Towards Highly Scalable X10 Based Spectral Clustering

Hidefumi Ogata, Miyuru Dayarathna, and Toyotaro Suzumura

Suzumura Laboratory
Department of Computer Science
Graduate School of Information Science and Engineering
Tokyo Institute of Technology, Japan
Introduction – Large Graphs

• Applications that work with large graphs have become prevalent in recent years.

Social Network

Communication Network

Protein-Protein interaction

E.g., Facebook has one billion monthly active users by October 2012

Scalable Analysis of Large Graph data is very important.
Introduction – Graph Clustering

- Is the activity of grouping the vertices of a graph by considering the edge structure of the graph
  - Many edges within each cluster
  - Relatively few edges between clusters

- Application areas
  - Social Analytics
  - Data mining
  - Gene Analytics

Implement Spectral clustering on X10

Visualization of a 160-vertex relaxed caveman graph (S.E. Virtanen et al., Properties of nonuniform random graph models, May 2003)
Presentation Outline

• Introduction
• X10 and Graph Clustering
• Related Work
• X10 Based Scalable Spectral Clustering
• Performance Evaluation
• Discussion
• Conclusion and Future Work
X10

- A parallel programming language aimed for highly scalable applications running on Post-Petascale supercomputers.
- X10 enables programmers to develop parallelized high performance applications easily.
- X10 partitions memory space into some places.

![Diagram of X10](image)

(partitioned global address space (PGAS))

(P. Charles, et. al. 2005)
X10 Language Constructs for parallel programming

- Use \textit{at} statement to communicate with places
- Use \textit{async} statement to execute some code block in parallel
- Use \textit{finish} for synchronization
- Use \textit{atomic} for exclusive control

```
1 class HelloWholeWorld {
2   public static def main(args:Rail[String]) {
3       finish
4       for (p in Place.places())
5         at (p)
6         async
7         Console.OUT.println(p" says " + args(0));
8     }
9 }
```

\%
```
x10c++ HelloWholeWorld.x10
X10_NPLACES=4; ./a.out hello
```

Place 0 says hello
Place 2 says hello
Place 3 says hello
Place 1 says hello
Graph Clustering

- Graph $G$ is a pair of Sets, $G = (V, E)$
  - $V$ – set of all vertices in the graph $G$
  - $|V|$ - number of vertices ($n$)
  - $E$ – set of edges in $G$, an edge is a pair of vertices $(u, v)$
  - $w(u, v)$ – Weight between $u$ and $v$

- Graph clustering partitions the vertices of a graph into $k$ clusters $C_1, ..., C_k$

- The value of $k$ can be either provided as a parameter (K-Means, Spectral) or determined by the algorithm (Markov clustering)
Graph Clustering

- Graph clustering has an objective function, named **Normalized cut (NCut)** that we try to keep minimized.

\[
N\text{Cut}(C_1, \ldots, C_k) = \sum_{i=1}^{k} \frac{\text{Cut}(C_i, V \setminus C_i)}{\text{Assoc}(C_i)}
\]

\[
\text{Cut}(A, B) = \sum_{u \in A, v \in B} w(u, v)
\]

\[
\text{Assoc}(A) = \sum_{u \in A, v \in V} w(u, v)
\]

Good clustering algorithms provide small value of Ncut.
Graph Clustering

- **Cut (A,B)** – The sum of the edge weight between A and B
- **Assoc(A)** – Is similar to Cut(A,B) but it uses V instead of B. I.e., Assoc(A) = Cut(A,V)
- The value of Ncut becomes small when Cut is small compared to Assoc.
Spectral Clustering

• Minimizing NCut is NP-complete
  • there are many methods to calculate an approximate solution.
• The method proposed by Shi and Malik is one of them [16].
• It reduces NCut to the generalized eigenvalue problem,

\[ Lz = \lambda Dz \]

where

• \( D \) is a degree matrix of a graph
• \( L \) is a Laplacian matrix
• \( W \) similarity matrix
• \( z \) is the eigenvector we need to compute.

Why Spectral Clustering?

- Spectral clustering is a type of clustering used in image segmentation, gene analytics, etc.

- Spectral clustering reduces the dimensionality of the similarity of the data
  - Can be applied for datasets that do not have linear separability (K-Means only does not provide this advantage)

- Provides high quality clustering results
Spectral Clustering Cons.

(1) Need to solve a generalized eigen value problem
(2) Space Complexity (O(n^2)) and Time Complexity (O(n^3))

Solve by parallelizing the eigen value computation
Related Work

- Clustering eigen vectors of a similarity matrix of a graph is a bottleneck
- Nystrom method can approximate eigenvectors fast using a sub matrix of the similarity matrix [9].
- Graclus is a multi-level graph clustering algorithm [6]
  - Attempts to minimize the same objective function as spectral clustering


Related Work (Cont.)

- Parallel Spectral Clustering approximate Eigen vectors using ARPACK [4].
  - ARPACK – A library for solving large scale eigen value problems
- The aforementioned algorithms are implemented using C++/MPI
  - Need to be careful – Deadlocks, communication
  - X10 abstracts the MPI interface
  - X10 activities constitutes a tree → Deadlock free applications

Related Work (Cont.)

- Scalable Eigensolver proposed by Yoo et. Al.
  - Employees 2D MatVec operation (Matrix is distributed by 2D partitioning)
  - A potential candidate for implementing parallel spectral clustering

Algorithm

- **Input**: Similarity matrix $W$.
1. Compute the degree matrix $D$ and Laplacian matrix $L$.
2. Solve a generalized eigen value problem $Lz = \lambda Dz$.
3. Make a matrix $Z$ by stacking the first $k$ eigenvectors $z_1, \ldots, z_k$.
4. For each $i = 1, \ldots, n$, let $y_i \in \mathbb{R}^k$ be the vector corresponding to the $i$-th row of $Z$.
5. Apply K-means to the points $\{y_i\}_{i=1,\ldots,n}$ and cluster them into clusters $C_1, \ldots, C_k$. 
**X10 Based Spectral Clustering**

- **Graph Loading**
  - Load an edge list data and convert to the distributed intermediate data, that is like adjacency list.

Undirected Graph

Edge list data

Intermediate data

Place 0

Place 1
X10 Based Spectral Clustering

- Making correspondence between vertex ID and row (column) index
  - Since Place 0 does not know the row index of 5 and 8, place 1 sends them to place 0, and vice versa.

<table>
<thead>
<tr>
<th>Intermediate data</th>
<th>Place 0</th>
<th>Place 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 =&gt; [3, 5, 8]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 =&gt; [2, 5]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 =&gt; [2, 3]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 =&gt; [2]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Correspondence</th>
<th>Similarity Matrix (Adjacency Matrix)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 =&gt; 0</td>
<td>0 =&gt; [1, 2, 3]</td>
</tr>
<tr>
<td>3 =&gt; 1</td>
<td>1 =&gt; [0, 2]</td>
</tr>
<tr>
<td>5 =&gt; 2</td>
<td>2 =&gt; [0, 1]</td>
</tr>
<tr>
<td>8 =&gt; 3</td>
<td>3 =&gt; [0]</td>
</tr>
</tbody>
</table>
X10 Based Spectral Clustering

- Making matrix and solving eigen value problem
  - Make a Laplacian matrix from the matrix data.
  - Pass it to ScaLAPACK and get eigenvectors.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1, 2, 3]</td>
<td>[0, 2]</td>
<td>[0, 1]</td>
<td>[0]</td>
</tr>
</tbody>
</table>

Matrix data

\[
\begin{pmatrix}
3 & -1 & -1 & -1 \\
-1 & 2 & -1 & 0 \\
-1 & -1 & 2 & 0 \\
-1 & 0 & 0 & 1 \\
\end{pmatrix}
\]

Laplacian matrix

\[
\begin{pmatrix}
3 & 0 & 0 & 0 \\
0 & 2 & 0 & 0 \\
0 & 0 & 2 & 0 \\
0 & 0 & 0 & 1 \\
\end{pmatrix}
\]

Degree matrix

\[
\begin{pmatrix}
3 & -1 & -1 & -1 \\
-1 & 2 & -1 & 0 \\
-1 & -1 & 2 & 0 \\
-1 & 0 & 0 & 1 \\
\end{pmatrix}
\]

ScaLAPACK

\[
\begin{pmatrix}
3 & -1 & -1 & -1 \\
-1 & 2 & -1 & 0 \\
-1 & -1 & 2 & 0 \\
-1 & 0 & 0 & 1 \\
\end{pmatrix}
\]

eigenvectors
Solving Eigenvalue Problem by Using ScaLAPACK

Matrix (Laplacian Matrix/Degree Matrix)

Place 0

ScaLAPACK

Place 1

ScaLAPACK

Place 2

ScaLAPACK

Place 3

ScaLAPACK

Each place gets a sub matrix of eigenvectors

MPI Communication
Solving Eigenvalue Problem by Using ScaLAPACK

Construct the matrices

Set the matrices’ values in different places

Part of X10 source code where Laplacian and degree matrices are created.

```scala
finish for(place in Place.places())
async at(place) {

val matrixL = new Array[Double]
            (localRow * localCol, 0.0);
val matrixD = new Array[Double]
            (localRow * localCol, 0.0);
val matrixZ = new Array[Double]
            (localRow * localCol, 0.0);
val gMatrixD = GlobalRef(matrixD);
val gMatrixL = GlobalRef(matrixL);
val there = here;

for(p in neighbourList.dist.places()) at(p){
  for(pt in neighbourList.dist.get(p)) {
    val pair = neighbourList(pt);
    val m = pair.first;
    val neighbours = pair.second;
    if(m == -1) continue;
    val nNeighbours = neighbours == null
          ? 0 : neighbours.size;
    at(there){
      ScalAPACK.pdelset(gMatrixD(),
          m + 1, m + 1, gDesc(),
          nNeighbours + 1.0);
      ScalAPACK.pdelset(gMatrixL(),
          m + 1, m + 1, gDesc(),
          nNeighbours);
      for(npt in neighbours){
        val n:Int = neighbours(npt);
        ScalAPACK.pdelset(
            gMatrixL(),
            m + 1, n + 1, gDesc(),
            -1.0);
      }
    }
  }
}
```
Re-distributing the eigen vectors
X10 Based Spectral Clustering

- K-means
  - Each row of eigenvectors is regarded as a point of the Euclidean space.
  - K-means partitions them into $k$ clusters.
  - The result of K-means becomes the result of spectral clustering.

$$
\begin{pmatrix}
3 & -1 \\
-1 & 2 \\
-1 & -1 \\
-1 & 0
\end{pmatrix}
$$

Euclidean space
Parallel K-means

Algorithm 1 Parallel K-means

Input: The number of clusters $k$, the points $x_1, \ldots, x_n$, the places $P$.

Output: The clusters $C_1, \ldots, C_k$.

Distribute all the points into all $P$ places.
Master place randomly generates the initial centers of the clusters.

repeat

Master sends the centers to all $(P - 1)$ workers.
Each worker assigns own points to the nearest clusters.
Each worker re-computes the centers of the clusters.
Master gets all the centers from all workers, and re-computes their centers.

until the centers of clusters converge
Running K-means in Parallel

Place 0 (Worker)

Place 1 (Master)

Place 2 (Worker)

Place 3 (Worker)

Merge the centers of current clusters

Broadcast next centers of clusters
K-means : Non-Parallel in X10

```java
/* compute new clusters and counters */
for(i in points){
    /* compute which cluster is
     closest to each point */
    var minDist:Double = Double.MAX_VALUE;
    var closestCluster:Int = 0;
    for([j] in curClusters){
        var dist:Double = (curClusters(j) - points(i)).norm();
        if(dist < minDist){
            minDist = dist;
            closestCluster = j;
        }
    }
    /* add the point to the cluster */
    newClusters(closestCluster).
cellAdd(points(i));
    clusterCounts(closestCluster)++;
    result(i) = closestCluster;
}
```
K-means: Parallel in X10

```java
/* compute local new clusters */
and local counters */
finish for(p in Place.places()) {
    for(i in points.dist.get(p)) {
        /* compute which cluster is closest to each point */
        var minDist:Double = Double.MAX_VALUE;
        var closestCluster:Int = 0;
        for(var j:Int = 0; j < k; j++) {
            var dist:Double = (curClusters()(j) - points(i)).norm();
            if (dist < minDist) {
                minDist = dist;
                closestCluster = j;
            }
        }
        /* add the point to the cluster */
        newClusters()(closestCluster).cellAdd(points(i));
        clusterCounts()(closestCluster) =
            clusterCounts()(closestCluster) + 1;
        result(i) = closestCluster;
    }
}
```

X10 language constructs used to parallelize the K-means computation
X10 Spectral Clustering Evaluation

- **Data sets**
  - Kronecker Graph is an artificial graph and the others are real data.

<table>
<thead>
<tr>
<th>Graph</th>
<th>#Vertices</th>
<th>#Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kronecker Graph</td>
<td>19683</td>
<td>40.3 Million</td>
</tr>
<tr>
<td>PowerGrid</td>
<td>4941</td>
<td>6594</td>
</tr>
<tr>
<td>Internet</td>
<td>22963</td>
<td>48436</td>
</tr>
<tr>
<td>BlogCatalog3</td>
<td>10312</td>
<td>333983</td>
</tr>
</tbody>
</table>

- **Evaluation**
  - Performance evaluation
  - Precision analysis
Performance Evaluation

- Environment
  - 16 Nodes on TSUBAME 2.0 at Tokyo Institute of Technology
  - CPU: Intel Xeon 2.93GHz (6 cores) x2
  - Memory: 54GB
  - OS: SUSE Linux Enterprise Server 11 SP1
  - Network: QDR InfiniBand(40Gbps) x2
  - X10: version 2.2.2
  - MPI: MVAPICH2 version 1.8
Performance Evaluation

- The parameter $k$ is the number of clusters.
- In this experiment, we fixed $k = 2$.
- Increasing the number of nodes up to 16 nodes can reduce the elapsed time.

Kronecker graph, $k = 2$, each machine has 1 place

Elapsed time (hour)

Number of nodes

4 8 16
Precision Analysis

- A value of $NCut$ of a result tells how good the result is.

$$NCut(C_1,\ldots,C_k) = \sum_{i=1}^{k} \frac{Cut(C_i,V \setminus C_i)}{Assoc(C_i)}$$

- PowerGrid was partitioned into 2 clusters evenly, however Internet and BlogCatalog3 was not.

- It seems that Internet and BlogCatalog3 may not form the specific clusters (or communities).

| Graph          | Ncut($C_1$,$C_2$) | Cut($C_1$,$C_2$) | $|C_1|$ | $|C_2|$ |
|----------------|-------------------|------------------|--------|--------|
| PowerGrid      | 0.004             | 14               | 2197   | 2744   |
| Internet       | 0.057             | 6                | 26     | 22937  |
| BlogCatalog3   | 0.667             | 4                | 2      | 10310  |
Discussion

- We attempted to remove the bottleneck of spectral clustering by using ScaLAPACK.
- However ScaLAPACK cannot handle sparse matrices.
- It caused the memory shortage when we run our spectral clustering with graphs such as LiveJournal graph, which has 4.8 million vertices.
Discussion – Memory Bottleneck

- ScaLAPACK provides eigen solvers for only dense matrices and Banded matrices.
- We need matrices (L, D and Z)
- If the number of vertices is n, the size of the memory needed for the metrics is,

\[
\text{memory requirement} = (\#\text{vertices})^2 \times \text{sizeof(Double)} \times \#\text{matrices} \\
= n^2 \times 8 \times 3 \\
= 24n^2 \text{ [bytes]}
\]

Our spectral clustering cannot be applied to graphs which have more than 1779887 (~1.78M) vertices on Tsubame 2.0 even if we use its all 1408 nodes.
Discussion – Memory Bottleneck Elimination

- We are considering use of other library instead of ScaLAPACK
- Current Solution → ARPACK
- ARPACK is designed to compute few eigenvalues and vectors of a large square matrix.
- ARPACK Algorithm
  - Based on Implicitly Restart Arnoldi Method
  - Requires only the Matrix Vector product in Arnoldi process
  - We have to implement matrix-vector product by ourselves
  - But we can choose any structure, even sparse matrices for our implementation → Better scalability
Future Work

- To solve the memory shortage problem, we are trying to use ARPACK library instead of ScaLAPACK.
- ARPACK is designed to compute a few eigen values and eigenvectors of a large matrix. It can handle a sparse matrix.
Conclusion

- Implemented X10 based spectral clustering and evaluated its performance
- Our spectral clustering implementation achieves (for specific graphs)
  - Limited scalability
  - Good precision
- Yet our implementation cannot handle graphs with many vertices
- Solution → use sparse matrices for representing graph data
- We plan to create an ARPACK based spectral clustering implementation in future and evaluate its scalability and efficiency in terms of memory usage.
Thank You!