[CoolName++]: A Graph Processing Framework for Charm++

Hassan Eslami, Erin Molloy, August Shi, Prakalp Srivastava
Laxmikant V. Kale

Charm++ Workshop
University of Illinois at Urbana-Champaign

{*eslami2,emolloy2,awshi2,psrivas2,kale*}*@illinois.edu*

May 8, 2015

Graphs and networks

A graph is a set of vertices and a set of edges, which describe relationships between pairs of vertices. Data analysts wish to gain insights into characteristics of increasingly large networks, such as

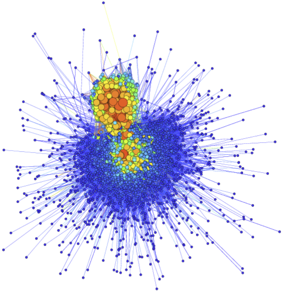
- roads
- utility grids
- internet
- social networks
- protein-protein interaction networks
- gene regulatory processes[1]

[1] X. Zhu, M. Gerstein, and M. Snyder. "Getting connected: analysis and principles of biological networks". In: *Genes and Development* 21 (2007), pp. 1010–24. DOI: 10.1101/gad.1528707.

Why large-scale graph processing?

Large social networks[2]

- 1 billion vertices, 100 billion edges
- 111 PB adjacency matrix
- 2.92 TB adjacency list
- 2.92 TB edge list



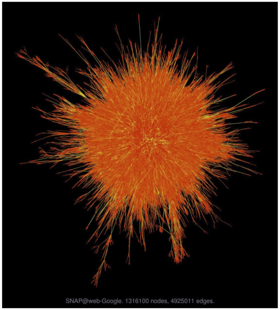
Twitter graph from Gephi dataset
(http://www.gephi.org)

[2] Paul Burkhardt and Chris Waring. *An NSA Big Graph Experiment*. Technical Report NSA-RD-2013-056002v1. May 2000.

Why large-scale graph processing?

Large web graphs[3]

- 50 billion vertices, 1 trillion edges
- 271 PB adjacency matrix
- 29.5 TB adjacency list
- 29.1 TB edge list



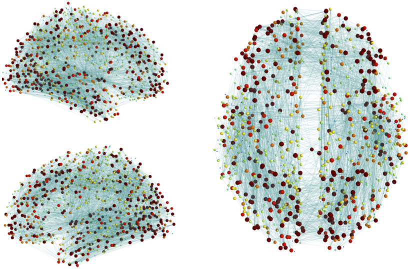
Web graph from the SNAP database (http://snap.stanford.edu/data)

[3] Paul Burkhardt and Chris Waring. *An NSA Big Graph Experiment*. Technical Report NSA-RD-2013-056002v1. May 2000.

Why large-scale graph processing?

Large brain networks[4]

- 100 billion vertices, 100 trillion edges
- 2.08 $mN_A \cdot bytes^2$ (molar bytes) adjacency matrix
- 2.84 PB adjacency list
- 2.84 PB edge list



Human connectome.
Gerhard et al., *Frontiers in Neuroinformatics* **5**(3), 2011

[4] Paul Burkhardt and Chris Waring. *An NSA Big Graph Experiment*. Technical Report NSA-RD-2013-056002v1. May 2000.

Challenges of parallel graph processing

Many graph algorithms result in[5]...

- ...a large volume of fine grain messages.
- ...little computation per vertex.
- ...irregular data access.
- ...load imbalances due to highly connected communities and high degree vertices.

[5] A. Lumsdaine et al. "Challenges in parallel graph processing". In: *Parallel Processing Letters* 17.1 (2007), pp. 5–20.

Vertex-centric graph computation

- Introduced in Google's graph processing framework, Pregel[6]
- Based on the Bulk Synchronous Parallel (BSP) model
- A series of global supersteps are performed, where each **active** vertex in the graph
    1. processes incoming messages from the previous superstep
    2. does some computation
    3. sends messages to other vertices
- Algorithm terminates when all vertices are **inactive** (i.e., they vote to halt the computation) and there are no messages in transit.
- Note that supersteps are synchronized via a global barrier
    - Costly
    - Simple and versatile

[6] G. Malewicz et al. "Pregel: a system for large-scale graph processing". In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. SCM, 2010, pp. 135–146.

Our contributions

- Implement and optimize a vertex-centric graph processing framework on top of Charm++
- Evaluate performance for several graph applications
  - Single Source Shortest Path
  - Approximate Graph Diameter
  - Vertex Betweenness Centrality
- Compare our framework to GraphLab[7]

[7] Yucheng Low et al. "Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud". In: *Proc. VLDB Endow.* 5.8 (Apr. 2012), pp. 716–727. ISSN: 2150-8097. DOI: 10.14778/2212351.2212354. URL: http://dx.doi.org/10.14778/2212351.2212354.

CoolName++ framework overview

- Vertices are divided amongst parallel objects (Chares), called Shards.
- Shards handle the receiving and sending of messages between vertices.
- Main Chare coordinates the flow of computation by initiating supersteps.

User API

Implementation of graph algorithms requires the formation of a

- vertex class
- compute member function

In addition, users **may** also define functions for

- graph I/O
- mapping vertices to Shards
- combining messages being sent to and received by the same vertex

Example vertex constructor

Algorithm 1 Constructor for SSSP

```
if vertex is the source vertex then
    setActive()
    distance = 0
else
    distance = ∞
end if
```

Example vertex compute function

Algorithm 2 Compute function for SSSP

```
min_dist = isSource() ? 0 : ∞
for each of your messages do
    if message.getValue() < min_dist then
        min_dist = message.getValue()
    end if
end for
if min_dist < distance then
    distance = min_dist
    sendMessageToNeighbors(distance + 1)
end if
voteToHalt()
```

Implementation - the .ci file

```
mainchare Main {
    entry Main(CkArgMsg* m);
    entry [reductiontarget] void start();
    entry [reductiontarget] void checkin(int n, int counts[n]);
};

group ShardCommManager {
    entry ShardCommManager();
}

array [1D] Shard {
    entry Shard(void);
    entry void processMessage(int superstepId, int length,
        std::pair<uint32_t, MessageType> msg[length]);
    entry void run(int mcount);
};
```

Implementation - run() function

```
void Shard::run(int messageCount) {
    // Start a new superstep
    superstep = commManagerProxy.ckLocalBranch()->getSuperstep();
    ...
    if (messageCount == expectedNumberOfMessages) {
        startCompute();
    } else {
        // Continue to wait for messages in transit
    }
}

void Shard::startCompute() {
    for (vertex in activeVertices) {
       vertex.compute(messages[vertex]);
    }
    for (vertex in inactiveVertices with incoming messages) {
        vertex.compute(messages[vertex]);
    }
    managerProxy.ckLocalBranch()->done();
}
```

Optimizations

Messages between vertices tend to be small but still incur overhead.

- Shards buffer messages
- User-defined message combine function (send/receive)

Example message combiner

Algorithm 3 Combine function for SSSP

```
if message1.getValue() < message2.getValue() then
    return message1
else
    return message2
end if
```

Applications

We consider three applications for the preliminary evaluation of our framework.

- Single Source Shortest Path (SSSP)
- Graph Diameter
  - Longest shortest path between any two vertices
  - We implement the **approximate** diameter with Flajolet-Martin(FM) bitmasks[8].
- Betweenness Centrality of a Vertex
  - Number of shortest paths between every two vertices that pass through a vertex divided by the total number of shortest paths between every two vertices
  - We implement Brandes' algorithm[9].

[8] P. Flajolet and G. N. Martin. "Probabilistic Counting Algorithms for Data Base Applications". In: *Journal of Computer and System Sciences* 31.2 (1985), pp. 182–209.

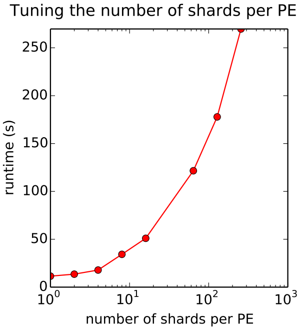
[9] U. Brandes. "A faster algorithm for betweenness centrality". In: *Journal of Mathematical Sociology* 25.2 (2001), pp. 163–177.

Tuning experiments

We want to tune parameters, specifically

- Number of Shards per PE
- Size of message buffer (i.e., the number of messages in the buffer)

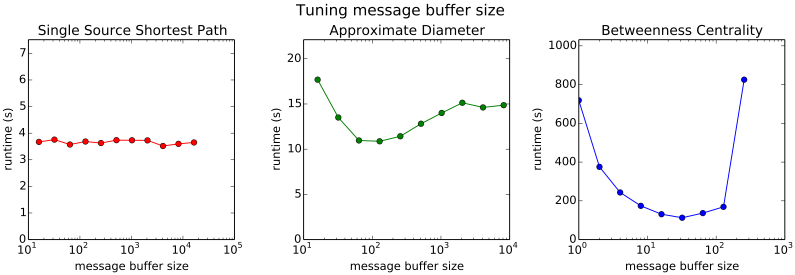
Number of Shards per PE



Approximate diameter on a graph of sheet metal forming (0.5M vertices, 8.5M edges).

All subsequent experiments use one shard per PE.

Size of message buffer



Varying message buffer size on a graph of sheet metal forming (0.5M vertices, 8.5M edges).

In the following experiments, we use a buffer size of 64 for SSSP, 128 for Approximate Diameter, and 32 for Betweenness Centrality.

Preliminary data for strong scalability

We examine three undirected graphs from the Stanford Large Network Dataset Collection (SNAP)[10].

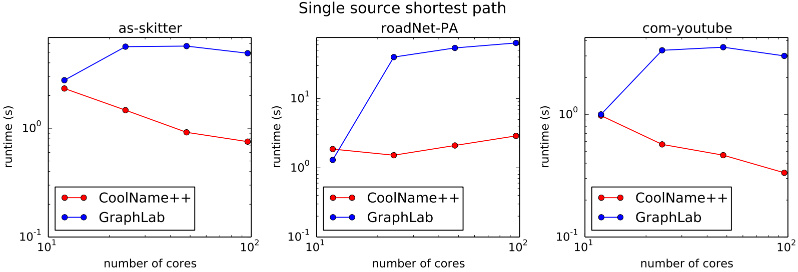
- "as-skitter"
  - Internet topology graph from trace-routes run daily in 2005
  - 1.7M vertices and 11M edges
- "roadNet-PA"
  - Road network of Pennsylvania
  - 1.1M vertices and 1.5M edges
- "com-Youtube"
  - Youtube online social network
  - 1.1M vertices and 3M edges

We compare our framework to GraphLab[11], a state-of-the-art graph processing framework originally developed at CMU.

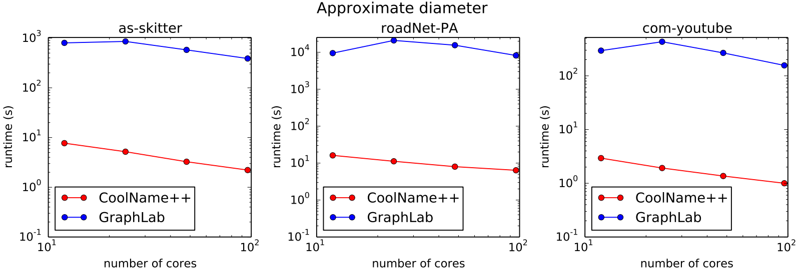
[10] Jure Leskovec and Andrej Krevl. *SNAP Datasets: Stanford Large Network Dataset Collection*. http://snap.stanford.edu/data. June 2014.

[11] Yucheng Low et al. "Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud". In: *Proc. VLDB Endow.* 5.8 (Apr. 2012), pp. 716–727. ISSN: 2150-8097. DOI: 10.14778/2212351.2212354. URL: http://dx.doi.org/10.14778/2212351.2212354.

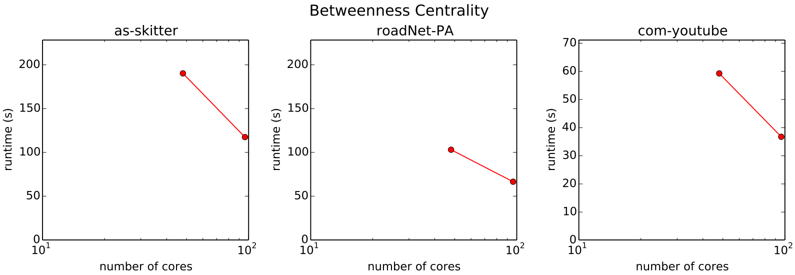
Strong scalability of single source shortest path (SSSP)



Strong scalability of approximate diameter



Strong scalability of betweenness centrality



Conclusions

We ...

- ...implemented a scalable vertex-centric framework on Charm++.
- ...implemented three applications using our framework.
- ...get promising preliminary results in comparison to GraphLab.
- ...hope to test on larger graphs and a greater number of compute cores.

Future work

- Parallel I/O
- Vectorization of compute function
- Aggregators (e.g., global variables computed across vertices)
- Graph mutability
    - Vertex addition/deletion
    - Edge addition/deletion
    - Edge contraction (message redirection)