L10/L11: Sparse Linear Algebra on GPUs

Administrative Issues

- Next assignment, triangular solve
  - Due 5PM, Tuesday, March 5
  - handin cs6235 lab 3 <probfile>"
- Project proposals
  - Due 5PM, Friday, March 8
  - handin cs6235 prop <pdffile>

Triangular Solve (STRSM)

```
for (j = 0; j < n; j++)
   for (k = 0; k < n; k++)
       if (B[j*n+k] != 0.0f) {
            for (i = k+1; i < n; i++)
              B[j*n+i] -= A[k * n + i] * B[j * n + k];
       }
```

Equivalent to:

```
cublasStrsm('l' /* left operator */, 'l' /* lower triangular */,
            'N' /* not transposed */, 'u' /* unit triangular */,
            N, N, alpha, d_A, N, d_B, N);
```

See: http://www.netlib.org/blas/strsm.f

A Few Details

- C stores multi-dimensional arrays in row major order
- Fortran (and MATLAB) stores multi-dimensional arrays in column major order
  - **Confusion alert:** BLAS libraries were designed for FORTRAN codes, so column major order is implicit in CUBLAS!

Dependences in STRSM

```
for (j = 0; j < n; j++)
  for (k = 0; k < n; k++)
     if (B[j*n+k] != 0.0f) {
         for (i = k+1; i < n; i++)
           B[j*n+i] -= A[k * n + i] * B[j * n + k];
     }
```

Which loop(s) “carry” dependences?
Which loop(s) is(are) safe to execute in parallel?

Assignment

- Details:
  – Integrated with simpleCUBLAS test in SDK
  – Reference sequential version provided
1. Rewrite in CUDA
2. Compare performance with CUBLAS library

Performance Issues?

- + Abundant data reuse
- - Difficult edge cases
- - Different amounts of work for different <j,k> values
- - Complex mapping or load imbalance

Project Proposal (due 3/8)

- Team of 2-3 people
  - Please let me know if you need a partner
- Proposal Logistics:
  – Significant implementation, worth 50% of grade
  – Each person turns in the proposal (should be same as other team members)
- Proposal:
  – 3-4 page document (11pt, single-spaced)
  – Submit with handin program:
    “handin CS6235 prop <pdf-file>”

Project Parts (Total = 50%)

- Proposal (5%)
  - Short written document, next few slides
- Design Review (10%)
  - Oral, in-class presentation 3 weeks before end
- Presentation and Poster (15%)
  - Poster session last week of class, dry run week before
- Final Report (20%)
  - Due during finals – no final for this class

Content of Proposal

I. Team members: Name and a sentence on expertise for each member

II. Problem description
  - What is the computation and why is it important?
  - Abstraction of computation: equations, graphic or pseudo-code, no more than 1 page

III. Suitability for GPU acceleration
  - Amdahl's Law: describe the inherent parallelism. Argue that it is close to 100% of computation. Use measurements from CPU execution of computation if possible.
  - Synchronization and Communication: Discuss what data structures may need to be protected by synchronization, or communication through host.
  - Copy Overhead: Discuss the data footprint and anticipated cost of copying to/from host memory.

IV. Intellectual Challenges
  - Generally, what makes this computation worthy of a project?
  - Point to any difficulties you anticipate at present in achieving high speedup

Projects from 2010

1. Green Coordinates for 3D Mesh Deformation
   Timothy George, Andrei Ostanin and Gene Peterson
2. Symmetric Singular Value Decomposition on GPUs using CUDA
   Gagandeep Singh and Vishay Vanjani
3. GPU Implementation of the Immersed Boundary Method
   Dan Maljovec and Varun Shankar
4. GPU Acclerated Particle System Representation for Triangulated Surface Meshes
   Manasi Datar and Brad Petersen
5. Coulombs Law on CUDA
   Torrey Atcitty and Joe Mayo
6. Bidomain Reaction-Diffusion Model
   Jason Briggs and Ayla Khan
7. Graph Coloring using CUDA
   Andre Vincent Pascal Grosset, Shusen Liu and Peihong Zhu
8. Parallelization API Performance Across Heterogeneous Hardware Platforms in Commercial Software Systems
   Toren Monson and Matt Stoker
9. EigenCFA: A CFA for the Lambda-Calculus on a GPU
   Tarun Prabhu and Shreyas Ramalingam
10. Anti-Chess
    Shayan Chandrashekar, Shreyas Subrmanya, Bharath Venkataramani

Projects from 2011

1. Counter Aliasing on CUDA, Dan Parker, Jordan Squire
2. Point Based Animation of Elastic Objects, Ashwin Kumar K and Ashok J
3. Data Fitting for Shape Analysis using CUDA, Qin Liu, Xiaoyue Huang
4. Model-Based Reconstruction of Undersampled DCE-MRI Tumor Data, Ben Felsted, Simon Williams, Cheng Ye
5. Component Streaming on Nvidia GPUs, Sujin Philip, Vince Schuster
6. Compensated Parallel Summation using Kahan's Algorithm, Devin Robison, Yang Gao
7. Implementation of Smoothness-Increasing Accuracy-Conserving Filters for DIscontinuous Galerkin Methods on the GPU, James King, Bharathan Rajaram, Supraja Jayakumar
8. Material Composites Optimization on GPU, Jonathan Bronson, Sheeraj Jadhav, Jihwan Kim
9. Grid-Based Fluid Simulation, Kyle Madsen, Ryan McAlister
10. Graph Drawing with CUDA to Solve the Placement Problem, Shomit Das, Anshul Joshi, Marty Lewis
11. Augmenting Operating Systems with the GPU: The Case of a GPU- augmented encrypted Filesystem, Weibin Sun, Xing Lin
12. Greater Than-Strong Conditional Oblivious Transfer Protocol using Graphics Processing Units, Prarthana Lakshmane Gowda, Nikhil Mishrikoti, Axel Rivera
13. Accelerating Dynamic Binary Translation with GPUs, Chung Hwan Kim, Srikanth Manikarnike, Vaibhav Sharma
14. Online Adaptive Code Generation and Tuning of CUDA Code, Suchit Maindola, Saurav Muralidharan
15. GPU-Accelerated Set-Based Analysis for Scheme, Youngrok Bahn, Seungkeol Choe
16. Containment Analysis on GPU, Anand Venkat, Preethi Kotari, Jacob Johns

Projects from 2012

1. "Cracking DES-Based Encryption Mechanisms Using Massively Parallel Processors," Grant Ayers, Nic McDonald, and Matt Strum.
2. "Point-Based Elastoplastic Simulation with CUDA", Ben Jones and Kyle Hansen.
3. "Traveling Salesman Problem on GPUs", Sidharth Kumar, Shruthi Sreenivas and Abhishek Tripathi.
4. "GPU Accelerated Aggregation for Multigrid", T. James Lewis, Samer Merchant, and Ben Felsted.
5. "Bucket Sorting on GPUs," Andrey Rybalkin, Chris Amevi Adjei and Mohammed Al-Mahfoudh.

Sparse Linear Algebra

$$a_i x_{i-1} + b_i x_i + c_i x_{i+1} = d_i,$$

where $a_1 = 0$ and $c_n = 0$. In matrix form, this system is written as

$$\begin{bmatrix} b_1 & c_1 & & & 0 \\ a_2 & b_2 & c_2 & & \\ & a_3 & b_3 & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ 0 & & & a_n & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_n \end{bmatrix}.$$

http://en.wikipedia.org/wiki/Tridiagonal_matrix_algorithm

GPU Challenges

- Computation partitioning?
- Memory access patterns?
- Parallel reduction

BUT, good news is that sparse linear algebra performs TERRIBLY on conventional architectures, so poor baseline leads to improvements!

Some common representations

DIA

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

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

Stores elements along a set of diagonals.

$$\begin{array}{ll} \texttt{data} & [1\ 2\ 5\ 6\ 7\ 8\ 3\ 4\ *\ *\ 9\ *] \\ \texttt{indices} & [0\ 1\ 0\ 1\ 1\ 2\ 2\ 3\ *\ *\ 3\ *] \\ \text{Iteration 0} & [0\ 1\ 2\ 3\ \quad\quad\quad\quad\quad\quad\quad\quad] \\ \text{Iteration 1} & [\quad\quad\quad\quad 0\ 1\ 2\ 3\ \quad\quad\quad\quad] \\ \text{Iteration 2} & [\quad\quad\quad\quad\quad\quad\quad\quad 0\ 1\ 2\ 3] \end{array}$$

. Each thread iterates over the diagonals
+ Avoids the need to store row/column indices
+ Guarantees coalesced access
- Can be wasteful if lots of padding required

ELL

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

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

-Efficiency rapidly degrades when the number of nonzeros per matrix row varies

$$\begin{array}{ll} \texttt{data} & [*\ *\ 5\ 6\ 1\ 2\ 3\ 4\ 7\ 8\ 9\ *] \\ \text{Iteration 0} & [\quad\quad 2\ 3\ \quad\quad\quad\quad\quad\quad\quad\quad] \\ \text{Iteration 1} & [\quad\quad\quad\quad 0\ 1\ 2\ 3\ \quad\quad\quad\quad] \\ \text{Iteration 2} & [\quad\quad\quad\quad\quad\quad\quad\quad 0\ 1\ 2\ \quad] \end{array}$$

Some common representations

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

CSR

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.

COO

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]

Store nonzero elements and their corresponding "coordinates".

+ Performance is largely insensitive to irregularity in the underlying data structure

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
Iteration 0	0	1	2	3					
Iteration 1					0	1	2	3	
Iteration 2									0

CSR Example

```
for (j=0; j<nr; j++)
   for (k = ptr[j]; k<ptr[j+1]-1; k++)
      t[j] = t[j] + data[k] * x[indices[k]];
```


ELL Example

```
for (j=0; j<nr; j++)
   for (k = 0; k<N; k++) // N is cols in ELL
      t[j] = t[j] + data[j*N+k] * x[indices[k]];
```

Benefits over CSR:
- Load balanced
- Inner loop has fixed bounds, better for compiler optimizations

COO Example

```
for (k=0; k<NNZs; k++)
   t[row[k]] += data[k] * x[indices[k]];
```

Benefits over CSR:
- More intuitive representation
- This approach probably worked better on traditional vector architectures, where access indirection less costly

DIA Example

```
for (j=0; j<ndiags; j++) {
    row = (offset[j] < 0) ? -offset[j] : 0;
    col = (offset[j] > 0) ? offset[j] : 0;
    for (k = 0; k<N; k++) // N is max diag length
       t[row+k] += data[j][k] * x[col+k];
}
```


Summary of Representation and Implementation

			Bytes/Flop	
Kernel	Granularity	Coalescing	32-bit	64-bit
DIA	thread : row	full	4	8
ELL	thread : row	full	6	10
CSR(s)	thread : row	rare	6	10
CSR(v)	warp : row	partial	6	10
COO	thread : nonz	full	8	12
HYB	thread : row	full	6	10

Table 1 from Bell/Garland: Summary of SpMV kernel properties.

Hybrid ELL/COO Format

- Tradeoff between ELL and COO
  - ELL format is well-suited to vector and SIMD
  - ELL efficiency rapidly degrades when the number of nonzeros per matrix row varies
  - Storage efficiency of the COO format is invariant to the distribution of nonzeros per row
- Hybrid format
  - Find a "K" value that works for most of matrix
  - Use COO for rows with more nonzeros (or even significantly fewer)

Other Sparse Matrix Representation Examples

- Blocked CSR
  - Represent non-zeros as a set of blocks, usually of fixed size
  - Within each block, treat as dense and pad block with zeros
  - Block looks like standard gemv
  - So performs well for blocks of decent size with few zeros

$$A = \begin{pmatrix} a_{00} & a_{01} & 0 & 0 & a_{04} & a_{05} \\ a_{10} & a_{11} & 0 & 0 & a_{14} & a_{15} \\ 0 & 0 & a_{22} & 0 & a_{24} & a_{25} \end{pmatrix}$$

$$\mathtt{b\_row\_ptr} = \begin{pmatrix} 0 & 2 & 4 \end{pmatrix}, \mathtt{b\_col\_idx} = \begin{pmatrix} 0 & 4 & 2 & 4 \end{pmatrix}$$

$$\mathtt{b\_value} = \begin{pmatrix} a_{00} & a_{01} & a_{10} & a_{11} & a_{04} & a_{05} & a_{14} & a_{15} & a_{22} & 0 & a_{32} & a_{33} & a_{24} & a_{25} & a_{34} & a_{35} \end{pmatrix}$$

Richard Vuduc, Attila Gyulassy, James W. Demmel, and Katherine A. Yelick. 2003. Memory hierarchy optimizations and performance bounds for sparse A^TAx. In Proceedings of the 2003 international conference on Computational science: PartIII (ICCS'03)

Stencil Example

What is a 3-point stencil? 5-point stencil? 7-point? 9-point? 27-point?

Examples:

a[i] = 2*b[i] – (b[i-1] + b[i+1]);
[-1 2 -1]

a[i][j] = 4*b[i][j] -(b[i-1][j] + b[i+1][j] + b[i][j-1] + b[i][j+1]);
[0 -1 0]
[-1 4 -1]
[0 -1 0]

Stencil Result (structured matrices)

See Figures 11 and 12, Bell and Garland

Unstructured Matrices

See Figures 13 and 14

Note that graphs can also be represented as sparse matrices.
What is an adjacency matrix?

This Lecture

- Exposure to the issues in a sparse matrix vector computation on GPUs
- A set of implementations and their expected performance
- A little on how to improve performance through application-specific knowledge and customization of sparse matrix representation