15-618 Final Project

Parallel Eigensolver for Graph Spectral Analysis on GPU

Heran Lin
lin1@andrew.cmu.edu

Yimin Liu
yiminliu@andrew.cmu.edu

Carnegie Mellon University

May 11, 2015

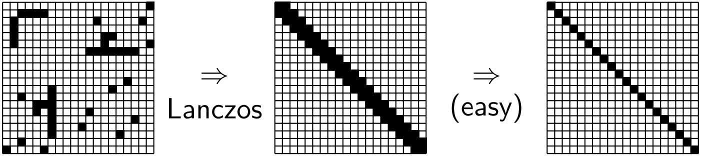
Overview

- Undirected graph $G = (V, E)$
- Symmetric square matrix $\mathbf{M}$ associated with graph G (adjacency matrix $\mathbf{A}$, graph Laplacian $\mathbf{L}$, etc.)
- Eigenvalues of $\mathbf{M}$ encodes interesting properties of the graph

$$\mathbf{M}\mathbf{x} = \lambda\mathbf{x}$$

Eigendecomposition Overview

- Transform $\mathbf{M}$ to a symmetric tridiagonal matrix $\mathbf{T}_m$
- Calculate eigenvalues of $\mathbf{T}_m$



The Lanczos Algorithm for Tridiagonalization

$$\mathbf{T}_m = \begin{pmatrix} \alpha_1 & \beta_2 & & \\ \beta_2 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_m \\ & & \beta_m & \alpha_m \end{pmatrix}$$

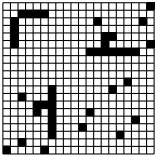
1. $\mathbf{v}_0 \leftarrow \mathbf{0}$, $\mathbf{v}_1 \leftarrow$ norm-1 random vector, $\beta_1 \leftarrow 0$
2. for $j = 1, \ldots, m$
   - $\mathbf{w}_j \leftarrow \mathbf{M}\mathbf{v}_j$
   - $\alpha_j \leftarrow \mathbf{w}_j^\top \mathbf{v}_j$
   - $\mathbf{w}_j \leftarrow \mathbf{w}_j - \alpha_j \mathbf{v}_j - \beta_j \mathbf{v}_{j-1}$
   - $\beta_{j+1} \leftarrow \|\mathbf{w}_j\|_2$
   - $\mathbf{v}_{j+1} \leftarrow \mathbf{w}_j / \beta_{j+1}$

Potential parallelism for CUDA: **matrix-vector product**, dot-product, SAXPY

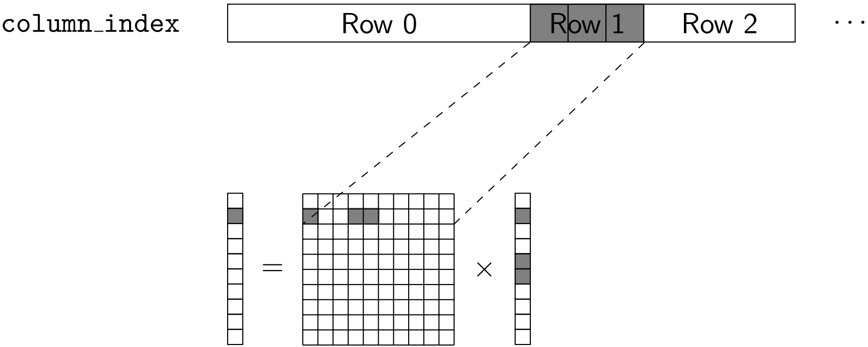
Challenges

Characteristics of $\mathbf{M}$

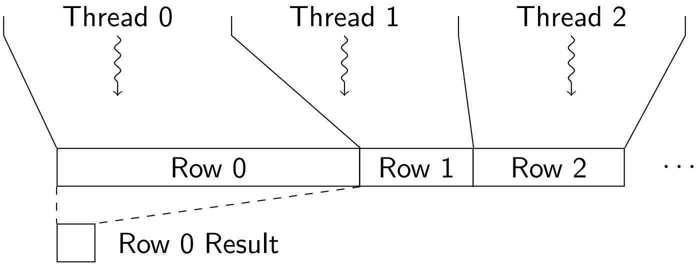
- Really sparse
- Skewed distribution of non-zero elements
  - Example: power-law node degree distribution in social networks



Compressed Sparse Row (CSR) Matrix-Vector Multiplication (SPMV)

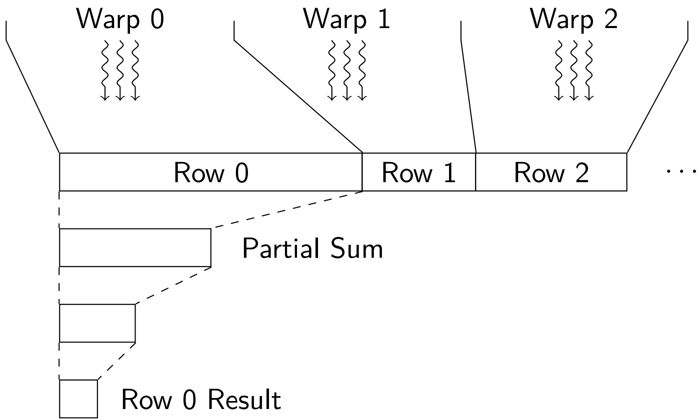


Naive Work Assignment



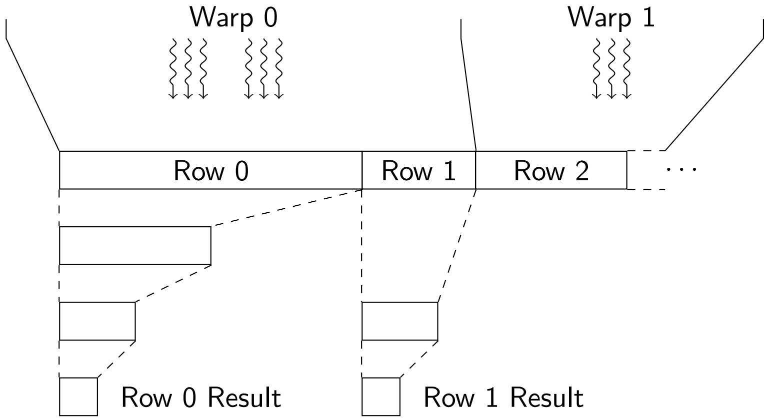
- Each thread is responsible for one row
- Work imbalance issues

Warp-based Work Assignment



- Each warp (32 threads) is responsible for one row
- Reduce partial sum in shared memory

Warp-based Work Assignment for Row Groups



- Each warp is responsible for a group of rows
- Group size depending on the average row sparsity of the matrix

Evaluation Environment

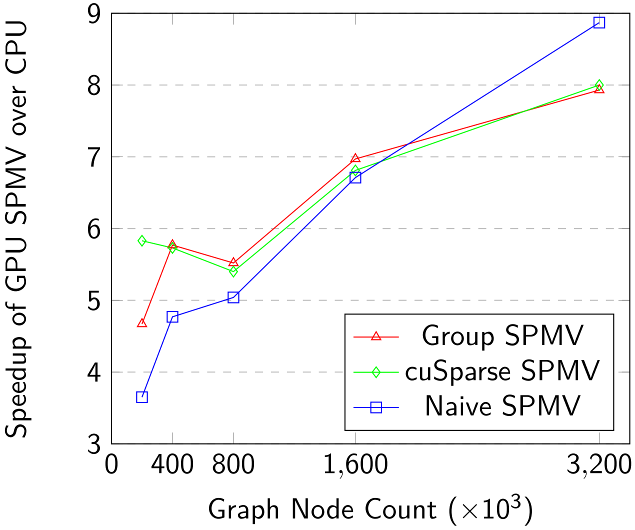
Amazon Web Service EC2 g2.2xlarge

- NVIDIA GK104 GPU, 1,536 CUDA cores, with CUDA 7.0 Toolkit installed
- Intel Xeon E5-2670 CPU, 8 cores, with gcc/g++ 4.8.2 installed, `-O3` optimization switched on

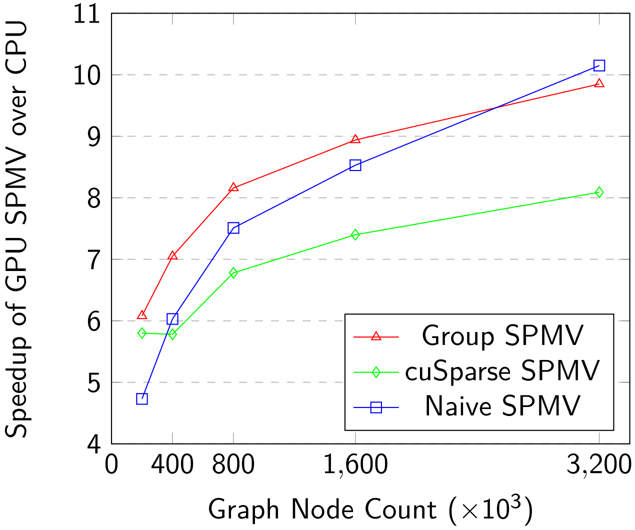
Competitive reference: SPMV implementation in cuSparse (`http://docs.nvidia.com/cuda/cusparse/`)

Dataset: generated scale-free networks based-on the Barabási-Albert model, using Python NetworkX

`float` SPMV Performance Similiar to cuSparse



`double` SPMV Performance Better than cuSparse



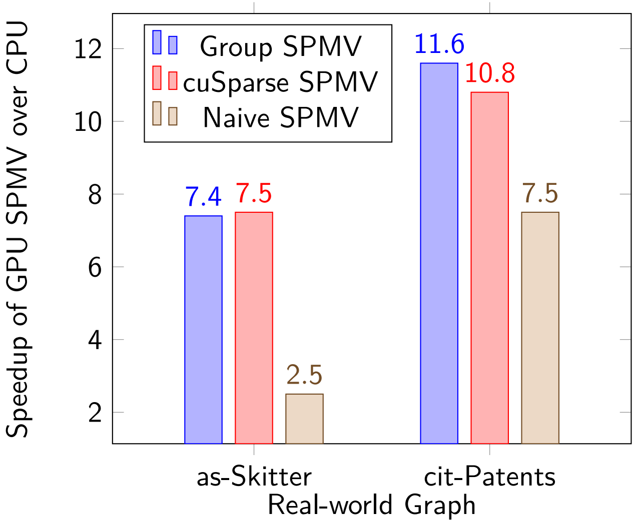
Real-world Graphs

- as-Skitter: $\sim$ 1,700,000 nodes, $\sim$ 11,000,000 edges
- cit-Patents: $\sim$ 3,800,000 nodes, $\sim$ 17,000,000 edges

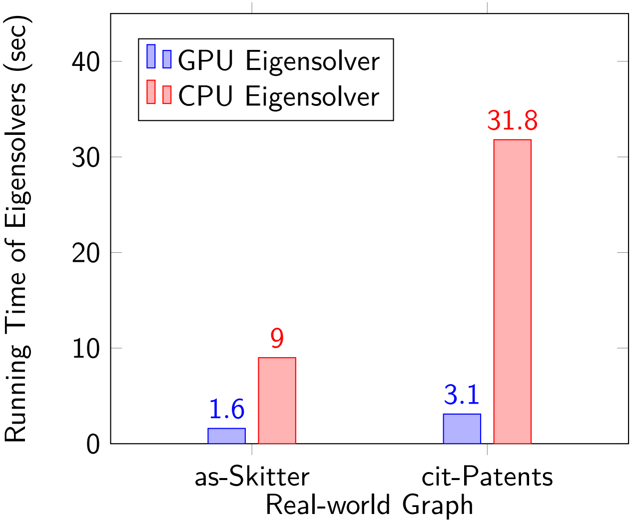
Converted to symmetric `double` adjacency matrices

Data source: SNAP
(`http://snap.stanford.edu/data/index.html`)

SPMV Better than cuSparse on Large Real-world Graphs



Faster Eigenvalue Solver on GPU



Discussion

SLEPc (`http://slepc.upv.es`)

- A state-of-the-art parallel CPU framework using MPI for sparse matrix eigenvalues solving
- Took 84.9 sec to solve 10 largest eigenvalues for the cit-Patents graph, while we took only 31.8 sec on CPU
- Unfair to compare?

- Many variants of the Lanczos algorithm
- Accuracy v.s. performance tradeoff