Offloading Partitioned Communication

Motivation and Background

Today's distributed memory machines are typically multiple interconnected many-core shared memory nodes. The most common programming model to progam distributed memory machines is MPI, whereas the most common programming model to program shared memory machines is OpenMP. One approach to use the full parallelism of these machines, and to guarantee overlap of communication with computation, is mixing MPI with OpenMP.

However, when a shared-memory programming model is used on top of MPI, the MPI library must be initialized with `MPI_THREAD_MULTIPLE` to provide full thread-safety, causing several performance-related issues, such as internal locking.

Recently, *Partitioned Communication* [1] was accepted to the MPI 4.0 standard to address some of these issues. Partitioned Communication is a form of *Persistent Communication*, which means that the communication pattern is static and can be reused, for example along multiple iterations of a loop. The communication buffer of partitioned communications is divided into multiple parts, each handled typically by a separate thread.

Another promising approach is *Software Offloading* [2], where one or more of the cores of the many-core node are dedicated only to communication. Compute threads forward their communication operations to a thread running on the dedicated communication core, which allows to avoid the internal locking and to guarantee progress in non-blocking communication for improved overlap of communication and computation.

Benefits

Performance matters! During this thesis you can:

- Gain hands-on experience in high-performance computing and computer architecture
- Have the opportunity to work with modern state-of-the-art hardware architectures
- Deepen your knowledge in the C/C++ programming languages and parallel programming

Goals and Tasks

We have developed a high-performance C++ library for software offloading at LMU named `libMMCSO` on top of MPI. However, Partitioned Communication, as well as the recently proposed Partitioned *Collective* Communication [3], are not implemented yet in `libMMCSO`. Additionally, the benefits of using Partitioned Communication with Offloading compared to other communication modes remain an open question.

In the context of this thesis, you will:

- Implement and integrate a prototype of the Partitioned Point-to-Point Communication primitives in `libMMCSO`
- Implement and integrate a prototype of the Partitioned Collective Communication primitives in `libMMCSO`
- Design and conduct experiments on a distributed memory machine (for example SuperMUC)
- Compare the performance of Partitioned Communication in `libMMCSO` with other communcation modes in micro-benchmarks and suitable use-cases

Note: The planned implementation should base on MPI 3.0 primitives, for example one-sided MPI 3.0 RMA. This provides the opportunity of extending MPI 3.0 implementations with Partitioned Communication using `libMMCSO` as a plugin.

Prerequisites

- Basic understanding and interest in distributed memory and shared memory programming
- Basic knowledge in high-performance and parallel computing (*Parallel and High-Performance Computing* or equivalent)
- Proficiency in Linux, and C/C++ (*Systempraktikum* or equivalent)

Starting Point Literature

- [1] Dosanjh et al. 2021 Implementation and evaluation of MPI 4.0 partitioned communication libraries
- [2] Vaidyanathan et al. 2015 Improving concurrency and asynchrony in multithreaded MPI applications using software offloading
- [3] Holmes et al. 2021 Partitioned Collective Communication
- [4] MPI Forum Partitioned Communication Examples

Organization

- Task setter: Prof. Dr. D. Kranzlmüller
- Master's thesis duration: 6 Month
- Students: 1
- Supervisors: Sergej Breiter and Dr. Karl Fürlinger
- Language: English