CUSPARSE Library

Linear algebra for sparse matrices. (A matrix is sparse if there are enough zeros to make it worthwhile to take advantage of them.)

Four types of operations:

Level 1: operations between a vector in sparse format and a vector in dense format

Level 2: operations between a matrix in sparse format and a vector in dense format

Level 3: operations between a matrix in sparse format and vectors in dense format

Conversion: operations that allow conversion between different matrix formats

#include <cusparse_v2.h>

nvcc -arch sm_30 -lcusparse filename.cu

CUSPARSE

The operations of CUSPARSE are done on the device. Assumes that the arrays are on the device through the usual means (cudaMalloc, cudaMemcpy,….)

The library is templatized: can use float (S), double (D), cuComplex (C), cuDoubleComplex(Z) or a generic type (X).

Operations:

cusparse<t>[<matrix data format>]<operation>[<output matrix data format>]

where
t = S,D,C,Z, or X;
<matrix data format> = dense, coo, csr, csc, or hsb, corresponding dense, coordinate, compressed sparse row, compressed sparse column, or hybrid formats;
<operation> = axpyi, doti, dotci, gthr, gthrz, roti, or sctr (Level 1), or mv or sv for Level 2, or mm or sm for Level 3.

Two Matrix Formats

Dense Format

Column-major* format for array A (pointr): m,n = rows and columns, ldx >= m
(if ldx > m, A is just a subset of larger matrix)

Coordinate Format (COO)

For A, nnz is the number of non-zero elements

cooValA is a pointer to array of length nnz containing the nonzero elements
(in **row-major** format)

cooRowIndA is pointer to array of length nnz of the row indices of elements

cooColIndA is pointer to arry of length nnz of the column indices of elements

* FORTRAN format—like CUBLAS

COO Format

$$\begin{bmatrix} 1.0 & 4.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 3.0 & 0.0 & 0.0 \\ 5.0 & 0.0 & 0.0 & 7.0 & 8.0 \\ 0.0 & 0.0 & 9.0 & 0.0 & 6.0 \end{bmatrix}$$

$$\text{cooValA} = [\ 1.0\ \ 4.0\ \ 2.0\ \ 3.0\ \ 5.0\ \ 7.0\ \ 8.0\ \ 9.0\ \ 6.0\ ]$$

$$\text{cooRowIndA} = [\ 0\quad 0\quad 1\quad 1\quad 2\quad 2\quad 2\quad 3\quad 3\ ]$$

$$\text{cooColIndA} = [\ 0\quad 1\quad 1\quad 2\quad 0\quad 3\quad 4\quad 2\quad 4\ ]$$

CUSPARSE

CUSPARSE runs algorithms asynchronously, so that CPU will move on to another process while running.

May need to use cudaDeviceSynchronize() to get CUSPARSE to finish.

Examine cuSparseExample.cu

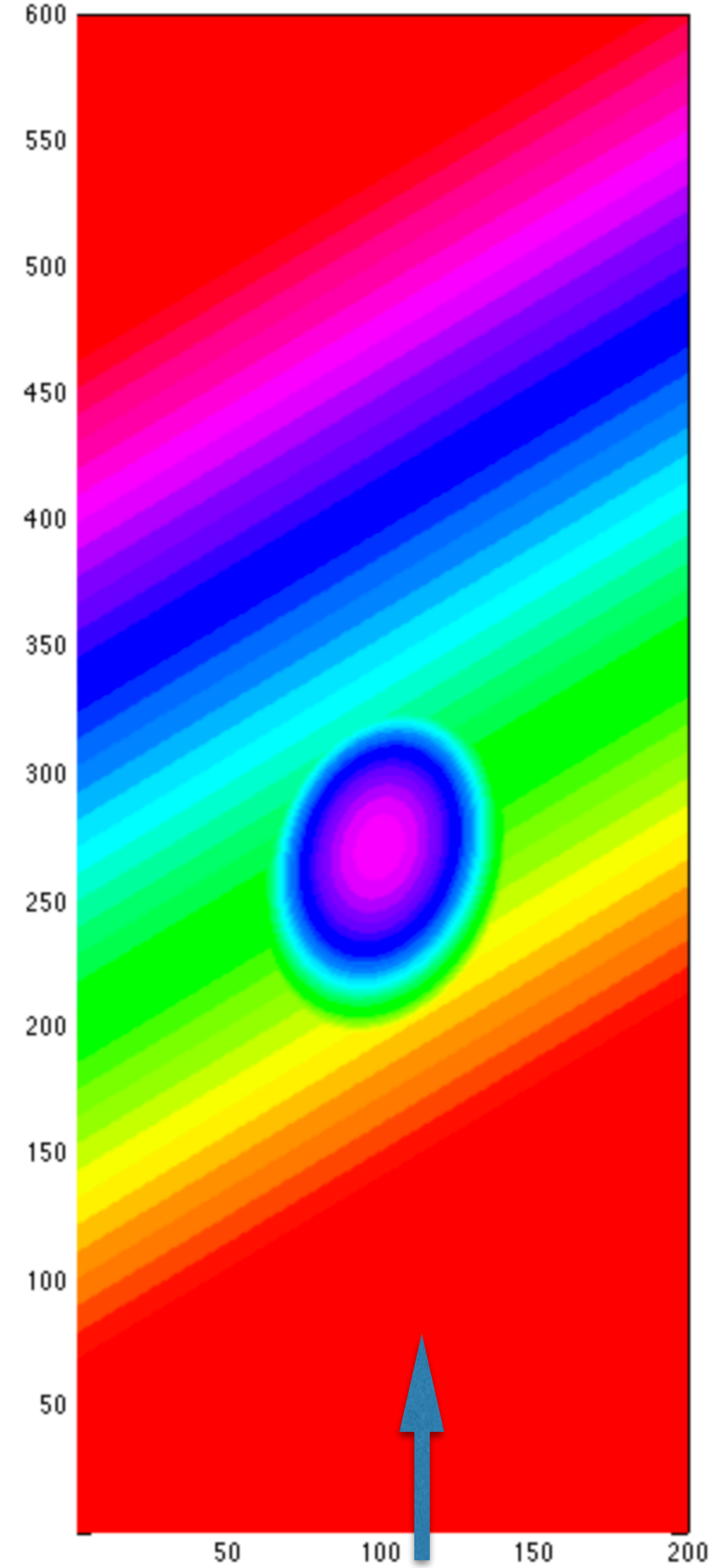
0.08 m

Refraction/Diffraction Problem

Canonical test of waves on a sloping beach with shoal

Waves incident from bottom in x direction

0.45 m



Example of REF/DIF

Solve for water waves over an irregular bathymetry

Illustrate the problem

Set up the simple equation

Mild-slope equation for water waves (Berkhoff, 1972):

$$\nabla_h \cdot \left(C C_g \nabla_h \phi \right) + \sigma^2 \frac{C_g}{C} \phi(x, y) = 0$$

where

$$\Phi(x, y) = \phi(x, y) \, e^{-i\sigma t}$$

∇_h horizontal gradient operator

Mild-slope Equation

Dispersion relationship relates *k(x,y)* to *h(x,y)* and σ :

$$\sigma^2 = gk\tanh(k(x,y)h(x,y))$$

$$C_g = \frac{1}{2}\left(1 + \frac{2kh}{\sinh(2kh)}\right)C$$

$$C = \frac{\sigma}{k}$$

Parabolic Version

Assume the waves propagate in the *x* direction

$$\phi(x, y) = A(x, y)\, e^{ikx}$$

where *A(x,y)* is slowly varying in *x* and *y* (if at all). Substituting:

$$2ikA_x + A_{yy} - kK'|A|^2 A = 0$$

Linearizing by removing cubic nonlinearity:

$$\boxed{2ikA_x + A_{yy} = 0}$$

Note: i is the $\sqrt{-1}$

$$x_i = i\,\Delta x \text{ and } y_j = j\,\Delta y \qquad A_j^i$$

$$A_x = \frac{i}{2k_o} A_{yy}$$

Using Crank-Nicholson approach (forward diff in x; central diff in y):

$$\Big(\frac{A_j^{i+1} - A_j^i}{\Delta x}\Big) = \frac{i}{2k_o}\,\frac{1}{2}\Big(\frac{A_{j+1}^{i+1} - 2A_j^{i+1} + A_{j-1}^{i+1}}{\Delta y^2} + \frac{A_{j+1}^{i} - 2A_j^{i} + A_{j-1}^{i}}{\Delta y^2}\Big)$$

$$O(\Delta x^2, \Delta y^2)$$

Finite Difference Equation

$$-irA_{i+1,j-1} + (1+2ir)A_{i+1,j} - irA_{i+1,j+1} = irA_{i,j-1} + (1-2ir)A_{i,j} + irA_{i,j+1}$$

$$r = \frac{\Delta x}{4k_o \Delta y^2}$$

Left hand side is a tridiagonal matrix for A_{i+1} and right hand side is known.

(Note: 2 i's. One is sqrt(-1), the other is spatial index.)

Complex Numbers

cuComplex
cuFloatComplex

Defined in **cuComplex.h**, which is included by cusparse_v2.h

```
make_cuFloatComplex( float r, float i);       //  z= r + i (i)
 { cuFloatComplex res;
    res.x = r;
    res.y = i;
    return res;
}
```

```
cuCmulf (x,y)
__host__ __device__ static __inline__ cuFloatComplex cuCmulf (cuFloatComplex x,
                                                              cuFloatComplex y)
{
    cuFloatComplex prod;
    prod = make_cuFloatComplex  ((cuCrealf(x) * cuCrealf(y)) -
                                 (cuCimagf(x) * cuCimagf(y)),
                                 (cuCrealf(x) * cuCimagf(y)) +
                                 (cuCimagf(x) * cuCrealf(y)));
    return prod;
}
```

10.3. cusparse<t>gtsv of CUSPARSE_Library

```
cusparseStatus_t
cusparseSgtsv(cusparseHandle_t handle, int m, int n,
              const float           *dl, const float           *d,
              const float           *du, float *B, int ldb)

cusparseStatus_t
cusparseDgtsv(cusparseHandle_t handle, int m, int n,
              const double          *dl, const double          *d,
              const double          *du, double *B, int ldb)
cusparseStatus_t
cusparseCgtsv(cusparseHandle_t handle, int m, int n,
              const cuComplex       *dl, const cuComplex       *d,
              const cuComplex       *du, cuComplex       *B, int ldb)
cusparseStatus_t
cusparseZgtsv(cusparseHandle_t handle, int m, int n,
              const cuDoubleComplex *dl, const cuDoubleComplex  *d,
              const cuDoubleComplex *du, cuDoubleComplex *B, int ldb)
```

This function computes the solution of a tridiagonal linear system

$$A * Y = a * X$$

with multiple right-hand-sides.

The coefficient matrix A of each of these tri-diagonal linear system is defined with three vectors corresponding to its lower (**ld**), main (**d**) and upper (**ud**) matrix diagonals, while the right-hand-sides are stored in the dense matrix X. Notice that the solutions Y overwrite the right-hand-sides X on exit.

Ref/Dif kernels

__global__ void **computeWaveNumber**(float4 *d_dem, float T, int num)

(given depth (d_dem), T = $2\pi/\sigma$

__global__ void **computeMatrix**(cuFloatComplex* d_amp, float4* d_dem, cuFloatComplex* du, cuFloatComplex* dl, cuFloatComplex* d, cuFloatComplex* b, int ncols, int i)

__global__ void **computeBforDamp**(cuFloatComplex* d_amp, cuFloatComplex* b, int ncols, int i)

__global__ void **computeWaves**(cuFloatComplex* d_amp, float* d_intkdx, int nrows, int ncols, unsigned char *ptr)

int main ( )

cusparseCgtsv(handle,ncols, 1, &dl[ncols], &d[ncols], &du[ncols], &b[ncols], ncols);

Graphics

```
 DataBlock   data;
CPUBitmap bitmap(ncols, nrows, &data );
unsigned char    *dev_bitmap;

cudaMalloc( (void**)&dev_bitmap, bitmap.image_size() ) ;


 data.dev_bitmap = dev_bitmap;

cudaMemcpy( bitmap.get_ptr(), dev_bitmap,
                       bitmap.image_size(),
                       cudaMemcpyDeviceToHost ) ;

cudaFree( dev_bitmap ) ;

bitmap.display_and_exit();
```

Compilation

```
nvcc -arch sm_30 -lcusparse -L/System/Library/Frameworks/OpenGL.framework/
        Libraries -lGL -lGLU -Xlinker -framework -Xlinker GLUT  refdif.cu
```