Scalable Fabrics Interface

Lessons learned from MVAPICH

Sayantan Sur, Intel

2nd Annual MVAPICH User Group Meeting (MUG)

8/27/2014
Legal Disclaimer & Optimization Notice

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.

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.

Copyright © 2014, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

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
The Path to Exascale …

Constrained power envelope

Data movement costs must be contained
i) Reduce power in the core to move data
ii) Reduce overheads of communication (finer-grain)

Communication stack is critical

Even in the current generation performance is gated by software overheads
Current OFA Verbs - Application send

**Significant SW overhead**

```
struct ibv_sge {
    uint64_t addr;
    uint32_t length;
    uint32_t lkey;
};
```

```
struct ibv_send_wr {
    uint64_t wr_id;
    struct ibv_send_wr *next;
    struct ibv_sge *sg_list;
    int num_sge;
    enum ibv_wr_opcode opcode;
    int send_flags;
    uint32_t imm_data;
    ...
};
```

**Application request**

- `<buffer, length, context>`

- 3 x 8 = 24 bytes of data needed
- SGE + WR = 88 bytes allocated

- Requests may be linked - next must be set to NULL
- Must link to separate SGL and initialize count
- App must set and provider must switch on opcode
- Must clear flags
- 28 additional bytes initialized

MUG 2014
Current OFA Verbs - Provider Send

- For each work request
- Check for available queue space
- Check SGL size
- Check valid opcode
- Check flags x 2
- Check specific opcode
- Switch on QP type
  - Switch on opcode
  - Check flags
- For each SGE
  - Check size
  - Loop over length
- Check flags
- Check
- Check for last request
- Other checks x 3

- Most often 1 (overlap operations)
- Often 1 or 2 (fixed in source)
- Artifact of API
- QP type usually fixed in source
- Flags may be fixed or app may have taken branches
- 19+ branches including loops
- 100+ lines of C code
- 50-60 lines of code to HW

MUG 2014
Develop an extensible, open source framework and interfaces
aligned with ULP and application needs for high-performance fabric services
Enable ...

- **Minimal footprint**
  - Reduced cache and memory footprint

- **High performance**
  - Optimized software path to hardware
    - Independent of hardware interface, version, features

- **App-centric**
  - Analyze application needs
    - Implement them in a coherent, concise, high-performance manner

- **Extensible**
  - More agile development
    - Time-based, iterative development
    - Application focused APIs
    - Adaptable
SFI Model

Framework defines multiple interfaces

Fabric Interfaces
- Control Interface
- Message Queue
- RMA
- Atomics
- CM Services
- Addressing Services
- Tag Matching
- Triggered Operations

Fabric Provider Implementation
- Control Interface
- Message Queue
- RMA
- Atomics
- CM Services
- Addressing Services
- Tag Matching
- Triggered Operations

Vendors or Community provide optimal implementations

MUG 2014
MVAPICH/MVAPICH2

MPI Implementation over InfiniBand, 10GE/iWarp, RoCE
Being developed since 2001
Continues tracking new developments in Verbs and HW
Rich history of strong research and publications
Huge impact to the Commodity HPC cluster community
Two major lessons from the research:
(i) Designing MPI/PGAS over verbs takes significant effort
(ii) Scaling MPI/PGAS over verbs also requires effort
MVAPICH Research and Design Topics

Performance
- Use hardware resources in an optimal fashion
- Optimize Protocols
- Collective Communications

Scalability
- Connection Management
- Buffer Management
- Flow Control

Other Topics
- Multiple generations of processors, adapters …
- Hybrid Programming Models

The cross-product of these are daunting!!

MUG 2014
Performance Lessons (1)

RDMA vs Send-Recv

- RDMA P2P is better than Send-Recv, but only Send-Recv scales
- *Message Based transport is desired, with better performance*
- *Tag matching semantic offload, zero copy (SFI Tagged interface)*

Eager vs Rendezvous

- Eager/Rendezvous protocol switch (MV2_HOMOGENEOUS_CLUSTERS)
- *Visibility of protocol is essential for interoperability (SFI info::protocol)*

Memory Registration

- Caching of recently used registration entries, intercept malloc, etc.
- *Remove memory registration from critical path (SFI Dynamic MR)*

Other optimizations

- Send queue sizes, huge pages, polling optimizations for fairness, …
- *Separate policy from code (SFI configuration file based optimizations)*
Performance Lessons (2)

Overlap of compute and communication

- RDMA w/ Interrupt, Asynchronous progress thread
- *Tagged API enables all provider optimizations*

Efficient completion mechanisms

- Completion processing can be expensive
- *Multiple completion formats (SFI FI_EQ_FORMAT_CONTEXT)*
- *Counters for aggregate completions*

Non-blocking Collectives with low noise

- Offload various steps of collective operations
- *Triggered operations to schedule communication using counters*

Locking and Progress

- Multi-threaded communication, internal threading
- *SFI threading modes, progress modes (FI_THREAD_PROGRESS, FI_PROGRESS_AUTO)*
Scalability Lessons

Transport Scalability

- Major design issue - RC, SRQ, XRC, UD, DCT, ...
- *Reliable Datagram support is desired with zero copy (SFI FID_RDM)*

Buffer Management and Flow Control

- Dynamic buffer pools, asynchronous buffer management, locks, ...
- Reduction in buffering/copying is better for performance
- *Tagged API tightly couples buffers at the provider level*

Connection Management

- User-level on-demand connection management and scalability
- Address resolution required for 3D torus and other topologies
- *Scalable and Integrated address resolution (SFI Address Vectors)*
An MPI Implementation example

/* Tagged provider */
hints.type = FID_RDM;
#ifdef MPIDI_USE_AV_MAP
  hints.addr_format = FI_ADDR;
#else
  hints.addr_format = FI_ADDR_INDEX;
#endif
hints.protocol = FI_PROTO_UNSPEC;
hints.ep_cap = FI_TAGGED |
   FI_BUFFERED_RECV |
   FI_REMOTE_COMPLETE |
   FI_CANCEL;
hints.op_flags = FI_REMOTE_COMPLETE;

Reliable unconnected endpoint
Address vector optimized for minimal memory footprint and no internal lookups
Transport agnostic
Behavior required by endpoint
Default flags to apply to data transfer operations
Querying and Setting Limits

optlen = sizeof(max_buffered_send);
fi_getopt(tagged_epfd, FI_OPT_ENDPOINT,
        FI_OPT_MAX_INJECTED_SEND,
        &max_buffered_send, &optlen);

optlen = sizeof(max_send);
fi_getopt(tagged_epfd, FI_OPT_ENDPOINT,
        FI_OPT_MAX_MSG_SIZE,
        &max_send, &optlen);
Small Message Send

```c
int MPIDI_Send(buf, count, datatype, rank, tag,
               comm, context_offset, **request)
{
    data_sz = get_size(count, datatype);
    if (data_sz <= max_buffered_send) {
        match_bits = init_sendtag(comm->context_id +
                                 context_offset,
                                 comm->rank, tag, 0);
        fi_tinjectto(tagged_epfd, buf, data_sz,
                     COMM_TO_PHYS(comm, rank),
                     match_bits);
    } else {
        ...
    }
}
```

Small sends map directly to tagged-injectto call

Fabric address provided directly to provider
Large Message Send

```c
int MPIDI_Send(buf, count, datatype, rank, tag,
    comm, context_offset, **request)
{
    /* code for type calculations, tag creation, etc */
    REQUEST_CREATE(sreq);
    fi_tsendto(MPIDI_Globa1.tagged_epfd, send_buf,
        data_sz,
        NULL,
        COMM_TO_PHYS(comm, rank),
        match_bits,
        &(REQ_OF2(sreq)->of2_context));
    *request = sreq;
}
```

Large sends require request allocation

SFI completion context embedded in request object
Co-chairs – Sean Hefty and Paul Grun

- Contacts: sean.hefty@intel.com / grun@cray.com
- Meets Tuesdays from 9-10 PST / 12-1 EST

Links

- Mailing list subscription
  - http://lists.openfabrics.org/mailman/listinfo/ofiwg
- Document downloads
  - https://www.openfabrics.org/downloads/OFIWG/
- libfabric source tree
  - https://github.com/ofiwg/libfabric.git

Request for direct participation as we move from architecture to implementation