CUSP – CUSP Library

CUSP is:

- An implementation of generic parallel algorithms for Sparse Matrix and Graph Computations
- An open-source project maintained by NVIDIA Research fellows: Nathan Bell et al.

Key elements

- Sparse Matrices
- Iterative Solvers
- Utilites

Build on top of Thrust Library:

- Raw pointers

Sparse Matrices

- Operates on the most common sparse formats (coo, csr, ell, dia, hyb)
- Transparent conversions between formats and copying from gpu to host and vice versa
- Support of 1D and 2D dense arrays
- Support for reading and writing MatrixMarket files
- Algorithms for multiplication and transposition

Iterative solvers

- State-of-art Krylov subspace methods are available (CG, GMRES, others)
- Monitors for determining convergence criteria (user can provide his own monitors)
- Preconditioners for improving the rate of convergence
- User-defined linear operators

Utilites

- Matrix generators
- Collection of level-1 sparse BLAS routines used by iterative solvers
- Basic support for printing matrix contents
- Matrix format verification

Availability

- Standalone codebase:

```
hg clone
https://code.google.com/p/cusp-library/
```

- Download page of the CUSP project

```
http://code.google.com/p/cusp-library/
downloads/list
```

Example #1: Input/Output

```
#include <cusp/io/matrix_market.h>
#include <cusp/array2d.h>
#include <cusp/coo_matrix.h>
#include <cusp/hyb_matrix.h>
#include <cusp/print.h>

int main(void)
{
    cusp::array2d<float, cusp::host_memory> A(3,4);

    A(0,0) = 10.0;  A(0,1) = 0.0;   A(0,2) = 20.0;  A(0,3) = 0.0;
    A(1,0) = 0.0;   A(1,1) = 30.0;  A(1,2) = 0.0;   A(1,3) = 40.0;
    A(2,0) = 50.0;  A(2,1) = 60.0;  A(2,2) = 70.0;  A(2,3) = 80.0;

    cusp::hyb_matrix<int, float, cusp::host_memory> C = A;

    cusp::io::write_matrix_market_file(C, "A.mtx");

    cusp::coo_matrix<int, float, cusp::device_memory> B;
    cusp::io::read_matrix_market_file(B, "A.mtx");

    cusp::print(B);

    return 0;
}
```

Example #1: Input/Output

```
#include <cusp/io/matrix_market.h>
#include <cusp/array2d.h>
#include <cusp/coo_matrix.h>
#include <cusp/hyb_matrix.h>
#include <cusp/print.h>
```

CUSP include files

```
int main(void)
{
    cusp::array2d<float, cusp::host_memory> A(3,4);

    A(0,0) = 10.0;  A(0,1) = 0.0;   A(0,2) = 20.0;  A(0,3) = 0.0;
    A(1,0) = 0.0;   A(1,1) = 30.0;  A(1,2) = 0.0;   A(1,3) = 40.0;
    A(2,0) = 50.0;  A(2,1) = 60.0;  A(2,2) = 70.0;  A(2,3) = 80.0;

    cusp::hyb_matrix<int, float, cusp::host_memory> C = A;

    cusp::io::write_matrix_market_file(C, "A.mtx");

    cusp::coo_matrix<int, float, cusp::device_memory> B;
    cusp::io::read_matrix_market_file(B, "A.mtx");

    cusp::print(B);

    return 0;
}
```

Example #1: Input/Output

```
#include <cusp/io/matrix_market.h>
#include <cusp/array2d.h>
#include <cusp/coo_matrix.h>
#include <cusp/hyb_matrix.h>
#include <cusp/print.h>

int main(void)
{
    cusp::array2d<float, cusp::host_memory> A(3,4);

    A(0,0) = 10.0;  A(0,1) = 0.0;   A(0,2) = 20.0;  A(0,3) = 0.0;
    A(1,0) = 0.0;   A(1,1) = 30.0;  A(1,2) = 0.0;   A(1,3) = 40.0;
    A(2,0) = 50.0;  A(2,1) = 60.0;  A(2,2) = 70.0;  A(2,3) = 80.0;

    cusp::hyb_matrix<int, float, cusp::host_memory> C = A;

    cusp::io::write_matrix_market_file(C, "A.mtx");

    cusp::coo_matrix<int, float, cusp::device_memory> B;
    cusp::io::read_matrix_market_file(B, "A.mtx");

    cusp::print(B);

    return 0;
}
```

Create simple matrix

Example #1: Input/Output

```
#include <cusp/io/matrix_market.h>
#include <cusp/array2d.h>
#include <cusp/coo_matrix.h>
#include <cusp/hyb_matrix.h>
#include <cusp/print.h>

int main(void)
{
    cusp::array2d<float, cusp::host_memory> A(3,4);

    A(0,0) = 10.0;  A(0,1) = 0.0;   A(0,2) = 20.0;  A(0,3) = 0.0;
    A(1,0) = 0.0;   A(1,1) = 30.0;  A(1,2) = 0.0;   A(1,3) = 40.0;
    A(2,0) = 50.0;  A(2,1) = 60.0;  A(2,2) = 70.0;  A(2,3) = 80.0;

    cusp::hyb_matrix<int, float, cusp::host_memory> C = A;

    cusp::io::write_matrix_market_file(C, "A.mtx");

    cusp::coo_matrix<int, float, cusp::device_memory> B;
    cusp::io::read_matrix_market_file(B, "A.mtx");

    cusp::print(B);

    return 0;
}
```

Transparent matrix format conversions

Example #1: Input/Output

```
#include <cusp/io/matrix_market.h>
#include <cusp/array2d.h>
#include <cusp/coo_matrix.h>
#include <cusp/hyb_matrix.h>
#include <cusp/print.h>

int main(void)
{
    cusp::array2d<float, cusp::host_memory> A(3,4);

    A(0,0) = 10.0;  A(0,1) = 0.0;   A(0,2) = 20.0;  A(0,3) = 0.0;
    A(1,0) = 0.0;   A(1,1) = 30.0;  A(1,2) = 0.0;   A(1,3) = 40.0;
    A(2,0) = 50.0;  A(2,1) = 60.0;  A(2,2) = 70.0;  A(2,3) = 80.0;

    cusp::hyb_matrix<int, float, cusp::host_memory> C = A;

    cusp::io::write_matrix_market_file(C, "A.mtx");

    cusp::coo_matrix<int, float, cusp::device_memory> B;
    cusp::io::read_matrix_market_file(B, "A.mtx");

    cusp::print(B);

    return 0;
}
```

Writing matrix to disk

Example #1: Input/Output

```
#include <cusp/io/matrix_market.h>
#include <cusp/array2d.h>
#include <cusp/coo_matrix.h>
#include <cusp/hyb_matrix.h>
#include <cusp/print.h>

int main(void)
{
    cusp::array2d<float, cusp::host_memory> A(3,4);

    A(0,0) = 10.0;  A(0,1) = 0.0;   A(0,2) = 20.0;  A(0,3) = 0.0;
    A(1,0) = 0.0;   A(1,1) = 30.0;  A(1,2) = 0.0;   A(1,3) = 40.0;
    A(2,0) = 50.0;  A(2,1) = 60.0;  A(2,2) = 70.0;  A(2,3) = 80.0;

    cusp::hyb_matrix<int, float, cusp::host_memory> C = A;

    cusp::io::write_matrix_market_file(C, "A.mtx");

    cusp::coo_matrix<int, float, cusp::device_memory> B;
    cusp::io::read_matrix_market_file(B, "A.mtx");

    cusp::print(B);

    return 0;
}
```

Reading matrix from disc

Example #1: Input/Output

```
#include <cusp/io/matrix_market.h>
#include <cusp/array2d.h>
#include <cusp/coo_matrix.h>
#include <cusp/hyb_matrix.h>
#include <cusp/print.h>

int main(void)
{
    cusp::array2d<float, cusp::host_memory> A(3,4);

    A(0,0) = 10.0;  A(0,1) = 0.0;   A(0,2) = 20.0;  A(0,3) = 0.0;
    A(1,0) = 0.0;   A(1,1) = 30.0;  A(1,2) = 0.0;   A(1,3) = 40.0;
    A(2,0) = 50.0;  A(2,1) = 60.0;  A(2,2) = 70.0;  A(2,3) = 80.0;

    cusp::hyb_matrix<int, float, cusp::host_memory> C = A;

    cusp::io::write_matrix_market_file(C, "A.mtx");

    cusp::coo_matrix<int, float, cusp::device_memory> B;
    cusp::io::read_matrix_market_file(B, "A.mtx");

    cusp::print(B);

    return 0;
}
```

Printing the result to console

Example #1: Input/Output

```
[user@tesla-cmc cusp]$ nvcc matrix_market.cu -o
matrix_market -I /opt/

[user@tesla-cmc cusp]$ ./matrix_market

sparse matrix <3, 4> with 8 entries

                0               0              10
                0               2              20
                1               1              30
                1               3              40
                2               0              50
                2               1              60
                2               2              70
                2               3              80
```

Example #2: GMRES Solver

```
#include <cusp/hyb_matrix.h>
#include <cusp/gallery/poisson.h>
#include <cusp/krylov/gmres.h>

typedef cusp::device_memory MemorySpace;

typedef float ValueType;

int main(void)
{
    cusp::hyb_matrix<int, ValueType, MemorySpace> A;
    cusp::gallery::poisson5pt(A, 10, 10);

    cusp::array1d<ValueType, MemorySpace> x(A.num_rows, ValueType(1));
    cusp::array1d<ValueType, MemorySpace> b(A.num_rows);
    cusp::multiply(A,x,b);

    thrust::fill( x.begin(), x.end(), ValueType(0) );

    cusp::verbose_monitor<ValueType> monitor(b, 100, 1e-6);
    int restart = 50;
    cusp::krylov::gmres(A, x, b, restart, monitor);

    return 0;
}
```

Example #2: GMRES Solver

```
#include <cusp/hyb_matrix.h>
#include <cusp/gallery/poisson.h>
#include <cusp/krylov/gmres.h>

typedef cusp::device_memory MemorySpace;

typedef float ValueType;

int main(void)
{
    cusp::hyb_matrix<int, ValueType, MemorySpace> A;
    cusp::gallery::poisson5pt(A, 10, 10);

    cusp::array1d<ValueType, MemorySpace> x(A.num_rows, ValueType(1));
    cusp::array1d<ValueType, MemorySpace> b(A.num_rows);
    cusp::multiply(A,x,b);

    thrust::fill( x.begin(), x.end(), ValueType(0) );

    cusp::verbose_monitor<ValueType> monitor(b, 100, 1e-6);
    int restart = 50;
    cusp::krylov::gmres(A, x, b, restart, monitor);

    return 0;
}
```

Defining correct rhs vector for the solution to be vector of ones

Example #2: GMRES Solver

```
#include <cusp/hyb_matrix.h>
#include <cusp/gallery/poisson.h>
#include <cusp/krylov/gmres.h>

typedef cusp::device_memory MemorySpace;

typedef float ValueType;

int main(void)
{
    cusp::hyb_matrix<int, ValueType, MemorySpace> A;
    cusp::gallery::poisson5pt(A, 10, 10);

    cusp::array1d<ValueType, MemorySpace> x(A.num_rows, ValueType(1));
    cusp::array1d<ValueType, MemorySpace> b(A.num_rows);
    cusp::multiply(A,x,b);

    thrust::fill( x.begin(), x.end(), ValueType(0) );

    cusp::verbose_monitor<ValueType> monitor(b, 100, 1e-6);
    int restart = 50;
    cusp::krylov::gmres(A, x, b, restart, monitor);

    return 0;
}
```

Using thrust::fill to set initial guess

Example #2: GMRES Solver

```
#include <cusp/hyb_matrix.h>
#include <cusp/gallery/poisson.h>
#include <cusp/krylov/gmres.h>

typedef cusp::device_memory MemorySpace;

typedef float ValueType;

int main(void)
{
    cusp::hyb_matrix<int, ValueType, MemorySpace> A;
    cusp::gallery::poisson5pt(A, 10, 10);

    cusp::array1d<ValueType, MemorySpace> x(A.num_rows, ValueType(1));
    cusp::array1d<ValueType, MemorySpace> b(A.num_rows);
    cusp::multiply(A,x,b);

    thrust::fill( x.begin(), x.end(), ValueType(0) );
    cusp::verbose_monitor<ValueType> monitor(b, 100, 1e-6);
    int restart = 50;
    cusp::krylov::gmres(A, x, b, restart, monitor);

    return 0;
}
```

Setting tolerance and maximum number of iterations

Example #2: GMRES Solver

```
#include <cusp/hyb_matrix.h>
#include <cusp/gallery/poisson.h>
#include <cusp/krylov/gmres.h>

typedef cusp::device_memory MemorySpace;

typedef float ValueType;

int main(void)
{
    cusp::hyb_matrix<int, ValueType, MemorySpace> A;
    cusp::gallery::poisson5pt(A, 10, 10);

    cusp::array1d<ValueType, MemorySpace> x(A.num_rows, ValueType(1));
    cusp::array1d<ValueType, MemorySpace> b(A.num_rows);
    cusp::multiply(A,x,b);

    thrust::fill( x.begin(), x.end(), ValueType(0) );

    cusp::verbose_monitor<ValueType> monitor(b, 100, 1e-6);
    int restart = 50;
    cusp::krylov::gmres(A, x, b, restart, monitor);

    return 0;
}
```

Solving the system

Example #3: Preconditioners

```
#include <cusp/precond/diagonal.h>
#include <cusp/krylov/cg.h>
#include <cusp/csr_matrix.h>
#include <cusp/io/matrix_market.h>
#include <iostream>

typedef cusp::device_memory MemorySpace;
typedef float ValueType;

int main(void)
{
    cusp::csr_matrix<int, ValueType, MemorySpace> A;

    cusp::io::read_matrix_market_file(A, "A.mtx");

    cusp::array1d<ValueType, MemorySpace> x(A.num_rows, 0);
    cusp::array1d<ValueType, MemorySpace> b(A.num_rows, 1);

    cusp::verbose_monitor<ValueType> monitor(b, 100, 1e-6);

    cusp::precond::diagonal<ValueType, MemorySpace> M(A);
    cusp::krylov::cg(A, x, b, monitor, M);

    return 0;
}
```

Example #3: Preconditioners

```
#include <cusp/precond/diagonal.h>
#include <cusp/krylov/cg.h>
#include <cusp/csr_matrix.h>
#include <cusp/io/matrix_market.h>
#include <iostream>

typedef cusp::device_memory MemorySpace;
typedef float ValueType;

int main(void)
{
    cusp::csr_matrix<int, ValueType, MemorySpace> A;

    cusp::io::read_matrix_market_file(A, "A.mtx");

    cusp::array1d<ValueType, MemorySpace> x(A.num_rows, 0);
    cusp::array1d<ValueType, MemorySpace> b(A.num_rows, 1);

    cusp::verbose_monitor<ValueType> monitor(b, 100, 1e-6);

    cusp::precond::diagonal<ValueType, MemorySpace> M(A);
    cusp::krylov::cg(A, x, b, monitor, M);

    return 0;
}
```

Defining preconditioner and passing it to CG method

Interop: CUDA -> Thrust, CUSP

- There's a way to avoid memory overhead when combining cuda kernels and cusp functions:

```
int *dX; cudaMalloc(&dX, 10 * sizeof(int));
thrust::device_ptr<int> wrapped_dX(dX);
cusp::array1d_view<thrust::device_ptr<int>>
cusp_dX (wrapped_dX, wrapped_dX + 10);
```

More examples — on google.code:

- Algorithms
- Gallery
- Matrix formats
- Views
- Linear operator
- Matrix assembly