A Multithreaded Communication Substrate for OpenSHMEM

OpenSHEM Users’ Group Workshop
Oct. 7, 2014, Eugene, OR

Aurélien Bouteiller, Thomas Hérault, George Bosilca
Motivation

- **UCCS**: Provide a common low-level scalable, robust, portable, simple and performance driven communication API for multiple parallel programming models over modern network interfaces.
- Provide common network code for implementing programming models, increasing code reusability and reducing development effort.
- Support hybrid programming environments efficiently, mixed programming models.
- Need to support efficient thread safety.
Thread safety: Posting Operations

- `uccs_put_large_contiguous_nb(ctx, ep, ..., desc)`
  - Creates a descriptor for the operation (aka a request)
  - Can be completed later by `uccs_wait`, `uccs_test`

- In the UCCS specification, all postings are thread safe (can be issued simultaneously from multiple threads, on the same communication channel)

- Implementation of most RMA operations uses fine grain locking (only active message posting is not yet), using OPAL lockless lists and data structures

- There remain some state sharing between threads (due to sharing the endpoint handle)
  - Current proposal in the OpenSHMEM community would help addressing these issues (endpoints, contexts)
Thread Safety with Progress

• UCCS specification: no guaranteed* progress if no thread is calling a progress function
  • uccs_wait, uccs_test, uccs_progress
  • To ensure the semantic of OpenSHMEM asynchronous progress, one thread can be delegated to progress UCCS
  • *The UCCS implementation may spawn threads of its own to progress RMA

• Multiple threads can enter progress simultaneously

• Active Message callbacks may be invoked from any thread that enters progress
  • The callbacks must ensure their own exclusive access to shared variables if needed

• 2 types of AM callbacks can be registered
  • Network_Protocol
    high priority functions, that demand no additional resource and only a few computations. Can run on any thread.
  • User
    for more demanding operations. Can run on User Threads only
RTE API

• UCCS specification provides a portable Runtime Environment API
  • Processor architecture, endianess, hardware locality info database
  • Network ID database
  • I/O forwarding
  • Out of band communication service
• UCCS Specification: assume all RTE routines are thread safe
• Can employ PMI, Open RTE, STCI, Slurm, etc. as the backend
• Unfortunately not all of these RTE backends are thread safe
• Used mostly when establishing new connections: RTE is not performance critical
• Implementation delegates the RTE in a separate thread. All commands delegated to the RTE thread
Performance Evaluation

- **Infiniband 20G cluster**
  - 2x Intel Xeon E5520 (Nehalem) 4cores (total 8 cores/node)
  - Linux CentOS 6.5

- **Intel MPI Benchmarks 4.0 (RMA)**
  - Open MPI 1.7.5 (non thread safe build)
  - Deploying $t$ processes per node (single threaded), each process pinned to a physical core (multi mode)
  - Put/Get bidirectional aggregate mode benchmarks are considered

- **UCCS benchmarks**
  - UCCS version (thread safe extended)
  - Deploying 1 process per node (with $t$ threads), each thread pinned to a physical core
  - Similar communication pattern to the IMB deployment
Active Message Injection Rate

- 1050 AM message sent from the origin, T threads are spinning in uccs_progress()
- When the 1050 messages have been received, an ack is sent back
- Some congestion is observed (some coarse grain locks still present in the implementation)
- In all threaded cases, injection rate improved with MT compared to single thread

UCCS Multithreaded Active Message Injection Rate (IB20G)
1 send thread, T recv threads, 4 bytes messages

- Maximum
- Standard Deviation
- Average
UCCS Fairness

- Same AM benchmark, 8 receive threads
- Perfect load balance: ~131 callbacks per threads
- No thread is favored, good spread
- Standard deviation between runs is low
- Is this a good thing? (think interrupt spread)
The RMA benchmarks

- MPI deployments use 1 process per core
- UCCS deployments use 1 thread per core

MPI processes pairs, even ranks on first node, odd ranks on second node

Flush

UCCS RMA

AM ack
The RMA benchmarks

- MPI deployments use 1 process per core
- UCCS deployments use 1 thread per core

MPI processes pairs, even ranks on first node, odd ranks on second node

Flush

AM ack
Starting point: large grain mutexes

- This is the poor performance with a coarse grain mutex protecting all routines...
- Not very enticing, can we do better?
Multithreaded Put benchmark

- 1 thread:
  - UCCS bandwidth slightly better than MPI

- 2 threads:
  - UCCS and MPI bandwidth improved (especially for medium messages)

- More threads:
  - MPI reaches best bandwidth for 4 threads, congestion starts to happen for 8 threads
  - UCCS also congested for 4/8 threads
Multithreaded Get benchmark

• Same observation hold for GET benchmark
Concluding Remarks

• Designed thread safety interface to UCCS specification

• Early performance results encouraging
  • Competes with multiple MPI processes (in a non-thread safe MPI build), with multiple non-shared endpoints
  • Observed good overall fairness, even when contentions are observed

• Future works
  • Observing behavior with multiple channels/qpairs to separate threads
  • Observing overlap in synthetic benchmarks
  • Observing OpenSHMEM application performance
  • Investigating “endpoints/PE/contexts” proposed extension to OpenSHMEM
  • Investigating “mixed model” programming (OpenSHMEM+MPI, OpenSHMEM + dataflow) when sharing the base communication substrate