Collective Communication on FPGA Clusters with Static Scheduling

Jiayi Sheng  Qingqing Xiong  Chen Yang  Martin C. Herbordt
Department of Electrical and Computer Engineering
Boston University, Boston, MA

Abstract—FPGA-centric clouds and clusters provide direct and programmable interconnects with obvious benefits for communication latency and bandwidth. One rarely studied aspect of DPI is that they facilitate application-aware routing: if communication patterns are static and known a priori, as is usually the case, then judicious routing can reduce congestion, latency, and the hardware required. In this study we explore applying the method of offline/static routing to collective operations, in particular, multicast and reduction. An entirely new communication infrastructure is proposed and implemented, including switch design and routing algorithm. A substantial improvement in performance is obtained, especially for multicast. We believe that this is one of the few general offline/static routing solutions for real HPC clusters, and FPGA-centric clusters in particular.

Keywords—Communication Collectives; Cluster Communication; Application-Aware routing; FPGA clusters;

I. INTRODUCTION

FPGA-centric clouds and clusters [1], [2] provide direct and programmable interconnects (DPI) among the FPGA accelerators with no intervention by host CPU or NIC; the benefits for communication latency and bandwidth are obvious. One rarely studied aspect of DPI, however, is that it facilitates application-aware routing: If communication patterns are static and known a priori, then routes can be determined statically, or offline, before run time. With judicious routing, congestion can be avoided and latency reduced. As has long been known [3], most High Performance Computing (HPC) applications meet these conditions; thus a priori knowledge of communication patterns affords potential performance benefits. These benefits, however, are as yet mostly untapped. In this study we explore applying the method of offline routing to collective operations, in particular, multicast and reduction. Collectives appear in wide range of applications and often throttle performance; this trend is likely to worsen as the gap between communication and compute capability widens.

There is a vast body of work in optimizing collective operations; in this very brief survey we restrict ourselves to studies based on hardware support and/or offline routing. The IBM Blue Gene series has hardware support for collectives, including specialized atomic instructions [4], but we are not aware of support based on communication pattern. This also appears to be the case with a number of programmable NICs (e.g., [5]). With FPGA clusters, collectives have been supported with softcores [6] and specialized hardware [7]. Our work differs from that of the latter in that we use a different switch architecture; also, while their design supports offline routing, we are not aware that they have used that capability. Anton2 is a special-purpose ASIC machine for Molecular Dynamic (MD) simulation; its network structures are optimized for the collective data movement specific to MD [8]. In this work we implement a general approach. Some recent work in statically scheduled routing on FPGA NoCs includes [9]. See also [10] and [11] for surveys of practical and theoretical work, respectively.

Our contributions include the following:

1) We propose and implement an entirely new communication infrastructure to support offline routing. This includes a new switch design that supports both online and offline routing (simultaneously), and a new offline routing algorithm for collectives.

2) We find that, compared with a state-of-the-art online switch architecture, our offline routing switch architecture saves FPGA logic resources while only requiring slightly more on-chip memory.

3) We find that a priori knowledge of communication can also be used to reduce buffer sizes and enable higher bandwidth utilization.

4) We find that, when compared with a state-of-the-art online routing algorithm, our new offline routing algorithm reduces latency of multicast by 15% and of reduction by 4%.

The overall significance of this work is that it provides a general static/offline routing solution for real HPC clusters, and FPGA-centric clusters in particular.

II. NETWORK ARCHITECTURE

A. Preliminaries

We have designed and implemented an entire solution for offline routing on an FPGA cluster. We assume that collective communication patterns have been extracted from the application (as is done, e.g., in [8]). We then use our offline routing algorithm to build an optimized tree topology for each collective operation. These are fed into scripts to generate the routing, multicast, and reduction tables. Finally, the routing data are downloaded into the appropriate tables within the switches.

Our target architecture is an FPGA-centric cluster such as Catapult [1] or the Novo-G# [2]. These are characterized by having direct connections among FPGAs through the Multi-Gigabit Transceivers (MGTs). Our target network is a 3D

This work was supported in part by the NSF through awards #CNS-1405695 and #CCF-1618303/7960, and through a grant from Microsoft Research. Email: (jysheng|qx|cyang90|herbordt)@bu.edu
torus as in the Novo-G#. Please see [12], [13] for low-level details and case studies. In the rest of this section, we describe the routing mechanism and the switch design.

B. Table-based Routing

Table-based routing can be implemented in two ways: source routing and node-table routing [14]. Since source routing requires packets to carry table indexes, which consumes extra bandwidth, we instead use node-table routing. There the resident routing table preserves a table entry for each incoming packet (see Figure 1).

In this example, node A dispatches a multicast packet that carries three fields: packet type, table index, and payload. The router routes the packet to either a unicast, multicast, or reduction table based on the packet type. In the corresponding table, multicast in this example, the router looks up the table entry based on the index field in the packet. The multicast table entry has slots for all of the six possible fanouts.

Figure 2 shows the routing table formats: (a) unicast (b) multicast (c) reduction

Figure 2. Routing table formats: (a) unicast (b) multicast (c) reduction

C. Switch Architecture

Our switch architecture is based on the classical four-stage pipelined Virtual-Channel (VC) switch [15], which has become a de facto standard. By adding support for multicast and reduction, the four-stage pipeline is extended to seven-stage pipeline. Its architecture is illustrated in Figure 3.

Figure 3. Switch architecture: (a) The switch is connected by seven input and seven output handlers. (b) The input handler has four stages: input buffer consumption, routing table lookup, multicast table lookup, and virtual channel allocation. (c) The output handler has three stages: switch allocation, reduction table lookup, and reduction table write-back.

Figure 4 shows the original and modified pipelines. The original switch has four stages: routing computation (RC), virtual channel allocation (VA), switch allocation (SA), switch traversal (ST). The first difference is that we divide the RC stage into three stages: input buffer consumption (IC), routing table lookup (RL), and multicast table lookup (ML). When a packet is injected into switch input buffer, during the IC stage, we spend one cycle to fetch the flit from the buffer. We examine its header, to obtain its routing table index; this is used during RL to find the routing table entry. At that point the switch knows whether the packet
is multicast or not. If so, then during ML we look up its multicast table entry based on the index from the routing table entry. The VA stage allocate VCs to all the multicast children that are generated during ML. If VA fails, the switch generates back pressure to stall the pipeline. In the SA stage, there might be multiple packets (up to 7) contending for the same output port. The packet with the highest priority, which is determined during the RL stage, wins the arbitration. Our current priority scheme is farthest-first.

Another difference between our switch and classical switch is that we divide ST stage into two stages: reduction table lookup (ReL) and reduction table write-back (ReW). If the packet is not a reduction packet, it still traverses the last two stages (bypass is certainly an option). If the packet is a reduction packet, it is routed to the reduction unit. We allocate one entry in the reduction table for each reduction operation. During ReL, the reduction packet checks for its corresponding entry and whether the expected number of downstream packets have arrived. If not, then the reduction operation is executed and the reduction table entry updated. If all the expected downstream packets have arrived, the reduction unit dispatches a new packet and injects it into the upstream link.

III. OFFLINE COLLECTIVE ROUTING ALGORITHM

Collective operations can be implemented either with unicast or multicast (see Figure 5). Figures 5 (a) and (c) show multicast and reduction using unicasts. All the packets are unicast but share the same source (or destination). Figure 5 (b) and (d) show multicast and reduction based on a tree topology: the communication burden is obviously drastically reduced.

Tree-based collective routing algorithms have been much studied recently [16]–[19]. Recursive Partition Multicast (RPM) in [18] appears to be the leading such algorithm. Developed for Networks-on-Chip, RPM only works for 2D meshes. Here, we extend RPM to deal with both multicast and reduction on a 3D torus; we also enhance the original algorithm for 2D.

RPM [18] provides a solution to generate the multicast pattern recursively on a 2D-mesh network. Their goal is to build a multicast tree that maximally reuses network links. We found, however, that there are two place in the algorithm that can be improved (see Figure 6).

In (a) and (b), dst 0 is northeast of src and dst 1 is southeast. The best multicast routing decision is to first send the packet east and then let this node distribute packets to dst 0 and dst 1 (as in (b)). RPM, however, sets North to always have the highest priority. The packet first goes north and then east to dst 0; this does not reuse the east link. In Figure (c) and (d), the two routing decisions have the same reusability of links. However, the north and south links are less congested than the west and east links, indicating that the routing decision made in (d) is preferred to the one in (c). RPM makes these suboptimal decisions because it does not account for link congestion and because of the policy giving priority to the north link.

Our new offline collective algorithm addresses these two drawbacks in RPM, and also extends it to tori and 3D. We call our algorithm offline collective routing (OCR). Pseudocode for OCR is shown in Figure 7. The first step is to determine the dimensionality of the topology. If 1D, then multicast routing is immediate. The algorithm for the 2D is described below. If 3D, then the next step is to find an optimal partition. There are three options corresponding to the number of dimensions. If the space is a 3D torus, we can always partition it into three parts because every node can be viewed as the center of the network.

As illustrated in Figure 8, we partition the space along three dimensions. We then count the number of outbound links exiting the source to all the destinations for each kind of partition. In this example, the partition along the yz plane requires only one outbound link, while the partition along the
xz and xy planes require two and three links, respectively. It is apparent that the partition along yz plane is the best partition method in this example. If there is more than one partition that has the minimal number of outbound links, we then select the one that results in a smaller variance in the loads on the six outbound links. If the partitions are still tied, we use a global round robin pointer. After we find the best partition, we partition the entire space into three parts: up space, middle plane and down space and distribute the destinations into the three subspaces depending on their coordinates. For the up and down space, we call the 3D OCR algorithm recursively. For the middle plane, we call the modified 2D OCR algorithm.

The 2D OCR algorithm is similar to RPM. As shown in Figure 9, we also partition the 2D space into 8 regions depending on the source location. If the space is 2D, we can always find the 8 regions since the source node could always be the center of the torus. We call regions 0, 2, 4, 6 corner regions and regions 1, 3, 5, 7 side regions. In the 2D plane, one source has at most four fan-outs, which means we can have at most 4 partitions among the 8 regions. One corner region must merge with either one of its two adjacent side regions. The first step is to count the number of nodes in all 8 regions. The next step is to determine whether to enable the links in the north, south, west and east directions and to determine which side region each corner region should merge with.

Figure 10 illustrates, the algorithm for north link, south link, and region 0. The merge direction of each corner region depends first of all on whether there are nodes in region 1, 2, 6 and 7, and second load on the north and east links. In the next step, the plane and the destination list are partitioned into up to four parts. The 2D OCR algorithm is called recursively for the four parts until they become 1D spaces.

IV. EVALUATION

A. Experimental Setup

In this section, we compare our OCR-based offline solution with an online routing solution based on RPM. Our targeted system is an FPGA cluster composed of potentially hundreds of Gidel ProceV boards (currently 128 [2]). Each board contains an Altera Stratix V 5GSMD8 chip and supports 6 links that use the FPGA’s Multigigabit Transceivers (MGT). Each MGT link has a bandwidth of 40Gbps and latency of 175 ns [13]. All the boards run at 156.25 MHz, the same as the MGTs. We have multiple switch designs, including one that supports just the online RPM routing logic and another with the OCR-based design described previously. We are currently targeting 4×4×4 and 8×8×8 clusters. Designs are coded in Verilog and synthesized with Quartus II 14.1.

Currently all paths in the switches are 256 bits to match the parallel I/O of the MGTs (256:1 ratio of frequencies). Modest tuning (in progress) will allow us to double the operating frequency of the switches and so halve the path widths and therefore the resources used. Smaller designs are also possible, but result in the MGTs being used suboptimally.

We use one-sided communication over MGT links rather than handshake communication. One reason is that the in-flight latency is already much longer than the latency in the switch (175ns versus about 45ns). In order to avoid packet drop, we must ensure that the buffers are sufficiently big. For
online routing, the input buffer must always be bigger than the worst-case requirement. For offline routing, however, we can select the buffer size that satisfies the requirements for a specific application. Buffer size is also reduced because the routing algorithm balances the link load.

Designs have been tested and validated on a 4 node subsystem; performance results below are from ModelSim simulations. We have run experiments using three synthetic communication pattern: random, bit-rotation, and nearest neighbor. More details are given below.

B. Hardware Cost

<table>
<thead>
<tr>
<th>Pattern</th>
<th>operation</th>
<th>table size (bits)</th>
<th>percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>All-to-all</td>
<td>multicast</td>
<td>6968</td>
<td>0.013%</td>
</tr>
<tr>
<td>All-to-all</td>
<td>reduction</td>
<td>61.7K</td>
<td>1.17%</td>
</tr>
<tr>
<td>Bit Rotation</td>
<td>multicast</td>
<td>1122</td>
<td>0.002%</td>
</tr>
<tr>
<td>Bit Rotation</td>
<td>reduction</td>
<td>10K</td>
<td>0.19%</td>
</tr>
<tr>
<td>Nearest Neighbor</td>
<td>multicast</td>
<td>2928</td>
<td>0.005%</td>
</tr>
<tr>
<td>Nearest Neighbor</td>
<td>reduction</td>
<td>25.5K</td>
<td>0.485%</td>
</tr>
</tbody>
</table>

Basic resource utilization is shown in Table I. We find that the offline router is able to save 5% of chip area by eliminating the routing computation logic. We measure the table sizes required by OCR algorithm for three typical collective patterns (see Table II). For all three cases, the routing tables consume at most 1.17% of the total on-chip memory. We observe that reduction requires much larger tables than multicast. This is because of that we need separate table entries for the different packets in the reduction to buffer temporary results.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>operation</th>
<th>table size (bits)</th>
<th>percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>All-to-all</td>
<td>multicast</td>
<td>1532</td>
<td>1132</td>
</tr>
<tr>
<td>All-to-all</td>
<td>reduction</td>
<td>367</td>
<td>288</td>
</tr>
<tr>
<td>Bit Rotation</td>
<td>multicast</td>
<td>91</td>
<td>57</td>
</tr>
<tr>
<td>Bit Rotation</td>
<td>reduction</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Nearest neighbor</td>
<td>multicast</td>
<td>277</td>
<td>157</td>
</tr>
<tr>
<td>Nearest neighbor</td>
<td>reduction</td>
<td>14</td>
<td>9</td>
</tr>
</tbody>
</table>

Table III compares the empirically determined worst-case buffer sizes of online and offline routing for the three patterns. The results of Table III show that offline routing can be expected to save around 20% to 30% input buffer size. Two other factors increase the advantage of the offline design. First, in a production design, the online buffers would need to be somewhat larger to deal with worst case scenarios. And second, while the offline buffer can be sized per application, the online account for the worst case across all applications. An alternative for the online design is to use backpressure, but this substantially increases latency.

Another advantage of offline routing is the packet size. In online broadcast, the packet header has to contain entire destination list, while the offline packet header only needs to carry a table index in header. For a network has N nodes, the online routing header needs to have N bits, while the offline routing header needs only log N bits.

C. Latency

We measure latency with respect to two types of loads, batch and continuous. For batch, each node transmits a fixed number of collective packets; latency is the time from when first packet is sent until the last packet is received. For continuous, each node generates collective with a certain injection rate; latency is the average packet latency.

![Figure 11](image)

Figure 11. Experiments with batch load for three typical benchmarks (all-to-all, nearest neighbor, and bit rotation) for two network sizes. Results shown are the speedups of the offline OCR versus online RPM designs.

Figure 11 shows the results for batched experiments (speedup of offline versus online). We apply three typical benchmarks (all-to-all, nearest neighbor, and bit rotation) for two network size: 4 × 4 × 4 and 8 × 8 × 8. For nearest neighbor in the 4 × 4 × 4 network, each source node communicates with nearest 26 neighbors (3^3 – 1); in the 8 × 8 × 8 network, each source node communicates with the nearest 124 neighbors (5^3 – 1). The patch size is set to 64 packets and injection rate is set to 1 packet per cycle per node. The results show that for the multicast operation, the latency of the offline routing solution is in all cases better than online routing with a geometric mean of over 15% for multicast and 4% for reduction. For the reduction operation, the improvement is limited by the fact that each node injects at most one packet per cycle but can consume more than one.

Figure 12 shows the results for the continuous multicast experiments. We add two patterns: these are similar to all-to-all, but with a subset of destinations selected at random. We
note that unloaded latency is maintained for higher loads for the offline design. Also, the offline design results in better average latency in nearly all cases. The exception is the nearest neighbor pattern. This is because that pattern is already symmetric and balanced leaving little room for improvement.

V. CONCLUSION

In this paper, we describe a complete communication infrastructure to support offline (statically scheduled) routing of collective communication on FPGA-centric clusters. We use table-based routing and a new switch design. We propose a new offline collective routing (OCR) algorithm that takes advantage of knowledge of communication patterns to load-balance network links and reduce congestion. The experiments show that this offline routing solution has significantly better performance and lower hardware cost than a state-of-the-art online routing solution. The OCR algorithm is not optimal; however, the infrastructure described supports improvements as they are developed with no change in design.

REFERENCES