GPU TECHNOLOGY CONFERENCE

High-Productivity CUDA Development with the Thrust Template Library

San Jose Convention Center | September 23rd 2010 | Nathan Bell (NVIDIA Research)

PRESENTED BY NVIDIA.

Diving In

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

int main(void)
{
    // generate 32M random numbers on the host
    thrust::host_vector<int> h_vec(32 << 20);
    thrust::generate(h_vec.begin(), h_vec.end(), rand);

    // transfer data to the device
    thrust::device_vector<int> d_vec = h_vec;

    // sort data on the device (846M keys per sec on GeForce GTX 480)
    thrust::sort(d_vec.begin(), d_vec.end());

    // transfer data back to host
    thrust::copy(d_vec.begin(), d_vec.end(), h_vec.begin());

    return 0;
}
```

GPU TECHNOLOGY CONFERENCE

Objectives

- Programmer productivity
  - Build complex applications quickly
- Encourage generic programming
  - Leverage parallel primitives
- High performance
  - Efficient mapping to hardware

PRESENTED BY NVIDIA.

What is Thrust?

- A template library for CUDA
  - Mimics the C++ STL
- Containers
  - On host and device
- Algorithms
  - Sorting, reduction, scan, etc.

Containers

- Concise and readable code
  - Avoids common memory management errors

```
// allocate host vector with two elements
thrust::host_vector<int> h_vec(2);

// copy host vector to device
thrust::device_vector<int> d_vec = h_vec;

// write device values from the host
d_vec[0] = 13;
d_vec[1] = 27;

// read device values from the host
std::cout << "sum: " << d_vec[0] + d_vec[1] << std::endl;
```

Containers

- Compatible with STL containers

```
// list container on host
std::list<int> h_list;
h_list.push_back(13);
h_list.push_back(27);

// copy list to device vector
thrust::device_vector<int> d_vec(h_list.size());
thrust::copy(h_list.begin(), h_list.end(), d_vec.begin());

// alternative method using vector constructor
thrust::device_vector<int> d_vec(h_list.begin(), h_list.end());
```

Namespaces

- Avoid name collisions

```
// allocate host memory
thrust::host_vector<int> h_vec(10);

// call STL sort
std::sort(h_vec.begin(), h_vec.end());

// call Thrust sort
thrust::sort(h_vec.begin(), h_vec.end());

// for brevity
using namespace thrust;

// without namespace
int sum = reduce(h_vec.begin(), h_vec.end());
```

Iterators

- Pair of iterators defines a *range*

```
// allocate device memory
device_vector<int> d_vec(10);

// declare iterator variables
device_vector<int>::iterator begin  = d_vec.begin();
device_vector<int>::iterator end    = d_vec.end();
device_vector<int>::iterator middle = begin + 5;

// sum first and second halves
int sum_half1 = reduce(begin, middle);
int sum_half2 = reduce(middle, end);

// empty range
int empty = reduce(begin, begin);
```

Iterators

- Iterators act like pointers

```
// declare iterator variables
device_vector<int>::iterator begin = d_vec.begin();
device_vector<int>::iterator end   = d_vec.end();

// pointer arithmetic
begin++;

// dereference device iterators from the host
int a = *begin;
int b = begin[3];

// compute size of range [begin,end)
int size = end - begin;
```

Iterators

- Encode memory location
  - Automatic algorithm selection

```
// initialize random values on host
host_vector<int> h_vec(100);
generate(h_vec.begin(), h_vec.end(), rand);

// copy values to device
device_vector<int> d_vec = h_vec;

// compute sum on host
int h_sum = reduce(h_vec.begin(), h_vec.end());

// compute sum on device
int d_sum = reduce(d_vec.begin(), d_vec.end());
```

Algorithms

- Elementwise operations
  - `for_each`, `transform`, `gather`, `scatter` …
- Reductions
  - `reduce`, `inner_product`, `reduce_by_key` …
- Prefix-Sums
  - `inclusive_scan`, `inclusive_scan_by_key` …
- Sorting
  - `sort`, `stable_sort`, `sort_by_key` …

Algorithms

- Process one or more ranges

```
// copy values to device
device_vector<int> A(10);
device_vector<int> B(10);
device_vector<int> C(10);

// sort A in-place
sort(A.begin(), A.end());

// copy A -> B
copy(A.begin(), A.end(), B.begin());

// transform A + B -> C
transform(A.begin(), A.end(), B.begin(), C.begin(), plus<int>());
```

Algorithms

- Standard operators

```
// allocate memory
device_vector<int>  A(10);
device_vector<int>  B(10);
device_vector<int>  C(10);

// transform A + B -> C
transform(A.begin(), A.end(), B.begin(), C.begin(), plus<int>());

// transform A - B -> C
transform(A.begin(), A.end(), B.begin(), C.begin(), minus<int>());

// multiply reduction
int product = reduce(A.begin(), A.end(), 1, multiplies<int>());
```

Algorithms

- Standard data types

```
// allocate device memory
device_vector<int>   i_vec = ...
device_vector<float> f_vec = ...

// sum of integers
int i_sum = reduce(i_vec.begin(), i_vec.end());

// sum of floats
float f_sum = reduce(f_vec.begin(), f_vec.end());
```

Custom Types & Operators

```
struct negate_float2
{
   __host__ __device__
   float2 operator()(float2 a)
   {
       return make_float2(-a.x, -a.y);
   }
};

// declare storage
device_vector<float2> input  = ...
device_vector<float2> output = ...

// create function object or ‘functor’
negate_float2 func;

// negate vectors
transform(input.begin(), input.end(), output.begin(), func);
```

Custom Types & Operators

```
// compare x component of two float2 structures
struct compare_float2
{
   __host__ __device__
   bool operator()(float2 a, float2 b)
   {
       return a.x < b.x;
   }
};

// declare storage
device_vector<float2> vec = ...

// create comparison functor
compare_float2 comp;

// sort elements by x component
sort(vec.begin(), vec.end(), comp);
```

Custom Types & Operators

```
// return true if x is greater than threshold
struct is_greater_than
{
   int threshold;

   is_greater_than(int t) { threshold = t; }

   __host__ __device__
   bool operator()(int x) { return x > threshold; }
};

device_vector<int> vec = ...

// create predicate functor (returns true for x > 10)
is_greater_than pred(10);

// count number of values > 10
int result = count_if(vec.begin(), vec.end(), pred);
```

Interoperability

- Convert iterators to raw 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!
```

Interoperability

- Wrap raw pointers with `device_ptr`

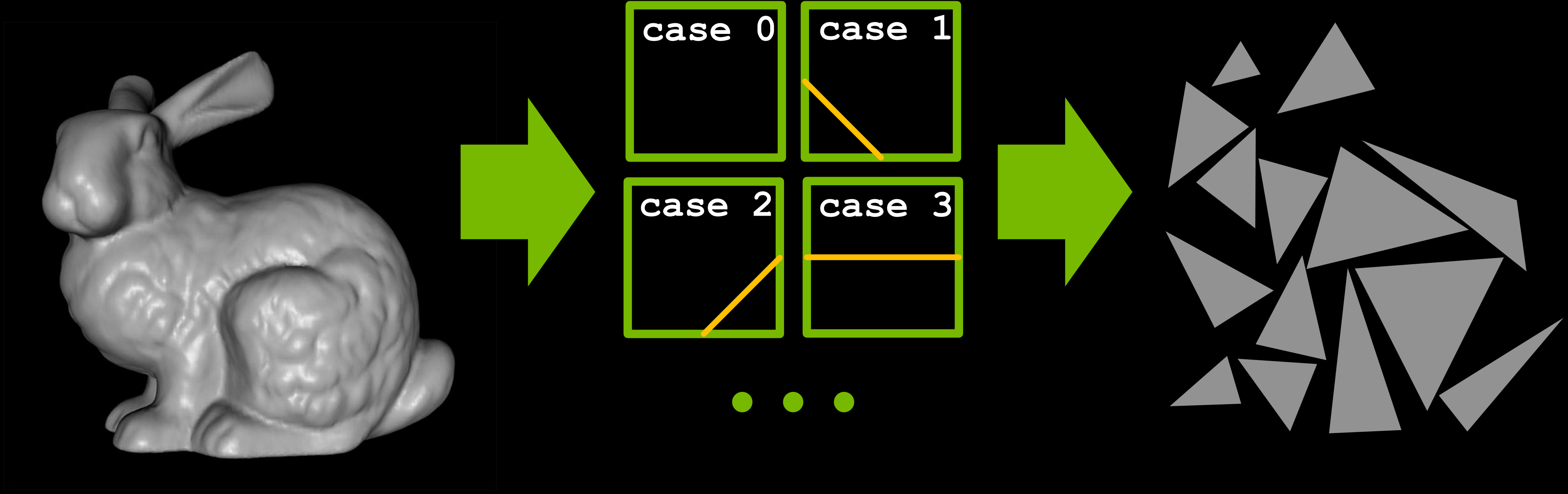
```
// raw pointer to device memory
int * raw_ptr;
cudaMalloc((void **) &raw_ptr, N * sizeof(int));

// wrap raw pointer with a device_ptr
device_ptr<int> dev_ptr(raw_ptr);

// use device_ptr in thrust algorithms
fill(dev_ptr, dev_ptr + N, (int) 0);

// access device memory through device_ptr
dev_ptr[0] = 1;

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

Recap

- Containers manage memory
  - Help avoid common errors
- Iterators define ranges
  - Know where data lives
- Algorithms act on ranges
  - Support general types and operators

Example: Weld Vertices

- Review: Marching Cubes



case 0

case 1

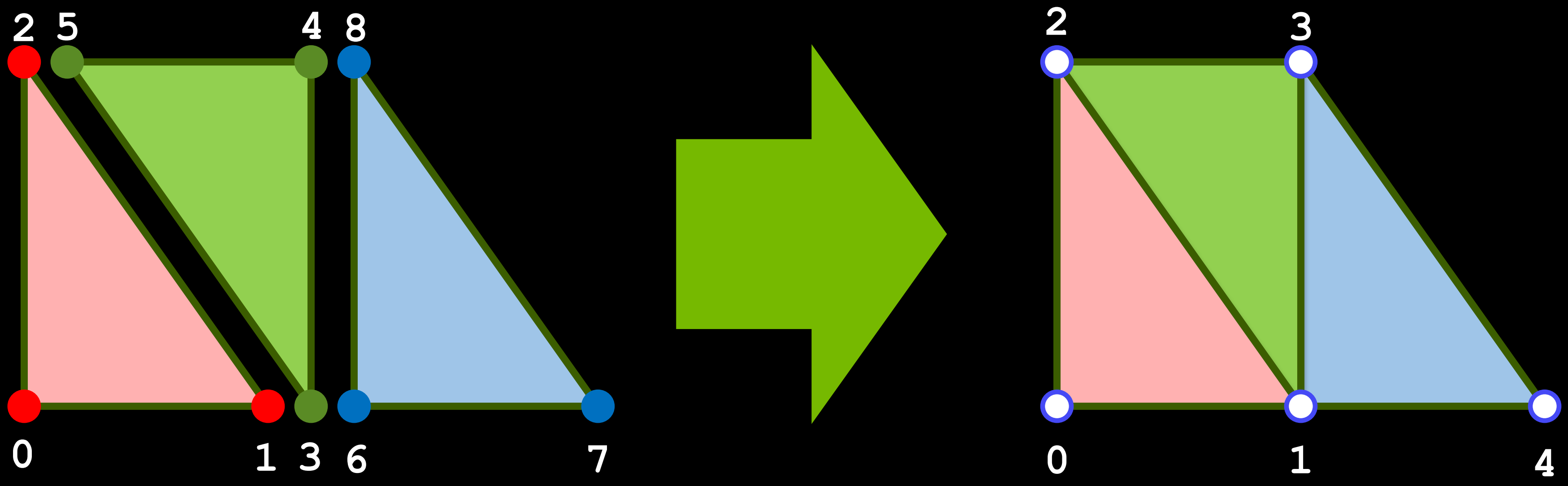
case 2

case 3

• • •

Example: Weld Vertices

- Problem: Marching Cubes produces “triangle soup”
- “Weld” redundant vertices together into a connected mesh



Example: Weld Vertices

Procedure:

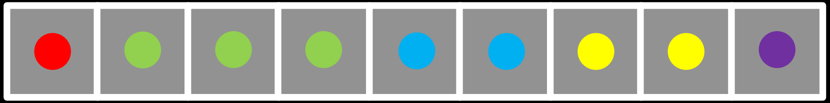
1. Sort triangle vertices
2. Collapse spans of like vertices
3. Search for each vertex’s unique index

Step 1: Sort triangle vertices

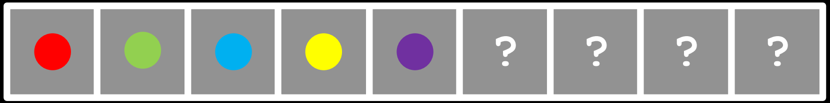
```
// a predicate sorting float2 lexicographically
struct float2_less
{
  __host__ __device__
  bool operator()(float2 a, float2 b)
  {
    if(a.x < b.x) return true;
    if(a.x > b.x) return false;
    return a.y < b.y;
  }
};

// storage for input
device_vector<float2> input = ...

// allocate space for output mesh representation
device_vector<float2> vertices = input;
device_vector<unsigned int> indices;

// sort vertices to bring duplicates together
sort(vertices.begin(), vertices.end(), float2_less());
```

Step 2: Collapse like vertices

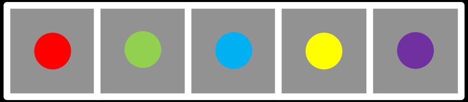
```
// an equivalence relation for float2
struct float2_equal_to
{
  __host__ __device__
  bool operator()(float2 a, float2 b)
  {
    return a.x == b.x && a.y == b.y;
  }
};
```



```
// find unique vertices
device_vector<float2>::iterator new_end;
new_end  = unique(vertices.begin(),
                  vertices.end(),
                  float2_equal_to());
```



```
// erase the redundancies
vertices.erase(new_end, vertices.end());
```

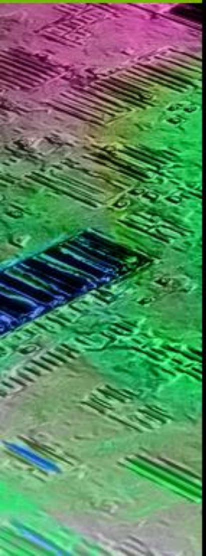


Step 3: Search for vertex indices

```
// find the index of each input vertex in the list of unique vertices
lower_bound(vertices.begin(), vertices.end(),
            input.begin(), input.end(),
            indices.begin(),
            float2_less());
```

Thinking Parallel

- Leverage generic algorithms
  - Sort, reduce, scan, etc.
  - Often faster than application-specific algorithms
- Best practices
  - Use fusion to conserve memory bandwidth
  - Consider memory layout tradeoffs
  - Attend *Thrust By Example* session for details!



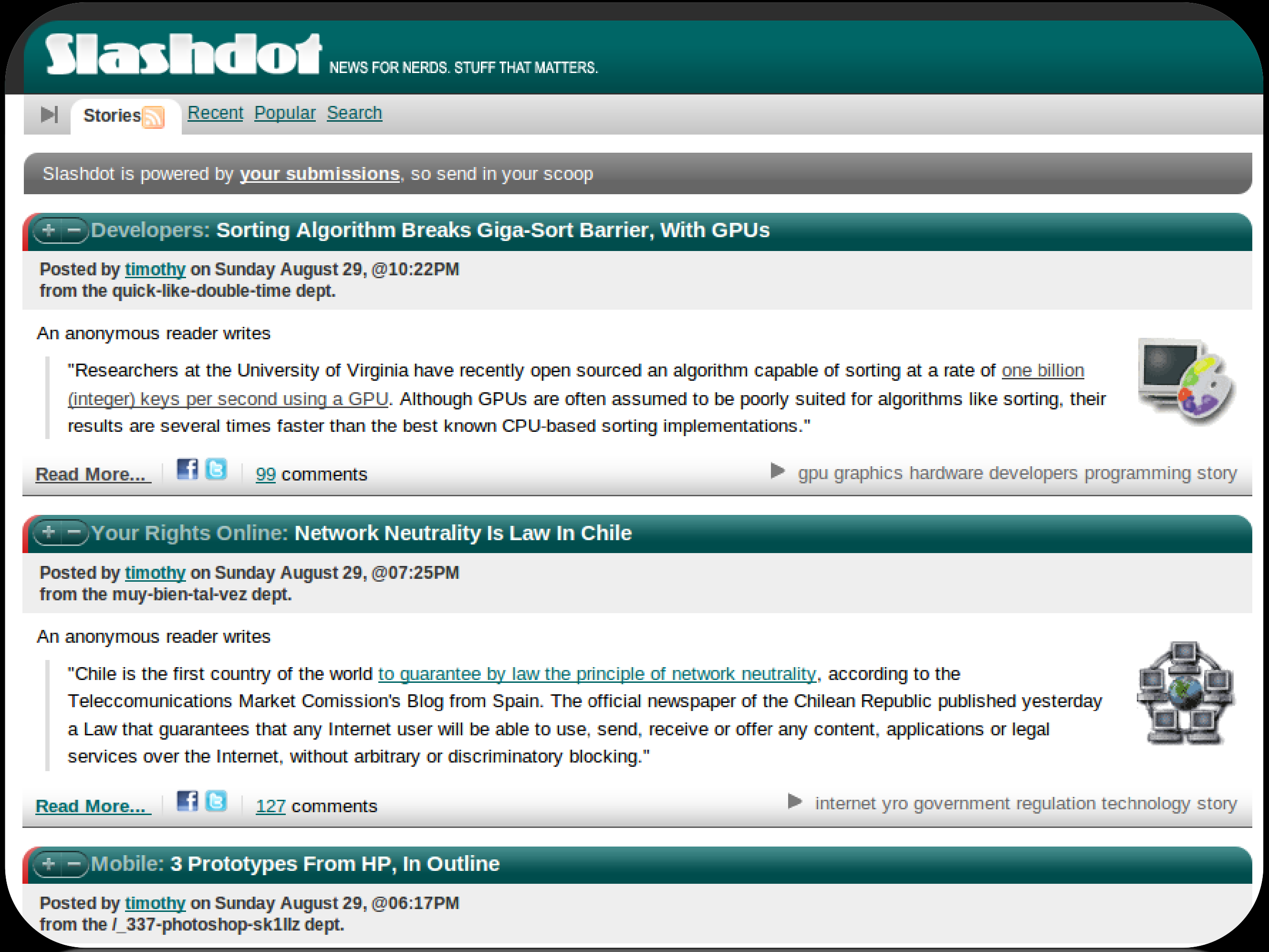
Leveraging Parallel Primitives

- Use `sort` liberally

data type	std::sort	tbb::parallel_sort	thrust::sort
`char`	25.1	68.3	3532.2
`short`	15.1	46.8	1741.6
`int`	10.6	35.1	804.8
`long`	10.3	34.5	291.4
`float`	8.7	28.4	819.8
`double`	8.5	28.2	358.9

Intel Core i7 950

NVIDIA GeForce 480



Slashdot is powered by your submissions, so send in your scoop

Developers: Sorting Algorithm Breaks Giga-Sort Barrier, With GPUs

Posted by timothy on Sunday August 29, @10:22PM
from the quick-like-double-time dept.

An anonymous reader writes

> "Researchers at the University of Virginia have recently open sourced an algorithm capable of sorting at a rate of one billion (integer) keys per second using a GPU. Although GPUs are often assumed to be poorly suited for algorithms like sorting, their results are several times faster than the best known CPU-based sorting implementations."

Read More... | 99 comments | gpu graphics hardware developers programming story

Your Rights Online: Network Neutrality Is Law In Chile

Posted by timothy on Sunday August 29, @07:25PM
from the muy-bien-tal-vez dept.

An anonymous reader writes

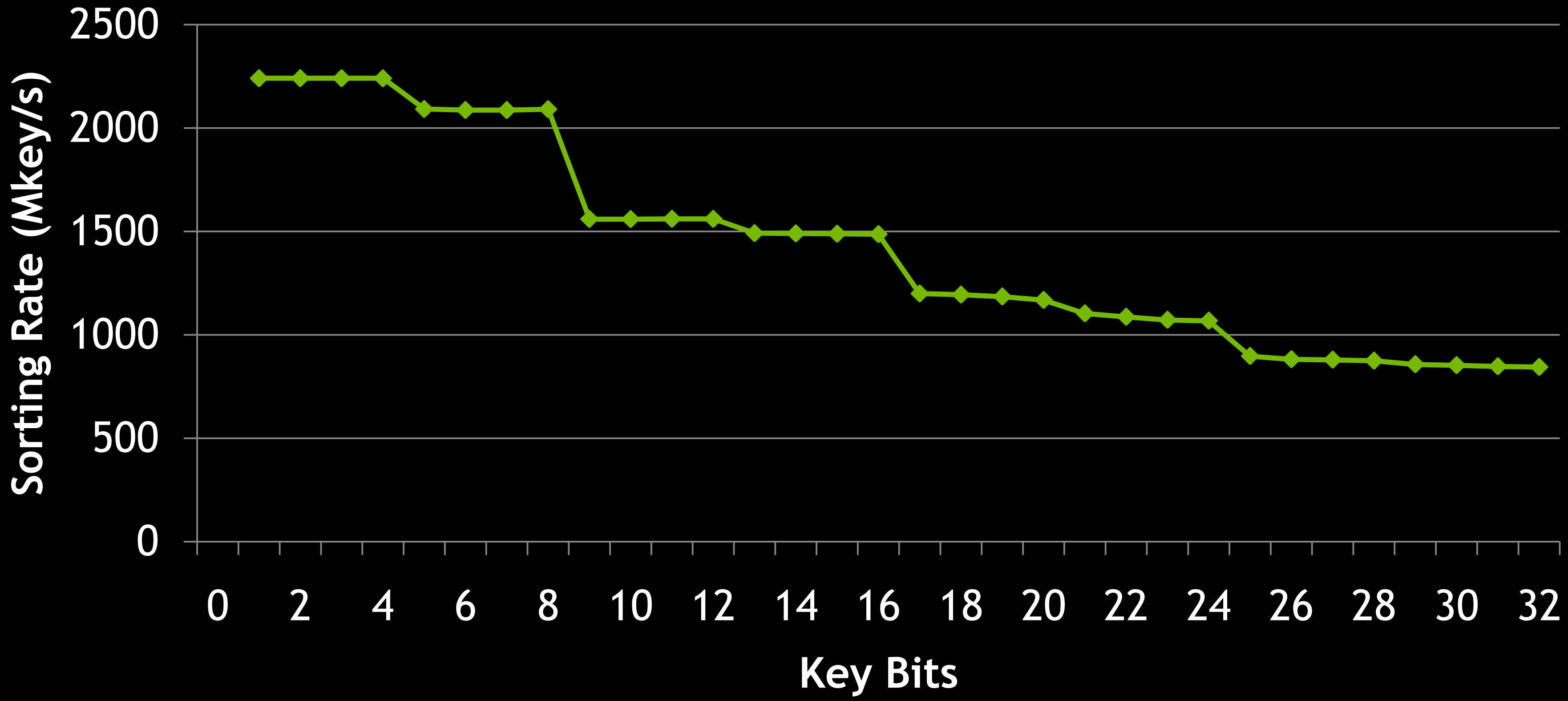
> "Chile is the first country of the world to guarantee by law the principle of network neutrality, according to the Teleccomunications Market Comission's Blog from Spain. The official newspaper of the Chilean Republic published yesterday a Law that guarantees that any Internet user will be able to use, send, receive or offer any content, applications or legal services over the Internet, without arbitrary or discriminatory blocking."

Read More... | 127 comments | internet yro government regulation technology story

Mobile: 3 Prototypes From HP, In Outline

Posted by timothy on Sunday August 29, @06:17PM
from the /_337-photoshop-sk1llz dept.

Input-Sensitive Optimizations

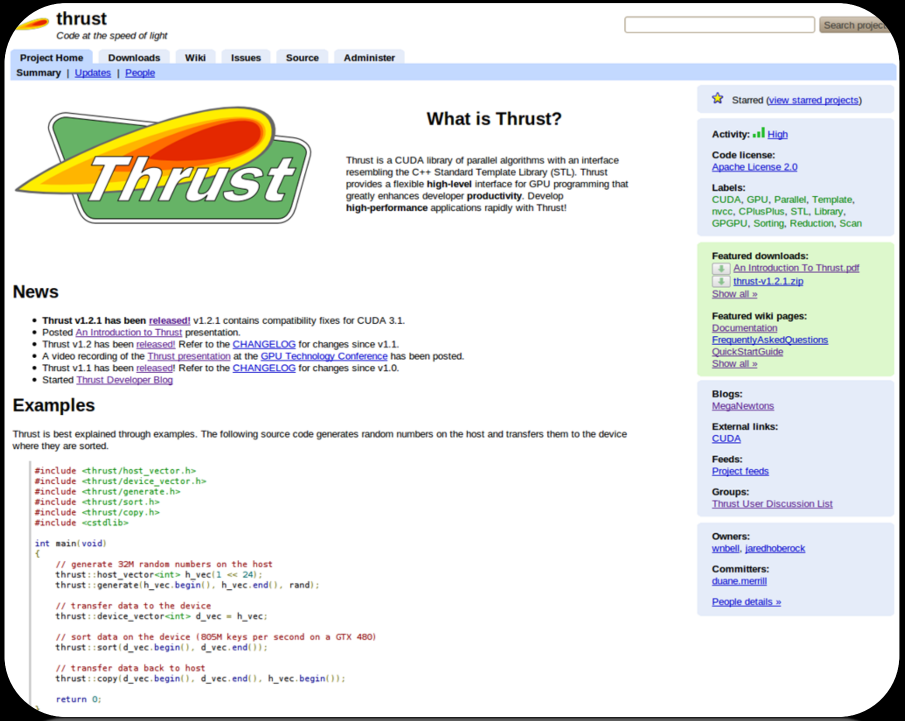


Reductions

Algorithm	Description
`reduce`	Sum of a sequence
`find`	First position of a value in a sequence
`mismatch`	First position where two sequences differ
`inner_product`	Dot product of two sequences
`equal`	Whether two sequences are equal
`min_element`	Position of the smallest value
`count`	Number of instances of a value
`is_sorted`	Whether sequence is in sorted order
`transform_reduce`	Sum of transformed sequence

Thrust on Google Code

- Quick Start Guide
- Examples
- Documentation
- Mailing List (thrust-users)



Starred (view starred projects)

Activity: High

Code license:
Apache License 2.0

Labels:
CUDA, GPU, Parallel, Template, nvcc, CPlusPlus, STL, Library, GPGPU, Sorting, Reduction, Scan

Featured downloads:
An Introduction To Thrust.pdf
thrust-v1.2.1.zip
Show all »

Featured wiki pages:
Documentation
FrequentlyAskedQuestions
QuickStartGuide
Show all »

Blogs:
MegaNewtons

External links:
CUDA

Feeds:
Project feeds

Groups:
Thrust User Discussion List

Owners:
wnbell, jaredhoberock

Committers:
duane.merrill

People details »

What is Thrust?

Thrust is a CUDA library of parallel algorithms with an interface resembling the C++ Standard Template Library (STL). Thrust provides a flexible **high-level** interface for GPU programming that greatly enhances developer **productivity**. Develop **high-performance** applications rapidly with Thrust!

News

- **Thrust v1.2.1 has been released!** v1.2.1 contains compatibility fixes for CUDA 3.1.
- Posted An Introduction to Thrust presentation.
- Thrust v1.2 has been released! Refer to the CHANGELOG for changes since v1.1.
- A video recording of the Thrust presentation at the GPU Technology Conference has been posted.
- Thrust v1.1 has been released! Refer to the CHANGELOG for changes since v1.0.
- Started Thrust Developer Blog

Examples

Thrust is best explained through examples. The following source code generates random numbers on the host and transfers them to the device where they are sorted.

```
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>
#include <thrust/generate.h>
#include <thrust/sort.h>
#include <thrust/copy.h>
#include <cstdlib>

int main(void)
{
    // generate 32M random numbers on the host
    thrust::host_vector<int> h_vec(1 << 24);
    thrust::generate(h_vec.begin(), h_vec.end(), rand);

    // transfer data to the device
    thrust::device_vector<int> d_vec = h_vec;

    // sort data on the device (805M keys per second on a GTX 480)
    thrust::sort(d_vec.begin(), d_vec.end());

    // transfer data back to host
    thrust::copy(d_vec.begin(), d_vec.end(), h_vec.begin());

    return 0;
```

What’s new in Thrust v1.3

- Performance improvements
  - 3x faster sorting
  - 2x faster stream compaction
- New features
  - Algorithms `mismatch`, `find_if`, `reverse`
  - System exceptions

Can I use Thrust?

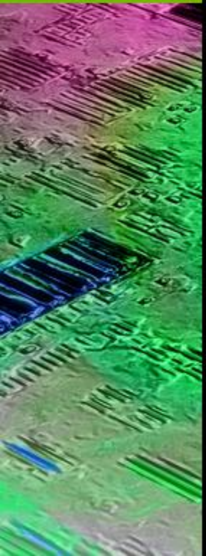
- Extensively tested
  - 600+ unit tests
- Open Source
  - Permissive License (Apache v2)
- Active community
  - 200+ members on cusp-users

Later Today…

2220 - Thrust by Example: Advanced Features and Techniques

Thrust is a parallel template library for developing CUDA applications which is modeled after the C++ Standard Template Library (STL). In this session we'll show how to implement decompose problems into the algorithms provided by Thrust. We'll also discuss the performance implications of "kernel fusion" and "array of structs" vs. "structure of arrays" memory layouts and how they relate to Thrust. Lastly, we'll present evidence that Thrust implementations are fast, while remaining concise and readable

Speaker:	**Jared Hoberock, NVIDIA**
Topic:	**Tools & Libraries**
Time:	**Thursday, September, 23rd, 14:00 - 14:50**
Location:	**Room B**



Acknowledgments

- Andrew Corrigan
- Bastiaan Aarts
- Boost Developers
- Duane Merrill
- Erich Elsen
- Gregory Diamos
- Mark Harris
- Michael Garland
- Nadathur Satish
- CUDA Group
- Ryuta Suzuki
- Shubho Sengupta
- Thomas Bradley