Software

Recent Advances in Open Fabric Interfaces

Sayantan Sur, Intel

Content from OFIWG presentations by Sean Hefty

Legal Disclaimer & Optimization Notice

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks.

INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

Open Fabric Interfaces

User-centric interfaces lead to innovation and adoption

Open Source

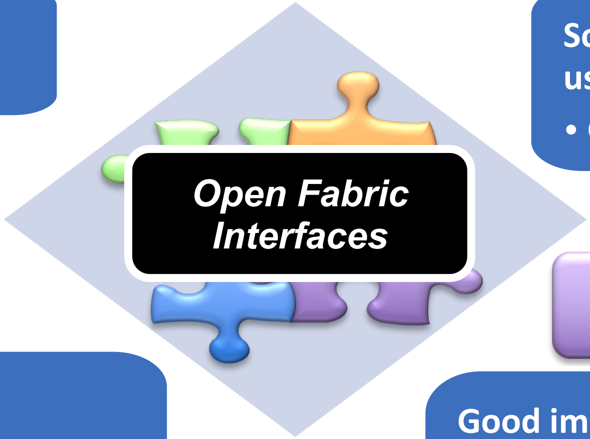
Inclusive development effort

- App and HW developers

User-Centric

Software interfaces aligned with user requirements

- Careful requirement analysis



Scalable

Optimized SW path to HW

- Minimize cache and memory footprint
- Reduce instruction count
- Minimize memory accesses

Implementation Agnostic

Good impedance match with multiple fabric hardware

- InfiniBand, iWarp, RoCE, raw Ethernet, UDP offload, Omni-Path, GNI, BGQ, …
- Works on Linux, Windows and MacOS

OFI – State of the Union

OFI Insulates applications from wide diversity of fabrics underneath

Intel® MPI Library | MPICH | Open MPI SHMEM | Sandia SHMEM | GASNet | Charm++ | Clang UPC | Chapel | PMDK#, Spark#, ZeroMQ#, TensorFlow#, MxNET#, NetIO, Intel MLSL, rsockets …

libfabric Enabled Middleware

OFI

Advanced application oriented semantics

- Tag Matching
- Scalable memory registration
- Triggered Operations
- Remote Completion Semantics
- Multi-Receive buffers
- Shared Address Vectors
- Unexpected Message Buffering
- Streaming Endpoints
- Reliable Datagram Endpoints

Sockets TCP, UDP | Verbs | Cisco usNIC* | Intel OPA | Shared Memory | Cray GNI* | AWS EFA* | Network Direct | IBM Blue Gene* | HPE Gen-Z* | RxM, RxD, Multi-Rail, Hooks …

Exploration

Endpoint Accelerators
Smart NICs, FPGA, GPU

Objectives

Expose common software APIs to apply data operations on network flows

Support offloaded accelerations in conjunction with network

- Smart NIC, FPGA, GPU, enhanced switches
- Local and/or remote accelerations
- Inline and look-aside

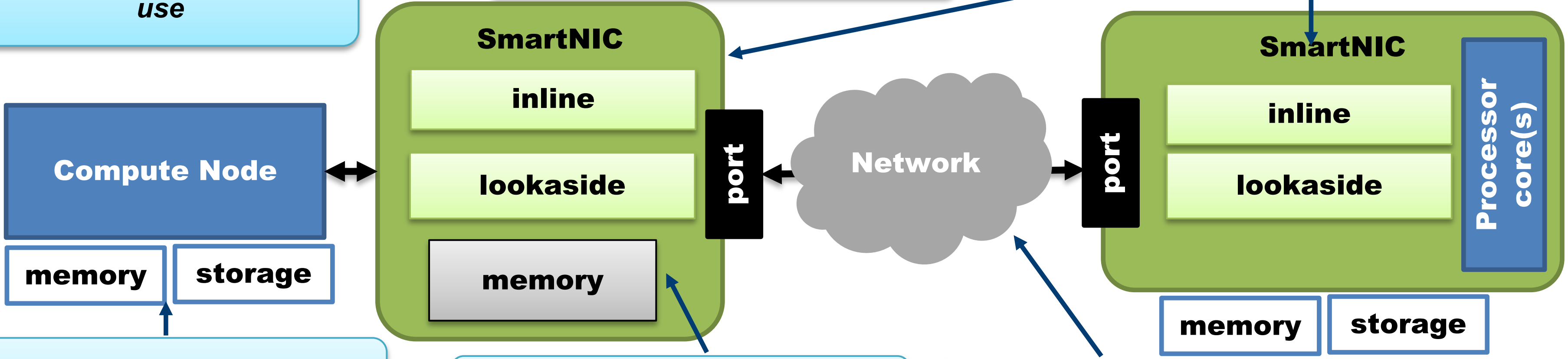
Discover available network functions

Enable functions at specific points in network data flows

This is not a general API for launching GPU/FPGA Kernels

Acceleration API requirements

- *Discovery mechanism – available vs active*
- *Select accelerator and function*
- *Support local and remote accelerations*
- *SmartNIC SW may need to program function prior to use*
- *Persistent vs on-demand functions*



- *Provide necessary input parameters and output results*
- *Support long-running functions (out-of-band execution)*
- *Network protocol may need enhancements*

Proposal (WIP)

Introduce new provider capability

Extend attributes to request/report available accelerations

Introduce new OFI object that corresponds to an acceleration

- Network function
- Generic base definition

Specify network function with data transfers

- Apply to all transfers of a specific type
- Specify per operation

Network Functions

New capability

```
#define FI_NETWORK_FUNC (1ULL << ?)
```

Define well-known functions, allow for extensions

- 'Chain' groups multiple functions together as a single larger function

```
enum {
      /* well known functions */
      fi_nf_noop,
      fi_nf_chain,
      …,
      /* OR in FI_PROV_SPECIFIC for
       * vendor specific functions
       */
};
```

Generic structure to request/report available functions

- Returned by existing fi_getinfo() call
- Extend domain attributes

```
struct fi_nf_info {
      struct fi_nf_info *next;
      int                type;
      uint64_t           caps;
      uint64_t           mode;
      uint64_t           flags;
      void              *data;
      size_t             data_len;
};
```

```
int fi_network_func(domain,
      struct fi_nf_info *nf_info,
      void * context,
      uint64_t flags,
      struct fid_nf **nf);

fi_ep_bind(ep, nf, flags);
e.g. flags = FI_SEND | FI_RECV
e.g. flags = FI_WRITE |
FI_REMOTE_WRITE
e.g. flags = 0
```

Open a network function

- Associate the function with an endpoint
- Specify types of data transfers the function applies to
- Or indicate that the function will be specified when submitting the transfer (flags=0)

Network Functions

Specify function to apply to the current data transfer via existing context parameters

- Provide any needed input/output parameters

```
struct fi_nf_context {
	struct fid_nf *nf;
	void          **params;
	size_t          param_cnt;
	size_t         *param_len;
	void           *reserved[4];
};
```

Re-use deferred work queues to execute long-running functions separate from data transfer

- Assumes results will be used by future transfers

```
struct fi_deferred_work { … }
fi_control(…)
FI_QUEUE_WORK
FI_SUBMIT_WORK
FI_CANCEL_WORK
FI_FLUSH_WORK
```

Network Accelerators Collective Offloads, etc.

Motivation

Support *fabric* based offloads

- Versus NIC based offloads

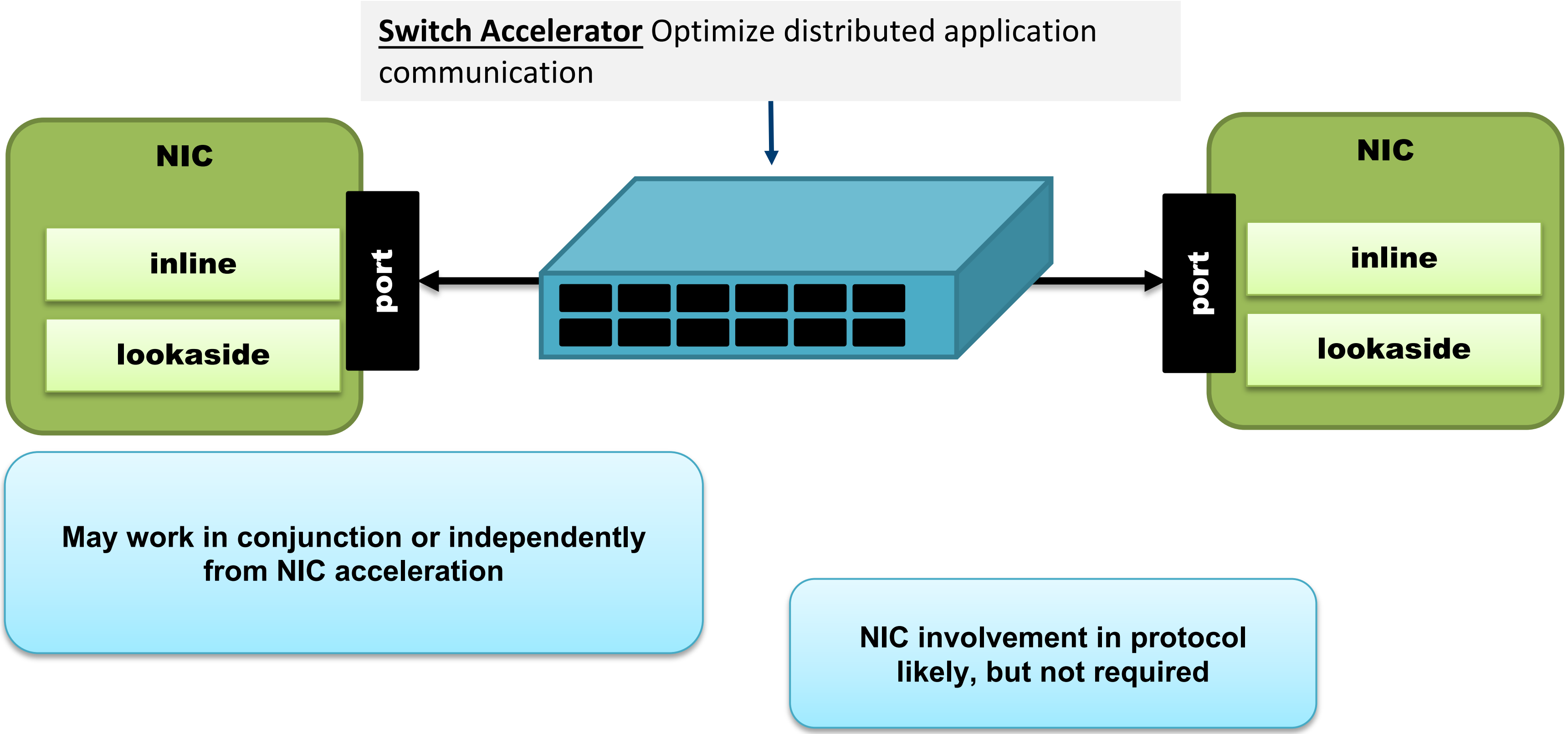
Immediate goal: accelerate collective operations

- SmartNIC / FPGA proposal focuses on endpoint accelerations
- This proposal focuses on network accelerations
- Separation is purely for convenience of discussion and focus

Solution should not be limited to collectives

Avoid trying to turn OFI into MPI

Network Accelerators



Collective Groups

Define a group based on fabric addresses

Provide the list of fabric addresses and stride information to define the group

Create the group, known as the AV Set

Each process creates the av_set

Derive new AV Sets using high-level APIs

The AV set operations correspond to MPI communicator operations

```
struct fi_av_set_attr {
        size_t        count;
        fi_addr_t     start_addr;
        fi_addr_t     end_addr;
        uint64_t      stride;
};
struct fi_ops_av {
        …
        int (*av_set)(struct fid_av *av, struct
                fi_av_set_attr *attr,
                struct fid_av_set **av_set,
                void *context);
};
struct fi_ops_set {
        int (*set_union)(struct fid_av_set *dst,
            const struct fid_av_set *src);
        int (*intersect)(…);
        int (*diff)(…);
        int (*insert)(struct fid_av_set *set,
                fi_addr_t addr);
        int (*remove)(…);
};
```

Group Membership and Query

Join a Multicast group for collective

Every process part of the collective needs to issue the join operation and get connected to the switch group

```
int fi_join_collective(struct fid_ep *ep,
      fi_addr_t coll_addr,
      const struct fid_av_set *set,
      struct fid_mc **mc,
      void *context);
```

Leave a Multicast group for collective

Simply close the fabric object representing the group

```
int fi_close(struct fid_mc mc);
```

Query Libfabric for collective support

Which combinations of datatype, operations, number of members etc.

```
int fi_query_collective(struct fid_domain *domain,
      enum fi_datatype datatype,
      enum fi_op op,
      struct fi_collective_attr *attr,
      uint64_t flags);
```

Collective Operations

Initial set of collectives defined that can be accelerated

- Barrier: no data transferred
- Broadcast: flags indicate FI_SEND/RECV based on whether the process is root
- Allreduce: non-void datatype required
- We will look at defining order of reductions

```
ssize_t fi_barrier(struct fid_ep *ep,
      fi_addr_t coll_addr, void *context);

ssize_t fi_broadcast(struct fid_ep *ep,
      void *buf, size_t count, void *desc,
      fi_addr_t coll_addr, enum fi_datatype dtype,
      enum fi_op op, uint64_t flags, void *context);

ssize_t fi_allreduce(..);

ssize_t fi_reduce_scatter(..);

ssize_t fi_alltoall(..);

ssize_t fi_allgather(..);
```

Communication from Accelerators

Overview

Support data transfers to/from/between peer devices

- GPUs, FPGAs, persistent memory
  - Examples only, solution is *mostly* independent from type of device
- PCI peer to peer transfers
  - E.g. GPU to RDMA NIC, GPU to GPU
  - Implementation agnostic, e.g. bounce buffers

Application examples:

- Machine learning, deep learning, AI, MPI

Scope is limited to fabric communication (not trying to be the accelerator interface itself)

Related

Support libfabric running on non-host cores

Invoke accelerator functionality as part of transfer

- Covered by separate SmartNIC proposal

File system backed memory regions

- Transfers to/from storage without mapping file into process VA space

Definitions

Proposal only - WIP

Heterogeneous memory (HMEM)

- Non-host memory (e.g. PCI device memory)

Unified address space

- Memory may be mapped into virtual address space of process
  - Without unified virtual address space, RMA may require FI_MR_RAW

Peer software interface

- Software interface to access peer device / allocate memory
- E.g. CUDA, OpenCL
- Compile and/or run time option?
- May need to support multiple SW interfaces simultaneously from single app

Requirements

SW may directly access HMEM device

- E.g. execute GPU kernels
- Most efficient if SW marks addresses as HMEM or not
  - Need to include device identifier

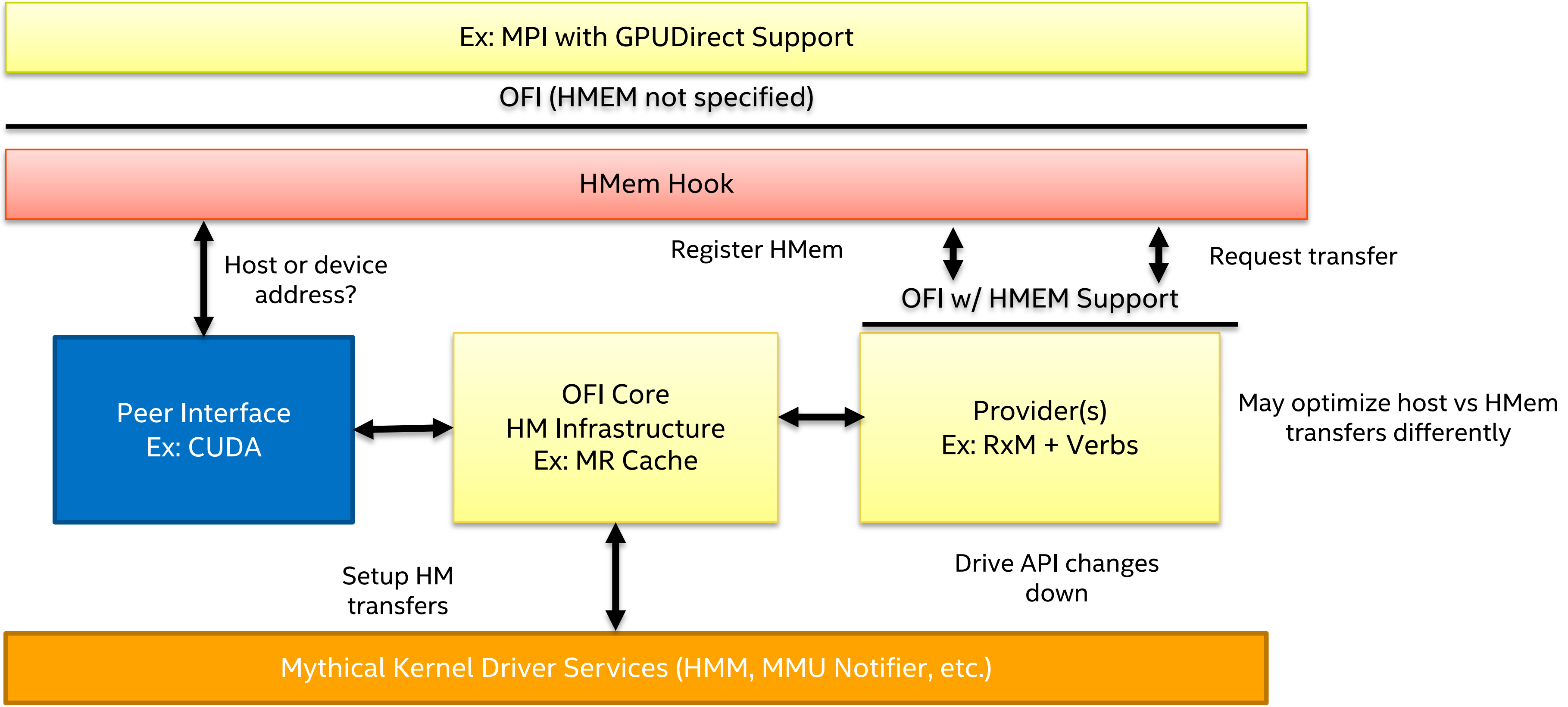
Middleware may lose HMEM association

- Rediscover HMEM properties
- Pass properties between app and OFI external from middleware
  - e.g. per-thread global variables

Memory *may* be allocated using specialized functions

- e.g. cudaMalloc vs malloc

HMEM Hooking provider



Summary

OFI community is moving to intercept fabric offloads and compute accelerators

The APIs being defined are generic and can be used for multiple vendors

Collective Offload APIs offer both endpoint and switch accelerations

Compute Accelerator APIs are in progress of being defined and will aim for both callable on Host and Accelerator cores

Participation is OFIWG is free, simple and no boards to join

http://libfabric.org