Towards Flexible Automatic Generation of Graph Processing Gateware

Nina Engelhardt, Hayden So
The University of Hong Kong

June 7, 2017
Introducing GraVF Graph Processing Framework

- Introduced last year [HEART 2016, FPL 2016]
- Several improvements now:
  - Off-Chip Memory (HMC)
  - Cleaner Synchronization/Network Protocol
  - Better Balancing/Partitioning
GraVF Overview
Algorithm is a vertex kernel operating from the POV of a vertex

Computation is organized in supersteps

Each superstep, all active vertices execute the vertex kernel concurrently, followed by global barrier

Vertices exchange messages along (outgoing) edges

Messages sent during superstep $n$ will be received during superstep $n + 1$

In GraVF: kernels divided in two phases apply and scatter
Dark Gray: user-provided algorithm-specific module
Using off-chip HMC memory to increase capacity
Approach to using off-chip memory

- move **edge storage** off-chip
- edge storage is larger, accesses are read-only and have some locality (more the higher the arity)
- vertex storage is smaller, accesses are completely random, R/W, and hazards induce more stalls if pipeline gets longer

Module to be replaced: adjacency list retrieval
HMC Memory Properties

- Hybrid memory cube: 3D stacked DRAM with memory controller on bottom logic layer

- Packet-based protocol: independent command and response channel, out-of-order response

- User assigns tag to request, same tag identifies response

- No flow control on response - only emit requests you have capacity to handle
New Module Overview

Legend
- input/output
- stage 1
- stage 2
- stage 3
Increasing the flexibility of the network protocol
Original Network Design

Floating Barrier:

Problems:

- $n$ input FIFOs per PE, i.e. $n^2$ total
- Messages need to be delivered in-order
Adaptations to Synchronization Protocol and Network

Permit reordering of messages:

- Order of messages within superstep not relevant, just need to ensure superstep separation

- New field in message header indicating which superstep message belongs to

- Sender counts messages sent to each PE, includes count in barrier message

- At receiving end, PE counts messages received and compares with count, waits for stragglers if necessary

How many bits does the new field need? 2 (up to 3 message sets at once)
Adaptations to Synchronization Protocol and Network

Required network guarantees:

- messages must be delivered (no packet drop)
- at least 3 channels to separate messages from different supersteps
- separate flow control per channel (message from later superstep may not block message from previous superstep)
Detecting termination in a distributed manner

Termination condition: no message is sent by any vertex during superstep

- Originally: PE sending 2 barriers in a row indicates local inactivity, AND signals from all PEs for global signal

- New: add bitfield to barrier messages indicating sending PE’s inactivity. Since PE already counts outgoing messages, it can set bitfield if total is zero

- Receiving side checks bitfield of barriers from all PEs. If all are set, terminate.
Balancing/Partitioning the Graph
Goal: same workload on all PEs

- motivated by switch to graph500 graph generator for better comparisons
- previously just filled PEs with vertices sequentially as they are read in
- 2 issues: completely empty PEs at end, all high-arity vertices in PE 0
- solution: assign vertices in round-robin to PEs instead
- additionally, size memories to fit data to maximize BRAM usage
Results
Evaluation setup

Benchmarks: Breadth First Search and PageRank

Input graphs:

- using graph500 kronecker generator (scale-free graphs)
- two variables: *scale* and *edgefactor*
- Size: graph generated with approx $2^{\text{scale}}$ vertices
- Edgefactor (average arity): fixed at default of 16

Evaluation platform:

- Micron AC-510 FPGA board
- Contains Xilinx Kintex Ultrascale KU060 FPGA and 4GB Micron HMC 1.1 chip
- 9 memory ports $\rightarrow$ 9 PEs
- increase graph size until resource limit reached
Results

- Original: before modifications
- BRAM: with network and partitioning updates, no HMC
- HMC: with all improvements including HMC

<table>
<thead>
<tr>
<th></th>
<th>Design</th>
<th>Scale</th>
<th>Runtime</th>
<th>MTEPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFS</td>
<td>Original</td>
<td>13</td>
<td>31 ms</td>
<td>229</td>
</tr>
<tr>
<td></td>
<td>BRAM</td>
<td>14</td>
<td>13.5 ms</td>
<td>1105</td>
</tr>
<tr>
<td></td>
<td>HMC</td>
<td>17</td>
<td>131 ms</td>
<td>1001</td>
</tr>
<tr>
<td>PR</td>
<td>Original</td>
<td>13</td>
<td>1155 ms</td>
<td>187</td>
</tr>
<tr>
<td></td>
<td>BRAM</td>
<td>14</td>
<td>561 ms</td>
<td>801</td>
</tr>
<tr>
<td></td>
<td>HMC</td>
<td>16</td>
<td>1073 ms</td>
<td>1789</td>
</tr>
</tbody>
</table>

For truly random access, HMC documentation projects performance bound of 1.6 GTEPS
Comparison with related works

Performance data extracted from papers in the literature, normalized per FPGA board:

<table>
<thead>
<tr>
<th>Design</th>
<th>Input Size</th>
<th>MTEPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphStep(*)</td>
<td>64K Edges</td>
<td>168-450</td>
</tr>
<tr>
<td>GraphGen</td>
<td>341K Edges</td>
<td>459</td>
</tr>
<tr>
<td>GraphSoC</td>
<td>126K Edges</td>
<td>approx. 100</td>
</tr>
<tr>
<td>ForeGraph(*)</td>
<td>490M Edges</td>
<td>464</td>
</tr>
<tr>
<td>GraVF</td>
<td>3.7M Edges</td>
<td>1001</td>
</tr>
</tbody>
</table>

(*) These works are simulation-only.
HMC can provide sufficient bandwidth to keep performance similar to BRAM version.

Larger graph size provides longer adjacency lists for high-arity nodes, gives boost to HMC performance.

Improved synchronization and termination mechanism now fully distributed, network layout independent.

Future improvements:

- Connecting multiple FPGAs together
- Moving vertex data off-chip
Q & A
How messages from 3 supersteps come to coexist

1. A and B run superstep $n - 1$, which sends messages for superstep $n$

2. Both A and B finish superstep $n - 1$ and send barriers indicating last message for superstep $n$

3. Messages for A are held up in the network, messages for B get delivered and cause B to run superstep $n$, which sends messages for $n + 1$

4. B receives barriers indicating last message for $n$ from both A and B, and terminates superstep $n$

5. B starts processing messages for the superstep $n + 1$ as it knows (due to the barrier for the superstep $n$) that there are no more messages for B in superstep $n$

6. B, during superstep $n + 1$, sends messages for superstep $n + 2$ to A