Московский государственный университет
имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики

Практикум по курсу «Распределенные системы»

Алгоритм MPI_Scatter для транспьютерной матрицы
Надежный алгоритм для задачи Gemver

ОТЧЕТ
о выполненном задании

Студента 420 учебной группы факультета ВМК МГУ
Попова Алексея Павловича

Москва, 2020

Оглавление

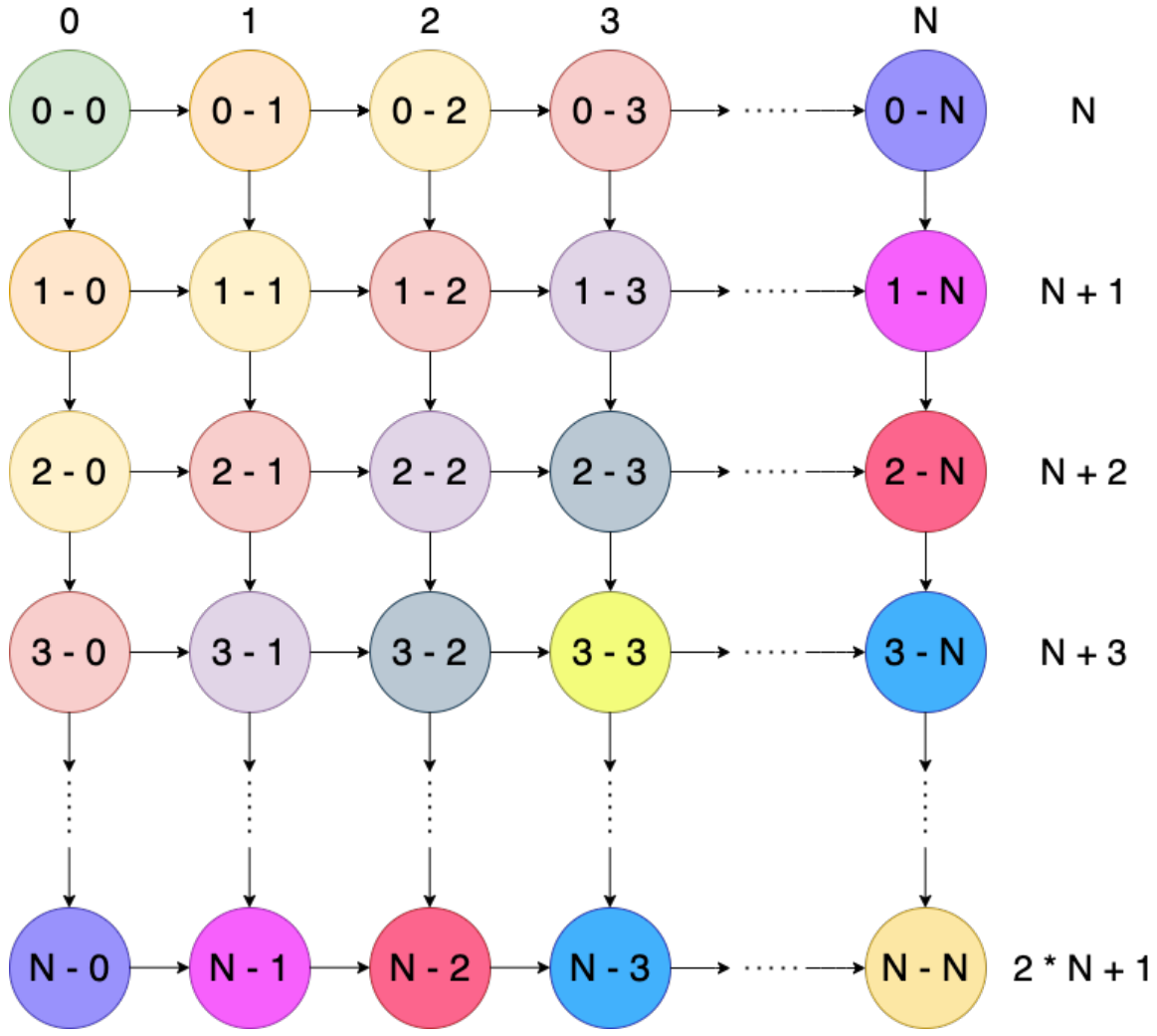
Постановка задачи

1) Необходимо промоделировать алгоритм *MPI_Scatter* на транспьютерной матрице размера 4 на 4, используя только примитивные операции пересылки данных типа: точка-точка (например: *MPI_Isend*, *MPI_Irecv*) используя средства технологии *MPI*.
2) Произвести теоретическую оценку временной сложности алгоритма *MPI_Scatter* для транспортерной матрицы размера 4 на 4.
3) Реализовать устойчивую к сбою в процессе версию программы *Gemver*: реализовать механизм сохранения контрольных точек, реализовать возможность продолжения работы программы при сбое в одном из процессов, посредствам создания запасного процесса и восстановления из контрольной точки.

Описание алгоритма

Алгоритм *MPI_Scatter*

Нам необходимо промоделировать работу алгоритма *MPI_Scatter* на транспьютерной матрице размера 4 на 4. Но для того, чтобы получить более общий алгоритм, было принято решение реализовать задачу в общем случае. Приведем схему (Изображение 1) по которой реализованы пересылки, заметим, что при использовании заданной схемы пересылок достигается минимальная временная оценка для алгоритма *MPI_Scatter*.



Изображение 1 (Схема устройства матрицы и очереди пересылок)

На схеме (Изображение 1) каждая диагональ обозначена собственным цветом. Каждому всем элементам на одной диагонали данные из нулевого процесса отправляются за 1 раз. В предложенной реализации процессы из нулевого столбец транспьютерной матрицы выполняют пересылки и вправо и вниз, все остальные процессы могут отправлять данные только вправо. Такое ограничение позволяет четко структуризовать пересылки.

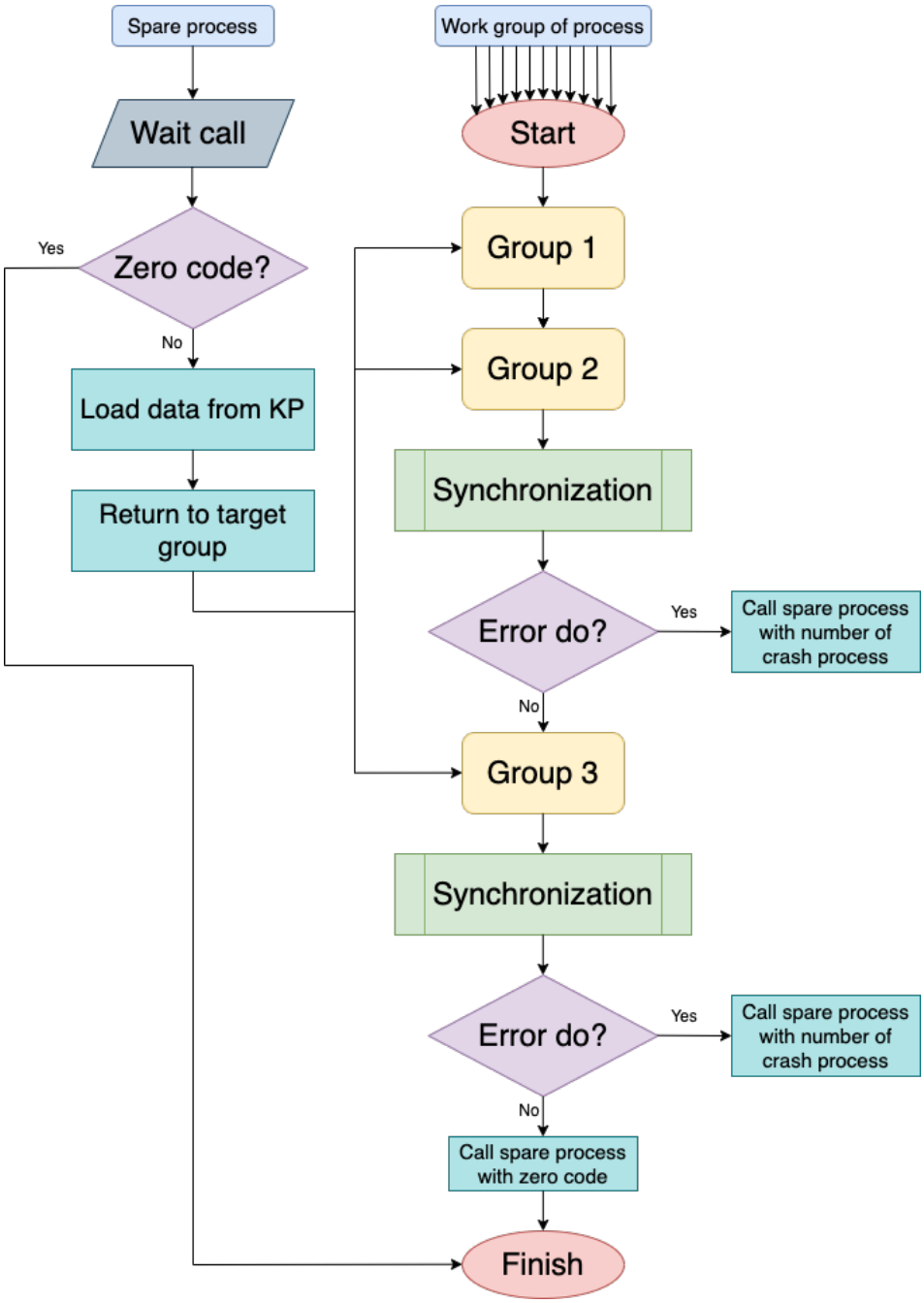
В начальный момент времени процесс стоящий в левом верхнем углу осуществляет отправку данных процессу, который стоит в самой удаленной диагонали с номером *2N-1*, таким образом, эта пересылка займет больше всего времени. В следующий момент времени процесс с индексом *(0,0)* отправляет данные предназначенные процессам стоящим в диагонали с номером *2N-1,* а процесс с индексом *(1,0)* получает данные от процесса (0,0) и отправляет их дальше по цепочке, процессу с индексом *(2,0)*, дальше шаги повторяются.

В заданной топологии, каждый процесс знает в какой позиции он находится, процесс распределитель помечает каждое отправляемое сообщение, помещая в тег номер процесса, которому окончательно принадлежит отправляемое данное. Каждый процесс, который получает данное, знает, каким процессам он должен на данном шаге работы алгоритма отправлять данные, поэтому он принимает только данные, которые принадлежат ему и отправляет их дальше по цепочке пересылок. В случае, если процесс получает порцию данных, которая принадлежит ему, значит далее он больше не будет принимать участие в пересылках, так как часть матрицы с номерами диагоналей большей, чем номер диагонали в которой стоит текущий процесс, уже обработана.

Важным преимуществом выбранной топологии является то, что каждый процесс может с легкостью вычислить какие процессы из транспьютерной матрицы он должен обслуживать на данном этапе алгоритма, а в случае, если ему больше некому пересылать данные, он может завершить работу.

Надежный алгоритм *Gemver*

В данном пункте задания, нам была поставлена задача реализовать надежный алгоритм *Gemver*, который осуществляет матричные операции с высокой степенью параллелизма и распределения данных по процессам. В курсе «Суперкомпьютеры и параллельная обработка данных» в рамках практикума, была реализована *MPI* версия этого алгоритма, которая не учитывает возможные сбои внутри вычислительных процессов. Доработаем эту реализацию. Для тестирования программы, в которой могут происходить сбои используется специальная версия компилятора *OpenMPI ULFM*.



Изображение 2 (Схема работы надежной реализации *Gemver*)

Для решения проблемы надежности была разработана схема восстановления программы, после сбоя. Общая схема работы алгоритма указана на Изображении 2.

Все процессы сохраняют свои данные, посредствам механизма асинхронных контрольных точек, а именно, в предопределенных местах работы алгоритма они выгружают все данные вычислений в файл с именем *proc_#_KT.txt,* где # - номер процесса, который создал эту контрольную точку. В файл записываются все данные, которые необходимы для восстановления вычислений с точки, в которой была сохранена последняя контрольная точка.

Во время выполнения расчетов, запасной процесс ждет сигнала от главного процесса, который осуществляет синхронизацию данных между всеми процессами, если на этапе синхронизации выяснилось, что один из процессов не может отправить свои данные, так как неожиданно завершил работу с ошибкой, главный процесс отправляет запасному процессу сообщение с номером процесса, который завершил работу со сбоем, если же ошибок не произошло, и мы не нуждаемся в услугах запасного процесса, то в конце вычислений запасному процессу отправляется сообщение, получая которое запасной процесс завершает свою работу.

Если запасной процесс получил сообщение, которое содержит в себе номер процесса, завершившегося со сбоем, то он считывает данные контрольной точки сбойного процесса и начинает переходит к тому блоку, в котором была сохранена последняя контрольная точка. Тем временем главные процесс ждет, пока запасной процесс догоняет все остальные процессы, чтобы получить от него необходимы для продолжения вычислений данные.

Таким образом, в независимости от того, произошло ошибка во время вычислений или нет, программа завершается успешно и выдает правильный результат вычислений.

Временная оценка

Нам была поставлена задача вывести теоретическую временную оценку работы алгоритма *MPI_Scatter* на транспьютерной матрицы, с учетом того, что процесс с индексом *(0,0)* должен отправить всем процессам из транспьютерной матрицы сообщение длинной 4 байта. Так как у нас получилось реализовать алгоритм для квадратной транспьютерной матрицы произвольного размера, то мы выведем временную оценку для этого случая. Мы имеем схему, в которой самой дальней точкой является процесс, который стоит в *2N-1* диагонали, и соответственно отправка данных ему, займет больше всего времени. Примем во внимание, что у нас заданы начальные параметры системы, а именно: $T_s = 100ms$, $T_b = 1ms$. Используя эти данные и выводы, которые мы получили выше запишем формулу временной сложности пересылок $T = (2N - 2)(T_s + 4T_b)$, коэффициент *4* перед T_b определен тем, что в нашей задачи мы отправляем каждому процессу 4 байта. Таким образом подставляя заданные значения вместо T_s и T_b получаем: $T = (2N - 2)(100 + 4)$ и для случая, где $N = 4$ имеем следующее значение: $T = 6(100 + 4) = 624ms$.

Мы получили очень оптимистичную временную оценку и нам удалось реализовать алгоритм, который должен ее достигать, так как работает по той же схеме, что и алгоритм, оценку которого мы производили теоретически.

Заключение

Нам удалось реализовать алгоритм *MPI_Scatter* с использованием пересылок типа точка-точка и средств *MPI*. Также мы вывели теоретическую временную оценку предложенного нами алгоритма.

Мы реализовали надежную версию алгоритма *Gemver*, которая устойчива к смерти одного процесса. Этого удалось достичь, так как был реализован метод создания асинхронных контрольных точек и восстановления из контрольной точки умершего процесса.

Приложение 1

Файл с программной реализацией *MPI_Scatter (scatter.c)*.

Приложение 2

Файлы с надежной реализацией алгоритма *Gemver (gemver.c, gemver.h)*.