On-The-Fly Parallel Data Shuffling for Graph Processing on OpenCL-based FPGAs

Xinyu Chen\textsuperscript{1}, Ronak Bajaj\textsuperscript{1}, Yao Chen\textsuperscript{2}, Jiong He\textsuperscript{3}, Bingsheng He\textsuperscript{1}, Weng-Fai Wong\textsuperscript{1}, Deming Chen\textsuperscript{4}

\textsuperscript{1}National University of Singapore, \textsuperscript{2}Advanced Digital Sciences Center, \textsuperscript{3}Alibaba Group, \textsuperscript{4}University of Illinois at Urbana-Champaign
Graph processing is widely used in variety of application domains.

- Social networks
- Cybersecurity
- Machine learning

Accelerating graph processing on FPGA has attracted a lot of attention benefiting from:

- Fine grained parallelism
- Low power consumption
- Extreme configurability
Graph processing on HLS-based FPGAs

- Previous RTL-based FPGAs development.
  - Time-consuming
  - Deep understanding of hardware

- To ease the use of FPGAs, HLS tools have been proposed.
  - High-level programming model
  - Hide hardware details
  - Both Intel and Xilinx have HLS tools

- Graph processing on OpenCL-based FPGAs.
GAS model for graph processing

- **Scatter**: for each edge, an update tuple is generated with the format of \(<\text{destination}, \text{value}\>\).
  - E.g. \(<2, x>, <7, y>\) for vertex 1

- **Gather**: accumulate the value to destination vertices.
  - E.g. \(\text{Op}(P_2 , x), \text{Op}(P_7 , y)\)

- **Apply**: an apply function on all the vertices.

GAS model on FPGAs

- **BRAM caching**
  - avoid random memory accesses to property array.
- **Multiple PEs**
  - each PE processes a part of cached data and runs independently.

Example graph

**Data shuffling**

Update tuples to process for vertex 1

\(<2, x>, \ <7, y>\)
Data shuffling

- Widely used for irregular applications.
- The data generated with format of `<dst, value>` is dispatched to ‘dst’ PEs to process.
- Challenges:
  - Run-time data dependency
  - Parallelism

![Diagram showing data shuffling](image)

* Arrows with different colours show a few shuffling examples.
OpenCL does not natively support shuffling

- Fine-grained control logic is not available for OpenCL.

- No vendor-specific extension for shuffling [1].

- OpenCL only does static analysis at compile time, thus cannot extract parallelism in functions with run-time dependency [2].


Potential shuffling solutions with OpenCL

- **Polling**
  - Each PE checks the tuples serially.
  - ‘Bubbles’ are introduced.
  - 8 cycles for dispatching a set of 8 tuples.
Potential shuffling solutions with OpenCL

- **Convergence kernel from [1]**
  - Each PE writes wanted tuples to local BRAM in parallel.
  - The run-time data dependency is not resolved.
  - Initiation interval (II) equals to 284 cycles.

Insights

- Polling: introduces ‘bubbles’.
- Convergence kernel: the run-time dependency is still there.

- What if we know the positions and number of wanted tuples?
  - PEs can directly access the wanted tuples.
  - Cycles needed equal to number of wanted tuples.

- How to know the positions and number of wanted tuples?
  - Decoder based solution.
  - E.g. $2^8$ possibilities, for a set of 8 tuples, since each tuple has two statuses only.
Proposed shuffling

• Calculate the destination PEs.

• Compute an 8-bit MASK by comparing destination PEs with the id of current PE, 0.

• Decode the positions and number of wanted tuples.

• Collect the wanted tuples without “bubbles”.

An example for a set of 8 tuples on PE₀
Proposed shuffling

• No ‘bubbles’ - no cycle wasted on unwanted tuples.

• Resolve the run-time dependency.

• All the modules are pipelined.
Proposed graph processing framework with shuffle

Scatter

Shuffle

N-way PE selection

Data Duplication

Validation

Decoder

Filter

gPE

Gather

DDR

Apply

aPE

Func

C

(2N*32-bit) / read

1

2

3
**Experimental configuration**

- Our experiments are conducted on a Terasic DE5-Net board.
- BFS, SSSP, PageRank and SpMV are used as applications.

<table>
<thead>
<tr>
<th>FPGA</th>
<th>Altera Stratix V GX</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenCL</td>
<td>Intel FPGA SDK 16.1 for OpenCL</td>
</tr>
<tr>
<td>Memory Bandwidth</td>
<td>17GB/s (Peak); 12.5GB/s (at 200MHZ)</td>
</tr>
</tbody>
</table>

**TABLE I: Details of hardware.**

**TABLE II: Graph dataset.**

| Graphs                | $|V|$ | $|E|$ | $D_{avg}$ | $D_{max}$ |
|-----------------------|-----|-----|----------|----------|
| rmat-19-32 (R19) [34] | 524K| 17M | 32       | 90K      |
| rmat-21-32 (R21) [34] | 2M  | 67M | 32       | 211K     |
| mouse-gene (MG) [35]  | 43K | 14.5M | 670   | 8K       |
| web-google (GG) [35] | 875K| 5.1M| 11       | 6.4K     |
| pokec (PK) [35]       | 1.6M| 31M | 37       | 20K      |
| wiki-talk (WT) [35]   | 2M  | 5M  | 4        | 100K     |
| live-journal (LJ) [35]| 4.8M| 69M | 13       | 3K       |
| twitter-2010 (TW) [35]| 41M | 1.4B| 35       | 770K     |

Efficiency of shuffle

- Theoretical throughput = memory_bandwidth / tuple_size
- The performance is close to the theoretical throughput.
Efficiency of shuffle

- The throughput of our shuffle is much higher than existing shuffling solutions.

End to end performance

- Compare the performance of graph frameworks with different shuffling solutions.
- Speedup of PageRank is up to $100\times$ of [1], and $6\times$ of Polling.

![Graph showing speedup comparison]

Resource utilization

- BRAMs are well utilized for vertex caching.
- PR and SpMV consume DSPs.

<table>
<thead>
<tr>
<th>Algo.</th>
<th>Freq.(Mhz)</th>
<th>BRAM</th>
<th>Logic</th>
<th>DSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR</td>
<td>171.5</td>
<td>2,329</td>
<td>125,811 (54%)</td>
<td>8 (3%)</td>
</tr>
<tr>
<td>SpMV</td>
<td>165.4</td>
<td>2,394</td>
<td>125,854 (54%)</td>
<td>8 (3%)</td>
</tr>
<tr>
<td>BFS</td>
<td>172.6</td>
<td>2,030</td>
<td>127,085 (54%)</td>
<td>0 (0%)</td>
</tr>
<tr>
<td>SSSP</td>
<td>170.6</td>
<td>1,926</td>
<td>123,457 (53%)</td>
<td>0 (0%)</td>
</tr>
</tbody>
</table>
Compare with RTL-based works

- Our approach achieves throughput that is comparable or even better than RTL-based graph processing designs.

**TABLE V: Comparison with state-of-the-art implementations.**

<table>
<thead>
<tr>
<th>Algo.</th>
<th>Graph</th>
<th>Others</th>
<th>Throughput</th>
<th>Ours</th>
<th>Impro.</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR</td>
<td>LJ</td>
<td>ForeGraph [14]</td>
<td>1193</td>
<td>1110</td>
<td>0.93×</td>
</tr>
<tr>
<td></td>
<td>WT</td>
<td>[11]</td>
<td>279</td>
<td>584</td>
<td>2.09×</td>
</tr>
<tr>
<td>SpMV</td>
<td>WT</td>
<td>GraphOps [38]</td>
<td>190</td>
<td>551</td>
<td>2.90×</td>
</tr>
<tr>
<td>SSSP</td>
<td>WT</td>
<td>[13]</td>
<td>657</td>
<td>618</td>
<td>0.94×</td>
</tr>
<tr>
<td></td>
<td>LJ</td>
<td>[13]</td>
<td>872</td>
<td>1129</td>
<td>1.29×</td>
</tr>
</tbody>
</table>

Conclusion

• Data shuffling on OpenCL-based FPGAs is challenging due to the run-time data dependency.

• We propose an efficient OpenCL-based data shuffling method.

• The performance of graph processing framework integrated our shuffling is comparable to state-of-the-art RTL based works.
Acknowledgement

• This work is supported by a MoE AcRF Tier 1 grant (T1 251RES1824) and Tier 2 grant (MOE2017-T2-1-122) in Singapore. This work is also partly supported by the National Research Foundation, Prime Minister’s Office, Singapore under its Campus for Research Excellence and Technological Enterprise (CREATE) programme, and the SenseTime Young Scholars Research Fund.

• We also thank Intel for hardware accesses and donations.
Thanks
Data shuffling on RTL-based FPGAs

- Fine-grained control logic based NoCs.
Outline

• Introduction to data shuffling
• Data shuffling on OpenCL-based FPGAs
  • Motivations
  • Design and implementation
• Graph processing framework with proposed shuffling
• Evaluation
• Conclusion
• Acknowledgement