Scaph: Scalable GPU-Accelerated Graph Processing with Value-Driven Differential Scheduling

Long Zheng\textsuperscript{1}, Xianliang Li\textsuperscript{1}, Yaohui Zheng\textsuperscript{1}, Yu Huang\textsuperscript{1}, Xiaofei Liao\textsuperscript{1}, Hai Jin\textsuperscript{1}, Jingling Xue\textsuperscript{1}, Zhiyuan Shao\textsuperscript{1}, and Qiang-Sheng Hua\textsuperscript{1}

\textsuperscript{1}Huazhong University of Science and Technology
\textsuperscript{2}University of New South Wales

July 15-17, 2020
Graph Processing Is Ubiquitous

Relationship Prediction

Recommendation Systems

Information Tracking

Knowledge Mining
Graph Processing: CPU vs. GPU

GPU V100

- **Performance**: Double-Precision: 7.8TFLOPS, Single-Precision: 15.7TFLOPS
- **InterConnect Bandwidth**: NVLINK 300GB/s
- **Memory**: 32GB HBM2, 1134GB/s


<table>
<thead>
<tr>
<th>Application</th>
<th>Metric</th>
<th>Test Modules</th>
<th>Bigger is better</th>
<th>Dual Cascade Lake 6240 (CPU-Only)</th>
<th>1x V100 32GB SXM2</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMBER [PME-Cellulose_NPT_4fs]</td>
<td>ns/day</td>
<td>DC-Cellulose_NPT</td>
<td>yes</td>
<td>4.73</td>
<td>88.29</td>
</tr>
<tr>
<td>AMBER [PME-Cellulose_NPT_4fs]</td>
<td>NRF</td>
<td>DC-Cellulose_NPT</td>
<td>yes</td>
<td>1x</td>
<td>19x</td>
</tr>
<tr>
<td>AMBER [DC-Cellulose_NVE]</td>
<td>ns/day</td>
<td>PME-Cellulose_NVE</td>
<td>yes</td>
<td>4.73</td>
<td>100</td>
</tr>
<tr>
<td>AMBER [DC-Cellulose_NVE]</td>
<td>NRF</td>
<td>PME-Cellulose_NVE</td>
<td>yes</td>
<td>1x</td>
<td>21x</td>
</tr>
<tr>
<td>AMBER [DC-FactorX_NPT]</td>
<td>ns/day</td>
<td>FactorIX (NPT)</td>
<td>yes</td>
<td>22.88</td>
<td>400</td>
</tr>
<tr>
<td>AMBER [DC-FactorX_NPT]</td>
<td>NRF</td>
<td>FactorIX (NPT)</td>
<td>yes</td>
<td>1x</td>
<td>17x</td>
</tr>
<tr>
<td>AMBER [DC-FactorX_NVE]</td>
<td>ns/day</td>
<td>FactorIX (NVE)</td>
<td>yes</td>
<td>23.41</td>
<td>454</td>
</tr>
</tbody>
</table>

GPU often offers 10x at least speedup over CPU for graph processing.
Graph Processing: CPU vs. GPU

GPU V100

Many real-world graphs cannot fit into GPU memory to enjoy high-performance in-memory graph processing.


<table>
<thead>
<tr>
<th>Application</th>
<th>Metric</th>
<th>Test Modules</th>
<th>Bigger is better</th>
<th>Dual Cascade Lake 6240 (CPU-Only)</th>
<th>1x V100 32GB SXN2</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMBER [PME-Cellulose_NPT_sfa]</td>
<td>ns/day</td>
<td>DC-Cellulose_NPT</td>
<td>yes</td>
<td>4.73</td>
<td>38.29</td>
</tr>
<tr>
<td>AMBER [PME-Cellulose_NPT_sfa]</td>
<td>NSE</td>
<td>DC-Cellulose_NPT</td>
<td>yes</td>
<td>1x</td>
<td>19x</td>
</tr>
<tr>
<td>AMBER [DC-FactorIX_NPT]</td>
<td>NSE</td>
<td>FactorIX (NPT)</td>
<td>yes</td>
<td>21x</td>
<td>100</td>
</tr>
<tr>
<td>AMBER [DC-FactorIX_NVE]</td>
<td>NSE</td>
<td>FactorIX (NVE)</td>
<td>yes</td>
<td>400</td>
<td>21x</td>
</tr>
<tr>
<td>AMBER [DC-FactorIX_NVE]</td>
<td>NSE</td>
<td>FactorIX (NVE)</td>
<td>yes</td>
<td>400</td>
<td>454</td>
</tr>
</tbody>
</table>

Performance

- Double-Precision: 7.8 TFLOPS
- Single-Precision: 15.7 TFLOPS

InterConnect Bandwidth

- NVLINK 300GB/s

Memory

- 32GB HBM2, 1134GB/s

GPU often offers 10x at least speedup over CPU for graph processing.
The significant performance gap between CPU and GPU may severely limit the performance potential expected on the GPU-accelerated heterogeneous architecture.
Existing Solutions on GPU-Accelerated Heterogeneous Architecture

• Totem (PACT’12)
  – Partitioned into two large subgraphs, one for CPU, one for GPU
  – Significant load unbalance

• Graphie (PACT’17)
  – Subgraphs are partitioned and streamed to GPU
  – All subgraphs are transferred in their entirety
  – Bandwidth is wasted

• Garaph (USENIX ATC’17)
  – All the subgraphs are processed on CPU if the active vertices in the entire graph have a lot of (50%) outgoing edges
  – Processed on the host otherwise
A Generic Example of Graph Processing Engine

```
Procedure SimpleSubgraphEngine(Graph G)

Load G's subgraphs in \{G_1, \ldots, G_n\} into the host

VertexInitialization(G)

\( \tilde{G}_{active} \leftarrow \text{FindActiveSubgraph}(G) \)

while \( \tilde{G}_{active} \neq \emptyset \) do

    foreach \( \tilde{G}_i \in \tilde{G}_{active} \) do

        stream \leftarrow \text{DispatchStream}(\tilde{G}_i)

        if \( \tilde{G}_i \) is not resident in GPU memory then

            GBuf \leftarrow \text{AllocateDeviceMemory}()

            TransferData(stream, GBuf, \tilde{G}_i, CPU2GPU)

        Kernel (stream, \tilde{G}_i)

    \tilde{G}_{active} \leftarrow \text{FindActiveSubgraph}(G)

/* Graph Processing Kernel on the GPU */

Procedure Kernel(Subgraph \( \tilde{G} \))

foreach \( v \in \tilde{G}.\text{SetOfVertices} \) do

    if \( v \) is active then

        foreach \( e \in v.\text{outedges} \) do

            if Update(\( v, e \)) = SUCCESS then

                Activate(\( e.\text{destination}_\text{vertex} \))
```

Vertices reside in GPU memory

Edges are streamed to GPU on demand

A graph is partitioned into many slices
A Generic Example of Graph Processing Engine

Vertices reside in GPU memory

Edges are streamed to GPU on demand

A graph is partitioned into many slices

In an iteration, all active subgraphs will be transferred entirely to GPU and processed there.
A Generic Example of Graph Processing Engine

Vertices reside in GPU memory
Edges are streamed to GPU on demand
A graph is partitioned into many slices
In an iteration, all active subgraphs will be transferred entirely to GPU and processed there.
These active subgraphs processed on GPU will activate more destination vertices possibly.
Motivation

This simple graph engine wastes a considerable amount of limited host-GPU bandwidth, limiting performance and scalability further.

<table>
<thead>
<tr>
<th>Algo.</th>
<th>Used</th>
<th>Unused</th>
</tr>
</thead>
<tbody>
<tr>
<td>TW</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC</td>
<td>12.15GB</td>
<td>21.44GB</td>
</tr>
<tr>
<td>SSSP</td>
<td>22.74GB</td>
<td>77.42GB</td>
</tr>
<tr>
<td>MST</td>
<td>25.78GB</td>
<td>106.27GB</td>
</tr>
<tr>
<td>UK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC</td>
<td>43.41GB</td>
<td>688.43GB</td>
</tr>
<tr>
<td>SSSP</td>
<td>81.64GB</td>
<td>1302.85GB</td>
</tr>
<tr>
<td>MST</td>
<td>134.93GB</td>
<td>2099.25GB</td>
</tr>
</tbody>
</table>

Only 6.29%~36.17% transferred data are used

Perf. can be plateaued quickly at #SMX=4

Little gains when more powerful GPUs are used
Characterization of Subgraph Data

The data of a subgraph are changing:

- **Useful Data (UD)**
  - associated with active vertices
  - must be transferred to GPU
- **Potentially Useful Data (PUD)**
  - associated with all future active vertices
  - (not) used in future (current) iteration
- **Never Used Data (NUD)**
  - Converged
  - Never be active again
Characterization of Subgraph Data

The data of a subgraph are changing

- **Useful Data (UD)**
  - associated with active vertices
  - must be transferred to GPU
- **Potentially Useful Data (PUD)**
  - associated with all future active vertices
  - (not) used in future (current) iteration
- **Never Used Data (NUD)**
  - Converged
  - Never be active again

PUD is substantial in earlier iterations but discarded
Contributions

Scaph A scale-up graph processing for large-scale graph on GPU-accelerated heterogeneous platforms

• Value-Driven differential Scheduling
  – distinguish high- and low-value subgraphs in each iteration adaptively

• Value-Driven Graph Processing Engines
  – exploit the most value out of high- and low-value subgraphs to maximize efficiency
Quantifying the Value of a Subgraph

• Conceptually, the value of a subgraph can be measured by its UD used in the current iteration and its PUD used in future iterations.

• The value of a subgraph from the current iteration and MAX-th iteration can be defined as:

\[ Val(\tilde{G}) = \sum_{i=\text{Cur}}^{\text{MAX}} \sum_{v\in \tilde{G}.\text{SetOfVertices}} D(v) \times A^i(v) \]

• The value of a subgraph depends upon its active vertices and their degrees
Value-Driven Differential Scheduling

Procedure VDDSEngine(Graph G)

1. Distribute G’s subgraphs \( \{G_1, \cdots, G_n\} \) to NUMA nodes
2. VertexInitialization(G)
3. \( \tilde{G}_{active} \leftarrow \text{FindActiveSubgraph}(G) \)
4. Transfer VertexStates from GPU to CPU

while \( \tilde{G}_{active} \neq \emptyset \) do

1. foreach \( \tilde{G} \in \tilde{G}_{active} \) do

   if \( \text{Predictor}(\tilde{G}) = \text{"HIGH-VALUE"} \) then

      Push(HVworklist, \( \tilde{G} \))

   else

      Push(LVworklist, \( \tilde{G} \))

   end

2. HVSPEngine(HVworklist)
3. LVSPEngine(LVworklist, VertexStates)
4. \( \tilde{G}_{active} \leftarrow \text{FindActiveSubgraph}(G) \)
5. Transfer VertexStates from GPU to CPU

- G is partitioned and distributed on NUMA nodes
- Vertices on GPU, edges streamed
- Estimate the value of an active subgraph
- Differential Scheduling
  - High-Value Subgraph Engine
  - Low-Value Subgraph Engine
- Updated vertices will be transferred from GPU to CPU. Edges, not modified, are not transferred
Checking If a Subgraph is High Value

- Suppose a subgraph $G$ is a high-value subgraph, its throughput can be measured below:
  $$ T_{HV}(\tilde{G}) = \frac{|UD| + \lambda|PUD|}{|\tilde{G}|/BW + t_{\text{barrier}}} $$

- Suppose a subgraph $G$ is a low-value subgraph, its throughput can be measured below:
  $$ T_{LV}(\tilde{G}) = \frac{|UD|}{|UD|/BW + t_{\text{barrier}}} $$

- Now, $G$ is a high-value subgraph if $T_{HV}(\tilde{G}) \geq T_{LV}(\tilde{G})$. Thus, we need to analyze:
  $$ |UD| + \lambda|PUD| \left(1 + \frac{t_{\text{barrier}}}{|UD|/BW}\right) > |\tilde{G}| $$

- This condition is heuristically simplified below:
  - $|UD|/|\tilde{G}| > \alpha$, which indicates UD is dominant.
  - $|UD_{current}| - |UD_{last}| > 0$ and $|UD|/|\tilde{G}| > \beta$. UD remains a medium level but is growing increasingly over iteration.
  - $\alpha = 50\%, \beta = 30\%$
High-Value Subgraph Processing

- Inspired from CLIP (ATC’17), each high-value subgraph can be scheduled multiple times to exploit *intrinsic value* of a subgraph.
- In a GPU context, subgraph sizes are small.
- We propose a delayed scheduling to exploit PUD *across the subgraphs*.
- Queue-assisted multi-round processing:
  - *k*-level priority queue ($PQ_1$, …, $PQ_k$)
  - Subgraph streamed to *TransSet* asynchronously.
  - A subgraph in $PQ_1$ is scheduled first. Its priority drops by one once processed.
  - Subgraph transfer and scheduling are executed concurrently.
Complexity Analysis

• Time Complexity
  – The queue depth $k$ is expected to be bounded by $BW’/BW$
  – For a typical server ($BW’=224\text{GB/s}$ and $BW=11.4\text{GB/s}$), $k$ can be less than 20, which is typically small.

• Space Complexity
  – $k$-level queue maintains only the indices of the active subgraphs
  – The worst complexity is $O\left(\frac{\text{Mem}_{\text{GPU}} \times \text{sizeof(SubgraphIndex)}}{|\tilde{G}|}\right)$
  – For P100 (GPU memory: $16\text{GB}$, Index size: $4\text{B}$, subgraph size: $32\text{M}$), the space overhead of the queue is $2\text{KB}$, which is small.
Low-Value Subgraph Processing

• NUMA-Aware Load Balancing
  – Intra-node load balancing: The UD extraction for each subgraph is done in its own thread.
  – Inter-node load balancing: A NUMA node is duplicated an equal number of randomly selected subgraphs from the other nodes

• Bitmap-based UD extraction
  – All vertices of a subgraph is stored in a bitmap
  – 1 (0) indicates the corresponding vertex is active (inactive)

• To reduce the fragmentation of the UD-induced subgraphs, we divide each chunk to store a subgraph into smaller tiles.
Limitations (More details in the paper)

• Graph Partition
  – A greedy vertex-cut partition

• Out-of-core solution
  – Using the disk as secondary storage is promising to support even larger graphs

• Performance Profitability
Experimental Setup

- **Baselines**
  - Totem, Graphie, Garaph

- **Graph Size:** 32MB

- **Graph Applications**
  - Typical algorithms: SSSP/CC/MST
  - Actual workloads: Two NNDR/GCS

- **Datasets**
  - 6 real-world graphs:
  - 5 large synthesized RMAT graphs

- **Platforms**
  - Host: E5-2680v4 (512GB memory, two NUMA nodes)
  - GPU: P100 (56 SMXs, 3584 cores, 16GB memory)
Efficiency

• Scaph vs. Totem
  – UD and PUD exploited more fully
  – yields 2.23x~7.64x speedups

• Scaph vs. Graphie
  – Exploit PUD and NUD is discarded
  – yields 3.03x~16.41x speedups

• Scaph vs. Garaph
  – Removing NUD transferred
  – yields 1.93x~5.62x speedups
Effectiveness

- **Scaph-HVSP**: All the low-value subgraphs are misidentified as high-value subgraphs
- **Scaph-LVSP**: All the high-value subgraphs are misidentified as low-value subgraphs
- **Scaph-HBASE**: Differential processing is used but queue-based scheduling is not applied
- **Scaph-LBASE**: A variation of Scaph-LVSP except that every subgraph is streamed entirely

- **Scaph-HBASE vs. Scaph-HVSP**: Significant performance difference shows the effectiveness of our delay-based subgraph scheduling
- **Scaph vs. Scaph-LVSP and Scaph-HVSP**: Scaph obtains the best of both worlds, showing the effectiveness of differential subgraph scheduling
**Sensitivity Study**

- **Varying #SMXs**
  - Significantly more scalable
- **Varying Graph Sizes**
  - Slower performance reduction rate
- **Varying GPU memory**
  - Scaph is nearly insensitive to the GPU memory used
- **GPU generations**
  - Enables significant speedups
Sensitivity Study (con't)

- A1: Scaph-HVSP
- A5: Scaph-LVSP
- A3 represents a nice point for yielding good performance results.
• **VDDS**: The cost of computing the subgraph value is negligible
• **HVSP**: Queue management cost per iteration is as small as 0.79% of total time
• **LVSP**: CPU-GPU bitmap transfer cost per iteration represents 4.3% of total time
Conclusion

Scaph: Scale up graph processing for large graphs on GPU-accelerated heterogenous architectures.

- Subgraph Value Characterization, which quantifies the value of a subgraph adaptively and dynamically
- Value-Driven Differential Scheduling, which adaptively distinguishes high- and low-value subgraphs and dispatches them to an appropriate graph processing engine
- Value-Driven Graph Processing Engines, which squeeze the most value out of high- and low-value subgraphs to maximize efficiency
- It outperforms state-of-the-art heterogeneous graph systems, Totem (4.12×), Graphie (8.93×), and Garaph (3.71×).
Thanks!
longzh@hust.edu.cn