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 parallel 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. We evaluated scalability and precision, and we found that our implementations are scalable in terms of execution time and precise for real data.

I. INTRODUCTION

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

Clustering is a type of graph analysis that partition a graph into some sub graphs whose vertices are highly connected. It is used in many area, for example, social analytics, data mining, and gene analytics. 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 [6], 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, called Partitioned Global Address Space (PGAS). Normally, one place corresponds to one computer. 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 partitioning the vertices of a graph into $k$ clusters $C_1, ..., C_k$ with $|V| = \sum_{i=1}^{k} |C_i|$. Some clustering algorithm have to be specified the number of clusters $k$ before cluster a graph, however other clustering algorithm can find appropriate $k$ for the graph. For example, the algorithm like K-means algorithm and spectral clustering needs $k$ as input. On the other hand, those like Markov clustering [4] 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 a objective function, named normalized cut (NCut),

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

where,

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

$Cut(A, B)$ 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 [10] proved that normalized cut is reduced to the generalized eigenvalue problem,

$$ Lz = \lambda Dz $$

where $D$ is the degree matrix of a graph, $L$ is the unnormalized Laplacian matrix $D - W$ and $W$ is the similarity matrix. Employing this relaxation, spectral clustering makes clusters by using first $k$ eigenvectors of (4). The algorithm of spectral clustering is as follows.

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 [9]. 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. It 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 solution have been proposed. Nyström method can approximate eigenvectors fast using a sub matrix of the similarity matrix [5].

Gracius is a multilevel graph clustering algorithm, which attempt to minimize the same objective function as spectral clustering [3].

Parallel Spectral Clustering (PSC) [1] approximate eigenvectors by using ARPACK [7], 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 [8] 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 to worry 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. This is our motivation to use X10.

IV. X10 BASED SCALABLE SPECTRAL CLUSTERING

The algorithm of spectral clustering was described in section II-C. In our spectral clustering, We use ScalAPACK [2] 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 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,

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

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

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

where $A$ is symmetric and $B$ is symmetric positive definite. Using a Cholesky factorization of $B$ as $B = U^T U$, we have

$$Az = ABz \quad (8)$$

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

$$\Rightarrow Cy = \lambda y \quad (10)$$

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

In our spectral clustering, we construct dense matrices which is one of the available structure in ScalAPACK. Each processor has sub matrices of the dense matrices as 1-dimensional arrays. Because making the matrices can be done independently, our spectral clustering do this in parallel. The source code of this part is shown as Fig. 1.

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 $x_1, \ldots, x_n$ 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.?. In this code, the cluster assigned to each vertex is updated asynchronously. For comparison, single-process version of this code is shown as Fig.?. 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

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

/* 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;
}

Algorithm 1 Parallel K-means
Input: The number of clusters k, the points $x_1, \ldots, x_n$.
Output: The clusters $C_1, \ldots, C_k$.
Distribute all the points into all places.
Master place randomly generates the initial centers of the clusters.
repeat
    Master send the centers to all 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 recompute their centers.
until the centers of clusters converge

TABLE I
TSUBAME 2.0 ENVIRONMENT

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel Xeon 2.93GHz (6 cores) x2</td>
</tr>
<tr>
<td>Hardware threads</td>
<td>24</td>
</tr>
<tr>
<td>Memory</td>
<td>54GB</td>
</tr>
<tr>
<td>OS</td>
<td>SUSE Linux Enterprise Server 11 SP1</td>
</tr>
<tr>
<td>Network</td>
<td>QDR InfiniBand (80Gbps) x2</td>
</tr>
<tr>
<td>X10</td>
<td>version 2.2.2</td>
</tr>
<tr>
<td>MPI</td>
<td>MVAPICH2 version 1.8</td>
</tr>
</tbody>
</table>

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 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 it, 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 an 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 eigensolver, 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

TABLE II
DATA SETS

<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>

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  |

Fig. 2. Partial Code of Parallel K-means

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

We attempted to remove the bottleneck of spectral clustering by using ScaLAPACK, because implementing an 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 eigensolver, 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.

\[
(\text{#vertices})^2 \times (\text{sizeof(Double)}) \times (\text{#matrices}) = n^2 \times 8 \times 3 = 24n^2 [\text{bytes}]
\]  
(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 [7]. 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 evaluate its performance. Our spectral clustering could achieve limited scalability and good precision for specific graphs. However it is not appropriate for the graphs have too many vertices. To solve this problem, we have to use sparse matrices instead of dense matrices. 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.

REFERENCES


