# Towards Highly Scalable X10 Based Spectral Clustering

Hidefumi Ogata*, Miyuru Dayarathna*, and Toyotaro Suzumura*†
*Department of Computer Science
Tokyo Institute of Technology/†IBM Research - Tokyo,
Email: {ogata.h.ac, dayarathna.m.aa}@m.titech.ac.jp, suzumura@cs.titech.ac.jp

***Abstract*—Large graph analysis has become a widely studied area in recent years. Clustering is one of the most important types of analysis that has versatile applications such as community detection in social networks, image segmentation, graph partitioning, etc. However, existing clustering algorithms do not intend for large scale graphs. To solve this problem, we implemented spectral clustering in X10, that is a programming language aimed for developing highly scalable applications on Post-Petascale supercomputers. Our spectral clustering is based on the algorithm proposed by Shi and Malik. After evaluating scalability and precision, we found that our implementations are scalable in terms of execution time and precise for analyzing real data.**

## I. Introduction

Applications which deal with large graphs (such as online social networking services) have become prevalent in recent years. The social networks grow bigger day by day, for example there are more than 125 billion friend connections by the end of March 2012 [8]. Though analyzing large social network supplies various information for us, it is difficult for billion scale graphs.

Graph clustering is the activity of grouping the vertices of a graph by considering the edge structure of the graph in such a way that there should be many edges within each cluster and realtively few between the clusters [15].

It is used in many area, for example, social analytics, data mining, and gene analytics [2]. In the case of social analytics, a graph represents relations between users of social networking service. An edge denotes that two users connected by the edge are friends on Facebook or follower on Twitter or so on. Clustering such graph can detect some communities in the social network.

We implemented spectral clustering on X10 [10], which is a parallel programming language aimed for highly scalable applications running on Post-Petascale supercomputers.

## II. X10 and Graph Clustering

### A. X10

X10 partitions memory space into some places (A place in X10 corresponds to a processing element with attached local storage [13]), called Partitioned Global Address Space (PGAS) [3]. Normally, one place corresponds to one compute node. One can use *at* statement to communicate with places, and *async* statement to execute some code block in parallel. Also *finish* and *atomic* do synchronization and exclusive control respectively. Such statements abstract the remote communication so that programmers can easily develop parallelized high performance applications.

### B. Graph Clustering

A graph $G$ is a pair of sets, $G = (V, E)$. $V$ is a set of all vertices in the graph $G$ and $|V|$ is the number of vertices $n$. $E$ is a set of all edges in $G$ and an edge is a pair of vertices $(u, v)$. The weight between $u$ and $v$ is denoted by $w(u, v)$. Graph clustering partitions the vertices of a graph into $k$ clusters $C_1, ..., C_k$ with $|V| = \Sigma_{i=1}^{k}|C_i|$. Some clustering algorithm have to be specified with the number of clusters $k$ before running clustering a graph. However, some other clustering algorithms can find appropriate $k$ for the graph. For example, the algorithms like K-means and spectral clustering need $k$ as input. On the other hand, those like Markov clustering [7] can cluster a graph without $k$.

### C. Spectral Clustering

Spectral clustering is a type of clustering used in image segmentation, gene analytics and so on. There is an objective function, named normalized cut (NCut) [16],

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

where,

$$Cut(A, B) = \sum_{u \in A, v \in B} w(u, v) \quad (2)$$

$$Assoc(A) = \sum_{u \in A, v \in V} w(u, v) \quad (3)$$

$A$ and $B$ are subsets of $V$. $Cut(A, B)$ is the sum of edge weight between $A$ and $B$. $Assoc(A)$ is similar to $Cut(A, B)$ but it uses $V$ instead of $B$. It means $Assoc(A)$ equals to $Cut(A, V)$. Since the value of $NCut$ becomes small when $Cut$ is small compared with $Assoc$, a good clustering provides a small value of $NCut$. Though minimizing normalized cut is NP-complete, it can be relaxed into an eigenproblem. Shi and Malik [16] proved that normalized cut is reduced to the generalized eigenvalue problem,

$$Lz = \lambda Dz \quad (4)$$

where $D$ is the degree matrix of a graph, $L$ is the unnormalized Laplacian matrix $D - W$ and $W$ is the similarity matrix.

978-1-4673-2371-0/12/$31.00 ©2012 IEEE

$z$ is the eigenvector we need to compute. Employing this relaxation, spectral clustering makes clusters by using first $k$ eigenvectors of (4). The algorithm of spectral clustering is as follows.

**Spectral Clustering**

Input: Similarity matrix $W \in \boldsymbol{R}^{n \times n}$, number of clusters $k$.

- Compute the degree matrix $D$ and the Laplacian matrix $L = D - W$.
- Solve a generalized eigenvalue problem $Lz = \lambda Dz$ and get the first $k$ eigenvectors $z_1, ..., z_k$.
- Make matrix $Z$ by stacking the eigenvectors as columns.
- For each $i = 1, ..., n$, let $y_i \in \boldsymbol{R}^k$ be the vector corresponding to the $i$-th row of $Z$.
- Apply K-means algorithm to the points $\{y_i\}_{i=1,...,n}$ and cluster them into clusters $C_1, ..., C_k$.

Output: Clusters $C_1, ..., C_k$.

Since spectral clustering reduces the dimensionality of the similarity of the data, it can be applied for data sets that do not have linear separability [14]. This is an advantage of spectral clustering; other clustering algorithms like K-means do not have such an advantage. Thus spectral clustering can provide a high-quality clustering than many other clustering algorithms. However, spectral clustering needs to solve a generalized eigenvalue problem, its space and time complexity are $O(n^2)$ and $O(n^3)$ respectively. This is a big bottleneck of spectral clustering. To apply spectral clustering to a large scale graph, it is very important to solve an eigenproblem fast. We attempt to deal with this bottleneck by parallelizing the computation of the eigenproblem.

## III. Related Work

Calculating eigenvectors of a similarity matrix of a graph is a bottleneck in spectral clustering. To eliminate this bottleneck, many solutions have been proposed. Nyström method can approximate eigenvectors fast using a sub matrix of the similarity matrix [9].

Graclus is a multilevel graph clustering algorithm, which attempts to minimize the same objective function as spectral clustering [6].

Parallel Spectral Clustering (PSC) [4] approximate eigenvectors by using ARPACK [11], that is a library for solving large scale eigenvalue problems. To make the best use of ARPACK, PSC reduces the similarity matrix to a sparse one by the t-nearest-neighbor approach [12] and distributes it into some computers.

These algorithms are implemented by C++ and MPI. However, MPI programming requires to be careful of deadlock or efficient communications. It is one of the causes to complicate MPI programming. In contrast, X10 let programmers not be worried about MPI, because MPI interface is abstracted. Furthermore, X10 activities (or threads) constitute a tree; all activities except root have one parent activity. Therefore, X10 make it easier to product deadlock-free applications [1]. This is our motivation to use X10.

There is also a new scalable eigensolver proposed by Yoo et. al. [17]. This eigensolver employs 2D MatVec operation, where the matrix is distributed by 2D partitioning. 2D MatVec operation can be used for any other eigensolvers that need a matrix-vector multiplication. PSC, we described in this section, is one of the application of 2D MatVec operation.

## IV. X10 Based Scalable Spectral Clustering

### A. Graph Loading

We assumed that graph data represented by edge list format are distributed into several files. Our spectral clustering load them in parallel. Edge list format is one of the representations of graph data, whose line consists of a pair of vertex's ID. Vertex's ID is a unique number for all vertices, but it is not always to begin from zero and not always to be continuous. We make a matrix from this graph data. Therefore, we have to correspond vertex's ID to an index of row (or column) of a matrix; continuous number begin from zero.

### B. Making Correspondence Between Vertex ID and Index

After graph loading, we get a list of all vertices divided into X10 places. Each X10 place numbers its vertices and gets neighbors of them independently. Since neighbors of some vertices in place A are probably not in place A, it occurs communications between current place and other places like MPI_ALLTOALL.

### C. Making a Matrix and Solving Eigenvalue Problem

In our spectral clustering implementation, we construct dense matrices which is one of the available structures in ScaLAPACK [5]. The detail of ScaLAPACK is discribed afterward. Each processor has sub matrices of the dense matrices as 1-dimensional arrays. Because making the matrices can be done independently, our spectral clustering algorithm does this in parallel. The source code of this part is shown as Fig. 1.

We use ScaLAPACK for solving a generalized eigenvalue problem. ScaLAPACK is a library of scalable linear algebra routines for parallel distributed memory machines. It solves dense and banded linear systems, least squares problems, eigenvalue problems, and singular value problems. Matrices data are divided by 2D block-cyclic distribution and each of the processors placed into a grid form which has the sub matrices of it. Then each processor solves one problem in cooperation with the other processors using MPI communication. The ScaLAPACK routine we need for spectral clustering is *pdsygvx*, that is a driver routine of generalized symmetric definite eigenvalue problems. It solves following types of problems,

$$Az = \lambda Bz \quad (5)$$

$$ABz = \lambda z \quad (6)$$

$$BAz = \lambda z \quad (7)$$

```
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);
                }
            }
        }
    }
}
```

Fig. 1. A part of our source code where Laplacian matrix and degree matrix are created.

where $A$ is symmetric, $B$ is symmetric positive definite, $\lambda$ is the eigenvalue and $z$ is the eigenvector. Using a Cholesky factorization of $B$ as $B = U^T U$, we have

$$Az = \lambda Bz \tag{8}$$

$$\Rightarrow \quad ((U^{-1})^T A U^{-1})(Uz) = \lambda(Uz) \tag{9}$$

$$\Rightarrow \quad Cy = \lambda y \tag{10}$$

where C is symmetric matrix $(U^{-1})^T A U^{-1}$ and y is $Uz$. Thus, *pdsygvx* solves the standard eigenproblem $Cy = \lambda y$ reduced from $Az = \lambda Bz$ and recovers $z$ from $y$.

### D. K-means

After Laplacian matrix and degree matrix are prepared, *pdsygvx* computes first $k$ eigenvectors of the eigenproblem $Lz = \lambda Dz$. Next, letting $U$ be the matrix containing the eigenvectors as columns, $i$-th row of $U$ is considered as a point $x_i$ to be clustered. Then K-means partitions the points $x_1, ..., x_n$ into $k$ clusters. We implemented a parallel version of K-means algorithm as Algorithm 1.

The partial source code of K-means is shown as Fig.2. In this code, the cluster assigned to each vertex is updated

**Algorithm 1** Parallel K-means

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

**Output:** The clusters $C_1, ..., C_k$.

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

TABLE I
TSUBAME 2.0 ENVIRONMENT

| | |
|---|---|
| CPU | Intel Xeon 2.93GHz (6 cores) x2 |
| Hardware threads | 24 |
| Memory | 54GB |
| OS | SUSE Linux Enterprise Server 11 SP1 |
| Network | QDR InfiniBand (40Gbps) x2 |
| X10 | version 2.2.2 |
| MPI | MVAPICH2 version 1.8 |

asynchronously. For comparison, single-process version of this code is shown as Fig.3. As you can see, these codes are very similar with each other. In X10 programming, it is easy to convert single-process code to multi-process code by small modification. This is one of the advantages to use X10 programming model.

## V. PERFORMANCE EVALUATION

### A. Evaluation Environment

We executed our X10 implementation of spectral clustering on TSUBAME 2.0 supercomputer at Tokyo Institute of Technology in Japan. Its environment is shown in TABLE I. The data sets for the evaluation is shown in TABLE II. Kronecker graph was used for scalability evaluation, and the other graphs are used for precision analysis.

We evaluated the scalability of our spectral clustering with a Kronecker graph, its number of vertices is 19683. The scalability result is shown in Fig.4, 5. It is found that increasing the number of nodes up to 16 nodes can reduce the elapsed time. However, if the number of nodes is more than 16, the performance did not improve. The same result was observed in the case of increasing the place (Fig.5). It

TABLE II
DATA SETS

| Graph | Vertices | Edges |
|---|---|---|
| Kronecker Graph | 19683 | 40.3 Million |
| PowerGrid | 4941 | 6594 |
| Internet | 22963 | 48436 |
| BlogCatalog3 | 10312 | 333983 |

```
/* compute local new clusters
                and local counters */
finish for(p in Place.places()) async at(p) {
    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;
    }
}
```

Fig. 2. Partial Code of Parallel K-means

```
/* 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;
}
```

Fig. 3. Partial Code of Non-Parallel K-means

TABLE III
NORMALIZED CUT VALUES OF OUR SPECTRAL CLUSTERING WITH $k = 2$

| 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 |

means that the communication cost becomes higher when machine nodes increase.

### B. Precision Analysis

As described in section II-C, spectral clustering minimize the normalized cut (1). Because of this, we can evaluate the precision of spectral clustering results by the value of the normalized cut (i.e. the smaller normalized cut is, the better result is.) The result of our spectral clustering is shown in TABLE III. In the case of PowerGrid graph, we got a good result; normalized cut is 0.004, each cluster has roughly 2500 vertices and cut of them has only 14 edges. However, other graphs' results seem to be not good. It may happen for the individual graph structure. That is, Internet and blogcatalog graphs may not form the specific clusters. We are investigating about it.

## VI. DISCUSSION

We attempted to remove the bottleneck of spectral clustering by using ScaLAPACK, because implementing a parallel eigensolver in X10 has a big cost; it is difficult to develop an algorithm which makes the best use of X10 in a short term. However, since we employ ScaLAPACK as eivensolver, there is a limitation of our spectral clustering in terms of memory usage. The matrix structure of the spectral clustering is dense matrix, because ScaLAPACK provides eigensolvers for only dense matrices and banded matrices. We need three matrices as Laplacian matrix L, degree matrix D and the matrix Z contains the eigenvectors. If the number of vertices is $n$, the size of memory needed for the matrices is calculated by following expression.

$$\begin{aligned} & (\#vertices)^2 \times (sizeof(Double)) \times (\#matrices) \\ = \quad & n^2 \times 8 \times 3 = 24n^2[bytes] \end{aligned} \tag{11}$$

Since TSUBAME 2.0 has 1408 nodes, each has 54GB memory, the maximum number of $n$ is 1779887. It means that our spectral clustering cannot be applied to the graph which has more than 1779887 vertices on TSUBAME 2.0.

Thus, we are considering to use other library instead of ScaLAPACK. Our current solution is ARPACK [11]. ARPACK is designed to compute a few eigenvalues and eigenvectors of a large square matrix. Its algorithm is based on "Implicitly Restart Arnoldi Method". It only requires the matrix-vector product in Arnoldi process, thus it is appropriate for structured matrices, whose matrix-vector product requires $O(n)$ rather than the usual $O(n^2)$. Though we have to implement matrix-vector product by ourself, we can choose any structure, even sparse matrix, for our implementation. Employing ARPACK will achieve better scalability.

## VII. CONCLUSION AND FUTURE WORK

We implemented X10 based spectral clustering and evaluated its performance. Our spectral clustering could achieve limited scalability and good precision for specific graphs. However, it is not appropriate for the graphs having too many vertices. To solve this problem, we have to use sparse matrices instead of dense matrices for representing graph data. ARPACK can be the solution. We plan to create an ARPACK based spectral clustering implementation in future and evaluate its scalability and efficiency in terms of memory usage.

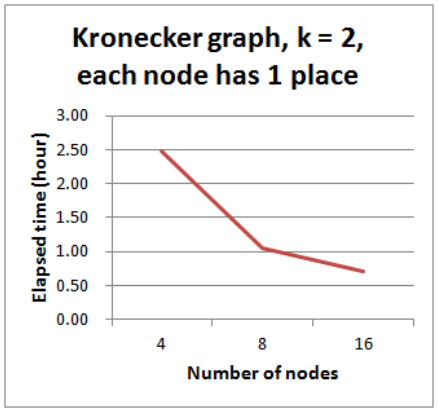


Fig. 4. Scalability evaluation of our spectral clustering with Kronecker graph. Each node has 1 place.

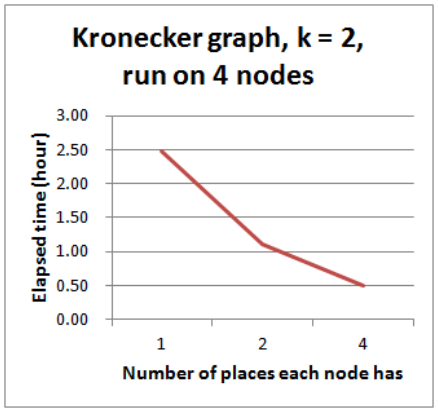


Fig. 5. Scalability evaluation of our spectral clustering with Kronecker graph. The number of nodes is 4.

## REFERENCES


[1] Shivali Agarwal, Rajkishore Barik, Dan Bonachea, Vivek Sarkar, Rudrapatna K. Shyamasundar, and Katherine Yelick. Deadlock-free scheduling of x10 computations with bounded resources. *SPAA*, pages 229–240, 2007.

[2] Charu C. Aggarwal and Haixun Wang. A survey of clustering algorithms for graph data. In Charu C. Aggarwal, Haixun Wang, and Ahmed K. Elmagarmid, editors, *Managing and Mining Graph Data*, volume 40 of *The Kluwer International Series on Advances in Database Systems*, pages 275–301. Springer US, 2010.

[3] Philippe Charles, Christian Grothoff, Vijay Saraswat, Christopher Donawa, Allan Kielstra, Kemal Ebcioglu, Christoph von Praun, and Vivek Sarkar. X10: an object-oriented approach to non-uniform cluster computing. *SIGPLAN Not.*, 40(10):519–538, October 2005.

[4] Wen-Yen Chen, Yangqiu Song, Hongjie Bai, Chih-Jen Lin, and Edward Y. Chang. Parallel spectral clustering in distributed systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(3):568–586, Mar 2011.

[5] J. Choi, J. Demmel, I. Dhillon, J. Dongarra, S.Ostrouchov, A. Petitet, K. Stanley, D. Walker, and R.C. Whaley. Scalapack: a portable linear algebra library for distributed memory computers - design issues and performance. *Computer Physics Communications*, 97(1-2):1–15, Aug 1996.

[6] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. Weighted graph cuts without eigenvectors: A multilevel approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(11):1944–1957, Nov 2007.

[7] S. Van Dongen. A cluster algorithm for graphs. Technical report, 2000. CWI Amsterdam, The Netherlands.

[8] Facebook. Facebook's latest news, announcements and media resources. URL: http://newsroom.fb.com/, May 2012.

[9] Charless Fowlkes, Serge Belongie, Fan Chung, and Jitendra Malik. Spectral grouping using the nyström method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2):214–225, Feb 2004.

[10] IBM. X10 language specification version 2.2. Jun 2012.

[11] R. B. Lehoucq, D. C. Sorensen, and C. Yang. Arpack users' guide: Solution of large scale eigenvalue problems with implicitly restarted arnoldi methods. Oct 1997.

[12] U. Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[13] J. Milthorpe, V. Ganesh, A.P. Rendell, and D. Grove. X10 as a parallel language for scientific computation: Practice and experience. In *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pages 1080 –1088, may 2011.

[14] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. *Advances in Neural Information Processing Systems*, 2:849–856, 2002.

[15] Satu Elisa Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27 – 64, 2007.

[16] Jianbo Shi and Jitendra Malik. Normalized cut and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, Aug 2000.

[17] Andy Yoo, Allison H. Baker, Roger A. Pearce, and Henson Van Emden. A scalable eigensolver for large scale-free graphs using 2d graph partitioning. In *SC*, page 63, 2011.