Distributed Sparse Matrices and MST in Julia

George Xing

Massachusetts Institute Of Technology

December 14, 2011

Overview

- Sparse Matrices
- Distributed Sparse Matrices
- Minimal Spanning Trees and Prim's Algorithm
- Future Work

Sparse Matrices

- We'd like to be able to store the nonzero values of a matrix that has a "lot" of zeros.
- Considerations:
  - memory
  - structure?
  - iteration over nonzero values

Compressed Sparse Column (CSC) Format

- Store indices with nonzero values in dictionary order (column, then row) and their corresponding values in matrix.
- Instead of an $m \times n$ matrix A with nnz nonzero values, maintain three fields: `colptr`, `rowval`, `nzval`
  - `colptr`: array of length $n+1$. Column i's values are in the indices from $\texttt{colptr}[i]$ to $\texttt{colptr}[i+1]-1$.
  - `rowval`: array of length nnz. Stores a row index.
  - `nzval`: array of length nnz. Stores the nonzero value; if $\texttt{colptr}[i] \leq j \leq \texttt{colptr}[i+1]-1$, then $\texttt{nzval}[j] = A[\texttt{rowval}[j], i]$.
- Good: arithmetic operations, general linear algebraic operations, column slicing
- Bad: referencing, assigning, structured matrices

Distributed Sparse Matrices

- Distribute the memory of a (large) sparse matrix over multiple processors. (currently limited to distributing contiguous blocks of columns)
- Each processor stores a local piece (in CSC format), a list of all processors involved, and a mapping of where each piece is located.
    - `pmap` and `dist`, where the i-th block is columns $\texttt{dist}[i]$ to $\texttt{dist}[i+1]-1$ and on processor $\texttt{pmap}[i]$

Operations on Distributed Sparse Matrices

- Goal: make end user not have to think too hard about coordinating parallelism
- Matrix operations automatically coordinate the processors with relevant data (ex. `multiply`, `ref`, `assign`)

The Minimal Cost Spanning Tree Problem

- Given a connected, undirected graph $G = (V, E)$, a *spanning tree* is a subset of edges $T \subseteq E$ such that $|T| = |V| - 1$ and T contains no cycles.
- Given G and a cost function $c : E \to \mathbb{R}$ on the edges, let $c(T) = \sum_{e \in T} c(e)$. We are interested in finding the minimum value of $c(T)$ over all spanning trees T.
- Applications: network design/routing, image segmentation, subroutine for harder problems

Prim's Algorithm (serial)

Given $G = (V, E)$ with cost function c:

- Set $V_{done} = \{v_0\}$ for an arbitrary $v_0 \in V$. Set d to be an array indexed by the vertices, with all values initialized to ∞. Set $cost = 0$.
- For $v \in V\{v_0\}$ such that $(v, v_0) \in E$, set $d[v] = c(v_0, v)$.
- While $V_{done} \neq V$:
    - Set $u = \text{argmin}\{d[v] | v \in V \setminus V_{done}\}$.
    - Set $V_{done} = V_{done} \cup \{u\}$.
    - For $v \in V \setminus V_{done}$ such that $(v, u) \in E$, set $d[v] = \min(d[v], c(u, v))$.
    - Set $cost = cost + d[u]$.

Implementing Prim's Algorithm (serial)

- Assume all costs are positive. $(c' = c + N)$
- Represent graph G as adjacency matrix A, where $A[i,j] = A[j,i] = c(i,j)$ if $(i,j) \in E$, and $A[i,j] = A[j,i] = 0$ otherwise.
- Sparse graph? Make the matrix sparse!

Implementing Prim's Algorithm (parallel)

- Opportunity for parallelism in the while loop (finding the next node, and updating values).
- Distribute the matrix A and the vector d.
- At each step, each processor submits its local closest vertex, and a parallel reduce finds the overall minimum. This choice is broadcast to all processors, which then independently update.

(Possible) Improvements

- Better data structure for d improves asymptotics:
  - Original: $O(|V|^2/p + |V| \log p)$, where p is number of processors
  - Binary heap: $O((|E| \log |V|)/p + |V| \log p)$
  - Fib heap: $O((E + |V| \log |V|)/p + |V| \log p)$.
- initial pruning (solving minimum spanning forest locally)
- load balancing, dynamic reallocating

Caveats

- Much better asymptotics possible: $O(|E|\alpha(|E|))$ in serial (Chazelle), or $O(\sqrt{|V|}\log^* |V| + D)$ for distributed (Kutten, Peleg), where D is graph diameter
- Current preduce doesn't use a log-depth tree
- Parallel implementation still incomplete (tsk, tsk)

Future Work

Local and distributed sparse functionality: a (giant) work in progress

- integrating SuiteSparse
- optimization, benchmarking versus Matlab (and others)
- log-depth broadcast, preduce
- (improved) interface for reallocation

Acknowledgements, Q&A

- Many thanks to Professor Edelman, Jeff Bezanson, Viral Shah, and Stefan Karpinski.

- Questions?