INTRODUCING SCALEGRAPH : AN X10 LIBRARY FOR BILLION SCALE GRAPH ANALYTICS

Miyuru Dayarathna, Charuwat Houngkaew, and Toyotaro Suzumura

Department of Computer Science
Graduate School of Information Science and Engineering
Tokyo Institute of Technology
Japan

X10 Workshop 2012

6/14/2012

This research was partly supported by the Japan Science and Technology Agency (JST) Core Research of Evolutionary Science and Technology (CREST)
Background

- Massive graph mining and Management has become an important research issue in recent years.
Background

- HPC programmer productivity is considered one of the important goals in achieving the Exascale computational capabilities.
- PGAS languages are an example for such initiatives.
- It is important for having a complex network analysis software APIs in such languages.
- However, there are no such libraries currently available.
Current Libraries for Complex Network Analysis

- Do not aim at solving large graph problems (Beyond the scale of Billions of Vertices and Edges)
- Do not provide a complete mix of graph algorithms
- Famous example libraries,
  1. **Igraph** – by Gabor Csardi et al.
  2. **JUNG** (Java Universal Network/Graph Framework) – by Joshua O’Madadhain et al.
  3. **GraphStream** - Stefan Balev et al.
  4. **The Boost Graph Library (BGL)** – by Jeremy Siek et al.
  5. **JGraphT** - Barak Naveh et al.
  6. **Ruby Graph Library (RGL)** – by Horst Duchene
  7. **LEMON** – Alpar Juttner et al.
  8. **NetworkX** – Hagberg et al.
  9. **NG4J** – Bizer et al.
Research Problem

Comprehensive support for HPC programmers to specify highly productive, distributed, scalable graph analysis tasks for billion scale graphs has not been achieved yet.

Possible Solutions
• Create high level language wrappers for existing low level graph analysis libraries (E.g., Knowledge Discovery Toolbox [45])
Presentation Outline

- Introduction
- Research Problem
- Proposed Solution
- Related Work
- Background (X10)
- Library’s Design
- Implementation
- Evaluation
- Conclusion
Aim and Objectives of ScaleGraph

• **Aim** - Create an X10 graph processing library which can efficiently process massive graphs (beyond the scale of billions of vertices and edges).

• **Objectives**
  - To define concrete abstractions for Massive Graph Processing
  - To investigate use of X10 (i.e., PGAS languages) for massive graph processing
  - To support significant amount of graph algorithms including algorithms (E.g., structural properties, clustering, community detection, etc.)
  - To create well defined interfaces to Graph Stores
  - To evaluate performance of each measurement algorithms and applicability of ScaleGraph using real/synthetic graphs in HPC environments.
Goal and Contributions of the Paper

• Establish the baseline architecture of ScaleGraph library

• Contributions

  1. We specify a graph API with graph representations, and algorithms for specifying graph processing in the scale of billions of vertices and edges.

  2. We cover a wide range of graph representation standards which will enable complex network analysts to easily use their Massive (ranging from GB to TB) datasets.

  3. We make an initial scalability study of our API in Peta scale computer systems.
ScaleGraph Architecture

X10 programmer creates X10 Graph program code.

X10 program code calls X10 Standard API.

X10 Standard API uses X10 C++ Compiler.

X10 C++ Compiler uses third party libraries.

Third party libraries uses ScaleGraph Application Executable.

ScaleGraph Application Executable uses GraphStore(s).

GraphStore(s) uses X10 Runtime.

X10 Runtime communicates with Computer Cluster.

Computer Cluster communicates with Computer Cluster.
Related Work (I)

- Complex Network Research - Igraph [15], SNAP [16]
  - Run only on workstations.
  - May scale only for few billion edges
- Graph Libraries - GGCL [17], BGL [18], JUNG [43]
  - Our library is for distributed processing
  - Vertex and Edge Attributes (Colorful Graphs)

Related Work (II)

- Distributed Graph Libraries – PBGL [21], ParGraph [24], ComBLAS [10]
  - Programmer productivity
- Shared Memory Graph Libraries – MTGL [7], SNAP (Georgia Tech) [46]
  - Need specialized hardware

Related Work (III)

- **Graph Analysis using X10 – Cong et al.**[13][14]
  - We focus on Graph API
- **Other Computational Models – Pregel**[35]
  - We can implement programming models like Pregel in X10
- **Importance of well defined abstractions – Kulkarni et al.**[28]

---


X10 – An Overview

- X10 is a PGAS language being developed by IBM Research in collaboration with academic partners.

X10 provides a programming model that can withstand architectural challenges posed by multiple cores, hardware accelerators, clusters, and super computers.

Increased programming productivity for future systems such as Exascale computing systems.
X10 – An Overview

• X10 Language Features
  • Strongly typed
  • Object-oriented
  • Static type-checking
  • Static expression of program invariants
    • Supports the motivation of improving programmer productivity and performance
  • Latest Major Release X10 2.2 – source-to-source compilation
    • ScaleGraph uses native X10
  • Supports GPU
    • Currently ScaleGraph does not use GPU programming features
X10 – An Overview (Contd.)

- **X10 Language Features**
  - **Place** – A collection of non-migrating mutable data objects and the activities that operate on the data

![Diagram showing X10 language features](chart.png)

---

(P. Charles, et. al. 2005)
X10 – An Overview (Contd.)

- **DistArray**
  - Used for creating graph abstractions

- **Annotation system of X10 allows extensions**
  - We use `@Native(lang, code)` for implementing C++ language specific functions that are not implemented in current X10
    - Directory listing
    - GML Reader

- **GlobalRef**
  - Used as a support for coordinating activities between different places
Library Design

- Aim: Define solid abstractions for billion scale graph processing
ScaleGraph Application types

• SMALL ($n > 0, n \in \mathbb{N}$)
  - Graph applications that run in a single place
  - To support complex network analysis community at large
    • Use the library in single node settings
  - Entire graph is stored in place 0.
  - Maximum $2^n$ vertices
  - E.g., $n = 16$, $2^{16} = 65,536$ vertices

• MEDIUM ($m > 0, m \in \mathbb{N}$)
  - In memory graphs that is stored in multiple places
  - Maximum $(2^m \times $numberOfPlaces$)$ vertices
  - E.g. $m = 24$, $(2^{24} \times 128) = 2,147,483,648$ vertices
ScaleGraph Application types

- **LARGE**
  - End user does not have enough compute resources to instantiate sufficient amount of resources to hold billion scale graphs
    - Users with small compute clusters
    - Resourceful clusters such as super computers when the processed graphs need to reside on disks

**Why three scales?**

Performance tradeoffs and resource availability issues present in many graph analysis applications

**LARGE scale with four machines each machine holds 32 places (i.e., Total 128 places). However only a portion of the graph is loaded on to the machines.**
Software Design

- Current Design consist if six main categories of classes: graph, I/O, generators, metrics, clustering, and communities
Software Design : Graph Representation

• Graph is just a data structure. Graph algorithms are coded separately.

• Graphs are represented as adjacency lists.
  • Most of the real world graphs are sparse
Software Design: Data Representation of AttributedGraph
Software Design : Data Representation of PlainGraph

Place ID
0 . . . i

Array of vertex records

Source vertices

Neighborhood vertex IDs of \( A_{(i,j)} \)

Vertex records

\( A_{(i,P)} \)
\( A_{(i,j)} \)
\( A_{(i,0)} \)

Place ID

Place ID
0 . . . i

Array of vertex records

Destination vertices

Neighborhood vertex IDs of \( B_{(i,j)} \)

Vertex records

\( B_{(i,P)} \)
\( B_{(i,j)} \)
\( B_{(i,0)} \)

\( M \) : Total supported vertices
\( N \) : Number of Places
\( P \) : Vertices per Place (\( P = (M/N) \))
\( i \) : Place ID (\( N > i \geq 0 \))
\( P > j \geq 0 \)
Software Design: Graph Storage Formats

• There are variety of graph storage formats in use.

<?xml version="1.0" encoding="UTF-8"?>
<gexf xmlns="http://www.gexf.net/1.1draft"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.gexf.net/1.1draft http://www.gexf.net/1.1draft/gexf.xsd" version="1.1">
  <graph mode="static" defaultedgetype="undirected">
    <nodes>
      <node id="4941" label="YBR236C"/>
      <node id="4942" label="YOR151C"/>
      <node id="4943" label="YML010W"/>
      <node id="4944" label="YNR016C"/>
      <!-- Rest of the Contents .... -->
      <edge id="20367" source="7276" target="7277"/>
      <edge id="20368" source="7278" target="7279"/>
      <edge id="20369" source="7293" target="7294"/>
    </edges>
  </graph>
</gexf>

GEXF

<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
  <graph id="G" edgedefault="undirected">
    <node id="n0"/>
    <node id="n1"/>
    <node id="n2"/>
    <node id="n3"/>
    <edge source="n0" target="n2"/>
    <edge source="n1" target="n2"/>
    <edge source="n2" target="n3"/>
  </graph>
</graphml>

GraphML

Creator "Mark Newman on Sat Jul 22 05:41:45 2006"
graph
[  directed 0
node
[    id 0
      label "8001"
  ]
node
[    id 1
      label "64666"
  ]
node
[    id 2
      label "7018"
  ]
]

GML

% US power grid - unweighted network
% from Panayiotis Tsaparas:
% adapted for Pajek, V. Batagelj, March 19, 2006
% 0 -> 4941
*vertices 4941
*edges
4941 386 395 451 1 3553 3586 3587 3637
2 3583
3 4930
4 88
5 13 120
6 8
7 8
8 6 7 9
9 8 10 61 75 205 208

Pajek
Software Design : Graph Storage
Readers/Writers

• A set of classes for reading and writing graph files located at org.scalegraph.io

• E.g.
  • EdgeListReader, EdgeListWriter
  • ScatteredEdgeListReader, ScatteredEdgeListWriter
  • GEXFReader, GEXFWriter
  • GMLReader, GMLWriter

<table>
<thead>
<tr>
<th>Attributed Graphs</th>
<th>Non-attributed Graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td>GML</td>
<td>CSV</td>
</tr>
<tr>
<td>GEXF</td>
<td>DIMACS</td>
</tr>
<tr>
<td>GraphML</td>
<td>LGL</td>
</tr>
<tr>
<td>CSV</td>
<td>Pajek</td>
</tr>
<tr>
<td>GDF</td>
<td></td>
</tr>
<tr>
<td>GraphViz</td>
<td></td>
</tr>
</tbody>
</table>
Software Design : Graph Generators

• Include a collection of synthetic graph generators
• Have implemented R-MAT generator
• Working on
  • BarabasiAlbertGenerator
  • CitationgraphGenerator
  • ErdosRenyiGenerator
Software Design: Graph Structural Properties

- Graphs contain specific topological features which characterize their connectivity.
- Implemented
  - Degree Distribution Calculation (in-degree, out-degree, in/out-degree)
  - Betweenness Centrality (BC)
  - PageRank/RWR
  - Clusters (E.g., Spectral Clustering)
- Planned other metrics
  - Diameter
  - Density
  - Complexity
  - Cliques
  - Kcores
  - Mincut
  - Connected Component
Implementation : Background – Degree Distribution Calculation, R-MAT Scale

- If one denotes degree by k, then the degree distribution can be represented by $p_k$.

- R-MAT scale is an integer that specifies the number of vertices available in a graph. E.g. Scale 10 graph has 1024 vertices.
Implementation: Background – Betweenness Centrality (BC)

- BC measures the extent to which a vertex lies on paths between other vertices.

- If \( n_{st}^i \) be the number of geodesic paths from \( s \) to \( t \) that pass through \( i \) (\( s, t, \) and \( i \) are vertices of the graph, \( s \neq t \neq i \)).

- If total number of geodesic paths from \( s \) to \( t \) is denoted as \( g_{st} \).

- Betweenness Centrality can be specified as follows,

\[
x_i = \sum_{st} \frac{n_{st}^i}{g_{st}}
\]

BC score of \( i \) = \( 2/2 \)
Implementation: An example for use of AttributedGraph

```scala
val attrArray:ArrayList[Attribute] = null;
schema: AttributeSchema = new AttributeSchema();
schema.add("fname", AttributeSchema.StringAttribute);
schema.add("email_add", AttributeSchema.StringAttribute);
schema.add("age", AttributeSchema.IntAttribute);
attrArray = new ArrayList[Attribute]();
attrArray.add(new StringAttribute("fname", "Alice"));
attrArray.add(new StringAttribute("email_add", "alice@gmail.com"));
v0:Vertex = new Vertex(attrArray);

attrArray = new ArrayList[Attribute]();
attrArray.add(new StringAttribute("fname", "Bob"));
attrArray.add(new StringAttribute("email_add", "bob@gmail.com"));
v1:Vertex = new Vertex(attrArray);
```

Define attribute schema

First vertex

Second vertex

Alice

Bob

alice@gmail.com

bob@gmail.com
Implementation: An example for use of AttributedGraph (Contd.)

```java
AttributedGraph g = AttributedGraph.make();
g.setVertexAttributeSchema(schema);
g.addVertex(v0);
g.addVertex(v1);

AttributeSchema schema = new AttributeSchema();
schema.add("title", AttributeSchema.DateAttribute);
schema.add("dtime", AttributeSchema.DateAttribute);
g.setEdgeAttributeSchema(schema);

ArrayList<Attribute> attrArray = new ArrayList<Attribute>();
attrArray.add(new StringAttribute("title", "Meeting"));
attrArray.add(new DateAttribute(2012, 2, 10));
e0: Edge = new Edge(v0, v1, attrArray);

g.addEdge(e0);
```

Initialize the graph and add the two vertices

Create edge attribute schema

Create the edge

Add the edge

Alice

Bob

alice@gmail.com

bob@gmail.com
Implementation: Run Betweenness Centrality on AttributedGraph

val graph: AttributedGraph;

//Load the graph data from secondary storage
graph = GMLReader.loadFromFile("/data/power_grid.gml");

//Run the Betweenness Centrality calculation
val result = BetweennessCentrality.run(graph, false);
Implementation : Betweenness Centrality on PlainGraph

finish {

    val distVertexList: DistArray[Long] = this.plainGraph.getVertexList();
    val localVertices : Array[Long]{self.rank == 1} =
        distVertexList.getLocalPortion();
    val numLocalVertices: Int = localVertices.size;
    val numThreads = Runtime.NTHREADS;
    val chunkSize = numLocalVertices / numThreads;
    val remainder = numLocalVertices % numThreads;

    var startIndex: Int = 0;

    for(threadId in 0..(numThreads - 1 ) ) {
        async doBfsOnPlainGraph(threadId, numThreads, localVertices);
    }
}
Implementation: Betweenness Centrality on PlainGraph (Contd.)

// If undirected graph divide by 2
if(this.plainGraph.isDirected() == false) {
    if(this.isNormalize) {
        // Undirected and normalize
        betweennessScore.map(betweennessScore, (a: Double) => a / (((numVertex - 1) * (numVertex - 2))));
    } else {
        // Undirected only
        betweennessScore.map(betweennessScore, (a: Double) => a / 2);
    }
} else {
    if(this.isNormalize) {
        // Directed and normalize
        betweennessScore.map(betweennessScore, (a: Double) => a / ((numVertex - 1) * (numVertex - 2)));
    }
}

Team.WORLD.allreduce(here.id, betweennessScore, 0, betweennessScore, 0, betweennessScore.size, Team.ADD);
Evaluation : Environment

- Conducted on Tsubame 2.0 (5th ranked super computer on November 2011 top 500 list) on 4 nodes

<table>
<thead>
<tr>
<th>CPU/Core count</th>
<th>Two Intel®Xeon®X5670 @ 2.93GHz CPUs each with 6 cores.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total 12 cores per node/24 hardware threads</td>
</tr>
<tr>
<td>RAM</td>
<td>54GB per node</td>
</tr>
<tr>
<td>Interconnect</td>
<td>Infinibband Network (Voltaire Grid Director 4700)</td>
</tr>
<tr>
<td>Secondary storage</td>
<td>GPFS/Luster file system</td>
</tr>
<tr>
<td>OS</td>
<td>SUSE Linux Enterprise Server 11 SP1</td>
</tr>
<tr>
<td>X10 version</td>
<td>X10.2.2.2</td>
</tr>
<tr>
<td>X10 Runtime</td>
<td>X10 native, MPI runtime. Used MPICH 2.1.4.</td>
</tr>
<tr>
<td></td>
<td>X10 was built with following options:</td>
</tr>
<tr>
<td></td>
<td>-DNO_CHECKS=true  –Doptimize=true squeakyclean</td>
</tr>
<tr>
<td>X10 environment varaibles</td>
<td>X10_STATIC_THREADS=true</td>
</tr>
<tr>
<td></td>
<td>X10_NTHREADS=22</td>
</tr>
</tbody>
</table>
Evaluation: Elapsed time on single place

Betweenness Centrality

In/Out Degree Distribution

Scale 16 has a knee because it has more edges compared to scale 18
Evaluation: Elapsed time of BC of ScaleGraph on multiple nodes
Evaluation: Elapsed time of BC of ScaleGraph on multiple nodes
Evaluation: What is X10 BC?

- A benchmark implementation of Betweenness Centrality
- Available from X10 source distribution from

http://x10.svn.sourceforge.net/viewvc/x10/benchmarks/trunk/BC/
Evaluation: Elapsed time of X10 BC on multiple nodes

Elapsed time for X10 BC on multiple nodes

Graph Scale:
- RMAT

Evaluation

- Introduction
- Research Problem
- Related Work
- X10
- Library Design
- Implementation
- Evaluation
- Conclusion
Evaluation: Elapsed time of X10 BC on multiple nodes

Elapsed time for X10 BC on multiple nodes

RMAT
Graph Scale

Evaluation
Conclusion
Evaluation : Degree Distribution calculation on KAIST Twitter dataset

- Contains 41.7 million user profiles represented as follower/followee relationship
- Contains 1.47 billion edges
- The dataset of 11GB (on GPFS) was scattered into 5454 files each of 2MB in size
- Results (Three times average)
  - Data loading : 40 minutes
  - Get vertex count : 81 seconds
  - Get edge count : 93 seconds
  - In/out degree calculation: 1 hour and 12 minutes
Conclusion

- Objective of this paper: Introduce the design and some initial experiment results of ScaleGraph
- Concrete abstractions for representing graph data on distributed environments while providing simple API for X10 application developer community
- Distinguishing feature: Graph is distributed across places
  - Difficult to load.
  - Solved by graph scattering
Current status and Future Work

- Five Developers (2 part-time)
- 14,000 lines of X10 code
- Currently working on
  - Improving scalability of Algorithms. Experiments are done on Tsubame 2.0
    - Degree, BC, Spectral Clustering, PageRank, Random Walk With Restart
  - Improving scalability of Data representation
    - CyclicPlainGraph
  - Implement other graph algorithms
    - Graph pattern matching, graph property calculation algorithms
  - Getting ready for Release 1.0 soon. Also planning for release 2.0.
- In Future
  - Support for other complex graph algorithms and analysis techniques
  - Usage of heterogenous hardware
Acknowledgement

• This research was partly supported by the Japan Science and Technology Agency (JST) Core Research of Evolutionary Science and Technology (CREST)