Using Thrust and Cusp

Lesson 3

Nathan Bell

A Productivity-Oriented Library for CUDA

THRUST

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 * 1024 * 1024);
    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;
}
```

Objectives

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

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

- 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

- 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

- 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 v)
   {
       return make_float2(-v.x, -v.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

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
  - See *Thrust By Example* slides for details

Leveraging Parallel Primitives

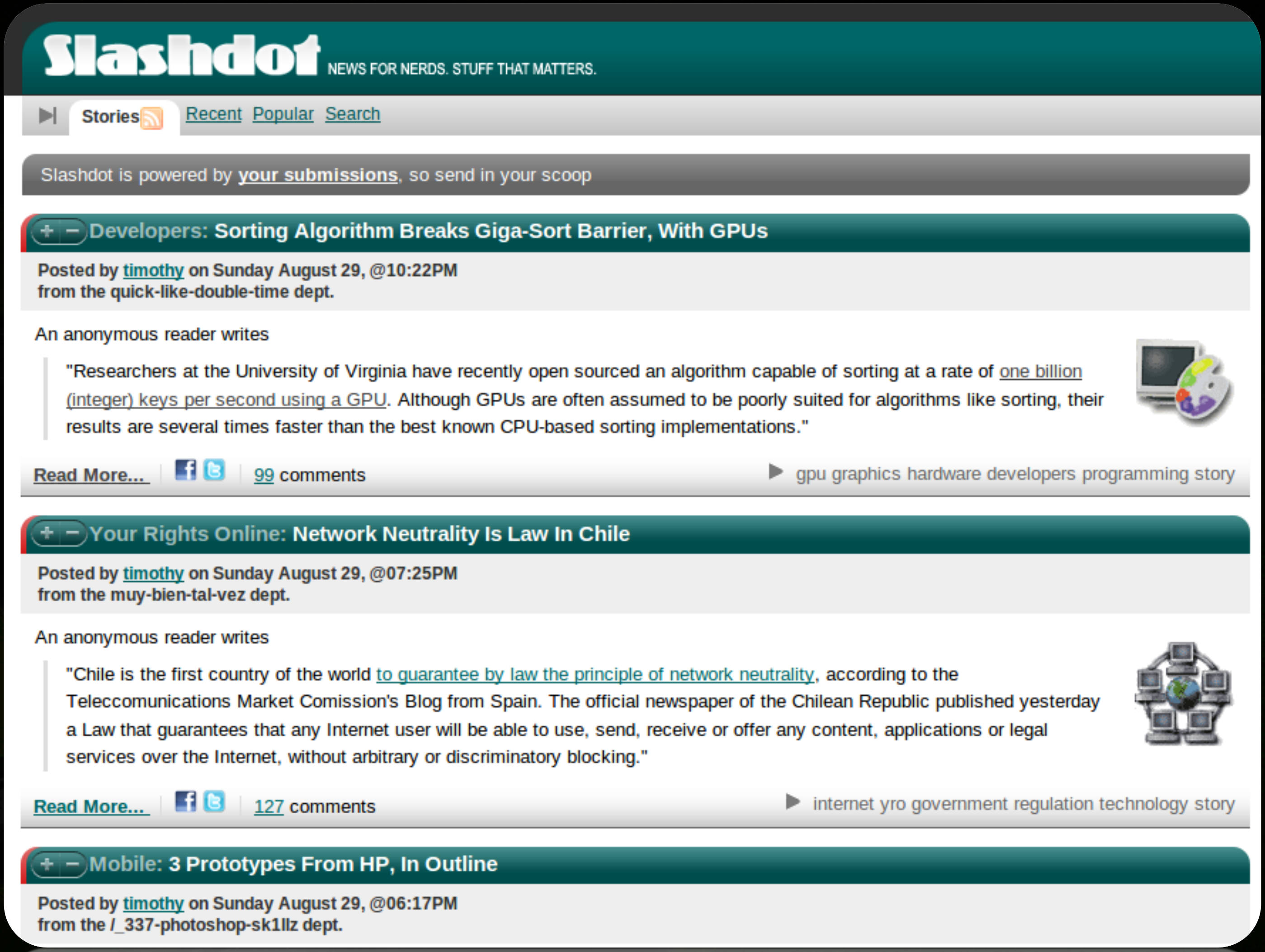
- Use `sort` liberally

Sorting Performance in Millions of Keys / Second

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 NEWS FOR NERDS. STUFF THAT MATTERS.
Stories Recent Popular Search

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."

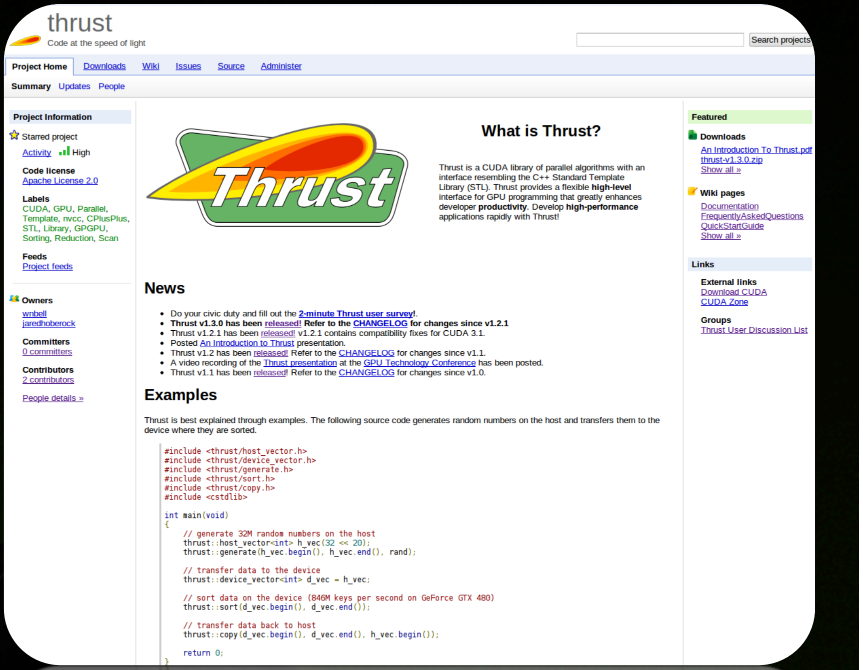
Read More... | 127 comments ▶ internet yro government regulation technology story

Mobile: 3 Prototypes From HP, In Outline

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

Thrust on Google Code

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



Project Information

Starred project
Activity High

Code license
Apache License 2.0

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

Feeds
Project feeds

Owners
wnbell
jaredhoberock

Committers
0 committers

Contributors
2 contributors

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

- Do your civic duty and fill out the **2-minute Thrust user survey!**.
- **Thrust v1.3.0 has been released! Refer to the CHANGELOG for changes since v1.2.1**
- 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.

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(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 second 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;
}
```

Featured

Downloads
An Introduction To Thrust.pdf
thrust-v1.3.0.zip
Show all »

Wiki pages
Documentation
FrequentlyAskedQuestions
QuickStartGuide
Show all »

Links

External links
Download CUDA
CUDA Zone

Groups
Thrust User Discussion List

Generic Parallel Algorithms for

Sparse Matrix and Graph Computations

CUSP

Diving In

```
#include <cusp/hyb_matrix.h>
#include <cusp/io/matrix_market.h>
#include <cusp/krylov/cg.h>

int main(void)
{
    // create an empty sparse matrix structure (HYB format)
    cusp::hyb_matrix<int, float, cusp::device_memory> A;

    // load a matrix stored in MatrixMarket format
    cusp::io::read_matrix_market_file(A, "5pt_10x10.mtx");

    // allocate storage for solution (x) and right hand side (b)
    cusp::array1d<float, cusp::device_memory> x(A.num_rows, 0);
    cusp::array1d<float, cusp::device_memory> b(A.num_rows, 1);

    // solve the linear system A * x = b with the Conjugate Gradient method
    cusp::krylov::cg(A, x, b);

    return 0;
}
```

Sparse Matrix Containers

- COO – Coordinate format
- CSR – Compressed Sparse Row Format
- DIA – Diagonal Format
- ELL – ELLPACK Format
- HYB – Hybrid ELL + COO Format

Dense Containers

```
#include <cusp/array1d.h>
#include <cusp/array2d.h>

int main(void)
{
    // allocate storage for 4 values (uninitialized)
    cusp::array1d<float, cusp::host_memory> A(4, -1.0f);

    // allocate storage for 4 values initialized to -1.0
    cusp::array1d<float, cusp::host_memory> A(4, -1.0f);

    // array1d is just like thrust::{host,device}_vector
    A[0] = 10.0f;  A[1] = 20.0f; A[2] = 30.0f; A[3] = 40.0f;
    B[0] = 10.0f;  B[1] = 20.0f;

    // A now contains the following values
    //    [10 20 30 40]

    // B now contains the following values
    //    [10 20 -1 -1]

    return 0;
}
```

Dense Containers

```
#include <cusp/array1d.h>
#include <cusp/array2d.h>

int main(void)
{
    // allocate storage for (4,3) matrix filled with zeros
    cusp::array2d<float, cusp::host_memory> B(4, 3, 0.0f);

    // set array2d entries on host
    B(0,0) = 10;
    B(0,2) = 20;
    B(2,2) = 30;
    B(3,0) = 40;
    B(3,1) = 50;
    B(3,2) = 60;

    // B now represents the following matrix
    //     [10  0 20]
    //     [ 0  0  0]
    //     [ 0  0 30]
    //     [40 50 60]

    return 0;
}
```

Dense Containers

```
#include <cusp/array1d.h>
#include <cusp/array2d.h>

int main(void)
{
    // allocate storage for (4,3) matrix filled with zeros
    cusp::array2d<float, cusp::host_memory, cusp::column_major> B(4, 3, 0.0f);

    // set array2d entries on host
    B(0,0) = 10;
    B(0,2) = 20;
    B(2,2) = 30;
    B(3,0) = 40;
    B(3,1) = 50;
    B(3,2) = 60;

    // B now represents the following matrix, stored in column-major order
    //    [10  0 20]
    //    [ 0  0  0]
    //    [ 0  0 30]
    //    [40 50 60]

    return 0;
}
```

Sparse Matrix Containers

```
#include <cusp/coo_matrix.h>

int main(void)
{
    // allocate storage for (4,3) matrix with 6 nonzeros
    cusp::coo_matrix<int, float, cusp::host_memory> A(4,3,6);

    // initialize matrix entries on host
    A.row_indices[0] = 0; A.column_indices[0] = 0; A.values[0] = 10.0f;
    A.row_indices[1] = 0; A.column_indices[1] = 2; A.values[1] = 20.0f;
    A.row_indices[2] = 2; A.column_indices[2] = 2; A.values[2] = 30.0f;
    A.row_indices[3] = 3; A.column_indices[3] = 0; A.values[3] = 40.0f;
    A.row_indices[4] = 3; A.column_indices[4] = 1; A.values[4] = 50.0f;
    A.row_indices[5] = 3; A.column_indices[5] = 2; A.values[5] = 60.0f;

    // A now represents the following matrix
    //    [10  0 20]
    //    [ 0  0  0]
    //    [ 0  0 30]
    //    [40 50 60]

    return 0;
}
```

Format Conversion

```
#include <cusp/coo_matrix.h>

int main(void)
{
    // allocate storage for (4,3) matrix with 6 nonzeros
    cusp::coo_matrix<int, float, cusp::host_memory> A(4,3,6);

    // initialize matrix entries on host
    A.row_indices[0] = 0; A.column_indices[0] = 0; A.values[0] = 10.0f;
    A.row_indices[1] = 0; A.column_indices[1] = 2; A.values[1] = 20.0f;
    A.row_indices[2] = 2; A.column_indices[2] = 2; A.values[2] = 30.0f;
    A.row_indices[3] = 3; A.column_indices[3] = 0; A.values[3] = 40.0f;
    A.row_indices[4] = 3; A.column_indices[4] = 1; A.values[4] = 50.0f;
    A.row_indices[5] = 3; A.column_indices[5] = 2; A.values[5] = 60.0f;

    // convert COO->CSR on the host and transfer to the device
    cusp::csr_matrix<int, float, cusp::device_memory> B = A;

    // convert CSR->ELL on the device
    cusp::ell_matrix<int, float, cusp::device_memory> C;
    cusp::convert(B, C);

    return 0;
}
```

Input / Output

```
#include <cusp/coo_matrix.h>
#include <cusp/io/matrix_market.h>

int main(void)
{
    // allocate empty COO container
    cusp::coo_matrix<int, float, cusp::device_memory> A;

    // load a matrix stored in MatrixMarket format
    cusp::io::read_matrix_market_file(A, “my_matrix.mtx");

    ...

    // store a matrix in MatrixMarket format
    cusp::io::write_matrix_market_file(B, “some_file.mtx");

    return 0;
}
```

Algorithms

```
#include <cusp/coo_matrix.h>
#include <cusp/array1d.h>
#include <cusp/multiply.h>

int main(void)
{
    size_t M   = 10;
    size_t N   = 15;
    size_t NNZ = 43;

    // allocate 10x15 COO matrix and vectors
    cusp::coo_matrix<int, float, cusp::device_memory> A(M, N, NNZ);
    cusp::array1d<float, cusp::device_memory> x(N);
    cusp::array1d<float, cusp::device_memory> y(M);

    // initialize A and x
    ...

    // compute matrix-vector product y = A * x
    cusp::multiply(A, x, y);

    return 0;
}
```

Algorithms

```
#include <cusp/array1d.h>
#include <cusp/blas.h>

int main(void)
{
    size_t N = 15;

    // allocate vectors
    cusp::array1d<float, cusp::device_memory> x(N);
    cusp::array1d<float, cusp::device_memory> y(N);

    // initialize vectors
    ...

    // compute vector 2-norm ||x||
    float x_norm = cusp::blas::nrm2(x);

    // compute y = y + 3 * x
    cusp::blas::axpy(x, y, 3.0f);

    return 0;
}
```

Algorithms

- Multiply
  - Sparse Matrix * Vector
  - Sparse Matrix * Sparse Matrix
- Level 1 BLAS
- Transpose
- Maximal Independent Sets
- More to come

Solvers

```
#include <cusp/coo_matrix.h>
#include <cusp/array1d.h>
#include <cusp/krylov/cg.h>

int main(void)
{
    size_t N   = 15;
    size_t NNZ = 43;

    // allocate 10x15 COO matrix and vectors
    cusp::coo_matrix<int, float, cusp::device_memory> A(N, N, NNZ);
    cusp::array1d<float, cusp::device_memory> x(N);
    cusp::array1d<float, cusp::device_memory> b(N);

    // initialize A and b
    ...

    // solve A * x = b to default tolerance with CG
    cusp::krylov::cg(A, x, b);

    return 0;
}
```

Monitors

```
// set stopping criteria of default_monitor:
//  iteration_limit    = 100
//  relative_tolerance = 1e-6
cusp::default_monitor<float> monitor(b, 100, 1e-6);

// solve A * x = b to specified tolerance
cusp::krylov::cg(A, x, b, monitor);
```

```
// set stopping criteria of verbose_monitor:
//  iteration_limit    = 100
//  relative_tolerance = 1e-6
cusp::verbose_monitor<float> monitor(b, 100, 1e-6);

// solve A * x = b to specified tolerance
cusp::krylov::cg(A, x, b, monitor);
```

Monitors

- Verbose monitor output

```
  Iteration Number   | Residual Norm
                 0        1.000000e+01
                 1        1.414214e+01
                 2        1.093707e+01
                 3        8.949319e+00
                 4        6.190056e+00
                 5        3.835190e+00
                 6        1.745481e+00
                 7        5.963548e-01
                 8        2.371135e-01
                 9        1.152524e-01
                10        3.134468e-02
                11        1.144415e-02
                12        1.824177e-03
                13        1.758425e-04
                14        5.735052e-06
Successfully converged after 14 iterations.
```

Preconditioners

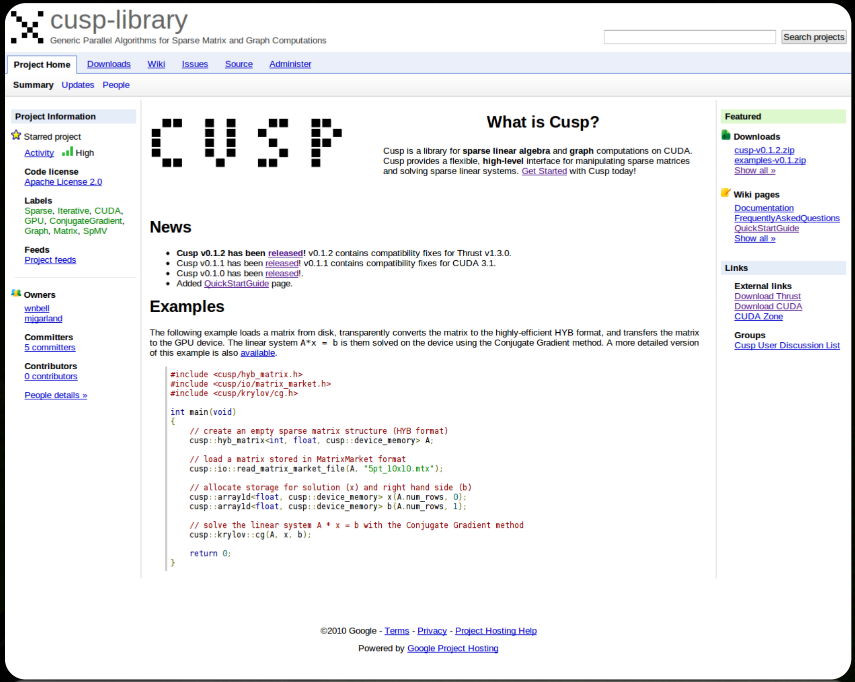
```
#include <cusp/krylov/cg.h>
#include <cusp/precond/smoothed_aggregation.h>

...

// set stopping criteria
//  iteration_limit = 100
//  relative_tolerance = 1e-6
cusp::default_monitor<float> monitor(b, 100, 1e-6);

// setup preconditioner
cusp::precond::smoothed_aggregation<int, float, cusp::device_memory> M(A);

// solve A * x = b to default tolerance with preconditioned CG
cusp::krylov::cg(A, x, b, monitor, M);
```

Views

- Containers “own” their memory
  - Copying containers is expensive
- Views reference other memory
  - Copying views is cheap
- Interfacing
  - External data must be copied into a container
  - Views can wrap external data in-place

Cusp on Google Code

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



cusp-library

Generic Parallel Algorithms for Sparse Matrix and Graph Computations

Project Information

Starred project

Activity High

Code license
Apache License 2.0

Labels
Sparse, Iterative, CUDA, GPU, ConjugateGradient, Graph, Matrix, SpMV

Feeds
Project feeds

Owners
wnbell
mjgarland

Committers
5 committers

Contributors
0 contributors

People details »

What is Cusp?

Cusp is a library for **sparse linear algebra** and **graph** computations on CUDA. Cusp provides a flexible, **high-level** interface for manipulating sparse matrices and solving sparse linear systems. Get Started with Cusp today!

News

- **Cusp v0.1.2 has been released!** v0.1.2 contains compatibility fixes for Thrust v1.3.0.
- Cusp v0.1.1 has been released! v0.1.1 contains compatibility fixes for CUDA 3.1.
- Cusp v0.1.0 has been released!.
- Added QuickStartGuide page.

Examples

The following example loads a matrix from disk, transparently converts the matrix to the highly-efficient HYB format, and transfers the matrix to the GPU device. The linear system `A*x = b` is them solved on the device using the Conjugate Gradient method. A more detailed version of this example is also available.

```
#include <cusp/hyb_matrix.h>
#include <cusp/io/matrix_market.h>
#include <cusp/krylov/cg.h>

int main(void)
{
    // create an empty sparse matrix structure (HYB format)
    cusp::hyb_matrix<int, float, cusp::device_memory> A;

    // load a matrix stored in MatrixMarket format
    cusp::io::read_matrix_market_file(A, "5pt_10x10.mtx");

    // allocate storage for solution (x) and right hand side (b)
    cusp::array1d<float, cusp::device_memory> x(A.num_rows, 0);
    cusp::array1d<float, cusp::device_memory> b(A.num_rows, 1);

    // solve the linear system A * x = b with the Conjugate Gradient method
    cusp::krylov::cg(A, x, b);

    return 0;
}
```

Featured

Downloads
cusp-v0.1.2.zip
examples-v0.1.zip
Show all »

Wiki pages
Documentation
FrequentlyAskedQuestions
QuickStartGuide
Show all »

Links

External links
Download Thrust
Download CUDA
CUDA Zone

Groups
Cusp User Discussion List

©2010 Google - Terms - Privacy - Project Hosting Help

Powered by Google Project Hosting