GraphBIG: Understanding Graph Computing in the Context Of Industrial Solutions

Lifeng Nai, Hyesoon Kim (Georgia Tech)
Yinglong Xia, Ilie Tanase, Ching-Yung Lin (IBM Research)
This is the Big Data era

Big Data are linked
WHAT IS GRAPH COMPUTING

Graph traversal?

This is NOT the FULL picture
The GRAPH can be
- Big or Small
The GRAPH can be
- Static or Dynamic
The GRAPH can be
- Property or Bayesian
Graph computing contains a BIG scope

- *Traversal ≠ Graph Computing*

Understand **Full-spectrum** Graph Computing
Understand **full-spectrum** graph computing

- Diverse workloads + Framework

Propose an open-source benchmark suite: **GraphBIG**

- Workloads from real-world use cases
- Cover major graph computing types and data types
- Both CPU and GPU implementations

An open-source graph framework: **OpenG**

- Designed from scratch
- Similar design methodology as **IBM System G** commercial toolkits
OUTLINE

<table>
<thead>
<tr>
<th>Motivation</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphBIG: Key factors</td>
</tr>
<tr>
<td>Characterizations</td>
</tr>
<tr>
<td>Conclusion</td>
</tr>
</tbody>
</table>
OpenG Framework

Vertex-centric Data Representation

Representative Graph Workloads

Graph Datasets
GRAPHBIG

OpenG Framework

Vertex-centric Data Representation

Representative Graph Workloads

Graph Datasets
GRAPHBIG: FRAMEWORK

Graph applications ➔ Framework primitives

OpenG: IBM System G-like Framework

% of Execution Time in Framework

Average 76%

IBMP System G GeorgiaTech comparch
GRAPHBIG

OpenG Framework

Vertex-centric Data Representation

Representative Graph Workloads

Graph Datasets
GRAPHBIG: DATA REPRESENTATION

(a) Graph G

(b) CSR Representation of G

(c) Vertex-centric Representation of G

Vertices

Edges

Edge Properties

Vertex Properties

1 2 3 4 5

1 2 6 8 10

2 1 3 4 5 2 5 2 5 2 3 4

Vertex Adjacency List

Vertex 1

Vertex 2

Vertex 3

Vertex 4

Vertex 5

2

1 3 4 5

2 5

2 5

2 3 4

2 3 4

2 3 4
GRAPHBIG

OpenG Framework

Vertex-centric Data Representation

Representative Graph Workloads

Graph Datasets
GRAPHBIG: WORKLOAD SELECTION

Coverage
- Workloads cover all computation types

Representativeness
- Workloads are selected from real-world use cases
GRAPHBIG: COMPUTATION TYPES

<table>
<thead>
<tr>
<th>Computation on graph structure (CompStruct)</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Example: Breadth-first search</td>
</tr>
<tr>
<td>- Irregular access pattern, heavy read access</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Computation on graph property (CompProp)</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Example: Belief propagation</td>
</tr>
<tr>
<td>- Heavy numeric operations on graph property</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Computation on dynamic graph (CompDyn)</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Example: Streaming Graph</td>
</tr>
<tr>
<td>- Dynamic graph structure, dynamic memory usage</td>
</tr>
</tbody>
</table>
GRAPHBIG: WORKLOAD SELECTION

Selected from **21 real-world use cases** of IBM System G

(A) # of Use Cases with Each Workload

(B) Distribution of Selected Use Cases in 6 Categories
# GRAPHBIG: WORKLOADS

<table>
<thead>
<tr>
<th>Category</th>
<th>Workload</th>
<th>Computation Type</th>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Graph traversal</strong></td>
<td>BFS</td>
<td>CompStruct</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td>DFS</td>
<td>CompStruct</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td><strong>Graph update</strong></td>
<td>Graph construction (GCons)</td>
<td>CompDyn</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Graph update (GUp)</td>
<td>CompDyn</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Topology morphing (TMorph)</td>
<td>CompDyn</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td><strong>Graph analytics</strong></td>
<td>Shortest path (SPath)</td>
<td>CompStruct</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td>kCore</td>
<td>CompStruct</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td>Connected component (CComp)</td>
<td>CompStruct</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td>Graph coloring (GColor)</td>
<td>CompStruct</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td>Triangle counting (TC)</td>
<td>CompProp</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td>Gibbs Inference (GI)</td>
<td>CompProp</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td><strong>Social analytics</strong></td>
<td>Betweenness Centrality (BCentr)</td>
<td>CompStruct</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td>Degree Centrality (DCentr)</td>
<td>CompStruct</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>
GRAPHBIG

- OpenG Framework
- Vertex-centric Data Representation
- Representative Graph Workloads
- Graph Datasets
**GRAPHBIG: DATA TYPES**

Big Data includes all sorts of Networks

<table>
<thead>
<tr>
<th>Type 1</th>
<th>Type 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Social/Economic/Political Network</td>
<td>Information/Knowledge Network</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type 3</th>
<th>Type 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nature/Bio/Cognitive Network</td>
<td>Man-Made Technology Network</td>
</tr>
</tbody>
</table>
## GRAPHBIG: DATASETS

<table>
<thead>
<tr>
<th>Data set</th>
<th>Type</th>
<th>Vertex #</th>
<th>Edge #</th>
</tr>
</thead>
<tbody>
<tr>
<td>Twitter Graph</td>
<td>Type 1</td>
<td>120M</td>
<td>1.9B</td>
</tr>
<tr>
<td>IBM Knowledge Repo</td>
<td>Type 2</td>
<td>154K</td>
<td>1.72M</td>
</tr>
<tr>
<td>IBM Watson Gene Graph</td>
<td>Type 3</td>
<td>2M</td>
<td>12.2M</td>
</tr>
<tr>
<td>CA Road Network</td>
<td>Type 4</td>
<td>1.9M</td>
<td>2.8M</td>
</tr>
<tr>
<td>LDBC Graph</td>
<td>Synthetic</td>
<td>Any</td>
<td>Any</td>
</tr>
</tbody>
</table>
CHARACTERIZATION

Methodology

- Real machine + hardware performance counters
- CPU: tool integrated within benchmarks
- GPU: CUDA nvprof
## CHARACTERIZATION

<table>
<thead>
<tr>
<th>Processor</th>
<th>Type</th>
<th>Xeon E5-2670</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>2.6 GHz</td>
<td></td>
</tr>
<tr>
<td>Core #</td>
<td>2 sockets x 8 cores x 2 threads</td>
<td></td>
</tr>
<tr>
<td>Cache</td>
<td>32KB L1, 256KB L2, 20MB L3</td>
<td></td>
</tr>
<tr>
<td>Memory BW</td>
<td>51.2 GB/s (DDR3)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GPU</th>
<th>Type</th>
<th>Nvidia Tesla K40</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA Core</td>
<td>2880</td>
<td></td>
</tr>
<tr>
<td>Memory</td>
<td>12 GB</td>
<td></td>
</tr>
<tr>
<td>Memory BW</td>
<td>288 GB/s</td>
<td></td>
</tr>
<tr>
<td>Frequency</td>
<td>Core-745 MHz, mem-3 GHz</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>System</th>
<th>Memory</th>
<th>192 GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk</td>
<td>2 TB HDD</td>
<td></td>
</tr>
<tr>
<td>OS</td>
<td>RHEL 6</td>
<td></td>
</tr>
</tbody>
</table>
CHARACTERIZATION

Showcase (Data: LDBC-graph 1M vertices)
- CPU execution time breakdown
- CPU core analysis
- CPU cache performance
- GPU divergence
- GPU speedup

More experiment results can be found in the paper
- More analysis (memory bandwidth, IPC, etc.)
- Input data sensitivity (all data sets are evaluated)
CPU: EXECUTION TIME BREAKDOWN

Four categories:
- Frontend, Backend, Bad Speculation, and **Retiring**
CPU: EXECUTION TIME BREAKDOWN

Backend is the bottleneck: memory sub-system issue

- CompProp is different: TC-triangle counting  Gibss-gibbs inference
CPU: CORE ANALYSIS

Significantly high DTLB penalty
ICache and Branch prediction: not a major bottleneck
CPU: CACHE PERFORMANCE

High cache MPKI because of irregular access pattern
GPU DIVERGENCE

<table>
<thead>
<tr>
<th>Branch divergence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch divergence rate = inactive threads per warp/warp size</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Memory divergence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory divergence rate = replayed instructions/issued instructions</td>
</tr>
</tbody>
</table>
High branch & memory divergence

Diverse behaviors across workloads
GPU SPEEDUP

Significant speedup over 16-core CPU
GRAPHBIG: TAKE AWAY

Graph computing has a wide scope, not just BFS

Multiple factors influence graph computing significantly, not only workload algorithms.
- Framework, data representation, datasets

Characterization
- CPU: irregular access pattern -> poor cache performance
- CPU: properly design code hierarchy can avoid ICache issue
- GPU: memory and branch divergence issue
- Diversity across workloads: both CPU and GPU sides
CONCLUSION

Graph Computing has a wide scope. To understand it, we have to consider multiple key factors in a holistic way.

We proposed GraphBIG, a suite of CPU/GPU graph benchmarks based on real-world industrial practices, and characterized it on real machines comprehensively.

GraphBIG is open sourced (BSD license)

- Check: https://github.com/graphbig
- Workloads, datasets, and documents
THANK YOU!

GraphBIG
http://github.com/graphbig

IBM System G
http://systemg.research.ibm.com/

HPArch Lab
http://comparch.gatech.edu/hparch/
BACKUP SLIDES
GRAPHBIG FEATURES

Design
- Framework: *property graph frame based on industrial practices*
- Representativeness: *workloads selected from real-world use cases*
- Coverage: *cover major computation types, much more than just traversal*
- CPU + GPU workloads

Code
- C++ code base: *requiring only c++0x*
- Standalone package: *no external package dependencies*
- Integrated profiling tool: *profiling via hardware performance counters*
GRAPHBIG HANDS-ON

Fetch Code

- Code: https://github.com/graphbig/graphBIG
- Doc: https://github.com/graphbig/GraphBIG-Doc

```bash
-bash:~$ git clone https://github.com/graphbig/graphBIG.git GraphBIG
Cloning into 'GraphBIG'...
remote: Counting objects: 497, done.
remote: Compressing objects: 100% (110/110), done.
remote: Total 497 (delta 57), reused 0 (delta 0), pack-reused 386
Receiving objects: 100% (497/497), 2.07 MiB / 0 bytes/s, done.
Resolving deltas: 100% (229/229), done.
Checking connectivity... done.
-bash:~$
```
Compile

- Require: gcc/g++ (>4.3), gnu make
- Just “make all”
GRAPHBIG HANDS-ON

Test Run
- Just “make run”
- Using default “small” dataset

```
-bash:benchmark$ cd bench_BFS/
-bash:bench_BFS$ make run
Running bfs, output in output.log
```

```
-bash:bench_BFSS$ cat output.log

Benchmark: BFS
loading data...
== 1000 vertices  29790 edges
== time: 0.0530188 sec

BFS root: 31
BFS finish:
== time: 0.00118506 sec
PERF_COUNT_HW_CPU_CYCLES  =>  2581036
PERF_COUNT_HW_INSTRUCTIONS =>  774222
PERF_COUNT_HW_BRANCH_INSTRUCTIONS => 200529
PERF_COUNT_HW_BRANCH_MISSES  =>  10486
PERF_COUNT_HW_CACHE_L1D_READ_ACCESS  =>  309284
PERF_COUNT_HW_CACHE_L1D_READ_MISSS =>  99740
```
More Datasets

- Download: https://github.com/graphbig/graphBIG/wiki/GraphBIG-Dataset
- Untar and specify the correct path in benchmark argument "--dataset"
- Other 3rd party datasets (csv format) are also possible
SCALE UP VS. SCALE OUT

- Scale up before Scale out

Triangle Counting in Twitter Graph

- Total: 34.8 Billion Triangles
  - 40M Users
  - 1.2B Edges

- Hadoop: 1536 Machines, 423 Minutes
- GraphChi: 64 Machines, 1024 Cores, 1.5 Minutes
- GraphLab: 59 Minutes, 1 Mac Mini!
Diverse behaviors across different computation types

- MPKI: L1D, L2, L3
- Branch Miss %: A, B, C
- IPC: A, B, C
- DTLB Miss Cycle %: A, B, C

A – CompStruct  B – CompProp  C – CompDyn
CACHE BEHAVIORS

L1D Hit Rate

L2 Hit Rate

L3 Hit Rate

DTLB Miss Cycle %

IPC

twitter  knowledge  watson  roadnet  LDBC

BFS  kCore  CComp  SPath  DCentr  TC  GUp
GPU ARCH BEHAVIOR

- Cannot fully utilize available memory bandwidth
- Significantly low IPC
GPU DATA SENSITIVITY

Sensitive to input data

Memory divergence shows higher sensitivity

![Graph showing memory and branch divergence sensitivity](image-url)