Cuda-Accelerated ODETLAP
A Parallel Lossy Compression Implementation

Daniel N. Benedetti, W. Randolph Franklin, and Wenli Li

Department of Electrical, Computer, and Systems Engineering
Rensselaer Polytechnic Institute
Troy, NY 12180

October 25, 2013

Partially supported by NSF grant IIS-1117277.
The compression of high-dimensional data

- High-dimensional data collected for various domains
  - Environmental
  - Meteorological
  - CFD
- Large amount of data needs to be compressed
- Utilize auto-correlation in all data dimensions to improve the compression technique
- Allows for the transmission and storage of more data
Compression Basics

Lossy vs. lossless
- Lossless preserves entirety of original data
- Lossy is more compact, but discards some data
- Lossy is acceptable due to limited precision of input data

Typical compression techniques
- Designed with 2D data in mind
- High-dimensional data broken into two-dimensional slices before compression
- Does not utilize auto-correlation beyond two dimensions
Inspired by Laplace’s Equation. In two dimensions:

\[
\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0
\]

Reconstruct data using a small set of points

- Find best fit to an overdetermined linear system
- Construct using two types of equations:
  - Select set of known points:
    \[ z_{ij} = h_{ij} \]
  - Apply Laplace’s Equation to all non-boundary points:
    \[ 4z_{ij} = z_{i-1,j} + z_{i+1,j} + z_{i,j-1} + z_{i,j+1} \]
- Use weighting of the two types of equations to emphasize accuracy or smoothness
Advantages

Benefits to using ODETLAP:

- Preserves local extrema
- Avoids slope discontinuity
- Handles various input types:
  - isolated point data
  - contour data
  - missing-data holes
ODETLAP Process

**Input**
- 400x400 matrix of elevations
- Contour lines
- Any user-supplied points, even inconsistent

**ODETLAP point selection**

**Small point set ~1000**

**ODETLAP terrain reconstruction**

Compressed distributed data

Reconstructed data

**400x400 matrix of elevations**
Emphasis on accuracy
ODETLAP Example

- Emphasis on smoothness

Original Elevation Data

ODETLAP (R = 0.5)
Computational Requirements

- Computationally intensive
  - Need to solve a large, overdetermined, sparse, linear system
    - 400 x 400 data grid with 1000 known points selected requires solution of a 160000 x 161000 sparse system matrix (which is fast to solve)
    - Larger data sets can take a long time to solve
  - Approximate iterative solution is sufficient
  - Improve performance
    - ODE TLAP benefits from parallelization
    - Utilize the power of the GPU
Parallelization

Break problem into smaller tasks

- Run on multiple threads concurrently
- Many calculations can be carried out simultaneously
- A parallel implementation should arrive at an answer faster than a sequential implementation
  - This assumes that the problem can be broken down to a sufficiently large number of tasks
  - Overhead associated with splitting and merging threads must be accounted for
Use CUDA for parallel programming

- Allows for general purpose parallel computation on the GPU using high-level languages
- Libraries useful to ODETLAP available
  - Thrust - STL compliant template library for vector operations, including sorting, transformations, and reductions
  - CUSP - Sparse linear algebra library, containing tools for the manipulation of sparse matrices and the solving of sparse systems
- Can be used on any system with a CUDA-enabled GPU
Original CUDA implementation technique

- Construct ODETLAP sparse linear system matrix using MATLAB
- Compile CUSP C++ code containing linear system solver
- Compile using MATLAB executable (MEX) to call from MATLAB
- Requires a large amount of time for data transfer
Direct CUDA Implementation

Minimize data transfer time
- Construct linear system directly on the GPU
- Use Thrust device vectors for vectors and dense matrices
- Use CUSP coordinate matrices for sparse matrices

Solve using CUSP
- Use Generalized Minimum Residual (GMRES) method
- Provides approximate iterative solution
Computation Time Comparison

2D ODETLAP performance using MATLAB versus using CUDA

- MATLAB runs on 16 CPU threads at 3.1 GHz (Intel Xeon E5-2687)
- CUDA runs on 2688 GPU cores at 732 MHz (NVIDIA Tesla K20X)

<table>
<thead>
<tr>
<th>dataset</th>
<th>MATLAB</th>
<th>CUDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>400x400</td>
<td>16.4 s</td>
<td>2.3 s</td>
</tr>
<tr>
<td>1024x1024</td>
<td>425.6 s</td>
<td>16.7 s</td>
</tr>
<tr>
<td>2048x2048</td>
<td>1556.8 s</td>
<td>49.8 s</td>
</tr>
</tbody>
</table>

- Greater than 30 times faster for the 2048x2048 dataset
- Smaller datasets benefit less from parallelization
  - Larger impact of overhead
  - Less parallelism due to smaller problem size
Conclusion and Future Work

ODETLAP using CUDA

- ODETLAP is computationally intensive
- Use CUDA to make ODETLAP more practical
  - Parallelization greatly reduces computation time
  - Avoid cost of data transfer with direct GPU implementation
  - Size of compressible data set restricted by GPU memory
- Additional optimizations will further improve performance
  - Better utilization of Thrust vector operations
  - Explore point selection techniques
  - Improve handling of boundary points
Original Elevation Data

ODETLAP (R = 0.5)

<table>
<thead>
<tr>
<th>dataset</th>
<th>MATLAB</th>
<th>CUDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>400x400</td>
<td>16.4 s</td>
<td>2.3 s</td>
</tr>
<tr>
<td>1024x1024</td>
<td>425.6 s</td>
<td>16.7 s</td>
</tr>
<tr>
<td>2048x2048</td>
<td>1556.8 s</td>
<td>49.8 s</td>
</tr>
</tbody>
</table>

Input

400x400 matrix of elevations

contour lines

any user-supplied points, even inconsistent

ODETLAP point selection

Small point set ~1000

Compressed distributed data

Reconstructed data

400x400 matrix of elevations