MVAPICH

MPI, PGAS and Hybrid MPI+PGAS Library

A Collective Streaming Interface for Scale-Out of Dataflow-centric Acceleration

Nicholas Contini

{contini.26, queiser.5, ramesh.113, subramoni.1}@osu.edu

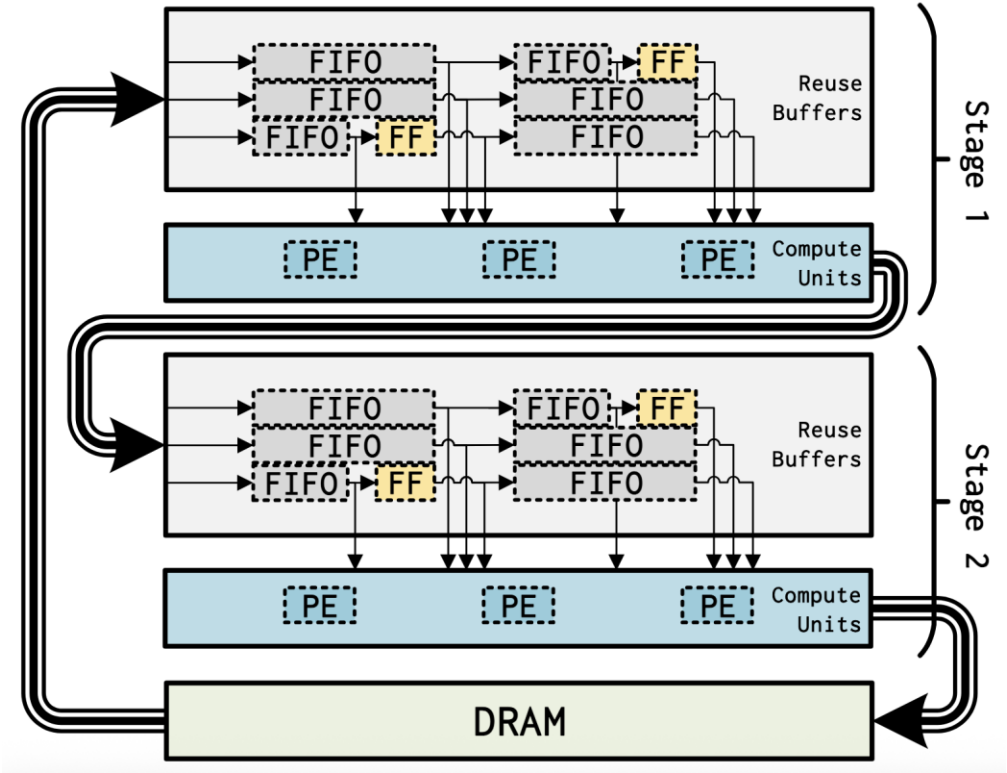
panda@cse.ohio-state.edu

Department of Computer Science and Engineering
The Ohio State University

Motivation

- The slowing of Moore's Law and end of Dennard Scaling has prompted exploration outside of generic processors
- Dataflow Architectures
  - Deviates from Von Neumann architecture
  - Instead of data bouncing between compute units (CUs) and the memory hierarchy, data is explicitly sent from one CU to another
  - Less time spent idling while fetching data
  - Fine-grained parallelism
  - Generally more energy/power efficient due to relative "slimness" of the architecture
  - Can achieve better performance for applications with irregular memory access

Example Dataflow Architecture



Chi, Yuze, et al. "SODA: Stencil with optimized dataflow architecture." *Proceedings of the International Conference on Computer-Aided Design*. 2018.

Motivation

- Research Gap: what about scale-out?
- There are very few works focusing on enabling high-performance communication between dataflow architectures
- There are some works leveraging direct/indirect networks directly attached to FPGAs
  - Issues with scalability
  - Issues with integrating with existing HPC systems
  - Issues with hardware resource usage
- What about MPI?
  - The compute-then-communicate flow of MPI applications is the anti-thesis of dataflow

Problem Statement

- Can we provide a communication interface that enables scale-out of dataflow architectures?
- Can we make it easy to use?
- Can we integrate it into existing HPC applications?

Solution

- Offload communication to host processor
  - Leverages pre-existing scale-out solutions
  - Avoids implementing networking stack on accelerator
- Provide APIs that are dataflow compatible
  - Want benefits of MPI with different semantics
  - Dataflow applications tend to transfer through streams
  - Target High Level Synthesis (HLS)

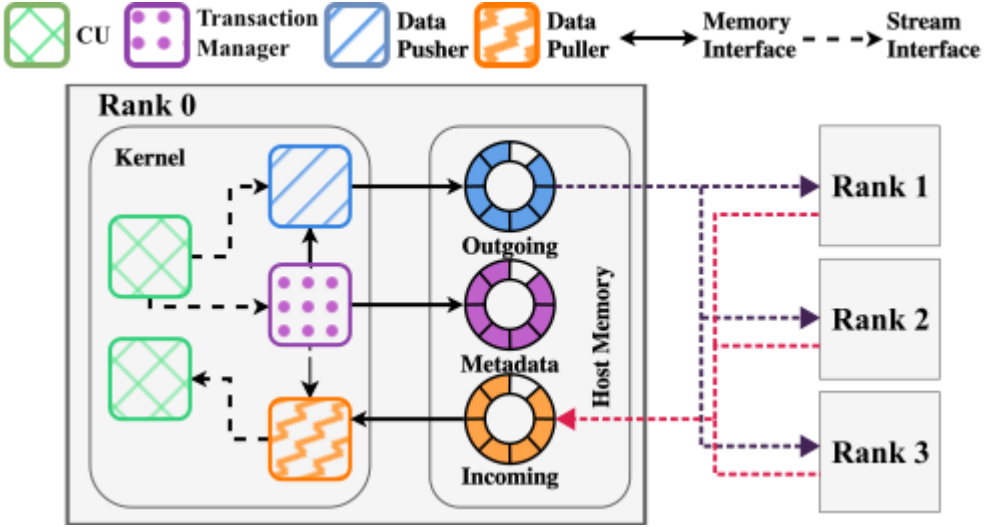
Design: APIs

- Inspired by hls::stream from Vitis HLS and persistent MPI
- Beginning of communication is declared by start calls
- CollectiveStream writes begin transfer
- CollectiveStream reads block until data is available

```
template <typename T>
class CollectiveStream{
  CollectiveStream(void *host_mem);
  void startSend<typename T>(int count, int
    rank, int tag, MPI_Comm comm);
  void startRecv<typename T>(int count, int
    rank, int tag, MPI_Comm comm);
  void startBcast<typename T>(int count,
    int root, MPI_Comm comm);
  void write(T val);
  T read();
  void progress();
}
```

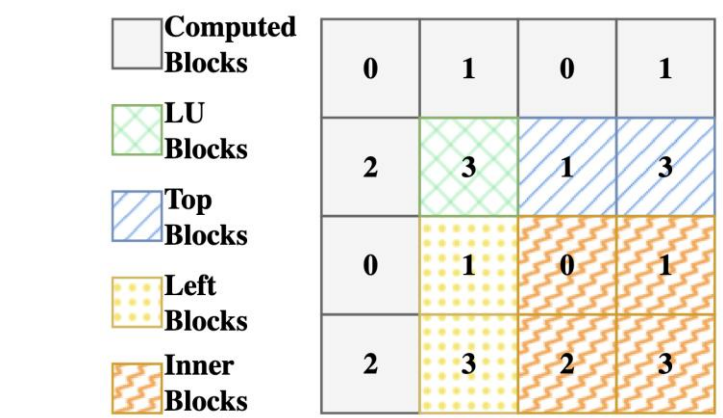
Design: Implementation

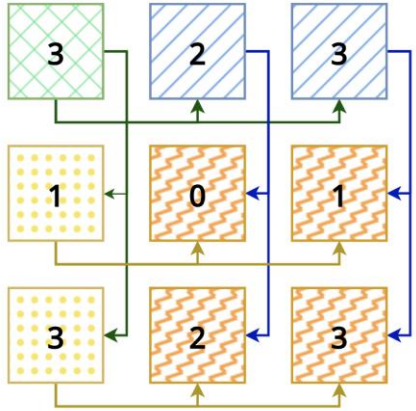
- Based on our API, it is clear we need fine-grained data transfers
- CUs write data to streams (usually implemented as FIFOs)
- The data pusher module internal to the CollectiveStream writes data to ring buffers in host memory
- A host thread awaits communication data and calls an appropriate MPI call
- The receiving CollectiveStream reads from host ring buffer into on-chip memory
- Receiving CUs can read from CollectiveStream

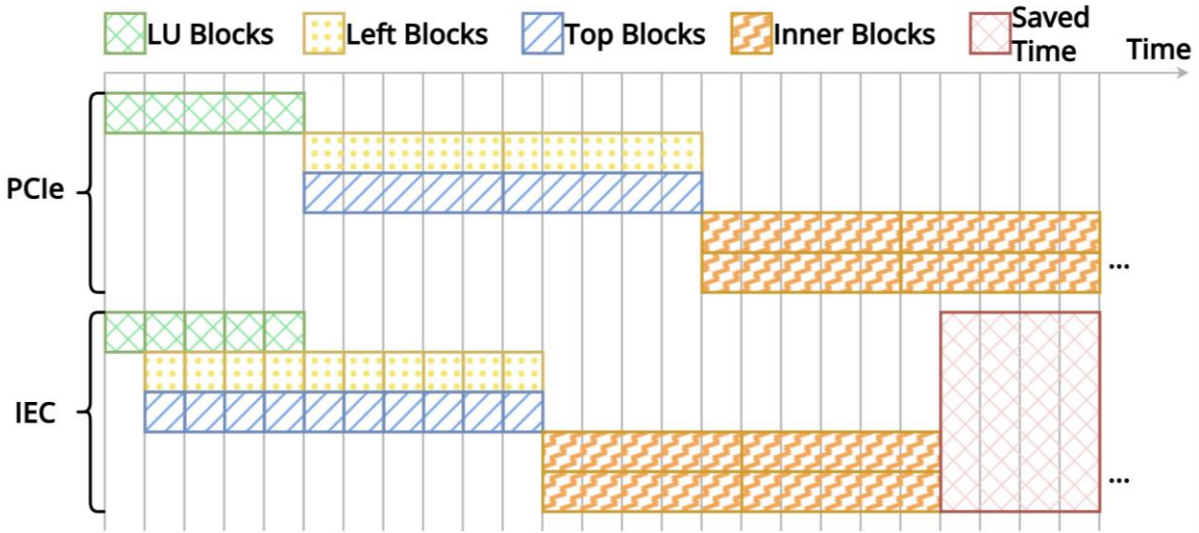


Application

- Consider LINPACK
- 4 core computations
  - LU Decomposition of top left block
  - Update top and left blocks
  - Update inner blocks
- Between each of these calculations broadcast is needed
- Without fine-grained transfers, communication doesn't occur until computation is complete

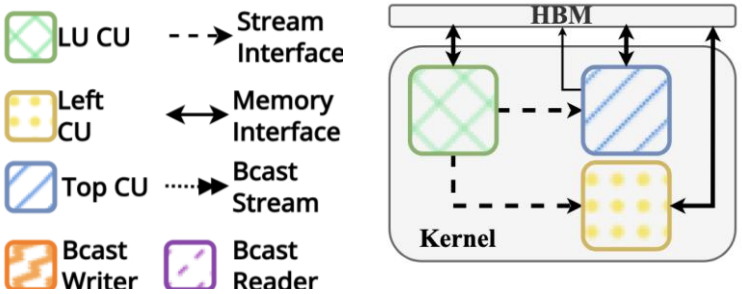


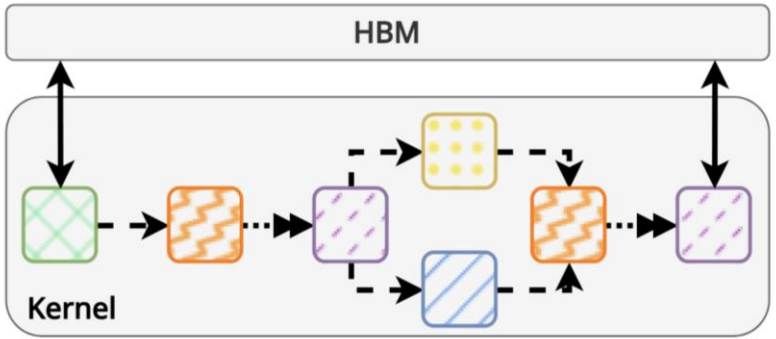




Application

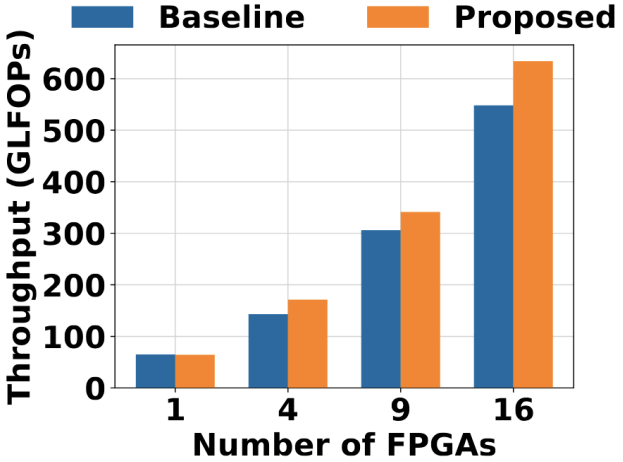
- We can avoid halting computation by following a dataflow-centric approach
- Without our contributions, we can easily implement the algorithm that allows data to flow immediately from one computation to the next within a single accelerator (top)
- However scaling out is not trivial...
- By integrating CollectiveStreams into the solution, the design can easily broadcast outputs to multiple accelerators with minimal changes to the design

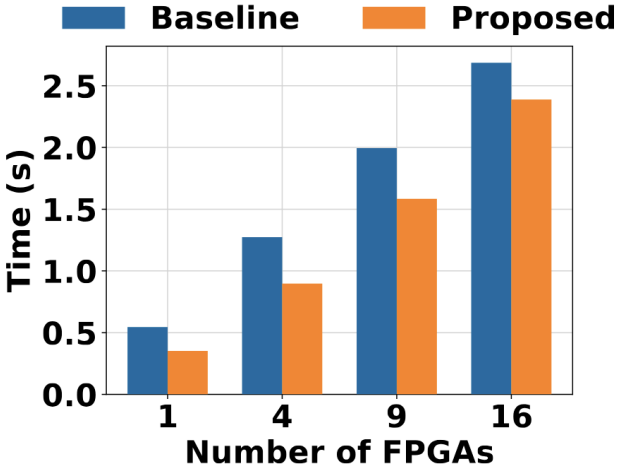




Results

- Using this design improved computational throughput by up to 18%, 14%, and 11% for 4, 9, and 16 accelerators respectively
- If we measure only the optimized portion of the application, we see a 5%, 29%, 19%, and 11% improvement in execution time for the 1, 4, 9, and 16 accelerators
- Improvement comes overlap of communication and computation, overlap of different computations, and reduced kernel launches

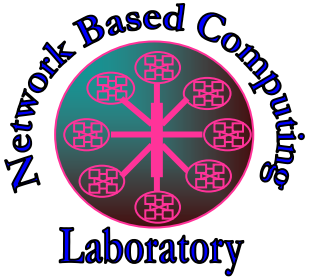




Future Work

- Equivalent GPU API
  - Allow GPUs to "stream" data to other GPUs or accelerators implementing the same API
- Multi-level networks
  - Use directly attached networks in conjunction with host network
- Exploration with other applications
  - Already publishing results on LINPACK and stencil microapplication, but graph computation or AI applications could have a use case

THANK YOU!



Network-Based Computing Laboratory
http://nowlab.cse.ohio-state.edu/

The High-Performance MPI/PGAS Project
http://mvapich.cse.ohio-state.edu/

The High-Performance Big Data Project
http://hibd.cse.ohio-state.edu/

The High-Performance Deep Learning Project
http://hidl.cse.ohio-state.edu/