Vertex-centric Graph Processing on FPGA

Nina Engelhardt, Hayden Kwok-Hay So
Department of Electrical and Electronic Engineering
University of Hong Kong
Email: {nengel, hso}@eee.hku.hk

***Abstract*—Past research and implementation efforts have shown that FPGAs are efficient at processing many graph algorithms. However, they are notoriously hard to program, leading to impractically long development times even for simple applications. We propose a vertex-centric framework for graph processing on FPGAs, providing a base execution model and distributed architecture so that developers need only write very small application kernels.**

I. Programming Model

We adopt a Pregel[1]-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 two parts, based on availability of data:

- *Apply* is called for each message that a vertex receives. It may access and modify the vertex' local data, and optionally produce an update to be broadcast to the vertex' neighbors.
- For each update, *Scatter* is called on each outgoing edge of the vertex. It can access the edge's data to finalize the message to be sent to each neighbor.

As an example, Fig. 1 shows how the PageRank algorithm is implemented in this model.

```
Apply(vertex, message):
1: vertex.sum += message.weight
2: vertex.nrecvd += 1
3: if vertex.nrecvd == vertex.indegree and superstep < 30 then
4:     update(sender=vertex.id,
       weight=0.15 / num_vertices + 0.85 * vertex.sum)
5:     vertex.sum = 0
6:     vertex.nrecvd = 0
7: end if
8: return vertex
Scatter(update, edge):
1: message(sender=update.sender, destination=edge.target,
   weight=update.weight/sender.outdegree)
```

Fig. 1. PageRank Vertex Kernel

II. Architecture

The framework implementation consists of a network of identical processing elements (PEs), as shown in Fig. 2. The architecture of the network and processing elements is fixed, with only the user-provided apply and scatter kernels (shown shaded) being switched out for each algorithm, and storage resized according to the user-defined data types. The apply-scatter split, combined with a floating barrier as presented in [2], allows us to relocate storage to the middle of the pipeline where data is most condensed, reducing the storage requirements to guarantee deadlock-free execution from $|V|^2$ to $2|V|$. Simulations on a graph with 2k vertices show good scaling behavior of our system up to 64 PEs for two benchmarks (cf. Fig 3).

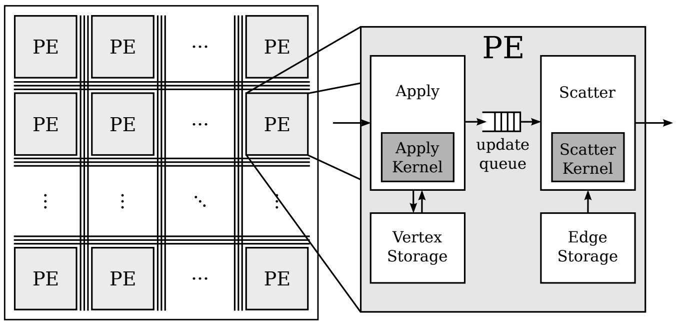
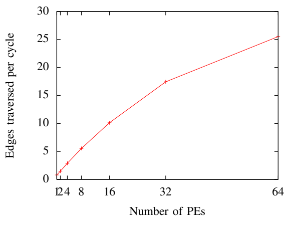
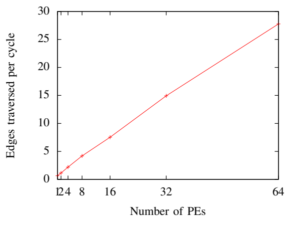


Fig. 2. Processing Element Architecture



(a) Breadth First Search



(b) PageRank

Fig. 3. Scaling behaviour

III. Conclusion

An FPGA-based vertex-centric framework for graph processing can improve development speed to the point where even casual users may take advantage of the platform's massive parallelism and energy efficiency.

References

[1] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*. ACM, 2010, pp. 135–146.
[2] Q. Wang, W. Jiang, Y. Xia, and V. Prasanna, "A message-passing multi-softcore architecture on FPGA for breadth-first search," in *Field-Programmable Technology (FPT), 2010 International Conference on*, Dec 2010, pp. 70–77.