Toward Operating System Support for Scalable Multithreaded Message Passing

Balazs Gerofi, Masamichi Takagi, Yutaka Ishikawa
RIKEN Advanced Institute for Computational Science,
Tokyo, JAPAN
Motivation

- Large (and growing) number of CPU cores in many-core chips
- Relatively small (and decreasing) proportion of per-core memory
- Prevalence of hybrid MPI+X (e.g., OpenMP) hybrid programming models
- Recent high-speed interconnection networks expect communication to be driven explicitly by multiple CPU cores
- Multiple threads interact with the MPI library, leading to resource contention on internal structures
- Scalability of MPI in presence of multiple threads is an issue
  - Re-designing/re-writing MPI for scalability is a significant effort
  - Can the operating system help?
Agenda

- Motivation
  - Background
  - Thread private shared library (TPSL)
  - Modifications to MPI
- Evaluation
- Discussion
- Conclusion
Background: Hybrid LWK Kernels

- Lightweight kernels in HPC
  - Low OS noise required for large scale bulk synchronous applications
  - Developed either from scratch or by eliminating features of a general purpose kernel (i.e., Linux) that inhibit scalability
  - Usually comes at the price of limited POSIX/Linux API support

- Applications increase in complexity
  - In-situ analysis/visualization, complex workflows, etc..
  - Heavily rely on POSIX/Linux

- How to achieve both at the same time?
  - Idea: run Linux and an LWK on compute nodes side-by-side and provide OS features selectively by the two kernels
Background: Hybrid LWK Kernels

- Lightweight kernels in HPC
  - Low OS noise required for large scale bulk synchronous applications
  - Developed either from scratch or by eliminating features of a general purpose kernel (i.e., Linux)
  - Usually comes at the price of limited POSIX/Linux API support

- Applications
  - In-situ analysis/visualization, complex workflows, etc..
  - Heavily rely on POSIX/Linux

- How to achieve both at the same time?
  - Idea: run Linux and an LWK on compute nodes side-by-side and provide OS features selectively by the two kernels

Plus: Lightweight kernels have small codebase, which enables rapid prototyping for supporting exotic hardware features and/or new software concepts!
Background: IHK/McKernel

- Interface for Heterogeneous Kernels (IHK)
  - Allows dynamically partitioning node resources (e.g., CPU cores, physical memory)
    - SMP chip and accelerator support (i.e., Xeon Phi)
  - Enables management of LWKs (assign resources, load, boot, destroy, etc..)
  - Provides inter-kernel communication (IKC), messaging and notification

- McKernel
  - A lightweight kernel developed from scratch, boots from IHK
  - Designed for HPC
    - Noiseless, simple, implements only performance sensitive system calls (roughly process and memory management) and the rest are offloaded to Linux
Agenda

✓ Motivation
✓ Background
  ▪ Thread private shared library (TPSL)
  ▪ Modifications to MPI
  ▪ Evaluation
  ▪ Discussion
  ▪ Conclusion
Threads, Address Spaces and Page Tables

- From a multithreaded standpoint the very central notion of a process is its *shared address space*.
- Page tables are just a representation of virtual to physical memory translation.
- Traditional operating systems use process wise page tables:
  - i.e., on a multicore chip, all CPU cores running threads of the same process refer to the same set of mappings.
  - Intel x86: CR3 register.
- **Main observation:**
  - Processes are abstract, software level constructs.
  - Page tables are specific to the HW.
  - The fact that threads are provided with the same view of virtual memory has nothing to do with HW page tables.

*There is nothing that prevents the usage of separate page tables just because the address space is shared!*
Proposal: Thread Private Shared Libraries

- Separate page tables allow mapping arbitrary virtual address to different physical memory on a per-thread basis
  - For instance: mapping a shared library in a thread private fashion!
- TPSL blurs the notion of processes and threads
  - From a regular mapping's point of view threads belong to the same process
  - From the standpoint of a TPSL mapped library each thread appears as a separate process
- Mapping MPI via TPSL results in per-thread MPI ranks
  - Eliminates multithreaded resource contention inside MPI without redesigning it!!
- glibc and OpenMP remain as usual
  - Heap is shared and OpenMP constructs are available even for threads with different MPI ranks
Agenda

✓ Motivation
✓ Background
✓ Thread private shared library (TPSL)
  ▪ Modifications to MPI
  ▪ Evaluation
  ▪ Discussion
  ▪ Conclusion
Modifications to MPI

- **Process Management Interface (PMI):**
  - It needs to be aware of the fact that TPSL constitutes multiple MPI ranks in a single OS process
  - Hydra: instead of PMI_RANK and PMI_FD, it passes PMI_RANKS and PMI_FDS vectors communicating multiple ranks
  - MPI_Init() receives a thread ID which is the offset into the vectors

- **Infiniband RDMA registration cache:**
  - IB requires RDMA buffers to be registered and deregistered
  - MPI implementations usually provide their own heap allocator (e.g., MVAPICH uses ptmalloc) so that they can track free() and munmap(), which could implicitly require deregistering buffers
  - Heap manager puts data structures into the BSS, but that becomes thread private with TPSL
  - Some data structures had to be moved into the heap so that they remain global across threads

- These modifications are not targeting the essence of message passing
  - Rather, sort of infrastructural changes..
Usage Example

- **Initialization**
  - requires thread ID

- **Create derived data types**

- **Compute remote rank based on thread ID**

- **Do data exchange**

```c
#pragma omp parallel
{
    MPI_Init(&argc, &argv, omp_get_thread_num());
}

MPI_Datatype sub_xz;
int sub_xz_size[3];
int sub_xz_start[3];
int xz_target;
...

#pragma omp parallel private(xz_target,
                           sub_xz_size, sub_xz_start)
{
    /* Subarray type creation */
    sub_xz_size[0] = Z_SIZE / omp_get_num_threads();
    sub_xz_size[1] = 2;
    sub_xz_size[2] = X_SIZE;
    sub_xz_start[0] = omp_thread_id *
                     (Z_SIZE / omp_get_num_threads());
    sub_xz_start[1] = 0;
    sub_xz_start[2] = 0;

    MPI_Type_create_subarray(3, sizes,
                             sub_xz_size, sub_xz_start,
                             MPI_ORDER_C, MPI_DOUBLE, &sub_xz);

    MPI_Type_commit(&sub_xz);

    xz_target = (rank + omp_get_num_threads())
                % num_ranks;

    /* Main loop */
    for (iter = 0; iter < NR_IERS; ++iter) {
        /* Computation */
        ...

        /* HALO exchange */
        MPI_Isend(data, 1, sub_xz, xz_target, ...);
        ...
    }
}
Agenda

- Motivation
- Background
- Thread private shared library (TPSL)
- Modifications to MPI
  - Evaluation
  - Discussion
  - Conclusion
Evaluation: Node Configuration

- **Intel Xeon Ivy Bridge (E5-2670 v2 @ 2.50GHz) CPUs**
  - Two sockets, ten cores per socket, 2 HW threads per core = 40 HW threads per node

- **64GB RAM, 2 NUMA domains**
  - All experiments were restricted to NUMA node 0
  - Same set of CPU cores both for Linux and McKernel

- **Mellanox Infiniband QDR (MT27500 ConnectX-3)**

- **Two MPI distributions:**
  - MPICH 3.1.3
    - Infiniband netmod developed by Masamichi Takagi while visiting ANL
  - MVAPICH 2.1
OSU latency and bandwidth benchmarks extended to support multithreaded measurements

Flat MPI messaging latency doesn’t change substantially with growing number of process pairs

Multithreaded performance is miserable, 2 orders of magnitude slower when running 16 threads

TPSL mapped MPI multithreaded performance → same as flat MPI!
Evaluation: Latency, Message Rate and Bandwidth

MVAPICH 2.1 Message Rate

- OSU latency and bandwidth benchmarks extended to support multithreaded measurements
- Flat MPI messaging rate increases substantially with growing number of process pairs
- Multithreaded performance becomes an order of magnitude slower with the growing number of threads

TPSL mapped MPI multithreaded performance → same as flat MPI!
Evaluation: Latency, Message Rate and Bandwidth

MVAPICH 2.1 Bandwidth

- OSU latency and bandwidth benchmarks extended to support multithreaded measurements
- Flat MPI bandwidth increases with growing number of process pairs (for small buffers)
- Multithreaded performance is an order of magnitude slower for 16 threads

TPSL mapped MPI multithreaded performance → same as flat MPI!
Evaluation: Latency, Message Rate and Bandwidth

**MPICH 3.1.3**

IB netmod not optimized for multithreaded execution..?

**Flat MPI (Linux)**

**Multithreaded MPI (Linux)**

**TPSL MPI (IHK/McKernel)**
Evaluation: Derived Datatype HALO Exchange

- Derived data types for exchanging HALO data of a three dimensional array of doubles (using subarray)

- Three shapes:
  - Cube: 512 x 512 x 512
  - LargeX: 16k x 128 x 64
  - LargeY: 128 x 16k x 64
  - LargeZ: 128 x 64 x 16k

- X-Y plane is contiguously in memory

- X-Z, Y-Z planes using 1 to 16 threads with TPSL ranks

for X-Z plane large Z dimension implies a lot of small data chunks scattered non-contiguously in memory
Evaluation: Derived Datatype HALO Exchange

- Derived data types for exchanging HALO data of a three dimensional array of doubles (using subarray)

- Three shapes:
  - Cube: 512 x 512 x 512
  - LargeX: 16k x 128 x 64
  - LargeY: 128 x 16k x 64
  - LargeZ: 128 x 64 x 16k

- X-Y plane is contiguous in memory

- X-Z, Y-Z planes using 1 to 16 threads with TPSL ranks

depending on the shape, increasing the number of threads translates to lower performance...
Agenda

- Motivation
- Background
- Thread private shared library (TPSL)
- Modifications to MPI
- Evaluation
  - Discussion
  - Conclusion
Discussion: Limitations

- Helper threads
  - TPSL cannot distinguish whether or not a pthread_create() (i.e., clone() syscall) is supposed to share address space with parent, they all are separated now
    - a new flag to clone() could indicate this

- Memory consumption
  - Extra memory for page tables is required
    - Only a designated part of the address space is separated, rest share mappings
  - Library is mapped with COW
  - MPI internal buffers are duplicated
    - Could MPI be aware of TPSL? Would that require lot of changes?..

- TLB contention
  - Although TLB is HW thread private, some architectures may share higher level TLB caches
  - TPSL increases contention on those resources

- OpenMP
  - The OpenMP standard doesn’t require to map threads to the same thread_id across different parallel regions (although it normally does)
  - Embedded loops may be a problem
Conclusion and Future Work

- Large number of CPU cores and modern interconnects favor multiple cores to drive the network simultaneously
- Scalable multithreaded message passing is required!

- Proposed “thread private shared library”, a new OS concept that helps eliminate contention on MPI internals in multithreaded code
- Orders of magnitude better messaging performance than current multithreaded MPI
- Requires minimal changes to the library itself
  - Demonstrated on MPICH and MVAPICH

- Future directions:
  - Evaluate on application code
  - Apply TPSL to OpenMPI
  - “Co-design” libraries with TPSL like solutions?
Thank you for your attention!
Questions?

Special acknowledgement:
Rolf Riesen, Dave Van Dresser, Evan Powers @ Intel mOS Team