Московский государственный университет имени
М.В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра автоматизации систем вычислительных комплексов

Отчет о выполнении практического задания по курсу “Распределенные системы”

Коваленко Анастасия Павловна
421 группа

Москва
2021

Постановка задачи:

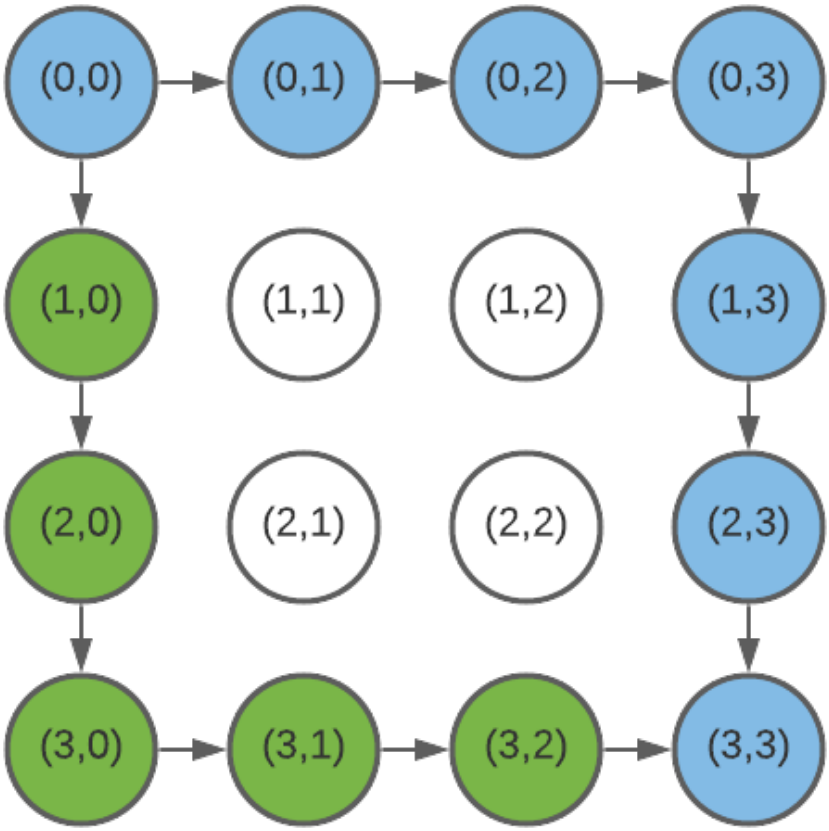
I. В транспьютерной матрице размером 4*4, в каждом узле которой находится один процесс, необходимо переслать очень длинное сообщение (длиной L байт) из узла с координатами (0,0) в узел с координатами (3,3). Реализовать программу, моделирующую выполнение такой пересылки на транспьютерной матрице с использованием режима готовности для передачи сообщений MPI. Получить временную оценку работы алгоритма, если 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.

II. Доработать MPI-программу, считающую определенный интеграл методом Симпсона, которая была реализована в рамках курса “Суперкомпьютеры и параллельная обработка данных”. Реализовать один из 3-х сценариев работы после сбоя: а) продолжить работу программы только на “исправных” процессах; б) вместо процессов, вышедших из строя, создать новые MPI-процессы, которые необходимо использовать для продолжения расчетов; в) при запуске программы на счет сразу запустить некоторое дополнительное количество MPI-процессов, которые использовать в случае сбоя.

I. Пересылка длинного сообщения в транспьютерной матрице

Требуется реализовать пересылку длинного сообщения (длиной L байт) из узла с координатами (0,0) в узел с координатами (3,3). В данном варианте задачи при отправке сообщений используется режим готовности. То есть операция send может быть выдана только после выдачи соответствующей операции receive, иначе программа считается ошибочной и результат ее работы не определен. В коде этот подход реализован с помощью функций MPI_Irsend и MPI_Irecv библиотеки MPI.

Для оптимизации операции будем осуществлять пересылку по двум путям, передавая по каждому пути половину сообщения. Маршрут пересылки сообщений представлен на схеме ниже.



Отправка одного сообщения между соседними процессами занимает t = Ts + L/2 * Tb. На каждом маршруте получается по 6 пересылок половины сообщения. В программе используются неблокирующие операции, поэтому можем считать, что всего происходит 6 параллельных пересылок. То есть T = 6 * t = 6 * (Ts + L/2 * Tb).

При значениях Ts = 100 и Tb = 1 общее время работы алгоритма получается T = 6 * (100 + L/2 * 1) = 600 + 300 * L.

II. Добавление обработки сбоя MPI-программы

В рамках курса “Суперкомпьютеры и параллельная обработка данных” была реализована программа, вычисляющая определенный интеграл методом Симпсона. Требуется добавить обработку ситуации, когда один из процессов выходит из строя. Для этого в стандарте MPI есть возможность задать обработчик ошибок с помощью функций MPI_Comm_create_errhandler и MPI_Comm_set_errhandler. Однако для того, чтобы определить, в каком именно процессе произошла ошибка и задать дальнейшее поведение программы, необходимо использовать расширение ULFM.

В данной реализации было принято решение продолжить выполнение программы на оставшихся исправных процессах. Для этого с помощью функций MPIX_Comm_failure_ack и MPIX_Comm_failure_get_acked находится номер неисправного процесса и с помощью функции MPIX_Comm_shrink он удаляется из коммуникатора.

В программе расставлены контрольные точки: количество витков цикла, приходящееся на каждый процесс, делится на количество контрольных точек(число посчитанных витков между каждой парой контрольных точек хранится в переменной part), и после каждых part вычислений промежуточное значение записывается в файл с именем “’номер процесса’_’номер контрольной точки’”.

Предполагается, что процесс с номером 0 не подвержен сбоям, поэтому после того, как все процессы вычислили свои значения и передали свои результаты процессу 0, этот процесс досчитывает то, что не успел досчитать убитый процесс. Он находит последнюю контрольную точку, считывает из файла последнее корректное значение, посчитанное убитым процессом, и вызывает функцию recovery_integral, в которой продолжается вычисление. Затем полученный результат добавляется к тому, что посчитали остальные процессы, это и есть итоговый ответ.