МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М. В. Ломоносова

Факультет вычислительной математики и кибернетики

Компьютерный практикум по учебному курсу

«Распределенные системы »

ЗАДАНИЕ № 1

Разработка параллельной версии программы передачи сообщения в транспьютерной матрице.

ЗАДАНИЕ № 2

Доработка параллельной версии программыRedBlack2D с возможностью для продолжения работы программы в случае сбоя.

Отчет выполнил:
студент 420 группы
Широков А. П.

Преподаватель:
Бахтин В. А.

Москва
2020

Содержание

Постановка задачи 2

Задание 1 2
- Алгоритм . . . 2
- Обоснование . . . 2
  - Почему только 2 пути? . . . 2
  - Зачем делить сообщения на части? . . . 3
- Оценка времени работы . . . 3
  - Об эффективном k . . . 3
- Реализация при помощи MPI . . . 3
  - Описание ролей . . . 4
- Запуск . . . 4

Задание 2 5
- Описание ролей . . . 5
- Контрольные точки . . . 5
- Стратегия обработки сбоя . . . 5
  - Поведение босса . . . 5
  - Поведение рабочих процессов . . . 5
- Случай нескольких ошибок . . . 6
- Запуск . . . 6

Обзор прикрепленных файлов 6

Выводы 6

Постановка задачи

1. В транспьютерной матрице размером 8*8, 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 (7,7).
   - 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.
2. Доработать MPI-программу, реализованную в рамках курса “Суперкомпьютеры и параллельная обработка данных”.
   - Добавить контрольные точки для продолжения работы программы в случае сбоя.
   - Реализовать следующий сценарий работы после сбоя: продолжить работу программы только на “исправных” процессах.

Задание 1

Алгоритм

1. Процесс (0, 0) делит сообщения на 2 * k частей;
2. Сообщения по частям отправляются 2 путями:
   - (0, 0) - > (0, 1) -> ... -> (0, 7) -> (1, 7) -> ... -> (7, 7)
   - (0, 0) - > (1, 0) -> ... (7, 0) -> (7, 1) -> ... -> (7, 7)
3. Сообщение собирается в процессе (7, 7). Сообщение доставлено.

Обоснование

Почему только 2 пути?

Транспьютерная матрица предоставляет нам широкий выбор путей передачи сообщения. Например, мы можем в каждой ячейке делить сообщения на 2 и передавать разными путями. Тем самым, кажется, мы можем эффективно использовать все процессы и получить выйгрыш в скорости.

На самом же деле это не так. Узким местом такой передачи является передача сообщения непосредственно конечному процессу (7, 7). Такой процесс одновременно может получать сообщения только по 2 путям, поэтому как бы много путей мы бы не сделали, все равно в конечном итоге работать они будут не быстрее чем с использованием 2 простых путей, описанных в алгоритме выше.

Зачем делить сообщения на части?

Сообщение делится на части, чтобы увеличить вовлеченность большего числа процессов в обмен в каждый момент времени. Если по одному пути передается только одна часть, то в каждый момент взаимодействуют 2 процессса. Если же частей не меньше, чем количество процессов в пути, то взаимодействуют все процессы в этом пути. Оптимальное значение k может быть посчитано для конкретных значений L, Ts и Tb.

Оценка времени работы

Последовательно построим оценку времени:

- Сообщение по условию очень длинное => мы можем пренебречь TS.
- Мы передаем сообщение параллельно по двум путям => нам достаточно построить оценку времени для предачи сообщения длины $L/2$ по одному пути;
- Первая часть сообщения дойдет за

$$14 * (Tb * L/2k)$$

, где 14 это количество обменов в пути;

- Оставшиеся части будут идти

$$(Tb * L/2k) * (k - 1)$$

- Получим оценку

$$14 * (Tb * L/2k) + (Tb * L/2k) * (k - 1)$$

Об эффективном k

Эффективное k можно нетрудно нати для конкретных значений L и Tb. Оценка времени – функция от k. Достаточно просто найти минимум этой функции.

Реализация при помощи MPI

Для реализации алгоритма на MPI при помощи передачи сообщений в режиме по готовности сделаем следующее:

- Раздадим всем процессам роли, в зависимости от их положения в матрице. Всего будет 5 ролей, которые будут описаны позже.
- Введем единицу взаимодействия, которую назовем период. Период делится на 3 окна.

- Каждый процесс в зависимости от своей роли в каждом окне может работать в режимах Ready (гоовится к приему), Receive (принемает кусок сообщения), Send (отправляет кусок сообщения дальше)
- В конце каждого окна происходит барьерная синхронизация, которая гарантирует то, что все процессы справились со своими задачами и готовы к обменам.

Описание ролей

- Sender - процесс (0, 0). За один период отправляет по одной части сообщения на каждый путь. Работает по сценарию -/Send/-, где - означает, что в окнах 1 и 3 он ничего не делает. Когда все сообщение отправлено, он преходил в режим ожидания (-/-/-).
- Receiver - процесс (7, 7). За один период принимает части сообщения от 2 путей и готовится к следующей передаче. Сценарий работы Ready/-/Receive.
- Internal3 - внутренние элеменнты матрицы, не лежащие ни на одном пути. Сценарий работы -/-/-.
- Internal2 - процессы, лежащие на путях передачи и имеющие четный порядковый номер в пути. Сценарий работы Ready/Send/Receive.
- Internal1 - процессы, лежащие на путях передачи и имеющие нечетный порядковый номер в пути. Сценарий работы Ready/Receive/Send.

Sender	Internal1	Internal2	Internal1	...	Internal1	Receiver
-	Ready	Ready	Ready	...	Ready	Ready
Send	Receive	Send	Receive	...	Receive	-
-	Send	Receiv	Send	...	Send	Receive

Таблица 1: Взаимодействие процессов.

Запуск

Количество процессов должно совпадать с количеством элементов матрицы. За размер матрицы отвечает макрос N.
Например, для N = 8.

```
mpicc Matrix.c -o run
mpiexec -n 64 ./run
```

Задание 2

Описание ролей

Процессы, которые использовалась в рамках задания по курсу СКиПОД, назовем рабочими. Они выполняют все необходимые вычисления.

Дополнительно добавим процесс босс, который организовывает надежную работу процессов. Обязанности босса:

- Сохранять в своей памяти состояния контрольных точек;
- Поддерживать актуальное описание решаемых рабочими процессами задач;
- Обрабатывать сбой. Перераспределять задачи между оставшимися рабочими процессами. Передавать им актуальное состояние, полученнное из контрольной точки.

Контрольные точки

Состояние процессов сохраняется раз в $backup_iter$ итераций. Рабочие процессы отправляют боссу посчитанные ими части матрицы.

Стратегия обработки сбоя

Благодаря использованию MPI_Barrier все процессы узнают о сбое одновременно.

Поведение босса

- Понять, кто перестал работать. Функция who_quit();
- Перераспределить задачи с учетом потери рабочего процесса. Процедура change_plans(int);
- Раздать актуальные задания рабочим процессам. Процедура give_new_tasks();

Поведение рабочих процессов

- Если рабочий процесс как-то взаимодействовал со сломавшимся, то сообщить боссу, кто именно сломался. Процедура send_info(int);
- Получить новое задание. Процедура get_new_task();
- Продолжить работать.

Случай нескольких ошибок

Данную стратегию можно применять для произвольного количество сбоев. Для этого необходимо отказаться от MPI_Barrier и реализовать его аналог, который ждет только работающие процессы. Это можно сделать также, как, была реализована операция редукции и broadcast. Прцедуры max_reduction() и broadcast(int) соответственно.

Запуск

Количество рабочих процессов должно совпадать со значением макроса NP, сделано это для удобства. Его значение может быть произвольным.
Например, для NP = 4, т.е 4 рабочих процесса и 1 процесс босс.

```
mpicc crash_recovery.c -o run
mpiexec -n 5 - -disable-auto-cleanup ./run
```

Обзор прикрепленных файлов

- Matrix.c – реализация задания 1
- crash_recovery.c – реализация задания 2
- MPI.c – исходная MPI программа

Выводы

Было сделано:

- Реализован алгоритм передачи сообщения в транспьютерной матрице. Оценено время его работы;
- Доработана MPI программа, добвалены возможности продалжать работу после сбоя.