Project 5, Due November 28 at 11:59PM

The code in sparse_matvec.c is a sequential version of a sparse matrix-vector multiply. The matrix is sparse in that many of its elements are zero. Rather than representing all of these zeros which wastes storage, the code uses a representation called Compressed Row Storage (CRS), which only represents the nonzeros with auxiliary data structures to keep track of their location in the full matrix.

I provide:
- Sparse input matrices which were generated from the MatrixMarket (see http://math.nist.gov/MatrixMarket/).
- Sequential code that includes conversion from coordinate matrix to CRS.
- An implementation of dense matvec in CUDA.
- A Makefile for the CADE Linux machines.

You write:
- A CUDA implementation of sparse matvec

Outline

- Sources for this lecture:

Administrative

- CUDA Project 5, due November 28 (no extension)
  - Available on CADE Linux machines (lab1 and lab3) and Windows machines (lab5 and lab6)
  - You can also use your own Nvidia GPUs
Sparse Linear Algebra

• Suppose you are applying matrix-vector multiply and the matrix has lots of zero elements
  - Computation cost? Space requirements?

• General sparse matrix representation concepts
  - Primarily only represent the nonzero data values
  - Auxiliary data structures describe placement of nonzeros in "dense matrix"

Some common representations

DIA: Store elements along a set of diagonals.

Compressed Sparse Row (CSR): Store only nonzero elements, with “ptr” to beginning of each row and “indices” representing column.

ELL: Store a set of K elements per row and pad as needed. Best suited when number non-zeros roughly consistent across rows.

COO: Store nonzero elements and their corresponding “coordinates”

Connect to dense linear algebra

Dense matvec from L18:
for (i=0; i<n; i++) {
  for (j=0; j<n; j++) {
    a[i] += c[j][i] * b[j];
  }
}

Equivalent CSR matvec:
for (i=0; i<n; i++) {
  for (j = ptr[i]; j < ptr[i+1]; j++)
    t[i] += data[j] * b[indices[j]];

Equivalent CSR matvec:
for (i=0; i<n; i++) {
  for (j = ptr[i]; j < ptr[i+1]; j++)
    t[i] += data[j] * b[indices[j]];
}