Assignment 3: MPI Point-to-Point and One-Sided Communication

Isaías A. Comprés Ureña
compresu@in.tum.de

Prof. Michael Gerndt
gerndt@in.tum.de

02-12-2016

1 Introduction

In the first assignment we focused on optimization through the use of compiler flags and pragmas, without modifying the source code in any significant way. In this and the final assignments we will work on optimization by modifying the parallel program. In this case, we will be optimizing the use of MPI point-to-point communication.

Point-to-point communication performance is of great importance in MPI applications. There are still many applications that are entirely implemented with this type of communication, where peers in a communicator exchange data directly. There are two models of direct peer-to-peer communication available in MPI: point-to-point and one-sided communication.

The point-to-point API consists of mainly send and receive operations. These operations have variants that are blocking, non-blocking, synchronous, ready and buffered. There are combined send and receive operations for swaps between peers. Additionally, there are wait and probe operations to check on the status of ongoing communication.

The one-sided API contains mainly put and get operations. These operations are all non-blocking with explicit transfers and synchronization. The processes need to create windows of memory that are then accessible by others with the put and get operations. In addition to these, there is also a collection of synchronization operations, such as locks and fences.

In this assignment, students will be provided a parallel MPI application that relies on blocking communication. The students will then need to transform it to use non-blocking point-to-point and one-sided communication. In both of these main tasks, the aim is to improve performance by improving MPI communication and allowing overlap of computation and communication.

1.1 Submission Instructions

Your third assignment submission for Programming of Supercomputers will consist of 2 parts:

- A 5 to 10 minute video with the required comments described in each task.
- A compressed tar archive with the required files described in each task.

2 Gaussian Elimination

Gaussian Elimination is a widely used technique used to solve systems of equations. To solve a linear system, transformations are applied so that the system matrix is transformed into an upper triangular form. After the matrix is in this form, back substitution is performed to find the solution vector.

An implementation is provided where the work is divided in equal parts among processes. This requires that the square matrix's size is divisible by the number of MPI processes evenly.

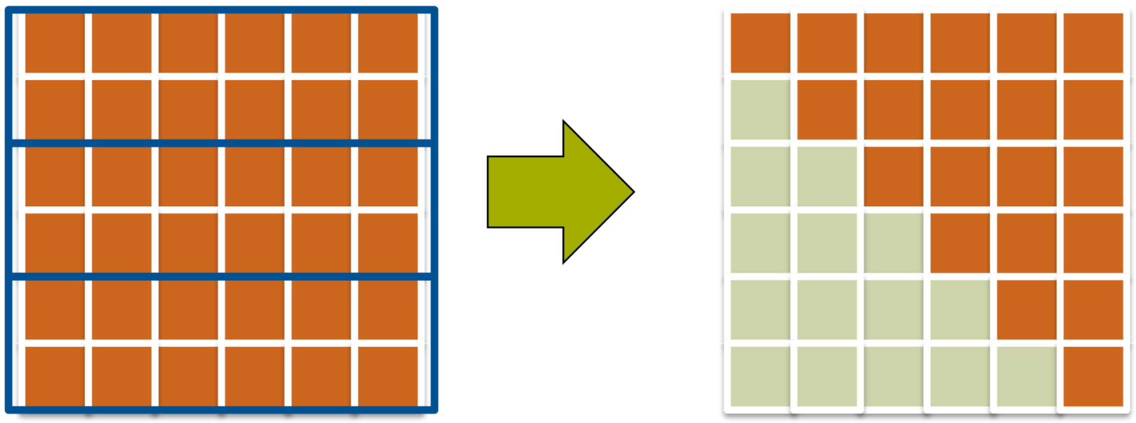

Figure 1: Upper triangular transformation with Gaussian Elimination

Figure 1 illustrates a 6x6 matrix that is partitioned across 3 processes, and then transformed into upper triangular for back substitution.

2.1 Provided MPI Implementation

You will be provided an implementation of the algorithm. This implementation relies on MPI blocking point-to-point communication and can therefore be run in distributed memory systems. The provided implementation takes as parameters the base name for the matrix and vector inputs. Input files are provided with the assignments' materials.

A Load-Leveler batch script is provided together with the implementation:

```
#!/bin/bash
#@ wall_clock_limit = 00:30:00
#@ job_name = pos-gauss-mpi-intel
#@ job_type = MPICH
#@ output = out_gauss_64_intel_$(jobid).out
#@ error = out_gauss_64_intel_$(jobid).out
#@ class = test
#@ node = 4
#@ total_tasks = 64
#@ node_usage = not_shared
#@ energy_policy_tag = gauss
#@ minimize_time_to_solution = yes
#@ notification = never
#@ island_count = 1
#@ queue

. /etc/profile
. /etc/profile.d/modules.sh
. $HOME/.bashrc

module unload mpi.ibm
module load mpi.intel

mpiexec -n 8 ./gauss ./gs_data/size64x64
date
mpiexec -n 16 ./gauss ./gs_data/size64x64
date
mpiexec -n 32 ./gauss ./gs_data/size64x64
date
...
```

In this assignment the tests will be performed with a maximum of 4 nodes and 64 MPI processes with the input files for sizes: 64x64, 512x512, 1024x1024, 2048x2048, 4096x4096 and 8192x8192. Make sure to update the script accordingly. You can use bash scripting for automation in the batch script, and later for postprocessing. For statistics, you can use the R language and infrastructure, or general mathematics software such as MatLab.

3 Provided Implementation and Baseline

During optimization activities, it is important to understand the problem that is being optimized and to measure a performance baseline. Take a look at the provided Load-Leveler script and modify it so that good measurements can be collected (hint: think of variance between runs) to determine an accurate application's performance baseline. Please note that the application already reports compute and MPI times for each participating process.

For these last assignments we will be working on Sandy Bridge nodes with the Intel MPI library and Intel compilers. Make sure that you have the correct modules at all times and that you are logged into the correct login nodes:

```
sb.supermuc.lrz.de
```

Copy the provided materials to your SuperMUC account and perform the following steps:

1. Make sure that you load the Intel MPI and Intel compiler modules.
2. Build the provided Gaussian Elimination implementation.
3. Run the updated Load-Leveler batch script.
4. Collect and plot the measured IO, setup, compute, MPI and total times.

3.1 Required Submission Files

Submit the updated Load-Leveler batch script and the performance plots. Provide the plots in PDF format.

3.2 Required Video Commentary

Please discuss the following topics in the video:

1. Gaussian Elimination and the provided implementation.
2. Changes applied to the provided Load-Leveler batch script.
3. Challenges in getting an accurate baseline time for Gaussian Elimination.
4. Compute and MPI times scalability with fixed process counts and varying size of input files.
5. Compute and MPI times scalability with fixed input sets and varying process counts.

4 MPI Point-to-Point Communication

After understanding the performance and scalability of the provided implementation and measuring a baseline for performance, we are now ready to optimize the application. In this task, the optimization will be based on converting blocking to non-blocking point-to-point communication. The aim is to reduce communication time as well as allow for the overlap of computation and communication.

Perform the following activities as part of this task:

1. Study the set of non-blocking operations in MPI's point-to-point API.
2. Select which operations to use in the benchmark.
3. Convert the communication parts of the application.
4. Analise the optimized version by following the steps in 3
5. Generate new plots with the new optimized binary.

4.1 Required Submission Files

Submit the updated `gauss.c` file and new performance plots (in PDF format).

4.2 Required Video Commentary

Answer the following questions in the video:

1. Which non-blocking operations were used?
2. Was communication and computation overlap achieved?
3. Was a speedup observed versus the baseline?

5 MPI One-Sided Communication

We continue our optimization efforts now with a conversion from blocking point-to-point to one-sided communication. The aim is again to reduce communication time and, if possible, to allow for computation and communication overlap. Additionally, one sided communication should in theory reduce synchronization costs and intermediate buffering.

Perform the following activities as part of this task:

1. Study the set of one-sided operations in MPI.
2. Select which operations to use in the benchmark.
3. Convert the communication parts of the application.
4. Analise the optimized version by following the steps in 3
5. Generate new plots with the new optimized binary.

5.1 Required Submission Files

Submit the updated `gauss.c` file and new performance plots (in PDF format).

5.2 Required Video Commentary

Answer the following questions in the video:

1. Which one-sided operations were used?
2. Was communication and computation overlap achieved?
3. Was a speedup observed versus the baseline?
4. Was a speedup observed versus the non-blocking version?