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)

```c
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](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
   Gogandaep Singh and Vidhay Varanjai
3. GPU Implementation of the Demesured Boundary Method
   Dan Mieljenov and Varun Shankar
4. GPU Accelerated Particle System Representation for Triangulated Surface Meshes
   Manasi Datar and Brad Peterson
5. Coulombs Law on CUDA
   Terrey Atcitty and Joe Mayo
6. Biomass Reaction-Diffusion Model
   Jason Briggs and Ayla Khan
7. Graph Coloring using CUDA
   Andre Vincent Pascal Grosset, Shuven Liu and Peibing 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
   Torun Prabhu and Shreyas Ramalingam
10. Anti-Chess
    Shayan Chandrashekar, Shreyas Subramany, 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 Ya
5. Component Streaming on Nvidia GPUs
   Sojin Philip, Vince Schuster
6. Compensated Parallel Summation using Karan’s Algorithm
   Devin, Robin, Yang Gao
7. Implementation of Smoothness-Increasing Accuracy-Conserving Filters for Discontinuous Galerkin Methods on the GPU
   James King, Bharathan Rajaram, Suprja Jayakumar
8. Material Composites Optimization on GPU
   Jonathan Bronson, Sheeraj Jadhav, Jeevan 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
    Prarthana Lakhotia, Nihal Mistry, Anil Ravish
13. Accelerating Dynamic Binary Translation with GPUs
    Chung Hwan Kim, Srikanth Manikam, Vishal Sharma
14. Online Adaptive Code Generation and Tuning of CUDA Code
    Suchit Maindola, Suchit Maindola, Sourav Muruvelkan
15. GPU-Accelerated Set-Based Analysis for Scheme
    Younguk Bahn, Seungkoo Cheon
16. GateCount Analysis on GPU
    Anand Venkat, Prithi Kotari, Jacob Juhos
Projects from 2012

3. "Traveling Salesman Problem on GPUs", Siddharth Kumar, Shruthi Sreenivas and Abhishek Tripathi.

Sparse Linear Algebra

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

Some common representations

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

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

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

**CSR**
-\[ A = \begin{bmatrix} 1700 \\ 0280 \\ 5039 \\ 0604 \end{bmatrix} \]
-\[ \text{ptr} = [0 2 4 7 9] \]
-\[ \text{indices} = [0 1 2 3 1 3] \]
-\[ \text{data} = [1 7 2 8 5 3 9 6 4] \]

Compressed Sparse Row (CSR):
- Store only nonzero elements, with "ptr" to beginning of each row and "indices" representing column.
- Performance is largely insensitive to irregularity in the underlying data structure.

**COO**
-\[ \text{row} = [0 0 1 2 3 3] \]
-\[ \text{indices} = [0 1 2 3 3] \]
-\[ \text{data} = [1 7 2 8 5 3 9 6 4] \]

- Store nonzero elements and their corresponding "coordinates".

**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

```cpp
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

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Granularity</th>
<th>Coalescing</th>
<th>32-bit</th>
<th>64-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIA</td>
<td>thread : row</td>
<td>full</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>ELL</td>
<td>thread : row</td>
<td>full</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>CSR(s)</td>
<td>thread : row</td>
<td>rare</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>CSR(v)</td>
<td>warp : row</td>
<td>partial</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>COO</td>
<td>thread : nonz</td>
<td>full</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>HYB</td>
<td>thread : row</td>
<td>full</td>
<td>6</td>
<td>10</td>
</tr>
</tbody>
</table>

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

\[
A = \begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{pmatrix}
\]

\[
b = \begin{pmatrix}
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
\end{pmatrix}
\]

Stencil Example

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

Examples:

\[ a[i] = 2 \cdot b[i] - (b[i-1] + b[i+1]); \]
\[ [-1 2 -1] \]

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

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