Implementing Performance Portable Graph Algorithms Using Task-Based Execution

Ümit V. Çatalyürek
Georgia Institute of Technology & Amazon Web Services*

Joint work with
Abdurrahman Yaşar, Georgia Institute of Technology & NVIDIA
Sivasankaran Rajamanickam & Jonathan Berry, Sandia National Laboratories

* This presentation describes work performed at Georgia Tech and is not associated with Amazon.
Graphs are Ubiquitous

They are growing. Up to billions of vertices and edges

Fast, efficient analysis is important and pervasive

Many graph processing frameworks have been proposed

Image credits:

Jenn Caulfield, Social network vector illustration, 2018
Gerhard et al., Frontiers in Neuroinformatics 5(3), 2011

Albert-László Barabási/BarabasiLab 2019
Caleb Jonson, How to Visualize Your Twitter Network, 2014
Heterogeneous Systems are Here

Our current target environment

A Single Computing Node

More are coming…
The Crux

How can we develop efficient parallel graph algorithms that run well on **shared-memory and heterogeneous systems** as well as distributed-memory systems?

Block-based graph algorithms offer a good compromise between efficient parallelism and architecture agnostic algorithm design.

Parallel Graph Algorithms by Block (PGAbB)

- Data/Computation Partitioning
- Performance Portability
- Block-Based Alg. Design
We have three design goals:

- An expressive programming model
- Execute graph kernel operations on different architectures.
  - Combine the results coming from different architectures
- Address major efficient parallel graph algorithm implementation challenges at behind the scenes.
System Overview

PGAbB API

Scheduler

Parallel Dispatch

Data Structures

Layout Manager

Partitioner

SARMA

I/O Handler

PIGO

An Overview of PGAbB

https://github.com/GT-TDAlab/PIGO

https://github.com/GT-TDAlab/SARMA

Çatalyürek "Implementing Performance Portable Graph Algorithms Using Task-Based Execution"
Algorithm Design Steps

Block List Composition

Attribute Assignment

Execution Handling

Kernel Development

$\mathcal{P}_G$, $\mathcal{P}_C$

PGABB API

$\mathcal{I}_B$, $\mathcal{I}_A$, $\mathcal{E}$

$\mathcal{K}_H$, $\mathcal{K}_D$

Generic and Custom Block-List Generators.

Required

Optional

Host and Device Kernels

$\mathcal{I}$ Before Iter. $\mathcal{A}$ After Iter. $\mathcal{E}$ WorkEstim.
### Execution Flow

**Layout Management**

- $B_0$, $B_1$, $B_2$
- $B_3$, $B_4$, $B_5$
- $B_6$, $B_7$, $B_8$

**Block List Composition**

- $P_G$ to $I_B$

**Symmetric Partitioner**

- TRUE

**I/O Handling**

- START

**Sorting; $E$**

- $L_5$, $L_1$, $L_3$, $L_0$, $L_2$, $L_4$

**Scheduling**

- GPU
- CPU
- CPU
Toy Graph
Symmetric Rectilinear Partitioning

<table>
<thead>
<tr>
<th>T₀,₀</th>
<th>T₀,₁</th>
<th>T₀,₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>T₁,₀</td>
<td>T₁,₁</td>
<td>T₁,₂</td>
</tr>
<tr>
<td>T₂,₀</td>
<td>T₂,₁</td>
<td>T₂,₂</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T₀,₀</th>
<th>T₀,₁</th>
<th>T₀,₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>T₁,₀</td>
<td>T₁,₁</td>
<td>T₁,₂</td>
</tr>
<tr>
<td>T₂,₀</td>
<td>T₂,₁</td>
<td>T₂,₂</td>
</tr>
</tbody>
</table>

A simple example

- Restricted rectilinear partitioning:
  - Can be obtained by **aligning the same partition vector** to rows and columns.
  - We showed this problem is **NP-Complete** too.
  - We proposed several heuristics and optimizations.
- PGAbB can be used with 1D and 2D partitioning. We will use 2D symmetric partitioning in this talk.
A block \((B_i)\): Set of edges.

Graph, \(G = \bigcup B_i\), and \(\cap B_i = \emptyset\)

\[B_0 = \{(0,1), (0,2), (1,0), (2,0)\}\]
\[B_1 = \{(1,3), (1,4), (2,3)\}\]
\[B_2 = \{(3,1), (3,2), (4,1)\}\]
\[B_3 = \{(3,4), (3,5), (4,3), (4,5), (5,3), (5,4)\}\]
Block List

Block list \((L_j = \langle B_i, B_1, \ldots, B_k \rangle)\) : list of ordered block references based on a rule.

\[ L_0 = \langle B_0 \rangle \]
\[ L_1 = \langle B_1, B_3 \rangle \]
\[ L_2 = \langle B_0, B_2, B_3 \rangle \]
\[ L_3 = \langle B_3 \rangle \]

Combined Example

Generalized Example
Categorizing Graph Algorithms

- PageRank
- HITS
- Shiloach-Vishkin
- Afforest

- kCore
- BFS
- kTruss

- Dijkstra
- MiniTri
- Jaccard Rank

- Single Block
- Bulk Sync.

- Activation Based

- Multi Block Pattern Based

Butterfly Counting
Triangle Counting
Floyd-Warshall
IA³ 2021 - Nov 15, 2021
A kernel is functor that takes a block list as input

\[ \text{PageRank} = \bigcup_{i} \text{PR}(\langle B_i \rangle) \]
A task, $T_i$, is defined with a kernel that operates on a block list.
Attributes

\[ \begin{array}{c|c|c}
B_0 & B_1 & B_2 \\
\hline
\text{Vertex} & \text{Edge} & \text{Global} \\
\hline
\text{Diagonal blocks} & \text{Blocks} & \text{Custom} \\
\text{Ref. to Source and Destination} & \text{Self} & \text{Custom} \\
\text{Reduction Methods} & \text{Reduction Methods} & \text{Custom} \\
\end{array} \]
## Implemented Algorithms

<table>
<thead>
<tr>
<th>Block-List</th>
<th>Attribute</th>
<th>Before Iter.</th>
<th>After Iter.</th>
<th>Host &amp; Device Kernels</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PageRank</strong></td>
<td>Single-Block</td>
<td>Vertex</td>
<td>-</td>
<td>Check Err.</td>
</tr>
<tr>
<td><strong>Shiloach-Vishkin</strong></td>
<td>Single-Block</td>
<td>Global: Array, counter</td>
<td>Reset Counter</td>
<td>Check counter</td>
</tr>
<tr>
<td><strong>Afforest</strong></td>
<td>Single-Block</td>
<td>Global: Array</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>BFS</strong></td>
<td>Activation</td>
<td>Global: Queues</td>
<td>-</td>
<td>Check Queue</td>
</tr>
<tr>
<td><strong>Triangle Counting</strong></td>
<td>Multi-Block</td>
<td>Global: Counter var.</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Triangle Counting Problem: Find the number of three-cycles (triangles) in an undirected graph $G$.

**Important kernel** which forms the core of:
- community detection,
- dense sub-graph discovery,
- $k$-truss decomposition,
- sub-graph isomorphism etc.
2D Partitioning

Cartesian  Symmetric Rectilinear

(u,v) in B_{0,1}
(v,w) in B_{1,2}
(u,w) in B_{0,2}

Block List: Triple of Blocks

B_{i,j} - B_{j,k} - B_{i,k}

i \leq j \leq k
How to Compose Task List

A Task: $LI((B_{i,j}, B_{j,k}, B_{i,k}))$

<table>
<thead>
<tr>
<th></th>
<th>$B_{0,0}$</th>
<th>$B_{0,1}$</th>
<th>$B_{0,2}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B_{0,0}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$B_{0,1}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$B_{0,2}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$B_{1,0}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$B_{1,1}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$B_{1,2}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$B_{2,0}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$B_{2,1}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$B_{2,2}$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Task list composition.

Workload estimation.

Sorting task list
Hybrid Execution

Heavier Tasks: GPU
from heavier to lighter

Lighter Tasks: CPU
from lighter to heavier

Execution Queue
Sequential Execution Time Comparison in CPU

bbTC [TPDS’21] is available at http://github.com/GT-TDAlab/bbTC

Latapy
Latapy; “Main-memory triangle computations for very large (sparse (power-law)) graphs”; TCS’08.

TCM
Shun and Tangwongsan; “Multicore triangle computations without tuning”; ICDE’15.

kkTri
Wolf et al.; “Fast linear algebra-based triangle counting with kokkoskernels”; HPEC’17.

TriCore (will be used next slide)
Liu et al.; “Tricore: Parallel triangle counting on gpus”; SC’18

Even sequential bbTC outperforms other algorithms in all graph instances.
Comparison with the state-of-the-art

Running on a system with 2 x Power9 + 2 V100s

Even bbTC-GPU outperforms fastest GPU code TriCore*

*TriCore starts everything in GPU memory, and it is highly unstable: deviates up to 40%.
**Related Work**

**Frameworks in Our Experiments**

**GAPBS**: Beamer, et al., 2015. “The GAP benchmark suite.”, ArXiV


**LAGraph**: Davis. 2019. “Algorithm 1000: SuiteSparse: GraphBLAS: Graph algorithms in the language of sparse linear algebra”, TOMS


Experimental Setup

- **Power9 (2 x 16 x 4)** CPUs with 2 Volta100 GPUs.
  - 320 GB Host Memory. 32 GB Device Memory.
  - CPU-GPU bandwidth: ~60GB/s

- **Dataset:** 44 graphs (real-world and synthetic), 100M-2.1B Edges
  - SuiteSparse, Konect, Snap
  - Converted to undirected and removed self-loops, duplicate edges.
  - In this talk: We are going to cover 7 of them in detail

- **Algorithms:** SV/LP, Best CC, PR, BFS, TC

- **PGAbB:** Kokkos at the backend with OpenMP (Host) and Cuda (Device)
## Selected Dataset

<table>
<thead>
<tr>
<th>Graph</th>
<th>Number of Vertices</th>
<th>Number of Edges</th>
<th>Number of Triangles</th>
<th>Clustering Coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>Twitter7</td>
<td>41.6 M</td>
<td>1.2 B</td>
<td>34.8 B</td>
<td>0.001</td>
</tr>
<tr>
<td>Com-Orkut</td>
<td>3 M</td>
<td>117 M</td>
<td>627 M</td>
<td>0.041</td>
</tr>
<tr>
<td>Sk-2005</td>
<td>50.6 M</td>
<td>1.8 B</td>
<td>84.9 B</td>
<td>0.002</td>
</tr>
<tr>
<td>Kmer_V1r</td>
<td>214 M</td>
<td>232 M</td>
<td>49</td>
<td>0.000</td>
</tr>
<tr>
<td>Europe-OSM</td>
<td>50.9 M</td>
<td>54.1 M</td>
<td>61 K</td>
<td>0.003</td>
</tr>
<tr>
<td>Myciel.19</td>
<td>393 K</td>
<td>451 M</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Kron-Scale21</td>
<td>2.1 M</td>
<td>91 M</td>
<td>8.8 B</td>
<td>0.044</td>
</tr>
</tbody>
</table>
## Experiments on Selected Graphs

<table>
<thead>
<tr>
<th></th>
<th>Social</th>
<th>Web</th>
<th>Gene</th>
<th>Road</th>
<th>Synthetic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>twitter7</td>
<td>Orkut</td>
<td>sk-2005</td>
<td>kmer_V1r</td>
<td>eu_osm</td>
</tr>
<tr>
<td><strong>Galois</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PR</td>
<td>0.83</td>
<td>1.01</td>
<td>1.01</td>
<td>0.89</td>
<td>1.03</td>
</tr>
<tr>
<td>SV/LP</td>
<td>8.40</td>
<td>1.71</td>
<td>1.68</td>
<td>2.29</td>
<td>1.81</td>
</tr>
<tr>
<td>CC</td>
<td>0.84</td>
<td>1.56</td>
<td>0.98</td>
<td>0.64</td>
<td>0.64</td>
</tr>
<tr>
<td>BFS</td>
<td>0.26</td>
<td>0.59</td>
<td>0.46</td>
<td>0.34</td>
<td>2.14</td>
</tr>
<tr>
<td>TC</td>
<td>0.69</td>
<td>1.06</td>
<td>0.63</td>
<td>0.90</td>
<td>1.21</td>
</tr>
<tr>
<td><strong>Ligra</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PR</td>
<td>0.39</td>
<td>0.60</td>
<td>0.99</td>
<td>0.43</td>
<td>0.53</td>
</tr>
<tr>
<td>SV/LP</td>
<td>1.24</td>
<td>0.70</td>
<td>1.05</td>
<td>0.18</td>
<td>0.02</td>
</tr>
<tr>
<td>CC</td>
<td>0.02</td>
<td>0.04</td>
<td>0.00</td>
<td>0.02</td>
<td>0.01</td>
</tr>
<tr>
<td>BFS</td>
<td>0.61</td>
<td>0.67</td>
<td>0.93</td>
<td>0.68</td>
<td>0.16</td>
</tr>
<tr>
<td>TC</td>
<td>0.31</td>
<td>0.35</td>
<td>0.12</td>
<td>0.30</td>
<td>0.17</td>
</tr>
<tr>
<td><strong>LAGraph</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PR</td>
<td>0.75</td>
<td>0.98</td>
<td>0.60</td>
<td>0.75</td>
<td>0.65</td>
</tr>
<tr>
<td>SV/LP</td>
<td>14.24</td>
<td>1.64</td>
<td>0.89</td>
<td>0.30</td>
<td>0.13</td>
</tr>
<tr>
<td>CC</td>
<td>0.17</td>
<td>0.21</td>
<td>0.12</td>
<td>0.14</td>
<td>0.05</td>
</tr>
<tr>
<td>BFS</td>
<td>0.79</td>
<td>0.33</td>
<td>0.77</td>
<td>0.27</td>
<td>0.33</td>
</tr>
<tr>
<td>TC</td>
<td>0.38</td>
<td>0.87</td>
<td>0.66</td>
<td>0.29</td>
<td>0.16</td>
</tr>
<tr>
<td><strong>Galois-GPU</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PR</td>
<td>0.00</td>
<td>2.72</td>
<td>0.00</td>
<td>1.01</td>
<td>1.49</td>
</tr>
<tr>
<td>SV/LP</td>
<td>0.00</td>
<td>3.67</td>
<td>0.00</td>
<td>2.43</td>
<td>2.71</td>
</tr>
<tr>
<td>CC</td>
<td>0.00</td>
<td>0.46</td>
<td>0.00</td>
<td>1.16</td>
<td>0.99</td>
</tr>
<tr>
<td>BFS</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>TC</td>
<td>1.03</td>
<td>0.85</td>
<td>0.90</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td><strong>Gumrock</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PR</td>
<td>0.00</td>
<td>1.28</td>
<td>0.00</td>
<td>1.44</td>
<td>1.34</td>
</tr>
<tr>
<td>SV/LP</td>
<td>0.00</td>
<td>1.88</td>
<td>0.00</td>
<td>3.18</td>
<td>1.22</td>
</tr>
<tr>
<td>CC</td>
<td>0.00</td>
<td>0.24</td>
<td>0.00</td>
<td>1.51</td>
<td>0.44</td>
</tr>
<tr>
<td>BFS</td>
<td>4.61</td>
<td>1.48</td>
<td>0.00</td>
<td>3.59</td>
<td>0.80</td>
</tr>
<tr>
<td>TC</td>
<td>0.00</td>
<td>0.74</td>
<td>0.00</td>
<td>0.04</td>
<td>0.02</td>
</tr>
<tr>
<td><strong>PGABB</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PR</td>
<td>4.64</td>
<td>4.67</td>
<td>0.80</td>
<td>0.53</td>
<td>0.64</td>
</tr>
<tr>
<td>SV/LP</td>
<td>18.02</td>
<td>5.95</td>
<td>1.90</td>
<td>5.73</td>
<td>2.95</td>
</tr>
<tr>
<td>CC</td>
<td>1.25</td>
<td>1.53</td>
<td>2.14</td>
<td>1.91</td>
<td>0.96</td>
</tr>
<tr>
<td>BFS</td>
<td>0.16</td>
<td>0.89</td>
<td>0.77</td>
<td>0.90</td>
<td>0.33</td>
</tr>
<tr>
<td>TC</td>
<td>3.02</td>
<td>3.01</td>
<td>1.69</td>
<td>1.11</td>
<td>3.91</td>
</tr>
</tbody>
</table>
Overall Comparison

PGAbB performs \textbf{1.6x to 5.7x} better than state-of-the-art in the median.

Galois performs the second. GAPBS performs the third.
Conclusion and Future Work

In this work we proposed PGAbB which provides

- an easy block-based programming model for leveraging heterogenous architectures.
- computation and data partitioning strategies for maximal usage of the available resources.
- simple and effective scheduling strategies for CPU and/or GPU processing of different graph kernels.

We are currently working on:

- Simplifying the user API.
- Memory hierarchy aware smarter block fetching.
- Open-source software release.
- Future work: Hypergraph-based locality aware different scheduling policies.
Thanks

- For more information
  - email umit@gatech.edu
  - Visit tda.gatech.edu

- Acknowledgement of Support