Практикум по курсу
“Распределенные системы”

Передача сообщения на транспьютерной матрице с использованием буферизируемого режима передачи сообщений MPI

Добавление контрольных точек и возможности восстановления работоспособности программы после сбоя.

Отчет
о выполненном задании

Студента 427 учебной группы факультета ВМК МГУ
Федяшкина Максима Алексеевича

Москва, 2020 г

Оглавление:

Задача № 1

1 Постановка задачи

Требуется разработать и реализовать алгоритм передачи очень длинного сообщения на транспьютерной матрице (5x5) от процесса (0,0) процессу (4,4), с использованием буферизируемого режима 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.

2 Краткий обзор файлов решения и команд запуска

Код доступен по ссылке : https://github.com/max181199/mpi/tree/main/mpi_one
В папке лежит 2 файла :

1) *host.txt* (файл конфигураций для запуска 25 процессов)
2) *matrix.c* (файл с решением)

Для компиляции использовалась команды:

1) *mpicc ./matrix.c*
2) *mpirun -hostfile ./host.txt -c 25 a.out*

Разработка и тестирование производилось на MacOS

3 Описание работы алгоритма

Пронумеруем элементы транспьютерной матрицы слева-направо сверху-вниз

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

Разобьем исходное сообщение на пакеты, каждый пакет имеет известный размер, задаваемый пользователем

Например сообщение из 1000001 бит разобьется на 101 пакет размером 10000 бит.

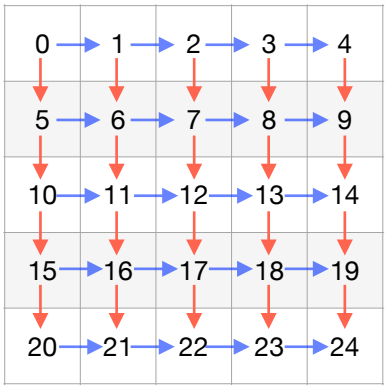
Левым соседом процесса n назовем процесс n-1, если n и n-1 расположены в одной строке иначе левым соседом n считается n.

Правым соседом процесса n назовем процесс n+1, если n и n+1 расположены в одной строке иначе правым соседом n считается n.

Верхним соседом процесса n назовем процесс n-5, если n и n-5 расположены в одном столбце иначе верхним соседом n считается процесс n.

Нижним соседом процесса n назовем процесс n+5, если n и n+5 расположены в одном столбце иначе нижним соседом n считается процесс n.

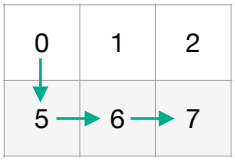
Каждый процесс получает данные от левого и верхнего соседа и отправляет их правому и нижнему (если соседи есть (номер соседа != номеру процесса)).

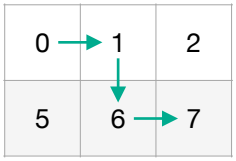


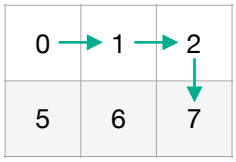
Введем понятия пути - путем называется 8 битный вектор (v) построенный следующим образом:

1) Изначально вектор равен 00000000 == 0
2) Для данных пришедших от левого соседа, то v = v << 1 + 1
3) Для данных пришедших от верхнего соседа, то v = v << 1 + 0

Например процесс с номером 6 достигают 2 пути 01 и 10, а процесс 7 три пути: 011,101, 110 (ниже примеры соответсвенно)







7 процесс оправит данные уже по 6 путям : 0110 0111 1010 1011 1100 1101

Данные распределяются следующим образом пусть с пути v1 пришло n пакетов, тогда по пути v2 = v1 << 1 + 1 отправится n / 2 + n % 2, а по пути v3 = v1 << 1 + 0 отправится n / 2 пакетов

При этом в процессах расположенных выше и на побочной диагонали данные разделяются, а на процессах ниже побочной диагонали собираются.

Передача пакетов по каждому пути реализована с помощью конвейера. Т.е. как только мы получили пакет от верхнего и левого (если они есть) соседей, передаем пакет сразу правому и нижнему соседу (если они есть). Процесс у которого нет нижнего и правого соседа собирает все пакеты у себя.

У каждого процесса есть буфер достаточный для хранения всех сообщений.

4 Расчет времени выполнения

Для подсчета времени достаточно подсчитать время выполнения каждого пути и найти среди них максимум (пути не зависимы друг от друга)

Пусть длина исходного сообщения L, тогда простая передача файла без конвейера по кратчайшему (8) пути займет : 800 + 8L, т.к. L очень большое, то будем считать, что займет 8L;

Для подсчета времени работы реализованного алгоритма потребуется посчитать время по всем путям и найти максимум, при этом длина всех путей одинакова, а размер и распределение пакетов одинаково для всех путей, как и время отправки. Поэтому найдем путь через, который суммарно проходит больше всего пакетов.

Пусть размер одного пакета равен M

Воспользуемся следующей матрицей:

(Пусть L является степенью двойки)

L/M	L/2M	L/4M	L/8M	L/16M
L/2M	L/2M	3L/8M	L/4M	3L/16M
L/4M	3L/8M	3L/8M	5L/16M	11L/32M
L/8M	L/4M	5L/16M	10L/32M	L/2M
L/16M	3L/16M	11L/32M	L/2M	L/M

Поскольку процессы выделенные красным являются узким горлом, то время потраченное на отправку данных равно :

7*(100+M) + L/2M*(100+M) == (100+M)(7+L/2M)

5 Выводы

1) Полученное решение позволяет ускорить программу
2) Решение является избыточным, поскольку достаточно было организовать всего 2 пути от 0 до 24 (с использованием конвейера)
3) Ускорение полученное до главной диагонали, полностью нивелируется узким горлышком (19 и 23 процесс)
4) В решении полностью достигнута поставленная задача
5) Преимущество решения в том, что если на процессах скорость передачи может замедляться (кроме 0 19 и 23), то всегда есть запасной путь, который позволит временно увеличивать скорость за счет отсутствия в нем замедления. Так можно получить ускорение если скорость отправки не постоянна.

Задача № 2

1 Постановка задачи

Необходимо доработать программу реализованную в рамках предыдущего курса, добавив контрольные точки и возможность восстановления работоспособности программы в случае сбоя.

2 Краткий обзор файлов решения и команд запуска

Код доступен по ссылке : https://github.com/max181199/mpi/tree/main/mpi_two
В папке лежит 2 файла :

1) *host.txt* (файл конфигураций для запуска 25 процессов)
2) second.*c* (файл с решением)

Для компиляции использовалась команды:

1) *mpicc ./second.c*
2) *mpirun -hostfile ./host.txt -c N a.out*

Разработка и тестирование производилось на MacOS

3 Описание работы алгоритма

Принцип работы самого алгоритма подробно был описан в рамках предыдущего курса.

При сбое программа пытается продолжить свое выполнения на оставшихся процессах, если во всех процессах произошел сбой программа завершает выполнение

Введем понятие состояния процесса - состоянием процесса называется глобальная переменная, которая принимает значение 0, если произошел сбой иначе 1.

Допущения :

1) Сбой может произойти только в определенной зоне
2) Процессы защищены от сбоев во время отправки и приема сообщений
3) Процессы со сбоем не участвуют в последующих расчетах, но могут отвечать на запросы о их состоянии
4) Финальная фаза сбора результатов вычисления со всех процессов считается безопасной от сбоев

Алгоритм можно разделить на 5 частей

1) Инициализация данных
2) Сохранение копий данных
3) Выполнение вычислений (место возможного сбоя)
4) Восстановление в случае сбоя
5) Сбор результатов и вывод ответа

Инициализация данных :

1) Инициализация матрицы, и инициализация ее копии
2) Инициализация worklist для каждого процесса
3) Инициализация массива для подсчета результата
4) Инициализация массива status
5) Инициализация переменной my_status
6) Инициализация дополнительных переменных

Изначально матрица и ее копия имеют одинаковые значения.

Worklist - вспомогательный массив, размерность которого соответствует кол-ву строк матрицы. Индекс массива - номер строки, а элемент массива - номер процесса обрабатывающего эту строку.
Изначально worklist[i] = i % process_count.

Массив для подсчета результата (minus) - имеет размерность равную кол-ву строк матриц. Изначально все элементы равны 0. Индекс номер столбца с максимальным элементом для строки лежащей в качестве значения по этому индексу.
status - массив по умолчанию заполнен единицами. Размерность массива равна кол-ву процессов. Индекс массива номер процесса. Элементом i

массива является либо 0, если в процессе с этим номером произошел сбой либо единица.

Сохранение копии данных:

1) Синхранизация текущего состояния матриц
2) Синхронизация текущего состояния массива minus
3) Запись в копию матрицы строк, которые могут быть изменены этим процессом

Сбой во время вычислений:

Сбоем во время вычислений называется ситуация при которой параметр my_status принимает значение 0.

Восстановление в случае сбоя:

1) Проверяется были ли сбои и кладем состояния процессов в массив status
2) Если сбоев не было, то ничего не меняем
3) Если были сбои выполняем следующие пункты
4) Отменяем изменения в массиве минус
5) Отменяем изменения в матрице используя данные сохраненные в копии матрицы
6) Пересчитываем worklist для всех процессов со статусом 1, при этом строки, которые принадлежали процессам со статусом 0 перераспределяются между процессами со статусом 1
7) Откатываемся к итерации цикла, которая была на момент сохранения

Подсчет результатов:

1) Выбирается случайный ROOT процесс со статусом 1 и собирает данные со всех процессов со статусом 1
2) Каждый процесс подсчитывает промежуточные результаты и отправляет ROOT процессу

Сохранения и восстановление данных очень трудозатратны, поэтому в программе предусмотрена возможность сохранения копий каждые N итераций по строкам матрицы (под итерациями понимается шаг при котором строка вычитается, из всех нижележащих)

4 Выводы

1) Задача сохранения и восстановления данных и продолжения работы была решена
2) Допущения сделаны из-за невозможности асинхронной обработки ошибок без использования ufml2
3) Сохранение и восстановление данных очень трудоемкий процесс, но это сглаживается возможностью уменьшения кол-ва точек сохранения