Московский Государственный Университет им. М.В. Ломоносова
Факультет Вычислительной Математики и Кибернетики
Кафедра Автоматизации Систем Вычислительных Комплексов

Задание по курсу "Распределённые системы"

Отчёт

Выполнил:
Звонов Андрей Денисович
421 группа

Москва, 2020 год

Содержание

1. Постановка задач

1.1. Задача 1: моделирование

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.

1.2. Задача 2: надёжность

Доработать MPI-программу, реализованную в рамках курса "Суперкомпьютеры и параллельная обработка данных". Добавить контрольные точки для продолжения работы программы в случае сбоя. 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.

В данной задаче был выбран вариант "в".

2. Пояснения к решениям

2.1. Моделирование

В транспьютерной матрице $4 * 4$ необходимо переслать сообщение L байт из $(0, 0)$ в $(3, 3)$. Ясно, что расстояние, на которое необходимо переслать данные - 6, т.е. 6 раз переслать даные от процессора к процесору. В дальнейшем при расчёте временной оценки будем считать, что во время работы алгоритма не произойдёт никаких ошибок; в частности, при вызове MPI_Rsend на процессоре-отправителе на соответствующем процессе получателе уже был запущен MPI_Recv, то есть пересылка пройдёт успешно.

Так как использовался режим передачи по готовности, до начала передачи на процессах, работающих на приём, было вызвано MPI_Recv, чтобы избежать неопределённого поведения.

В основе работы лежит следующая модель: сообщение длины L разбивается пополам, на 2 сообщения длиной $\frac{L}{2}$ каждое (на 2, так как у отправителя и получателя есть по 2 независимых друг от друга канала, которые являются в данном случае узким местом). Каждое из этих сообщений пересылается по непересекающимся маршрутам, проходя 6 пересылок.

Итак, необходимо 6 раз отправить сообщение длины $\frac{L}{2}$, что займёт $6 * Tb * \frac{L}{2}$. Кроме того, необходимо учесть и время старта $6 * Ts$. Итоговая временная сложность реализованного алгоритма: $6 * Ts + 3 * Tb * L$. Вообще говоря, можно отправлять сообщение не двумя цельными кусками, а дополнительно разбить на более мелкие блоки. За счёт такой конвейеризации, при разбиении сообщения на k более мелких, можно уменьшить сложность до $6 * Ts + 3 * Tb * \frac{L}{k}$. Однако, это не изменит полиномиальной временной сложности алгоритма, но логически усложнит сам алгоритм, поэтому конвейеризация в данной задаче реализована не была, но может быть добавлена в случае необходимости.

Временная сложность реализованного алгоритма — $6 * Ts + 3 * Tb * L$

2.2. Надёжность

Программа, написанная в рамках курса СКиПОД решала задачу нахождения ранга матрицы. Так как на момент написания программы не было известно, что в дальнейшем потребуется её модификация, была выбрана не очень удобная для обеспечения отказоустойчивости стратегия: каждый процесс работал с "полосой"в матрице (некоторым множеством её строк), причём обмен, собственно, данными между процессами был минимизирован. Так, было решено отказаться от обмена матричными полосами, так как накладные расходы были настолько велики,

что параллельная программа работала в разы медленнее. Иными словами, ни на каком этапе работы (только в конце головной процесс собирает часть данных) процессы не имеют никакой информации о данных, над которыми работают другие процессы.

Ввиду этого, наиболее правильным решением показалось сохранение промежуточных результатов работы каждого процесса в отдельный файл. В контрольной точке в бинарный файл сохраняется состояние "полосы обрабатываемой процессом. Это необходимо и достаточно для того, чтобы, в случае падения процесса, другой (резервный) процесс мог продолжить работу вышедшего из строя.

При запуске должен быть задан параметр $-np\ n$, где $n \in N$. Если n — степень двойки, то резервные процессы не создаются, а в противном случае будет создано r резервных процессов, где r — разница между n и ближайшей степенью двойки. Для создания аварийной ситуации заранее выбранный процесс "кладётся"с помощью вызова *SIGKILL*.

После выхода процесса из строя обработчик ошибок "ужимает"коммуникатор с помощью *MPIX_ Comm_ shrink*, то есть оставляет в нём только живые процессы, которые после чтения данных из файла (не обязательно с той же самой полосой, над которой они работали) продолжат работу.

3. Исходный код

В рамках задания по курсу "Распределённые системы"были реализованы две параллельные программы, исходный код которых можно найти репозитории *https://github.com/zvonand/dis_ systems*. Кроме того, файлы с исходными текстами программ прикреплены в трекере на сайте *dvmh.keldysh.ru*.