2010

Факультет вычислительной математики и кибернетики
427 группа

Сухов Иван Николаевич

Отчет по учебному курсу «Распределенные системы»

Москва, 2022

Содержание

1 Постановка задачи

Требуется сделать следующее:

- Реализовать операцию редукции **MPI_MAXLOC**, определить глобальный максимум и соответствующих ему индексов на транспьютерной матрице 4*4 при помощи пересылок **MPI** типа точка-точка.
- Доработать **MPI**-программу, реализованную в рамках курса Суперкомпьютеры и параллельная обработка данных". Добавить контрольные точки для продолжения работы программы в случае сбоя.

Для продолжения работы программы после сбоя использована следующая стратегия: при запуске программы на счет сразу запускается некоторое дополнительное количество **MPI**-процессов, которые используются в случае сбоя.

После реализации операции **MPI_MAXLOC** необходимо оценить сколько времени потребуется для её выполнения, если все процессы выдали эту операцию редукции одновременно. Время старта равно 100, время передачи байта равно 1 (Ts=100,Tb=1). Процессорные операции, включая чтение из памяти и запись в память, считаются бесконечно быстрыми

2 Реализация операции редукции и оценка ее сложности

В транспьютерной матрице размером 4*4, в каждом узле которой находится один процесс, необходимо выполнить операцию нахождения максимума среди 16 чисел (каждый процесс имеет свое число). Найденное максимальное значение и координаты первого процесса с этим значением должны быть получены каждым процессом.

Минимальное время оценивается через минимальное расстояние между двумя самыми дальними процессами в матрице. В нашем случае, чтобы пройти от процесса с координатами (0, 0) к процессу с координатами (3, 3), необходимо сделать 6 шагов. Это количество шагов является минимальным, так как есть алгоритм, реализующий операцию нахождения максимума за 6 шагов. Один из способов пересылки (который был реализован) показан на рисунке 1:

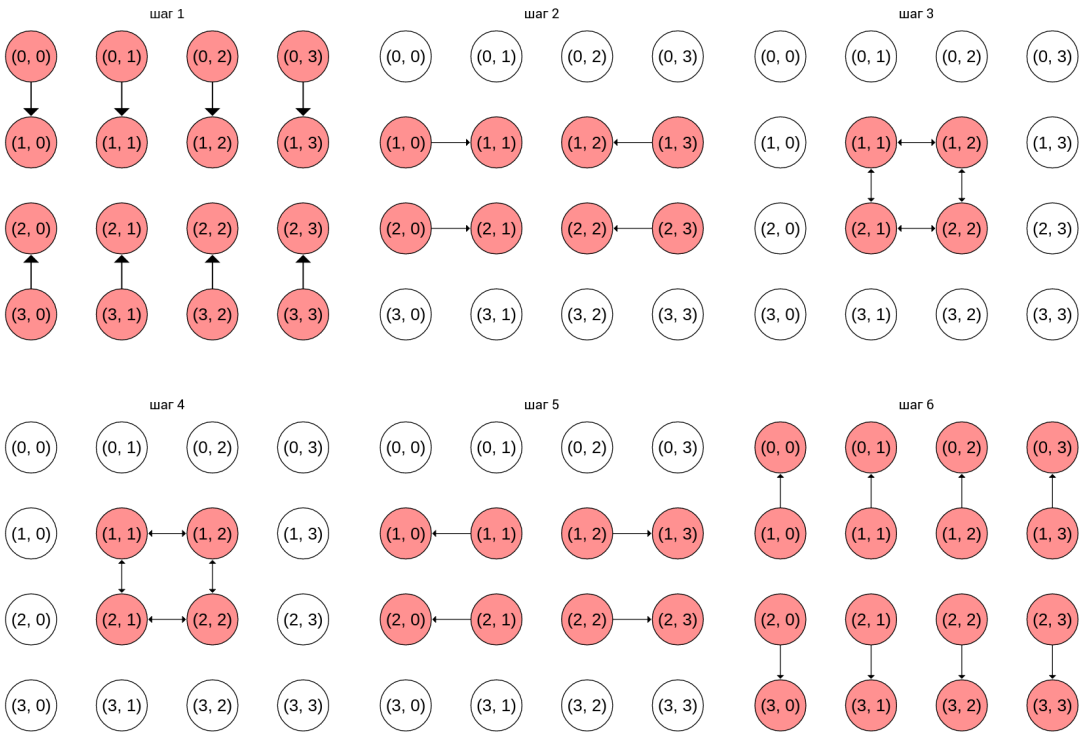

Рис. 1: Алгоритм нахождения максимума на транспьютерной матрице

Данный алгоритм был реализован с помощью функций **MPI_Isend** и **MPI_Recv**.

Получение топологии в виде транспьютерной матрицы произведено с помощью функции **MPI_Cart_rank**.

Оценим время работы алгоритма. Если время старта равно 100, время передачи байта равно 1 (Ts=100,Tb=1), то время выполнения операции рассчитывается следующим образом:

$$time = num\_steps \cdot (Ts + n \cdot Tb)$$

где n - размер передаваемого сообщения в байтах. В нашем случае сообщением является 2 числа, суммарный размер которых равен 8 байтам.

Таким образом, при n = 8, получаем:

$$time = 6 \cdot (100 + 8 \cdot 1) = 648$$

3 Добавление в программу возможности её продолжения в случае сбоя

Для того, чтобы при сбое одного из процессов программа не завершалась с ошибкой, а продолжала своё выполнение, необходимо написать обработчик ошибок, который будет срабатывать в таких ситуациях. Для этого в стандарте **MPI** существует специальные функции **MPI_Comm_create_errhandler** и **MPI_Comm_set_errhandler**. Однако стандарт не позволяет определить, в каком именно процессе произошла ошибка. Это можно сделать, используя расширенный функционал библиотеки **mpich**.

Программа была доработана следующим образом:

1. С помощью фунцкций **MPI_Comm_create_errhandler** и **MPI_Comm_set_errhandler** добавлен обработчик ошибок **err_handler**, о реализации которого будет сказано ниже.

2. В качестве резервного процесса используется последний процесс.

3. Далее следует не больше чем **MAT_IT** итераций цикла, на каждой итерации соседние процессы обмениваются данными друг с другом. В начале каждой итерации процессы считывают данные из резервного массива, используемого в реализации, работают с ними и в конце снова записывают в резервный массив. Если произошла какая-то ошибка (это выясняется с помощью специального флага), то все процессы начинают данный процесс сначала.

4. Для того, чтобы процессы находились на итерациях с одинаковым номером, были расставлены «барьеры» с помощью функции **MPI_Barrier**.

5. В обработчике ошибок на базе старого коммуникатора создается новый, не включающий в себя вышедшие из строя процессы (в нашем случае один процесс). Это делается с помощью функции, не входящей в стандарт **MPI**, **MPIX_Comm_shrink**.

6. После создания нового коммуникатора, каждый процесс получает новый номер и возможно работает с другими данными, однако это не влияет на результат работы программы.

7. Работа программы продолжается на оставшихся процессах и результат собирается на процессе с номером 0 с помощью функции **MPI_Gather**.

4 Заключение

Таким образом, была реализована операция **MPI_Reduce** на транспьютерной матрице при помощи пересылок **MPI** типа точка-точка и оценено время ее работы.

MPI-программа, разработанная в рамках работы прошлого года, была доработана так, чтобы ее работа продолжалась после выхода из строя одного из процессов. Для этого один из процессов изначально считается резервным и используется в случае необходимости.

Код обоих программ доступен на сайте.