Optimizing MPI Implementation on Massively Parallel Many-Core Architectures

Min Si*, Yutaka Ishikawa (Advisor)*, Pavan Balaji (Co-advisor)†

*Department of Computer Science, University of Tokyo, Tokyo, Japan, {msi@il.is.s, ishikawa@is.s}.u-tokyo.ac.jp
†Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, USA, balaji@mcs.anl.gov

I. Introduction

While multicore processor chips are the norm today, architectures such as the Intel Xeon Phi take such chips to a new level of parallelism, with dozens of cores and hundreds of hardware threads. To efficiently utilize such architectures, application programmers are increasingly looking at hybrid programming models comprising a mixture of processes and threads. In such models, one or more threads utilize a distributed-memory programming system, such as MPI, for their data communication.

My doctoral research focuses on exploiting the capabilities of these architectures on widely used MPI implementations. The broad vision is to investigate the characteristics of MPI on such massively threaded architectures and develop various design strategies that allow MPI implementations to perform efficiently. While my initial work is performed on the Intel Xeon Phi, we believe that the research theme as well as our designs are valid for a large variety of future architectures.

II. Research Plan

My doctoral research plan is broadly divided into three related pieces.

A. Internal Multithreading in MPI

A common mode of operation for hybrid MPI+threads applications involves using multiple threads to parallelize the computation, while one of the threads issues MPI operations. This mode often results in a model where most threads are idle during MPI calls. In our research, we plan to investigate techniques that allow MPI to share idle threads from user applications. We also plan to utilize lightweight user-level threads, to allow more flexible scheduling of MPI tasks without oversubscribing the number of threads.

B. Fine-Grained Consistency Management in MPI

An increasing number of applications are utilizing MPI in a multithreaded mode in which multiple threads can simultaneously make MPI calls. On massively parallel systems, this model can significantly reduce performance because the MPI implementation has to perform various locks and atomic operations in order to keep its internal state consistent. We plan to investigate various techniques that allow applications to request for a more restricted semantics for sharing of resources between threads, thus allowing the MPI implementation to partition resources to reduce contention.

C. MPI Tasklets

Some MPI applications still follow an MPI-only mode, in which an MPI process is launched for each available core. Such a model suffers not only from the partitioning of resources but also from loss of performance in a number of ways. In our research, we plan to study the concept of MPI tasklets, in which a single OS process can contain a large number of MPI processes, as user-level threads, for example.

III. Current Status of Research

My research has so far focused on the first technique (Section II-A). Specifically, in order to avoid thread idleness during MPI calls, we reuse the idle cores inside MPI when the user application is not using them.

In our initial work, we parallelized the MPI functionality used in stencil computation routines (specifically, noncontiguous data movement), as a case study. Preliminary results show that the parallel version provides up to a 40-fold speedup in MPI performance when parallelized across the entire Intel Xeon Phi board (Figure 1), compared with what the original application could achieve. We are also planning to implement parallelism inside one-sided MPI synchronization functions, an approach that is expected to improve the performance of many data-intensive applications.

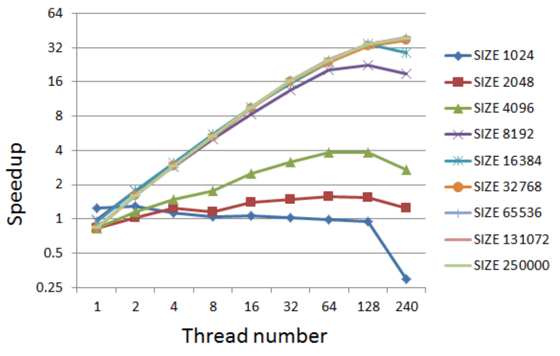

Figure 1. Speedup of halo exchange between 2 Intel Xeon Phi coprocessors.