PROTOCOLS FOR FULLY OFFLOADED COLLECTIVE OPERATIONS ON ACCELERATED NETWORK ADAPTERS

TIMO SCHNEIDER, TORSTEN HOEFLER, RYAN E. GRANT, BRIAN W. BARRETT, RON BRIGHTWELL
**WHY (COLLECTIVE) OFFLOAD**

- **Blocking Collectives:**
  - During communication CPU is idle
  - If one process is delayed, all have to wait (OS Noise)

See: Hoefler et. al.: “Characterizing the Influence of System Noise on Large-Scale Applications by Simulation”, SC’10
**WHY (COLLECTIVE) OFFLOAD**

- **Blocking Collectives:**
  - During communication CPU is idle
  - If one process is delayed, all have to wait
- **Solution:** Non-blocking collectives!
  - If the CPU does computations, who ensures progress?
**WHY OFFLOAD**

- Dedicate cores to communication?
- Speed: CPU <-> NIC can be up to 500ns!

See: Barrett et al.: The impact of hybrid-core processors on MPI message rate, EuroMPI’13
if (rank > 0) {
    send(NULL, 0, ..., i, ...);
    recv(NULL, 0, ..., MPI_ANY_SOURCE, ...);
}
/*The root collects and broadcasts the messages.*/
else {
    for (i = 1; i < size; ++i)
        irecv(NULL, 0, ..., ANY_SOURCE, ..., &reqs[i]);
    wait_all(size-1, reqs+1,...);
    for (i = 1; i < size; ++i)
        isend(NULL, 0, ..., i,...,&reqs[i]);
    wait_all(size-1, reqs+1, ...);
}

Open MPI Barrier for small Communicator sizes
How do collectives look like?

```c
if (rank > 0) {
    send(NULL, 0, ..., i, ...);
    recv(NULL, 0, ..., MPI_ANY_SOURCE, ...);
}
/*The root collects and broadcasts the messages.*/
else {
    for (i = 1; i < size; ++i)
        irecv(NULL, 0, ..., ANY_SOURCE, ..., &reqs[i]);
    wait_all(size-1, reqs+1,...);
    for (i = 1; i < size; ++i)
        isend(NULL, 0, ..., i,...,&reqs[i]);
    wait_all(size-1, reqs+1, ...);
}
```

Some dependencies are implicit

Some dependencies are explicit

How do I translate this to Portals 4? ConnectX? Quadrics?

Not a good programming model for collective offload!
HOW SHOULD COLLECTIVES LOOK LIKE

Data Movement + Dependencies

“Schedule” of operations (and dependencies among them) for each rank

Communication Topology

T. Schneider, T. Hoefler, R. Grant, B. Barrett, R. Brightwell
Backends for cDAG: MPI, DMAPP, Portals 4 Triggered Operations
PORTALS 4

Source:
Barrett et al.: The Portals 4.0
Network Programming Interface

Slide 9
PORTALS 4
PORTALS 4
PORTALS 4

1. Put Request

2. Data

Matching NI

Matching Portals Table

Priority List

Memory Descriptor (MD)

Counter

MD

Counter

Counter

Counter

Counter

Counter

ME

ME

ME

ME
PORTALS 4 TRIGGERED OPERATIONS

- During cDAG execution, the CPU must not be involved -> full offload

Matching Entries (MEs)

Counters

Threshold

Memory Descriptors (MDs)

Triggered Operations: i.e., Put, Set Counter, Increment Counter
TRANSLATING cDAG TO PORTALS

- $S \rightarrow S$
- $R \rightarrow S$
- $R \rightarrow R$

$\star = \text{#incoming edges}$

- $MD \rightarrow \star \rightarrow \text{Put}$
- $ME \rightarrow \star \rightarrow \text{Put}$
- $CtInc \rightarrow \star$
EAGER PROTOCOL
PRE-MATCHED RENDEZVOUS

Sender
CPU  NIC

Receiver
CPU  NIC

Data
Put(Data)
CTInc(e1)
CTInc(e2)

Synchronization

incoming deps of recv
Put(RTR)
CTInc(e3)

incoming deps of send

Data

RTR

- Put of Data triggered
- Inc. of enabled-counter for e1, e2 triggered

incoming deps resolved

Put of RTR triggered

Enabled-Counter of e3 incremented
MISSING FEATURE IN PORTALS

- Impossible to influence matching with Triggered Operations!
- We propose PtITriggeredMEAppend()
- Adds (“activates”) a pre-defined Match Entry
**RENDEZVOUS PROTOCOL**

Diagram showing the flow of messages and processes in a rendezvous protocol, including steps for sending and receiving data, with synchronization points indicated.

- **Sender-Early Case:**
  - RTS
  - RTR
  - DATA

- **Receiver-Early Case:**
  - MEApp(RTR)
  - Put(RTS)

- **Synchronization Points:**
  - incoming deps of send
  - incoming deps of recv

**Legend:**
- MEApp(RTR)
- Put(RTS)
- Send RTR
- Receive RTS
- Receive Data
OVERLAP

Broadcast, 2.9 Ghz AMD K10, Infiniband NICs

All rounds overlapped

First round is overlapped

Datasize [B]
- 8
- 1024
- 65536

Number of Processes
NON-OVERLAPPED LATENCY

Broadcast,
2.9 Ghz AMD K10,
Infiniband NICs

<table>
<thead>
<tr>
<th>Method</th>
<th>8</th>
<th>16384</th>
<th>65536</th>
</tr>
</thead>
<tbody>
<tr>
<td>cDAG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Latency [us]

Number of Processes

2x
CONCLUSIONS

- cDAG works well as a programming model for collective offload
- Translating cDAG to Portals 4 exposed “missing features” in current spec
- Good overlap and latency, even on software-emulated reference implementation
- cDAG backends for other offload engines (i.e., ConnectX) are future work
THANK YOU!

- Time for questions!
- Offline questions: timos@inf.ethz.ch
- Slides will be published, see Publication list at http://spcl.inf.ethz.ch