POSTER: a Generic Framework for Asynchronous Progression and Multithreaded Communications

Alexandre DENIS
Inria Bordeaux – Sud-Ouest, France, e-mail: Alexandre.Denis@inria.fr

Abstract—Recent cluster architectures include dozens of cores per node, with all cores sharing the network resources. To program such architectures, hybrid models mixing MPI+threads, and in particular MPI+OpenMP are gaining popularity. This imposes new requirements on communication libraries, such as the need for MPI_THREAD_MULTIPLE level of multi-threading support. Moreover, the high number of cores brings new opportunities to parallelize communication libraries, so as to have proper background progression of communication and computation/computation overlap. In this paper, we present pioman, a generic framework to be used by MPI implementations, that brings seamless asynchronous progression of communication by opportunistically using available cores. It uses system threads and thus is composable with any runtime system used for multithreading. Through various benchmarks, we demonstrate that our pioman-based MPI implementation exhibits very good properties regarding overlap, progression, and multithreading, and outperforms state-of-art MPI implementations.

I. INTRODUCTION

With the dramatic increase in the number of cores per node in clusters, communication libraries have to deal with multithreading, and may exploit cores to make communication progress. However, mixing threads and communication is not straightforward, and care must be taken to design a thread-aware communication library.

In this paper, we present pioman, a generic framework to be used by MPI implementations, that brings seamless asynchronous progression of communication by opportunistically using available cores. It uses system threads and thus is composable with any runtime system used for multithreading. Through various benchmarks, we demonstrate that our pioman-based MPI implementation exhibits very good properties regarding overlap, progression, and multithreading.

II. RELATED WORKS

People have studied parallelism in the communication library, and shown it may be an opportunity to hide the cost of communications [1], [2], [3], [4]. OpenMPI [5] supports MPI_THREAD_MULTIPLE only on TCP, and can overlap computation and communication only on the sender side on InfiniBand. RDMA-based MPI [6], [7] may overlap some parts of transfers thanks to the hardware. MT-MPI [8] is specific to Xeon Phi and to a given OpenMP runtime. Some solutions use multithreading to make communication progress [9] which is very restrictive. Our own previous work [10], [11] lacked genercity and was bound to the Marcel thread scheduler.

III. A MULTITHREADED COMMUNICATION ENGINE

Tasklets in user-space: Parallelizing network communication processing is needed for asynchronous progression and for multithreaded application having their communication actually progress in parallel. Such mechanisms are well known in Linux kernel and are known as bottom half since kernel 2.3.x series. They include tasklets, small tasks to be executed asynchronously at some time later. The kernel ensures some guarantees on concurrent operations, deadline, and on CPU placement. Tasklets opportunistically utilize available resources, and asynchronously make communication progress independently of the application execution flow.

We propose a full rewrite of pioman [11] using system threads (pthreads), so as to be compatible with multithreaded applications, whatever the multithreading runtime or the compiler. Its light tasks are called ltasks, which are inspired from tasklets but not completely mimics their behavior since user-space and kernel-space are different contexts with different requirements. These ltasks need to be executed at the following polling points: idle core, for an opportunistic resource usage; timer to ensure guaranteed reactivity; and explicit polling, for a progression at least as efficient as the no-ltask flavor. Idle uses a low-priority thread, and timer uses a high-priority thread with sleeps.

Locality: To reduce contention, we take locality into account. Since architectures is hierarchical, ltask queues are hierarchical, as a tree of queues attached to entities (core, cache, socket). Tasks are submitted in the local queue. For polling, ltasks are dequeued and executed from the most local queue, then queues from parents are recursively dequeued up to the root. To reduce contention near the root, we perform the recursive polling on the parent queue with a frequency divided by the number of siblings, taking into account that multiple children object will contribute to the polling on their parent.

Contention-free locking scheme: Concurrent ltask enqueue by the communication library and ltask execution by another thread cause contention, depending on the locking scheme. For lock-based scheme (mutex, spinlock), threads compete to acquire the lock. For lock-free scheme, queue traversal is not possible atomically, so polling means dequeue/enqueue, and thus competes with applications threads. We propose submission queues: a companion queue dedicated to submission is attach to polling queues. The submission queue is lock-free: the main queue has a spinlock. Tasks from the submission queues are dequeued by polling threads before an ltask execution round, and enqueued in the main queue once the spinlock is already held. Readers and writers use separate structures, changes from writers are incorporated later by readers. This solution is lock-free for task submission, spin-free for polling (uses only trylock, no need to wait if someone else is already polling). Spinning on locks or atomics, and shared variable for writing are avoided. Contention is actually mitigated.
Our benchmarks are performed on a dual Xeon E5-2650@2.0GHz with IB ConnectX3 FDR. We compare our pioman-enabled NewMadeleine communication library against OpenMPI 1.7.4 and MVAPICH2 2.0b.

**Progression Benchmarks:** Figure 1 reports overlap ratio with computation on both sender and receiver side. We observe that pioman-enabled NewMadeleine perfectly overlaps computation and communication as soon as the computation time equals the communication time.

**Multithreaded Benchmarks:** To evaluate multithreaded performance, we consider 1 sender to N receivers, N senders to N receivers, and 1 sender to 1 receiver with N computation threads. Figure 2 shows that the pioman-enabled NewMadeleine behaves well with a large number of threads, while MVAPICH2 has its latency much more impacted by threads. OpenMPI was not considered in this benchmark since it does not support MPI_THREAD_MULTIPLE on IB.

Figure 3 shows that for single-threaded communication with competing computation threads, pioman-enabled NewMadeleine and MVAPICH2 have a constant median latency, while OpenMPI has a median latency linear with the number of computing threads.

pioman exhibits better progression properties than state-of-the-art MPI implementations.

### V. Conclusion

We have presented pioman, a generic framework to be used by communication libraries, that brings seamless asynchronous progression of communication. We have proposed mechanisms that make communication progress on timer events, opportunistically on idle cores, and allows explicit polling. Implementation uses system threads and thus is composable with any runtime system used for multithreading. We have studied tasks concurrency and proposed two mechanisms that mitigate contention, based on locality and an original locking scheme. We have shown that pioman makes actually communication progress in background, thus allowing computation and communication to overlap. We have shown that it handles multithreaded load and does not collapse with massive number of threads. In future works, we plan to modify MPI applications to actually take benefit from multithreading and to overlap computation and communications on both sides. Finally, we are working on porting it to the Intel Xeon Phi.

**REFERENCES**


