Efficient Interoperability of OpenSHMEM on Multicore Architectures

Khaled Z. Ibrahim  
kzibrahim@lbl.gov

Lawrence Berkeley National Laboratory

OUG 2014: Oct. 7th, 2014
Supercomputing Architectures

Exascale Architectures: what to expect?
- Hybrid architectures
- Slower cores for power-efficiency (better to inject data while being produced to smooth traffic)
- Less physical memory per core (smaller transfers)

Exascale problems: what to expect?
- Strong scaling problems (smaller transfers)

Programming Model Challenge
- Need model composition and efficient support for small transfers.
General Conditions for Efficient Interoperability

OpenSHMEM and Interoperability
Runtime Abstraction - Pure processes

- Model:
  - Uniform MPI/SHMEM/UPC (using processes)

- Pros:
  - Should be easier for application developers.
  - Supporting distributed runtime on shared resources should be easier for runtime designers.
Why Pure Processes Not favorable on Shared Architectures?

- **Sharing is complex**
  - Multiple name spaces
  - Complex mechanisms for sharing data (via mmap/aliasing) or code (via remote procedure call (RPC))

- **Sharing not efficient**
  - Resource replication (less efficient memory use), **problematic at large scale**

- **Sharing not explicit**
  - Language runtimes cannot assume physical sharing with processes.
  - Ambiguity leads to conservative assumptions → performance penalty
  - Most openMP runtime and tasking libraries are based on threads. Also, parallel libraries (FFT, BLAS, etc) relies on threading within a node. Efficiency rather than productivity force us to use vendor optimized libraries.
Runtime Abstraction - processes + threads (funnel)

Model:
- Compose but Funnel: (compose but always switch between the main programming model and the $x$)
- Funneling $\rightarrow$ Master (or single) thread communicates on behalf of all (or a group of) threads.
- Extra synchronization, less injection parallelism, but ...

More efficient, why?
Model:
- MPI/SHMEM/UPC + x (persistent x, i.e. no model switch)

Pros:
- Expose architectural sharing (one name space) → memory efficiency

Could we communicate efficiently in a parallel region?
- Less application synchronization, but rarely used!
- 41× slowdown at concurrency level of 24 for small messages
- Difference decreases with the increase in message size → problem shifts to bandwidth
- Same for all MPI implementations and UPC implementations I am aware of.
Processes access to interconnect
  ▶ lock-free access points to the interconnect messaging system
  ▶ Separate injection resources.
HPC Runtimes with Threads (sharing)

- Threads share an access point
  - Serialization within the programming language runtime
  - Serialization to access the messaging system
- Rely on system library thread safety
  - e.g., Cray DMAPP (= serialization, system locks) → not matching processes performance for communication
What are the issues preventing efficient interoperability?

- **Addressability**: Influenced by the programming model
  - Are threads addressable entity? or are they affecting addressability?

- **Separability of communication paths**:
  - How to allocate and manage resources for independent transfers?

- **Full direct reachability**:
  - Could we have full reachability with separable communication paths?

Feasible for PGAS one-sided primitives. Difficult to satisfy for active messages or MPI two-sided.

More details in our ICS 2014 paper:

Addressability: In PGAS, the target is a physical memory → a target process (or node) is just to resolve affinity not for processing incoming traffic.

Separability: each thread could have its own communication domain and own resources

Reachability: need it full and possibly redundant.
Shared resources require thread safety
  ▶ Thread-safety through serialization
    ▶ Critical regions (lock/unlock)
    ▶ lock free data structures (atomic-based)
  ▶ Thread safety through resource split
    ▶ All levels of the stack should support resource split.
    ▶ Need to avoid implicit serialization by all used libraries, for instance memory allocation.
Up to $31 \times$ improvement for small messages compared with lock-free (atomic-based) algorithms

Monotonic improvement with the domain count

UPC is exploiting the GASNet features, and we plan to have better support for OpenSHMEM on GASNet.
IBM PAMI MultiContext and One-sided Performance

- Up to $16 \times$ improvement for small messages
- BGQ Single socket systems.
General Conditions for Efficient Interoperability

OpenSHMEM and Interoperability
OpenSHMEM and Interoperability - Programming Model Interface

shmem_int_put(IN endpoint, IN target, IN source, IN nelems, IN pe)

- Need an identifier for thread-specific resources, or lose hundreds of cycles for thread state lockup. Injection overheads are few tens of cycles.
  - MPI tried to adopt but is more suitable for PGAS.
- Direct mapping to IBM PAMI contexts, Cray Domains, Infiniband rails, Portal 4 VNICS.
- Application level allocation and query of thread-specific resources.
- Additional threads safety using locks should not hurt performance as long this lock does not migrate.
- GASNet already support such resource split on Cray Machines, released Nov. 2013.
- The challenge is at the target side. Need to match the injection rate with drainage rate.
Message layer domains (context, rails, or endpoints) are scarce resources on most systems!
- Cannot provide square of the threads to enable thread addressibility.

Linear communication paths require redundant reachability path to the same physical address.
- Redundant memory registration create redundant reachability paths.
- Portals 4 non-matching tables need to be coherent across all virtual NICs.
Need to avoid active target model

- Require service threads equal to the injection threads.
- Require mutual exclusion at the target if the service thread is not specified.
Conclusions

- **Interoperability is key for future extreme scale systems.**
  - Unless you want your programming model to be a superset of everything.

- **PGAS one-sided models provide efficient abstractions for interoperability.**
  - Addressibility should not be affected by the use of threads or processes.

- **We have a successful GASNet implementation for efficient interoperability.**
  - Berkeley UPC is benefiting from it and we are working on extending it to OpenSHMEM.

- **Conditions for efficient interoperability with pthreads:** Addressability, Separability, Reachability.

- **Target side requirements are tricky:** multi-reachability (coherent redundant symmetric heap tables), passive target, etc.
Thanks & Questions

This research used resources in Lawrence Berkeley National Laboratory, which is supported by the DOE Office of Advanced Scientific Computing Research under contract number DE-AC02-05CH11231. This research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357.