Performance-driven System Generation for Distributed Vertex-Centric Graph Processing on Multi-FPGA Systems

Nina Engelhardt, C.-H. Dominic Hung, Hayden K.-H. So

Department of Electrical and Electronic Engineering
The University of Hong Kong

Abstract

We present a multi-FPGA graph processing framework and an accompanying performance model. Our framework emphasizes programmability, requiring minimal user input beyond providing the application kernel and the dataset. The framework predicts the performance of the system based on the algorithm characteristics and problem size and automatically selects the optimal FPGA configuration. We implement our system on an experimental 4-FPGA platform and compare the results to the predicted performance.

Programming Model

We adopt a Pregel-like vertex-centric programming model. Users specify a *vertex kernel* that is executed in parallel for each vertex in the graph. Vertices can exchange messages only along edges, and can access only private local data. To enable optimization of data storage in our system, we ask users to split their implementation in three parts, based on availability of data:

- *Gather* is called for each message that a vertex receives. It updates the vertex' local data based on the received information.
- *Apply* is called on each vertex at the end of a superstep. It may read and modify the vertex' local data, and optionally produce an update to be broadcast to the vertex' neighbors.
- *Scatter* is called on each outgoing edge of the vertex if an update is issued. It can read the edge's data to finalize the message to be sent to each neighbor.

Example Apply Code: Breadth First Search

```
self.comb += [
  visited.eq(self.state_in.parent != 0),
  If(visited,
    self.state_out.parent.eq(self.state_in.parent),
    self.state_out.active.eq(self.state_in.active)
  ).Else(
    self.state_out.parent.eq(self.sender_in),
    self.state_out.active.eq(1)
  ),
  self.state_valid.eq(self.valid_in),
  self.nodeid_out.eq(self.nodeid_in),
  self.ready.eq(self.state_ack)
]
```

Roofline-Style Performance Model

There are four limiting factors on performance considered in this model. Whichever limit is most constraining in a given situation determines overall performance.

Processing element throughput

The cumulative throughput limit L_{PE} of all PEs in the system is expressed in the following formula:

$$T_{sys} \leq L_{PE} = n_{FPGA} \times n_{PE/FPGA} \times f_{clk}/CPE_{PE} \quad (1)$$

For the maximum number of PEs that fits in an FPGA, we obtain the computational limit of the FPGA, $L_{PE_{max}}$.

Memory bandwidth

When using off-chip memory to store the adjacency lists, the FPGA throughput T_{FPGA} may also be limited by the memory bandwidth. Traversing one edge requires loading one edge of size m_{edge} from memory.

$$T_{sys} \leq L_{mem} = n_{FPGA} \times \frac{BW_{mem}}{m_{edge}} \quad (2)$$

Network interface bandwidth

A messsage will be sent for each traversed edge. In the absence of elaborate partitioning, a message is equally likely to be sent to any other PE. Each FPGA's throughput T_{FPGA} is limited by its network interface(s):

$$T_{sys} \leq L_{if} = \frac{BW_{if}}{2 \times m_{message}} \times \frac{n_{FPGA}^2}{n_{FPGA} - 1} \quad (3)$$

Total network bandwidth

The overall network bandwidth of the switch is limited. The cumulative messages sent by all the FPGA boards cannot exceed it:

$$T_{sys} \leq L_{network} = \frac{BW_{network} \times n_{FPGA}}{(n_{FPGA} - 1) \times m_{message}} \quad (4)$$

Definitions

Variables	
n_{FPGA}	number of FPGAs to use
$n_{PE/FPGA}$	number of PEs per FPGA
Dependent Variables	
T_{FPGA}	throughput per FPGA (edges/s)
T_{sys}	total system throughput (edges/s)
Algorithm-dependent Parameters	
m_{vertex}	data storage per vertex (bits)
$m_{message}$	message size (bits/edge)
m_{edge}	edge size (bits)
$p_{msg/TE}$	messages per traversed edge (edges^{-1})

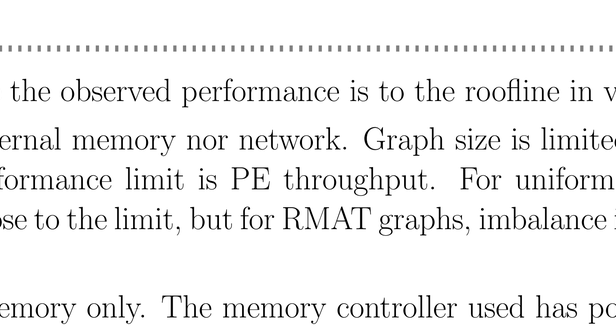
System Parameters	
CPE_{PE}	cycles per edge (edges^{-1})
f_{clk}	operating frequency (Hz)
BW_{if}	network interface bandwidth (bits/s)
BW_{if}	total network bandwidth (bits/s)
BW_{mem}	memory interface bandwidth (bits/s)
M_{board}	memory capacity per FPGA (bits)
Dataset-dependent Parameters	
$\|V\|$	number of vertices in the input graph
$\|E\|$	number of edges in the input graph

Evaluation

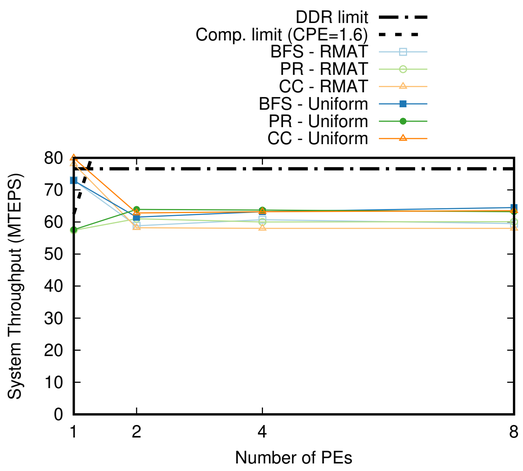
We check how close the observed performance is to the roofline in various conditions:

- Using neither external memory nor network. Graph size is limited by internal memory, but the only performance limit is PE throughput. For uniform graphs, the observed performance is close to the limit, but for RMAT graphs, imbalance in workload introduces inefficiency.
- Using external memory only. The memory controller used has poor bandwidth, so predicted performance is limited by memory bandwidth for $N_{PE} > 1$. Observed performance does not quite reach the limit, but is stable as expected.
- Using network and multiple FPGAs. Due to the increasing fraction of messages needing to be sent off-chip when adding more boards, the expected performance gain is limited by the network interface bandwidth. For RMAT graphs, the imbalance overhead grows more quickly than the system throughput, leading to reduced performance.

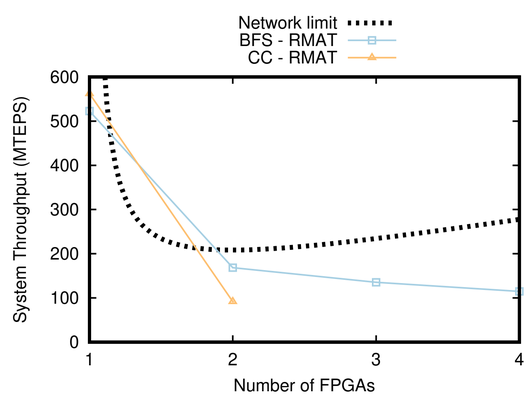
Results



(a) no memory, no network



(b) memory only



(c) network only

Conclusion

- Many factors influence the peak performance of a system, and evaluating them creates an additional burden on application developers. By baking a performance model into the framework, performance predictions can be presented and the optimal configuration automatically chosen.
- The model cannot account for all of the factors influencing performance, most notably those that depend on features of the input graph, but it nevertheless arrives at the correct conclusion as to the best configuration in the cases examined.