S. Di Girolamo, P. Jolivet, K. D. Underwood, T. Hoefler

Exploiting Offload Enabled Network Interfaces
We need an abstraction!

1980’s

Lossy Networks
Ethernet

2000’s

Lossless Networks
RDMA

2020’s

Device Programming
Offload

How to program QsNet?

How to offload in Portals 4?

How to offload in libfabric?
Communications (non-blocking)

Computations

Dependencies

L0: recv a from P1;
L1: b = compute f(buff, a);
L2: send b to P1;
L0 and CPU→ L1
L1 → L2
Performance Model

P0{
    L0: recv m1 from P1;
    L1: send m2 to P1;
}

P1{
    L0: recv m1 from P1;
    L1: send m2 to P1;
    L0 -> L1
}

Offloading Collectives

A collective operation is fully offloaded if:

1. No synchronization is required in order to start the collective operation
2. Once a collective operation is started, no further CPU intervention is required in order to progress or complete it.

Definition. A schedule is a local dependency graph describing a partial ordered set of operations.

Definition. A collective communication involving \( n \) nodes can be modeled as a set of schedules \( S = S_{\downarrow 1} , ... , S_{\downarrow n} \) where each node \( i \) participates in the collective executing its own schedule \( S_{\downarrow i} \)
Asynchronous algorithms, with their ability to tolerate memory latency, form an important class of algorithms for modern computer architectures.

Solo Collectives

- Synchronized collectives lead to the synchronization of the participating nodes
- A solo collective starts its execution as soon as one node (the initiator) starts its own schedule
Solo Collectives: Activation

- Root-Activation: the initiator is always the root of the collective
- Non-Root-Activation: the initiator can be any participating node
A Case Study: Portals 4

- Based on the one-sided communication model
- Matching/Non-Matching semantics can be adopted

A Case Study: Portals 4

Communication primitives
- Put/Get operations are natively supported by Portals 4
- One-sided + matching semantic

Atomic operations
- Operands are the data specified by the MD at the initiator and by the ME at the target
- Available operators: min, max, sum, prod, swap, and, or, …

Counters
- Associated with MDs or MEs
- Count specific events (e.g., operation completion)

Triggered operations
- Put/Get/Atomic associated with a counter
- Executed when the associated counter reaches the specified threshold
Experimental results

**Curie, a Tier-0 system**
- 5,040 nodes
- 2 eight-core Intel Sandy Bridge processors
- Full fat-tree Infiniband QDR
- **OMPI**: Open MPI 1.8.4
- **OMPI/P4**: Open MPI 1.8.4 + Portals 4 backend
- **FFLIB**: proof of concept library
- One process per computing node

More about FFLIB at http://spcl.inf.ethz.ch/Research/Parallel_Programming/FFlib/
Experimental results

Curie, a Tier-0 system
5,040 nodes
2 eight-core Intel Sandy Bridge processors
Full fat-tree Infiniband QDR
OMPI: Open MPI 1.8.4
OMPI/P4: Open MPI 1.8.4 + Portals 4 backend
FFLIB: proof of concept library
One process per computing node

More about FFLIB at http://spcl.inf.ethz.ch/Research/Parallel_Programming/FFlib/
Simulations

- **Why?** To study offloaded collectives at large scale
- **How?** Extending the LogGOPSim to simulate Portals 4 functionalities

---

**Simulations**

<table>
<thead>
<tr>
<th></th>
<th>Broadcast</th>
<th>Allreduce</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>L</td>
<td>o</td>
</tr>
<tr>
<td>P4-SW</td>
<td>5μs</td>
<td>6μs</td>
</tr>
<tr>
<td>P4-HW</td>
<td>2.7μs</td>
<td>1.2μs</td>
</tr>
</tbody>
</table>


Abstract Machine Model

Offloading Collectives

Solo Collectives

Mapping to Portals 4

Results

Co-Authors

P. Jolivet
K. D. Underwood
T. Hoefler
Backup slides
Multi-Version Scheduling

- Enables the multiple asynchronous execution of the same collective
  - It allows the pre-posting of k versions of the same schedule
  - Each version can have its own buffers
  - Each version can be activated by a different node

- Implemented as FIFO queue of schedules
  - Only one scheduled enabled at each time: $S\downarrow i$
  - When $S\downarrow i$ is activated, the next in the queue $S\downarrow i−1$ must be enabled

\[ \bigvee_{o_{i_k} \in I_i} \quad o_{i_k} \rightarrow o_{i_j} \quad \forall o_{i_j} \in I_{i-1} \]

Independent operations of schedule $S\downarrow i$

Independent operations of schedule $S\downarrow i−1$
Use Case: Multigrid

- Multilevel preconditioners are a dominant paradigm for large-scale partial differential equation simulations
  - Theoretically optimal
  - High communication and synchronization overheads

- Two-grid hierarchy
  - Only one process perform the coarsening

- Simple benchmark implementing the communication pattern
  - The introduction of solo-collective led to a 1.5x improvement in the completion time
  - A full benchmark would require a study of the convergence rate for such fully asynchronous approach
Solo Collectives

- Collective communications lead to the pseudo-synchronization of the participating nodes
  - Each node starts its own schedule at time $t\downarrow i$
  - The collective communication will terminate at a time $t\downarrow s \geq \max \downarrow i(t\downarrow i)$

- A solo collective starts its execution as soon as one node, the initiator, starts its own schedule
  - The schedule of other nodes is asynchronously activated
  - The initiator starts its schedule at time $t\downarrow init$
  - The collective communication will terminate at a time $t\downarrow a \geq t\downarrow init + \epsilon$
  - The term $\epsilon$ models the activation time: $\epsilon \leq \max(\epsilon \downarrow i)$
Solo Collectives: activation

- One active node
- No activation cost
- e.g., broadcast, scatter

Solo Collectives

- Multi-Source
  - Non-Root-Activated
  - Root-Activated

- Single-Source
  - Root-Activated
Experimental results

Curie, a Tier-0 system
5,040 nodes
2 eight-core Intel Sandy Bridge processors
Full fat-tree Infiniband QDR

OMPI: Open MPI 1.8.4
OMPI/P4: Open MPI 1.8.4 + Portals 4 backend
FFLIB: proof of concept library
One process per computing node
Experimental results

Curie, a Tier-0 system
5,040 nodes
2 eight-core Intel Sandy Bridge processors
Full fat-tree Infiniband QDR

OMPI: Open MPI 1.8.4
OMPI/P4: Open MPI 1.8.4 + Portals 4 backend
FFLIB: proof of concept library
One process per computing node
Simulations

- **Why?** To study offloaded collectives at large scale
- **How?** Extending the LogGOPSim to simulate Portals 4 functionalities

- FFLIB-HW uses $m=0.3\,\mu s$, discussed in [3] to model the incoming message processing time

---

Point-to-Point Protocols

Eager protocol
- Expected messages: priority list
- Unexpected messages: overflow list

Rendezvous protocol
- No shadow buffers are required
- Synchronization happens among OEs

```c
ptl_md_t rts;
ptl_me_t data;
PtlMEAppend(data, NONE);
PtlPut(rts);
```

Sender

```c
ptl_md_t data;
ptl_me_t rts;
PtlMEAppend(rts, ct_rts);
PtlTriggeredGet(data, ct_rts, 1);
```

Receiver
Offloading Point-To-Point Protocols

- P2P communications are building blocks of our abstract model
  - They can be implemented according with different protocols (i.e., eager, rendezvous)

*Can this protocols be fully offloaded to the OE (e.g., Portals 4-compliant)?*

**Eager**

**Expected:** the message is directly received in the user-provided buffer.

**Unexpected:** the message is received in a temporary buffer. It will copied in the user-provided one when the receive will be posted.

Portals 4 priority and overflow list can be used for a straightforward implementation of this protocol.

**Rendezvous**

Only the Ready-To-Send (RTS) control message can be unexpectedly received.
A Case Study: Portals 4

Communication primitives
- Put/Get operations are natively supported by Portals 4
- One-sided + matching semantic

Atomic operations
- Operands are the data specified by the MD at the initiator and by the ME at the target
- Available operators: min, max, sum, prod, swap, ...

Eager protocol
- Expected messages: priority list
- Unexpected messages: overflow list

Rendezvous protocol
- No shadow buffers are required
- Synchronization happens among OEs

Counters
- Associated with MDs or MEs
- Count specific events (e.g., operation completion)

Triggered operations
- Put/Get/Atomic associated with a counter
- Executed when the associated counter reaches the specified threshold

Can point-to-point protocols be fully offloaded?

Fully offloading:
No synchronization & No CPU intervention

Can point-to-point protocols be fully offloaded?