Automatic Transformation and Analysis Tool for Improving Legacy MPI Applications

Hadia Ahmed, Peter Pirkelbauer, and Anthony Skjellum
Department of Computer and Information Sciences, University of Alabama at Birmingham

Motivation

• Devise a tool that will allow automatic transformation (and/or semi-automatic, with minimal user intervention) of MPI programs to have the best MPI operations available at constant portability on newer systems with efficient MPI implementations.

• Goals
  – Reveal equivalent portable MPI programs
  – Gain more access of the underlying potential of high performance architectures
  – Reduce jitter associated by avoiding underlying polling

Introduction

• Legacy MPI applications are an important and economically valuable category of parallel software that rely on the MPI-1, MPI-2 (and, more recently, MPI-3) [1].
• Many of them have been developed or ported to MPI to achieve
  – high performance and scalability;
  – a high level of portability between diverse parallel architectures.

• However, they were often created using MPI in ways that exploited how a particular underlying MPI behaved at the time (such as those with polling progress or slow persistent operations).

• They did not take advantage of
  – describing latent concurrency
  – loosening the coupling of the application thread from the message scheduling and transfer.

Methodology

Preliminary Results

MPI Blocking Code

if ( taskid == 0 )
{
  ierr = MPI_Recv (recvbuff, buffsize, MPI_DOUBLE, ntasks-1,
                   MPI_ANY_TAG, MPI_COMM_WORLD, &status);
  recvtime = MPI_Wtime ();
  ierr = MPI_Send (sendbuff, buffsize, MPI_DOUBLE, taskid+1, 0,
                   MPI_COMM_WORLD);
}

Transformed MPI Non-Blocking

if ( taskid == 0 )
{
  MPI_Request reqs[2];
  MPI_Status status[2];
  ierr = MPI_Irecv (recvbuff, buffsize, MPI_DOUBLE, ntasks-1,
                    MPI_ANY_TAG, MPI_COMM_WORLD, &status);
  ierr = MPI_Isend (sendbuff, buffsize, MPI_DOUBLE, taskid+1, 0,
                    MPI_COMM_WORLD, &status);
  recvtime = MPI_Wtime ();
  MPI_Waitall (2, reqs, &status);
}

References


Future Research

• Add persistent communication to minimize overhead associated with repeatedly transferring data between the same end points.
• Add fault-tolerant code (FA-MPI and/or other techniques)
• Determine what is undecidable to be automatically transformed and how much developer intervention is needed to enable the desired transformations.

Discussion

• ROSE [3] source-to-source transformation tool is used to do the semantic checking and implement code transformations.
• Adapting legacy MPI applications to work well on newer architectures and use newer functionality in MPI-3 and MPI-4 are of concern to users.
• An important class of applications was designed years ago; this valuable software has to continue working well on newer systems subject to newer requirements.
• A first goal is to transform blocking send/receives to non-blocking; and to use non-blocking collectives. This will reveal potential optimizations involving overlapping of communication and computation.
• Older MPI implementations were mostly polling, so applications didn’t see benefits from using non-blocking operations... they do now and will in future.
• These preliminary results show how this tool will yield a faster, more efficient, less error prone and effective means for code transformation compared to manual refactoring.