МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М. В. Ломоносова
Факультет вычислительной математики и кибернетики

Практикум по курсу «Распределенные системы»

студентки 428 группы
Семеновой Анны Леонидовны

Москва
2024

Содержание

Задание 1

Постановка задачи

Задание 103
В транспьютерной матрице размером 4*4, в каждом узле которой находится один процесс, необходимо выполнить операцию рассылки данных всем процессам от одного (MPI_SCATTERV) — от процесса с координатами (0,0). Каждый i-ый процесс должен получить i чисел (длинной 4 байта).

Реализовать программу, моделирующую выполнение операции MPI_SCATTERV на транспьютерной матрице при помощи пересылок MPI типа точка-точка. Получить временную оценку работы алгоритма. Оценить сколько времени потребуется для выполнения операции MPI_SCATTERV, если все процессы выдали ее 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.

Алгоритм

В начале нулевой процесс ((0,0) - элемент транспьютерной матрицы) все сообщения, предназначенные 1-15 процессу, в порядке убывание номеров процессов отправляет 1 или 4 процессу. Далее каждый из процессов выводит свое сообщение и распространяет сообщение, предназначенное для другого процесса.

Схема отправки сообщений между процессами представлена на графике на следующей странице. Если остаток от деления на 4 (то есть столбец матрицы) целевого процесса (для которого было отправлено сообщение), совпадает с номером процесса, который отправляет сообщение, то оно передается следующему в столбце процессу, иначе сообщение отправляется следующему в строке процессу.

Каждый процесс должен получить соответсвующее его номеру количество чисел - элементов массива, поэтому процессы должны заранее знать, буфер какого размера им необходимо будет принять на вход. В программе это реализовано с помощью функции MPI_Probe, которая вызывается в процессе-получатель и принимает на вход аргумент $status$, а передав его на вход функции MPI_Get_count можно получить размер буфера. Процесс-получатель выделяет нужное число байт - буфер для сообщения соответствующего размера и принимает последовательность чисел.

Таким образом, нулевой процесс отправляет 15 сообщений, процессы с 1 по 15 проксируют сообщения для других процессов и получают свое собственное сообщение.

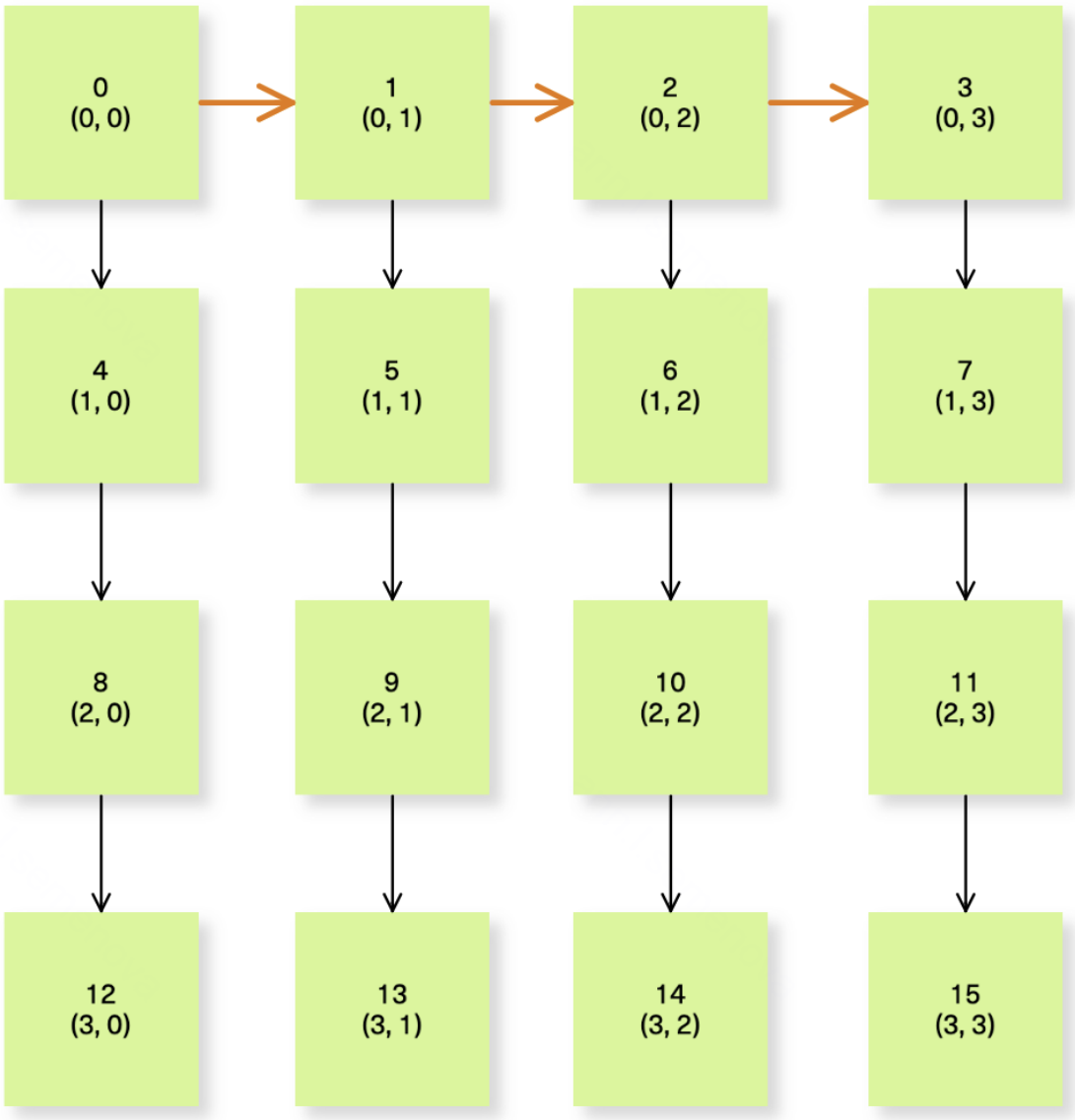

Схема отправки сообщений между процессами

Оценка времени работы алгоритма

Сообщение для 1-го, 2-го и 3-го процессов будут отправлены 1, 2 и 3 раза соответственно: $T_1+T_2+T_3 = (T_s+4*T_b)+2*(T_s+4*2*T_b)+3*(T_s+4*3*T_b) = 6T_s + 56T_b$.

Сообщение для 4-го, 8-го и 12-го процессов будут отправлены 1, 2 и 3 раза соответственно: $T_4+T_8+T_{12} = (T_s+4*4*T_b)+2*(T_s+4*8*T_b)+3*(T_s+4*12*T_b) = 6T_s + 224T_b$.

Сообщение для 5-го, 9-го и 13-го процессов будут отправлены 2, 3 и 4 раза соответственно: $T_5+T_9+T_{13} = 2*(T_s+4*5*T_b)+3*(T_s+4*9*T_b)+4*(T_s+4*13*T_b) = 9T_s + 356T_b$.

Сообщение для 6-го, 10-го и 14-го процессов будут отправлены 3, 4 и 5 раза соответственно: $T_6 + T_{10} + T_{14} = 3 * (T_s + 4 * 6 * T_b) + 4 * (T_s + 4 * 10 * T_b) + 5 *$

$(T_s + 4 * 14 * T_b) = 12T_s + 512T_b$.

Сообщение для 7-го, 11-го и 15-го процессов будут отправлены 4, 5 и 6 раза соответственно: $T_7 + T_{11} + T_{15} = 4 * (T_s + 4 * 7 * T_b) + 5 * (T_s + 4 * 11 * T_b) + 6 * (T_s + 4 * 15 * T_b) = 15T_s + 692T_b$.

Итого $T = 48T_s + 1840T_b = 6640$

Воспроизведение решения

Для компиляции программы необходимо использовать команду:

```
mpic++ -o task1 task1.cpp
```

Для запуска программы:

```
mpirun --oversubscribe -np 16 task1
```

В случае успешного завершения нулевой процесс выведет информацию об отправке всех сообщений, а остальные процессы должны вывести свой номер i и полученное сообщение - числа от 1 до i.

Задание 2

Постановка задачи

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:

- Продолжить работу программы только на “исправных” процессах;
- Вместо процессов, вышедших из строя, создать новые MPI-процессы, которые необходимо использовать для продолжения расчетов;
- При запуске программы на счет сразу запустить некоторое дополнительное количество MPI-процессов, которые использовать в случае сбоя.

Алгоритм

В случае сбоя программы был выбран третий сценарий - создание резервного процесса в начале работы. В программу прошлого года были добавлены:

- функция-обработчик ошибок $verbose\_errhandler$, в которой описана логика обработки в случае возникновения сбоя в одном из процессов. Если умирает процесс, то во всех остальных процессах управление переходит в $verbose\_errhandler$. Таким образом, после сбоя все процессы удаляют из своей рабочей группы мертвые процессы с помощью функции $MPIX\_Comm\_shrink$;
- функция чтения данных $data\_load$
- функция сохранения данных $data\_save$

На вход программе подается на 1 процесс больше, чем нужно для выполнение (дополнительный процесс для восстановления после сбоев). Запускается цикл перемножения матриц, на каждой итерацию для каждого процесса перемножаются матрицы A и B, а результат записывается в матрицу C. В файл data сохраняются все матрицы последнего процесса на данной итерации.

После сохранения данных проверяем итерацию алгоритма. Если номер процесса совпадает с заранее описанным в программе(в данном случае считаем, что номер такого процесса половине от общего числа), то убиваем последний процесс в процессорной решётке.

Когда оставшиеся в живых процессы доходят до барьера, они замечают потерю и вызывают функцию $verbose\_errhandler$. В ней реализована логика поиска упавшего процесса (он находится в другой группе). Затем вызывается функция $MPIX\_Comm\_shrink$, чтобы получить новый коммуникатор без мертвых процессов, а на его основе пересоздается $Cart$ с помощью функции MPI_Cart_create.

Для нового $Cart$ мы переопределяем номера процессов(упал последний следовательно новый просто получит его ранг). С помощью функции $load\_data$ этот новый процесс загружает сохраненные упавшем процессом данные.

Таким образом, новый процесс владеет данными и рангом упавшего процесса и может выполнять его задачи.

Оценка времени работы

Алгоритм был запущен на Macbook Air с процессором M2 и 16 GB оперативной памяти. Все измерения проводились с опцией –oversubscribe из-за нехватки мощности ноутбука. Каждое измерение проводилось 3 раза, в таблице отражены усредненные результаты времени выполнения.

Узлы, размер матрицы	500	1000	1500	2000
5	0.1683s	2.0896s	6.1060s	16.3353s
10	0.1408s	1.7359s	5.4454s	11.4455s
17	0.2317s	1.6854s	5.5092s	11.9001s
26	0.1600s	0.9326s	2.7783s	12.1309s

Таблица 1: Время работы программы

Воспроизведение решения

Для компиляции программы необходимо использовать команду:

```
mpic++ task2.c -o task2
```

Для запуска программы:

```
mpirun -v -np 5 --enable-recovery --with-ft=ulfm --oversubscribe ./task2 500
```