IN-NETWORK COLLECTIVE COMMUNICATION ACCELERATIONS

OVI COLLECTIVES

Sean Hefty
Intel Corporation
April 2022
OVERVIEW

Introduction:
How collectives differ
Collective operations

Software abstraction:
Identify collective membership
Setup communication groups
Invoke collective

Other thoughts:
Ensure efficient mappings

Focus on enabling the technology
Accelerations in switches, NICs, platforms, FPGAs
No discussion on effectiveness
HOW COLLECTIVES DIFFER
MULTICAST VS COLLECTIVE BROADCAST

Multicast

Limited to 1 MTU

Datagram endpoint

join multicast()
multicast send()

Does not know who receivers are

In-network support
• Packet replication
• Unreliable delivery

Datagram endpoint

recv()
join multicast()
recv()

Recv not matched with send

Uncoordinated (e.g. late join)

Datagram endpoint

...recv()...
join multicast()

Data
Multicast
Network
HOW COLLECTIVES DIFFER
MULTICAST VS COLLECTIVE BROADCAST

Collective broadcast

RDM endpoint

Message

data

Join coordinated

Knows who receivers are

In-network support
- Message replication
- Reliable delivery – ack coalescing

RDM endpoint

Join collective()

Broadcast recvfrom()

Join collective()

Broadcast recvfrom()

Join collective()

Broadcast recvfrom()

Recv matched with send

RDM – Reliable Datagram
COLLECTIVE OPERATIONS
CONCEPTUAL: “MULTICAST ATOMICS”

Definitions by example

- **data array**
  - Collectives not appearing on stage: gather, scatter, reduce, reduce-scatter

- **barrier** (global sync)
- **Same data types and operations as atomic APIs**
  - All reduce
- **Collectives not appearing on stage:**
  - Data replication
  - Computation – data format aware
  - Data coalescing and distribution

### Definitions by example

- **Peer 0**
  - 1
  - 5
  - 9
- **Peer 1**
  - 1
  - 5
  - 9
- **Peer 2**
  - 1
  - 5
  - 9

### barrier (global sync)

- **Peer 0**
  - 1
  - 5
  - 9
- **Peer 1**
  - 2
  - 6
  - 10
- **Peer 2**
  - 3
  - 7
  - 11

### Same data types and operations as atomic APIs

- **Peer 0**
  - 1
  - 5
  - 9
- **Peer 1**
  - 1
  - 5
  - 9
- **Peer 2**
  - 1
  - 5
  - 9

- **Peer 0**
  - 2
  - 6
  - 10
- **Peer 1**
  - 2
  - 6
  - 10
- **Peer 2**
  - 2
  - 6
  - 10

- **Peer 0**
  - 3
  - 7
  - 11
- **Peer 1**
  - 3
  - 7
  - 11
- **Peer 2**
  - 3
  - 7
  - 11

- **all gather**
- **sum**
- **all to all**
- **all reduce**

- **Additional in-network support**
  - *Data* replication
  - Computation – *data format* aware
  - Data coalescing and distribution
1. **Identify collective membership**
   - Select participating peers
   - Local operation – address vector sets

2. **Setup communication groups**
   - Coordinated join among members
   - Network operation (maybe) – 2 supported models

3. **Invoke collective**
   - Collective data transfer operation
IDENTIFY COLLECTIVE MEMBERSHIP
ADDRESS VECTOR SETS

Existing API

Application visible addresses of peer endpoints

<table>
<thead>
<tr>
<th>e.g.: IP : Port</th>
<th>Address Vector</th>
<th>Address Vector Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.0.1 : 7000</td>
<td>fi_addr_t</td>
<td>fi_addr_t</td>
</tr>
<tr>
<td>10.0.0.1 : 7001</td>
<td>0   100:3:50</td>
<td>0</td>
</tr>
<tr>
<td>10.0.0.2 : 7000</td>
<td>1   100:3:51</td>
<td>2</td>
</tr>
<tr>
<td>10.0.0.3 : 7003</td>
<td>2   101:3:83</td>
<td>4</td>
</tr>
<tr>
<td>...</td>
<td>3   102:3:64</td>
<td>6</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

AV
Represents the peer universe

Translated addresses

Unicast address used by API

Associated collective address

AV Set
Set of peers in a collective group

AV Set operations
- Insert
- Remove
- Union
- Intersect
- Difference

Local setup to identify members for new group

AV – Address Vector
SETUP COMMUNICATION GROUPS
JOIN COLLECTIVE

Universe (AV) or existing group

• **OFI-managed**
  • Provider implements coordination in-band
  • Provider reserves resources

• **Application-managed**
  • Group was established out-of-band
  • Resources already reserved

New collective multicast group

Current member

New member

fi_addr_t
struct fi_info *hints, *info;

hints fi_allocinfo();
<format hints>
hints->caps |= FI_COLLECTIVE;

fi_getinfo(FI_VERSION(1,14), hostname, NULL, FI_SOURCE, hints, &info);

<allocate fabric resources>

struct fi_collective_attr attr = {0};
attr.op = FI_SUM;
attr.datatype = FI_FLOAT;

fi_query_collective(domain, FI_ALLREDUCE, &attr, 0);
assert(attr.datatype_attr.count >= 100 && attr.max_members >= 50)
struct fid_av_set *av_set;
struct fi_av_set_attr attr = {0};
attr.start_addr = 2;
attr.end_addr = 100;
attr.stride = 2;
fi_av_set(av, &attr, &av_set, NULL);

struct fid_mc *mc;
fi_join_collective(ep, FI_ADDR_UNAVAIL, av_set, 0, &mc, NULL);

struct fi_eq_entry entry;
uint32_t event;
fi_eq_sread(eq, &event, &entry, sizeof(entry), -1, 0);
assert(event == FI_JOIN_COMPLETE);

fi_allreduce(ep, input_array, 100, NULL, result_array, NULL,
    fi_mc_addr(mc), FI_FLOAT, FI_SUM, 0, my_context);
OTHER THOUGHTS
ENSURE EFFICIENT MAPPINGS

Managing in-network resources
   Guarantee resources are available
   App may want to prioritize which collectives to accelerate
API object: collective resource tokens?

Priority
   Define impact on active collectives
   Preempt possible? Pause-resume or abort/cancel?
libfabric defines priority at the endpoint level
Do resource tokens act as a proxy?
Reproducibility of results
Order that data is fed into operations can produce different results
Relaxed reproducibility can reduce in-network memory
Setting: per-operation, group (AV set), resource token?

Sparse data
Avoid sending / storing null data
Define a compact, data aware SGL?
Other Thoughts

Ensure Efficient Mappings

Network topology
- Query collective support - local vs global?
- Peer endpoints relative to switches and accelerators
- Scope of the job or resource manager?

Programmable in-network accelerations
- Non-collective operations
- How does app specify operation and parameters?
- Entity responsible for programming switch/FPGA?
THANK YOU