L20: Sparse Matrix Algorithms, SIMD review

November 15, 2012

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

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 Sforage (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:
  - "Implementing Sparse Matrix-Vector Multiplication on Throughput Oriented Processors," Bell and Garland (Nvidia), SC09, Nov. 2009.

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

$$A = \begin{bmatrix} 1 & 7 & 0 & 0 \\ 0 & 2 & 8 & 0 \\ 5 & 0 & 3 & 9 \\ 0 & 6 & 0 & 4 \end{bmatrix}$$

$$\text{data} = \begin{bmatrix} * & 1 & 7 \\ * & 2 & 8 \\ 5 & 3 & 9 \\ 6 & 4 & * \end{bmatrix} \qquad \text{offsets} = [-2\ 0\ 1]$$

DIA: Store elements along a set of diagonals.

ptr = [0 2 4 7 9]
indices = [0 1 1 2 0 2 3 1 3]
data = [1 7 2 8 5 3 9 6 4]

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

$$\text{data} = \begin{bmatrix} 1 & 7 & * \\ 2 & 8 & * \\ 5 & 3 & 9 \\ 6 & 4 & * \end{bmatrix} \qquad \text{indices} = \begin{bmatrix} 0 & 1 & * \\ 1 & 2 & * \\ 0 & 2 & 3 \\ 1 & 3 & * \end{bmatrix}$$

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

row = [0 0 1 1 2 2 2 3 3]
indices = [0 1 1 2 0 2 3 1 3]
data = [1 7 2 8 5 3 9 6 4]

COO: Store nonzero elements and their corresponding "coordinates".

THE UNIVERSITY OF UTAH

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<nr; i++) {
   for (j = ptr[i]; j<ptr[i+1]-1; j++)
     t[i] += data[j] * b[indices[j]];
```