Thrust

- Integração numérica
- Iterators

Simple Numerical Integration: Example

```
thrust::device_vector<int> width(11, 0.1);
width       =  0.1   0.1   0.1   0.1   0.1   0.1   0.1   0.1   0.1   0.1   0.1

thrust::sequence(x.begin(), x.end(), 0.0f, 0.1f);
x           =  0.0   0.1   0.2   0.3   0.4   0.5   0.6   0.7   0.8   0.9   1.0

thrust::transform(x.begin(), x.end(), height.begin(), square());
height      =  0.0  0.01  0.04  0.09  0.16  0.25  0.36  0.49  0.64  0.81   1.0

thrust::transform(width.begin(), width.end(), height.begin(), area.begin(),
thrust::multiplies<float>())
area        =  0.0 0.001 0.004 0.009 0.016 0.025 0.036 0.049 0.064 0.081   0.1

total_area = thrust::reduce(area.begin(), area.end());
total_area =  0.385
```

```
thrust::inclusive_scan(area.begin(), area.end(), accum_areas.begin());
accum_areas =  0.0 0.001 0.005 0.014 0.030 0.055 0.091 0.140 0.204 0.285 0.385
```

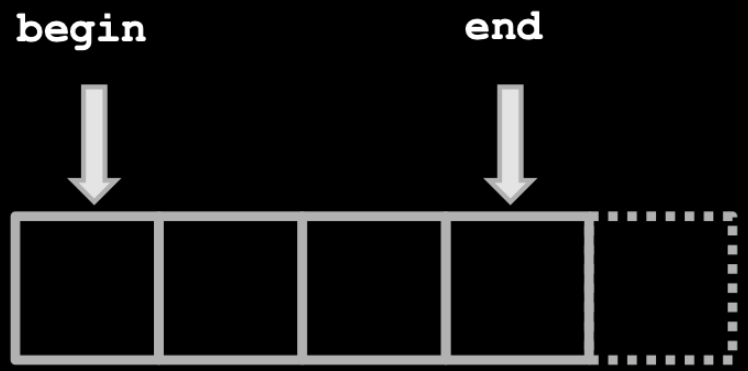
Iterators

```
// 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 size of sequence [begin, end)

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



Constant Iterators

- constant_iterator
  - Mimics an infinite array filled with a constant value

```
// create iterators
constant_iterator<int> begin(10);
constant_iterator<int> end = begin + 3;

begin[0]   // returns 10
begin[1]   // returns 10
begin[100] // returns 10

// sum of [begin, end)
reduce(begin, end);   // returns 30 (i.e. 3 * 10)
```

Counting Iterator

- counting_iterator
  - Mimics an infinite array with sequential values

```
// create iterators
counting_iterator<int> begin(10);
counting_iterator<int> end = begin + 3;

begin[0]   // returns 10
begin[1]   // returns 11
begin[100] // returns 110

// sum of [begin, end)
reduce(begin, end);   // returns 33 (i.e. 10 + 11 + 12)
```

Zip Iterator

zip_iterator

```
// initialize vectors
device_vector<int>  A(3);
device_vector<char> B(3);
A[0] = 10;  A[1] = 20;  A[2] = 30;
B[0] = 'x'; B[1] = 'y'; B[2] = 'z';

// create iterator (type omitted)
begin = make_zip_iterator(make_tuple(A.begin(), B.begin()));
end   = make_zip_iterator(make_tuple(A.end(),   B.end()));

begin[0]   // returns tuple(10, 'x')
begin[1]   // returns tuple(20, 'y')
begin[2]   // returns tuple(30, 'z')

// maximum of [begin, end)
maximum< tuple<int,char> > binary_op;
reduce(begin, end, begin[0], binary_op); // returns tuple(30, 'z')
```