OpenSHMEM with Threads: A Bad Idea?

Gabriele Jost, Ulf R. Hanebutte, James Dinan

Motivation
Stimulate discussion on multi-threaded execution in OpenSHMEM

- Typical motivation for multi-threading:
  - Exploit available shared address space
    - Use direct load/store access rather than calls to communication library
    - Enable compiler optimizations for better performance
    - No communication calls, communication buffers
  - Reduced memory requirements
  - Lore for Extreme Scale Systems: MPI + X \(\Rightarrow\) SHMEM+Threads?

- ... BUT: OpenSHMEM is based on PGAS:
  - Global address space within each node:
    - Direct access using `shmem_ptr`
    - Smart OpenSHMEM avoids unnecessary communication

- Why introduce extra baggage?
  - Thread start-up overhead, synchronization, thread-safety, other interoperability requirements, ...
SHMЕМ on SMP Clusters

OpenSHMEM memory model:

- Two shared, symmetric segments:
  - Data: statically declared shared objects
  - Heap: dynamically allocated share objects

- Access symmetric object through Processing Element (PE) local pointer
  - Within one SMP node segments of peer PEs can be mapped to each other (shmem_ptr)

Performance challenges on SMP nodes:

- “non-uniform” HPC systems:
  - Rich hierarchy of shared caches, cc NUMA domains, NUMA shared memory islands, multiple sockets, large number of cores, etc

- Application inherent multi-level parallelism:
  - Domain decomposition (coarse-grained)
  - Intra-domain solver routines (finer-grained)

- How do we map the application hierarchy onto the system hierarchy?
  - Topology-aware OpenSHMEM and/or hybrid programming?
Hybrid Programming Execution Modes

- **PE only**: 1 Process on each core per node
- **PE + Threads**: Multiple PEs per node, multiple threads per PE
- **PE + PE**: Communication calls between Nodes, Shared memory access within Node
  - Update through direct data copy within node
  - Update through direct load/store

**Master-Only**: Communication only outside of parallel regions

**Communicating threads inside of parallel regions**

- **Funneled**: Communication only on master-thread
- **Funneled with Full Load Balancing**: Multiple more than one thread may communicate

**Example**: NPB-MZ

**Example**: SNAP
Hybrid Programming Opportunities
Past experience mostly based on MPI+OpenMP

Lower memory requirements
- Reduced amount of replicated user data and reduced size of communication buffers -> often impacts computational performance
- Very important for systems with 100’s or 1000’s of nodes

Convenient way to exploit fine grained parallelism
- NPB-MZ LU: Multi-zone CFD code with limited #zones

Provide for flexible load-balancing on coarse and fine grain
- NPB-MZ BT: Assign multiple smaller zones to one PE
- PEs with larger workloads could employ more threads

Convenient way to exploit task parallelism at fine granularity
- OpenMP 4.0 tasking, OmpSS, Cilk,....
Some Thoughts on Thread Support

Thread-safety in OpenSHMEM is essential
- Different levels of thread support, similar to MPI, proposed by Cray

Just adding thread safety may not be enough:
- Threads share communication state of the PE
  - `shmem_fence/quiet`, barrier affect all threads, introduces unnecessary synchronization
- Threads share network-level resources and state

Desirable model would be:
- Multiple threads associated with one PE
- Memory resources depend on the #PEs only
- **Per-thread communication resources**
Threads in OpenSHMEM: Why we don’t need them

- Hybrid programming convenient, but is it the most efficient?
- OpenSHMEM pointers improve compute performance without threads
- OpenSHMEM extensions could address memory resource requirements
  - changes to the memory model
- OpenSHMEM process teams to exploit application hierarchies
- OpenSHMEM Standard extensions can do more:
  - Enable multiple communication streams for non-blocking communication efficiently e.g. via communication contexts
  - Exploit shared memory which is not cache coherent
  - ...
Threads in OpenSHMEM: Why we can’t live without them ... for now

Interoperability:

- Need to support OpenSHMEM + X
  - X = OpenMP, Cilk, ompSS, etc
  - Hybrid programming *evolutionary* path to bring existing applications up to Exascale

Hybrid programming *convenient* to exploit application multi-level parallelism

Reduce memory requirements

- Add a second, node-level programming API
- User can adjust node-level memory requirements
Questions for OpenSHMEM community

Characteristics of realistic OpenSHMEM applications:

- Likely to be different from legacy codes based on 2-sided MPI
- Computations tightly interleaved with non-blocking communication and atomic updates
- Need for realistic workloads to identify and investigate limits of OpenSHMEM

What will OpenSHMEM look like for Extreme Scale in 2020?

Evolutionary approach:

- Roadmap for OpenSHMEM evolution in small steps
- OpenSHMEM extensions to address limitation

Revolutionary approach:

- Start new and provide backward compatibility over time
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
OpenSHMEM Memory Model

PE 0
Symmetric Data Segment
Symmetric Heap
Private Memory

PE 1
Put

PE 2
Get

PE 3
“Non-Uniform” HPC Systems
Structure of NPB-MZ

- NPB-MZ: Multi-Zone Versions of the NAS Parallel Benchmarks
- Computational Fluid Dynamics
- Available at www.nas.nasa.gov
- “SHMEM”-like implementations in older distributions are available (e.g., NPB 3.1-MZ)
- All communication outside of parallel regions
NPB-MZ BT Code Example

call omp_set_numthreads (weight)
do step = 1, itmax
    call exch_qbc(u, qbc, nx,...)
    call to shmemb__put
end do

do zone = 1, num_zones
    if (iam .eq. pzone_id(zone)) then
        call zsolve(u,rsd,...)
    end if
end do

end do

...
SNAP Application Proxy Code Example

- Solves linear Boltzman transport equation
  - A wave-front is employed for parallelization
- OMP threads handle MPI communication independently
- Sending thread identified via sending processor and message tag

```c
!$OMP DO SCHEDULE(STATIC,1) PRIVATE(i)
DO i = 1, num_grth
  IF ( dogrp(i) == 0 ) CYCLE
  CALL sweep_recv_bdry ( dogrp(i), iop )
  CALL octsweep ( dogrp(i), iop, jd, kd, jlo, jhi, jst, &
                  klo, khi, kst )
  CALL sweep_send_bdry ( dogrp(i), iop )
END DO
$OMP END DO NOWAIT
```

Call MPI_RECV (value, ..., proc, tag, ..)