H.end());

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

  // print D
  for(int i = 0; i < D.size(); i++)
    std::cout << "D[" << i << "] = " << D[i] << std::endl;

  return 0;
}
```

The Thrust Library

Will Landau
Prof. Jarad Niemi

Getting Started

Iterators

Basic iterators
constant_iterator
counting_iterator
transform_iterator
permutation_iterator
zip_iterator

Containers

Algorithms

Transformations
Reductions
Scans
Reordering
Sorting

Output of vector2.cu l

```
[landau@impact1 vector2]$ make
nvcc vector2.cu -o vector2
[landau@impact1 vector2]$ ./vector2
D[0] = 0
D[1] = 1
D[2] = 2
D[3] = 3
D[4] = 4
D[5] = 9
D[6] = 9
D[7] = 1
D[8] = 1
D[9] = 1
[landau@impact1 vector2]$
```

Assignment

The Thrust Library

Will Landau
Prof. Jarad Niemi

Getting Started

Iterators

Basic iterators
constant_iterator
counting_iterator
transform_iterator
permutation_iterator
zip_iterator

Containers

Algorithms

Transformations
Reductions
Scans
Reordering
Sorting

- ▶ thrust::copy() copies a section of one vector into a section of another.
- ▶ thrust::fill() sets a range of elements to some fixed value.
- ▶ thrust::sequence() assigns equally-spaced values to a section of a vector.

The Vector Template Classes

- Declaring vectors:
  - `thrust::device_vector<T> D;` creates a vector `D` with entries of data type `T` on the device.
  - The analogous declaration for host vectors is `thrust::host_vector<T> H;`.
- An object `D` of the vector template class includes the following features:
  - A dynamic linear array of elements of type `T`.
  - Two *iterators*:
    - `D.begin()`
    - `D.end()`

The Thrust Library

Will Landau
Prof. Jarad Niemi

Getting Started

Iterators
Basic iterators
constant_iterator
counting_iterator
transform_iterator
permutation_iterator
zip_iterator

Containers

Algorithms
Transformations
Reductions
Scans
Reordering
Sorting

Outline

The Thrust Library

Will Landau
Prof. Jarad Niemi

Getting Started

Iterators
Basic iterators
constant_iterator
counting_iterator
transform_iterator
permutation_iterator
zip_iterator

Containers

Algorithms
Transformations
Reductions
Scans
Reordering
Sorting

The Thrust Library

Will Landau
Prof. Jarad Niemi

Getting Started

Iterators

Basic iterators
constant_iterator
counting_iterator
transform_iterator
permutation_iterator
zip_iterator

Containers

Algorithms
Transformations
Reductions
Scans
Reordering
Sorting

An iterator is a pointer with a C++ wrapper around it. The wrapper contains additional information, such as whether the vector is stored on the host or the device.

```
// allocate device vector
thrust::device_vector<int> d_vec(4);

d_vec.begin(); // returns iterator at first element of d_vec
d_vec.end(); // returns iterator one past the last element of d_vec

// [begin, end) pair defines a sequence of 4 elements
```

Iterators act like pointers.

```
// allocate device vector
thrust::device_vector<int> d_vec(4);

thrust::device_vector<int>::iterator begin = d_vec.begin();
thrust::device_vector<int>::iterator end = d_vec.end();

int length = end - begin; // compute the length of the vector

end = d_vec.begin() + 3; // define a sequence of 3 elements
```

The Thrust Library

Will Landau
Prof. Jarad Niemi

Getting Started

Iterators
Basic iterators
constant_iterator
counting_iterator
transform_iterator
permutation_iterator
zip_iterator

Containers

Algorithms
Transformations
Reductions
Scans
Reordering
Sorting

The Thrust Library

Will Landau
Prof. Jarad Niemi

Getting Started

Iterators
Basic iterators
constant_iterator
counting_iterator
transform_iterator
permutation_iterator
zip_iterator

Containers

Algorithms
Transformations
Reductions
Scans
Reordering
Sorting

```cpp
// allocate device vector
thrust::device_vector<int> d_vec(4);

thrust::device_vector<int>::iterator begin = d_vec.begin();

*begin = 13; // same as d_vec[0] = 13;
int temp = *begin; // same as temp = d_vec[0];

begin++; // advance iterator one position

*begin = 25; // same as d_vec[1] = 25;
```
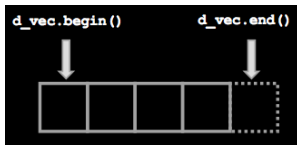
Wrap pointers to make iterators.

The Thrust Library

Will Landau
Prof. Jarad Niemi

Getting Started

Iterators

Basic iterators
constant_iterator
counting_iterator
transform_iterator
permutation_iterator
zip_iterator

Containers

Algorithms
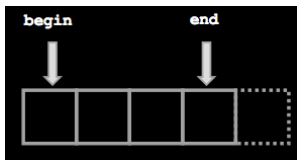
Transformations
Reductions
Scans
Reordering
Sorting

```
int 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_iter(raw_ptr); // dev_iter is now an
    iterator pointing to device memory
thrust::fill(dev_iter, dev_iter + N, (int) 0); // access device memory
    through device_ptr

dev_iter[0] = 1;

// free memory
cudaFree(raw_ptr);
```

Unwrap iterators to extract pointers.

```
// allocate device vector
thrust::device_vector<int> d_vec(4);

// obtain raw pointer to device vector s memory
int * ptr = thrust::raw_pointer_cast(&d_vec[0]); // use ptr in a CUDA C
    kernel
my_kernel<<<N/256, 256>>>(N, ptr);

// Note: ptr cannot be dereferenced on the host!
```

The Thrust Library

Will Landau
Prof. Jarad Niemi

Getting Started

Iterators

Basic iterators
constant_iterator
counting_iterator
transform_iterator
permutation_iterator
zip_iterator

Containers

Algorithms

Transformations
Reductions
Scans
Reordering
Sorting

constant_iterator

The Thrust Library

Will Landau
Prof. Jarad Niemi

Getting Started

Iterators
 Basic iterators
 constant_iterator
 counting_iterator
 transform_iterator
 permutation_iterator
 zip_iterator

Containers

Algorithms
 Transformations
 Reductions
 Scans
 Reordering
 Sorting

A constant_iterator is a pointer with some constant
value associated with it.

```cpp
#include <thrust/iterator/constant_iterator.h>
...
// create iterators
thrust::constant_iterator <int> first(10);
thrust::constant_iterator <int> last = first + 3;

first[0]; // returns 10
first[1]; // returns 10
first[100]; // returns 10

// sum of [first , last)
thrust::reduce(first , last); // returns 30 (i.e. 3 * 10)
```

counting_iterator

The Thrust Library

Will Landau
Prof. Jarad Niemi

Getting Started

Iterators
  Basic iterators
  constant_iterator
  counting_iterator
  transform_iterator
  permutation_iterator
  zip_iterator

Containers

Algorithms
  Transformations
  Reductions
  Scans
  Reordering
  Sorting

A counting_iterator is a pointer with the value
some_constant + offset associated with it.

```
#include <thrust/iterator/counting_iterator.h>
...

// create iterators
thrust::counting_iterator <int> first(10);
thrust::counting_iterator <int> last = first + 3;

first[0]; // returns 10
first[1]; // returns 11
first[100]; // returns 110

// sum of [first, last)
thrust::reduce(first, last); // returns 33 (i.e. 10 + 11 + 12)
```

transform_iterator I

A `transform_iterator` is a pointer with the value
`some_function(vector_entry)` associated with it.

```cpp
#include <thrust/iterator/transform_iterator.h>
...
thrust::device_vector <int> vec(3);
vec[0] = 10;
vec[1] = 20;
vec[2] = 30;

// create iterator
thrust::transform_iterator <int> first =
  thrust::make_transform_iterator(vec.begin(), negate<int>());

thrust::transform_iterator <int> last =
  thrust::make_transform_iterator(vec.end(), negate<int>());

first[0] // returns -10
first[1] // returns -20
first[2] // returns -30

thrust::reduce(first, last); // returns -60 (i.e. -10 + -20 + -30)

//same thing:
thrust::reduce(
  thrust::make_transform_iterator(
    vec.begin(), negate<int>()),
  thrust::make_transform_iterator(
```

transform_iterator II

The Thrust Library

Will Landau
Prof. Jarad Niemi

Getting Started

Iterators
Basic iterators
constant_iterator
counting_iterator
transform_iterator
permutation_iterator
zip_iterator

Containers

Algorithms
Transformations
Reductions
Scans
Reordering
Sorting

```
vec.end(), negate<int>()));
```

permutation_iterator I

A `permutation_iterator` is a pointer associated with a
permuted vector.

```cpp
#include <thrust/iterator/permutation_iterator.h>
...
thrust::device_vector <int> map(4);
map[0] = 3;
map[1] = 1;
map[2] = 0;
map[3] = 5;

thrust::device_vector <int> source(6);
source[0] = 10;
source[1] = 20;
source[2] = 30;
source[3] = 40;
source[4] = 50;
source[5] = 60;

typedef thrust::device_vector<int>::iterator indexIter;

thrust::permutation_iterator<indexIter, indexIter> pbegin =
  thrust::make_permutation_iterator(
    source.begin(), map.begin())

thrust::permutation_iterator<indexIter, indexIter> pend =
  thrust::make_permutation_iterator(
    source.end(), map.end())

int p0 = pbegin[0] // source[map[0]] = 40
```

The Thrust Library

Will Landau
Prof. Jarad Niemi

Getting Started

Iterators
Basic iterators
constant_iterator
counting_iterator
transform_iterator
permutation_iterator
zip_iterator

Containers

Algorithms
Transformations
Reductions
Scans
Reordering
Sorting

permutation_iterator II

The Thrust Library

Will Landau
Prof. Jarad Niemi

Getting Started

Iterators
Basic iterators
constant_iterator
counting_iterator
transform_iterator
permutation_iterator
zip_iterator

Containers

Algorithms
Transformations
Reductions
Scans
Reordering
Sorting

```
int p1 = pbegin[1]  // source[map[1]] = 20

int sum = thrust::reduce(pbegin, pend)

/* sum =
 * source[map[0]] + source[map[1]] + source[map[2]] + source[map[3]] =
 * source[3] + source[1] + source[0] + source[5] =
 * 40 + 20 + 10 + 60 =
 * 130 */
```

zip_iterator I

A zip_iterator is a pointer associated with a vector of tuples.

```cpp
#include <thrust/device_vector.h>
#include <thrust/tuple.h>
#include <thrust/iterator/zip_iterator.h>
#include <iostream>

#include <thrust/iterator/zip_iterator.h>

int main(){
    thrust::device_vector<int> int_v(3);
    int_v[0] = 0; int_v[1] = 1; int_v[2] = 2;

    thrust::device_vector<float> float_v(3);
    float_v[0] = 0.0; float_v[1] = 1.0; float_v[2] = 2.0;

    thrust::device_vector<char> char_v(3);
    char_v[0] = 'a'; char_v[1] = 'b'; char_v[2] = 'c';

    // typedef these iterators for shorthand
    typedef thrust::device_vector<int>::iterator    IntIterator;
    typedef thrust::device_vector<float>::iterator  FloatIterator;
    typedef thrust::device_vector<char>::iterator   CharIterator;

    // typedef a tuple of these iterators
    typedef thrust::tuple<IntIterator, FloatIterator, CharIterator>
        IteratorTuple;
```

The Thrust Library

Will Landau
Prof. Jarad Niemi

Getting Started

Iterators

Basic iterators
constant_iterator
counting_iterator
transform_iterator
permutation_iterator
zip_iterator

Containers

Algorithms

Transformations
Reductions
Scans
Reordering
Sorting

zip_iterator II

```
// typedef the zip_iterator of this tuple
typedef thrust::zip_iterator<IteratorTuple> ZipIterator;

// finally, create the zip_iterator
ZipIterator iter(thrust::make_tuple(int_v.begin(), float_v.begin(),
    char_v.begin()));

*iter;    // returns (0, 0.0, 'a')
iter[0]; // returns (0, 0.0, 'a')
iter[1]; // returns (1, 1.0, 'b')
iter[2]; // returns (2, 2.0, 'c')

thrust::get<0>(iter[2]); // returns 2
thrust::get<1>(iter[0]); // returns 0.0
thrust::get<2>(iter[1]); // returns 'b'

// iter[3] is an out-of-bounds error
}
```

The Thrust Library

Will Landau
Prof. Jarad Niemi

Getting Started

Iterators
Basic iterators
constant_iterator
counting_iterator
transform_iterator
permutation_iterator
zip_iterator

Containers

Algorithms
Transformations
Reductions
Scans
Reordering
Sorting

Outline

The Thrust Library

Will Landau
Prof. Jarad Niemi

Getting Started

Iterators
Basic iterators
constant_iterator
counting_iterator
transform_iterator
permutation_iterator
zip_iterator

Containers

Algorithms
Transformations
Reductions
Scans
Reordering
Sorting

Containers are fancy data storage classes used in the Standard Template Library (STL), the CPU C++ analogue of Thrust. Examples of containers include:

- ► `vector`
- ► `deque`
- ► `list`
- ► `tack`
- ► `queue`
- ► `priority_queue`
- ► `set`
- ► `multiset`
- ► `map`
- ► `multimap`
- ► `biset`

The Thrust Library

Will Landau
Prof. Jarad Niemi

Getting Started

Iterators
Basic iterators
constant_iterator
counting_iterator
transform_iterator
permutation_iterator
zip_iterator

Containers

Algorithms
Transformations
Reductions
Scans
Reordering
Sorting

container.cu I

Thrust only implements vectors, but it's still compatible with the rest of STL's template classes:

```cpp
#include <thrust/device_vector.h>
#include <thrust/copy.h>
#include <list>
#include <vector>

int main(void){
  // create an STL list with 4 values
  std::list<int> stl_list;
  stl_list.push_back(10);
  stl_list.push_back(20);
  stl_list.push_back(30);
  stl_list.push_back(40);

  // initialize a device_vector with the list
  thrust::device_vector <int> D(stl_list.begin(), stl_list.end());

  // copy a device_vector into an STL vector
  std::vector<int> stl_vector(D.size());
  thrust::copy(D.begin(), D.end(), stl_vector.begin());

  return 0;
}
```

Outline

The Thrust Library

Will Landau
Prof. Jarad Niemi

Getting Started

Iterators
Basic iterators
constant_iterator
counting_iterator
transform_iterator
permutation_iterator
zip_iterator

Containers

Algorithms
Transformations
Reductions
Scans
Reordering
Sorting

Transformations

The Thrust Library

Will Landau
Prof. Jarad Niemi

Getting Started

Iterators

Basic iterators
constant_iterator
counting_iterator
transform_iterator
permutation_iterator
zip_iterator

Containers

Algorithms

Transformations
Reductions
Scans
Reordering
Sorting

- A transformation is the application of a function to each element within a range of elements in a vector. The results are stored as a range of elements in another vector.
- Examples:
  - `thrust::fill()`
  - `thrust::sequence()`
  - `thrust::replace()`
  - `thrust::transform()`

transformations.cu I

```cpp
#include <thrust/device_vector.h>
#include <thrust/transform.h>
#include <thrust/sequence.h>
#include <thrust/copy.h>
#include <thrust/fill.h>
#include <thrust/replace.h>
#include <thrust/functional.h>
#include <iostream>

int main(void) {
  // allocate three device_vectors with 10 elements
  thrust::device_vector <int> X(10);
  thrust::device_vector <int> Y(10);
  thrust::device_vector <int> Z(10);

  // initialize X to 0,1,2,3, ....
  thrust::sequence(X.begin(), X.end());

  // compute Y = -X
  thrust::transform(X.begin(), X.end(), Y.begin(), thrust::negate<int>()
      );

  // fill Z with twos
  thrust::fill(Z.begin(), Z.end(), 2);

  // compute Y = X mod 2
  thrust::transform(X.begin(), X.end(), Z.begin(),
                    Y.begin(), thrust::modulus<int>());

  // replace all the ones in Y with tens
```

The Thrust Library

Will Landau
Prof. Jarad Niemi

Getting Started

Iterators

  Basic iterators
  constant_iterator
  counting_iterator
  transform_iterator
  permutation_iterator
  zip_iterator

Containers

Algorithms

  Transformations
  Reductions
  Scans
  Reordering
  Sorting

transformations.cu II

```
thrust::replace(Y.begin(), Y.end(), 1, 10);

// print Y
thrust::copy(Y.begin(), Y.end(), std::ostream_iterator<int>(std::cout,
    "\n"));
return 0;
}
```

Output:

```
[landau@impact1 transformations]$ make
nvcc transformations.cu -o transformations
[landau@impact1 transformations]$ ./transformations
0
10
0
10
0
10
0
10
0
10
[landau@impact1 transformations]$
```

Reductions

A reduction algorithm uses a binary operation to reduce an input vector to a single value. For example, here are equivalent ways to code the pairwise sum:

```
int sum = thrust::reduce(D.begin(), D.end(),
  (int) 0, thrust::plus<int>());

int sum = thrust::reduce(D.begin(), D.end(),
  (int) 0);

int sum = thrust::reduce(D.begin(), D.end())
```

- ▶ The third argument is the starting value of the reduction.
- ▶ The fourth argument is the binary operation that defines the kind of reduction.

The Thrust Library

Will Landau
Prof. Jarad Niemi

Getting Started

Iterators

Basic iterators
constant_iterator
counting_iterator
transform_iterator
permutation_iterator
zip_iterator

Containers

Algorithms

Transformations
Reductions
Scans
Reordering
Sorting

Counting

The Thrust Library

Will Landau
Prof. Jarad Niemi

Getting Started

Iterators
Basic iterators
constant_iterator
counting_iterator
transform_iterator
permutation_iterator
zip_iterator

Containers

Algorithms
Transformations
Reductions
Scans
Reordering
Sorting

Another reduction: use thrust::count() to count the
number of times a value appears in a vector

```cpp
#include <thrust/count.h>
#include <thrust/device_vector.h>
...

// put three 1s in a device_vector
thrust::device_vector <int> vec (5,0);

vec [1] = 1;
vec [3] = 1;
vec [4] = 1;

// count the 1s
int result = thrust::count(vec.begin(), vec.end(), 1);
// result is three
```

Scans

The Thrust Library

Will Landau
Prof. Jarad Niemi

Getting Started

Iterators

Basic iterators
constant_iterator
counting_iterator
transform_iterator
permutation_iterator
zip_iterator

Containers

Algorithms

Transformations
Reductions
Scans
Reordering
Sorting

A scan, also called a prefix-sum, applies a function to
multiple sub-ranges of a vector and returns the result in a
vector of the same size. The default function is addition.

```
#include <thrust/scan.h>
...
int data[6] = {1, 0, 2, 2, 1, 3};
thrust::inclusive_scan(data, data + 6, data);

/* data[0] = data[0]
 * data[1] = data[0] + data[1]
 * data[2] = data[0] + data[1] + data[2]
 * ...
 * data[5] = data[0] + data[1] +     + data[5]
 */

// data is now {1, 1, 3, 5, 6, 9}
```

Scans

There are *exclusive scans* in addition to *inclusive scans*:

```cpp
#include <thrust/scan.h>
...
int data[6] = {1, 0, 2, 2, 1, 3};
thrust::exclusive_scan(data, data + 6, data);

/* data[0] = 0
 * data[1] = data[0]
 * data[2] = data[0] + data[1]
 * ...
 * data[5] = data[0] + data[1] + ... + data[4]
 */

// data is now {0, 1, 1, 3, 5, 6}
```

The Thrust Library

Will Landau
Prof. Jarad Niemi

Getting Started

Iterators
Basic iterators
constant_iterator
counting_iterator
transform_iterator
permutation_iterator
zip_iterator

Containers

Algorithms
Transformations
Reductions
Scans
Reordering
Sorting

Reordering

The Thrust Library

Will Landau
Prof. Jarad Niemi

Getting Started

Iterators

Basic iterators
constant_iterator
counting_iterator
transform_iterator
permutation_iterator
zip_iterator

Containers

Algorithms

Transformations
Reductions
Scans
Reordering
Sorting

The "Reordering" utilities provides subletting and partitioning tools:

- ▶ thrust::copy_if(): copy the elements that make some logical function return true.
- ▶ thrust::partition(); reorder a vector such that values returning true precede values returning false.
- ▶ thrust::remove() and remove_if(): remove elements that return false.
- ▶ thrust::unique(): remove duplicates in a vector.

The Thrust Library

Will Landau
Prof. Jarad Niemi

Getting Started

Iterators

Basic iterators
constant_iterator
counting_iterator
transform_iterator
permutation_iterator
zip_iterator

Containers

Algorithms
Transformations
Reductions
Scans
Reordering
Sorting

```cpp
#include <thrust/partition.h>
...

struct is_even{
  __host__ __device__ bool operator()(const int x){
    return (x % 2) == 0;
  }
};

int main(){
  int A[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
  const int N = sizeof(A)/sizeof(int);
  thrust::partition(A, A + N,
                    is_even());

  // A is now {2, 4, 6, 8, 10, 1, 3, 5, 7, 9}
  return 0;
}
```

Sorting I

The Thrust Library

Will Landau
Prof. Jarad Niemi

Getting Started

Iterators

Basic iterators
constant_iterator
counting_iterator
transform_iterator
permutation_iterator
zip_iterator

Containers

Algorithms

Transformations
Reductions
Scans
Reordering
Sorting

`thrust::sort()`:

```
#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}
```

`thrust::sort_by_key()`:

```
#include <thrust/sort.h>
...
const int N = 6;
int keys[N] = { 1, 4, 2, 8, 5, 7};
char values[N] = {'a', 'b', 'c', 'd', 'e', 'f'};
thrust::sort_by_key(keys, keys + N, values);
// keys is now { 1, 2, 4, 5, 7, 8}
// values is now {'a', 'c', 'b', 'e', 'f', 'd'}
```

thrust::stable_sort()

The Thrust Library

Will Landau
Prof. Jarad Niemi

Getting Started

Iterators

Basic iterators
constant_iterator
counting_iterator
transform_iterator
permutation_iterator
zip_iterator

Containers

Algorithms

Transformations
Reductions
Scans
Reordering
Sorting

:

```cpp
#include <thrust/sort.h>
...
const int N = 6;
int keys[N] = { 1, 4, 2, 8, 5, 7};
char values[N] = {'a', 'b', 'c', 'd', 'e', 'f'};
thrust::sort_by_key(keys, keys + N, values);
// keys is now { 1, 2, 4, 5, 7, 8}
// values is now {'a', 'c', 'b', 'e', 'f', 'd'}
```

Materials

Will Landau
Prof. Jarad Niemi

These slides, a tentative syllabus for the whole lecture series, and all example code are available at
https://github.com/wlandau/gpu.

After logging into you home directory on impact1, type:

```
git clone https://github.com/wlandau/gpu
```

into the command line to download all the materials.

Sources

The Thrust Library

Will Landau
Prof. Jarad Niemi

Getting Started

Iterators
Basic iterators
constant_iterator
counting_iterator
transform_iterator
permutation_iterator
zip_iterator

Containers

Algorithms
Transformations
Reductions
Scans
Reordering
Sorting

1. Bell N. and Hoberock J. Thrust.
   http://developer.download.nvidia.com/CUDA/
   training/webinarthrust1.mp4

2. Savitch W. Absolute C++. Ed. Hirsch M. 3rd Ed.
   Pearson, 2008.

3. CUDA Toolkit 4.2 Thrust Quick Start Guide. March
   2012. http://developer.download.nvidia.com/
   compute/DevZone/docs/html/CUDALibraries/doc/
   Thrust_Quick_Start_Guide.pdf