Московский государственный университет имени М. В. Ломоносова

Факультет вычислительной математики и кибернетики

Компьютерный практикум по учебному курсу “Распределенные системы”

Алгоритм MPI_Ssend для транспьютерной матрицы

Доработка MPI-программы, реализованной в рамках курса “Суперкомпьютеры и параллельная обработка данных”. Исправная работа программы при возникновении сбоев в процессах.

Отчет

о выполненном задании

Студента 424 группы факультета ВМК МГУ

Барышева Артем Владимировича

Лекторы:

Профессор, д.ф.-м.н. Крюков Виктор Алексеевич

Доцент, к.ф.-м.н. Бахтин Владимир Александрович

Москва 2021

Содержание

1. Задание №1

1.1 Постановка задачи

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 синхронного режима 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.

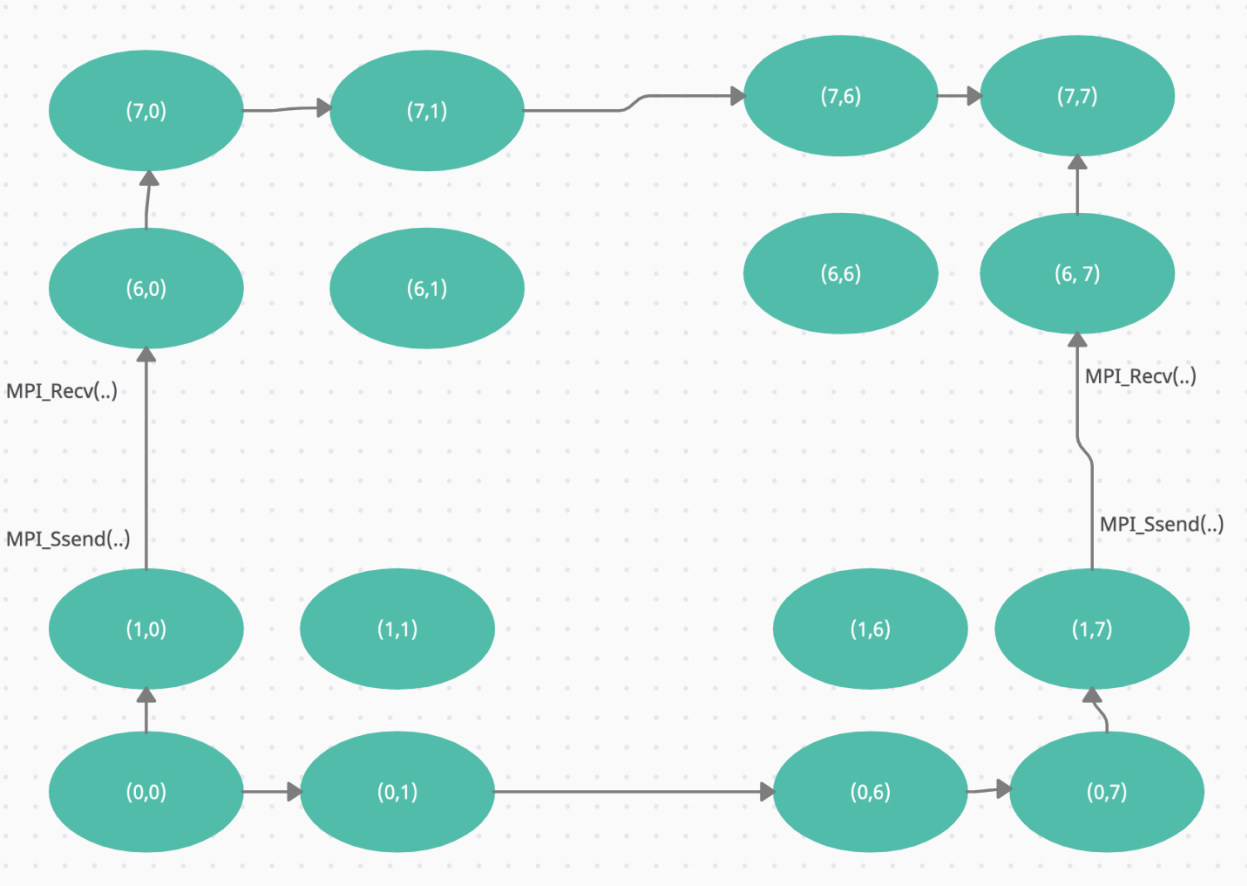
1.2 Описание алгоритма

В виду того, что у процесса с номером (7, 7) около углов узкое место, оптимально будет передавать данные из процесса с номером (0,0) двумя путями, а именно буквой Г. Данные разделяются на CNT_DIV_MESSAGE * 2 количество частей, которые должны в итоге оказался в процессе (7, 7).

Передача данных от одного процесса другому организована двумя функция: **MPI_Ssend** (организует синхронность в передаче данных) и функции получения данных **MPI_Recv**.

Код лежит в директории /**task_1_transputer_matrix_MPI/**. Главный файл - main.cpp.

В 0-м процессе формируется сообщение длинной SIZE_MESSAGE, в котором массив элемент с номером i = i. Инициализация сообщения происходит в функции **initialization_message()**. Проверка корректности полученных данных



проверяется в процессе (7, 7), то есть с номером 63 с помощью функцию **check_result()**.

1.3 Получение временной оценки

Так как сообщение является **очень длинным (длина L байт)**, то временем старта, временем разгона конвейера и длиной маршрута можно пренебречь для временной оценки. Таким образом у нас остается только Tb. Разобьем сообщение на K = CNT_DIV * 2 кусков и здесь несколько вариантов:

- Если рассматриваем 0-й процесс: пересылает по очереди 1 кусок соседу сверху, 2-й кусок соседу справа, и так далее пока не закончится сообщение
- Если рассматриваем любой другой процесс: ему нужно передать сообщение одному нужному соседу

1-й кусок дойдет до нужного процесса (7, 7) за временную оценку равную (TB * L / K) * 14. Так как сообщение очень длинное, получаем временную оценку равную

(TB * L / K). Получается последний кусок дойдет до нужного процесса (7, 7) за временную оценку равную (K - 1) * (TB * L / K) + (TB * L / K) == TB * L. Учитывая, что TB == 1 получаем (K - 1) * (L / K) + (L / K) == L. Такая оценка из-за того, что у нас синхронный режим передачи данных.

1.4 Инструкция по запуску программы

Исходный код доступен в открытом сервисе **github** по ссылке: **https://github.com/arteeemik/ROC_PracticalTask/tree/main/task_1_transputer_matrix_MPI**

Для запуска программы требуется open-mpi и требуется выполнить следующие команды:

- **mpic++ -o main main.cpp --std=c++17**
- **mpirun -np 64 ./main**

2. Задание №2

2.1 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:

a) продолжить работу программы только на “исправных” процессах;

b) вместо процессов, вышедших из строя, создать новые MPI-процессы, которые необходимо использовать для продолжения расчетов;

c) при запуске программы на счет сразу запустить некоторое дополнительное количество MPI-процессов, которые использовать в случае сбоя.

2.2 Описание оригинальной программы

Оригинальная версия программы находится в директории **/task_2_fault_tolerance/original_program/**. Данная программа производит определенные вычисления, которые нужно было распараллелить с помощью MPI.

В начале функции **int main(int argc, char** argv)** происходит создание инициализация исходных данных созданием массивов и вызовом функции init_array:

init_array (tmax, nx, ny,*ex, *ey, *hz, *_fict_);

Основные вычисление происходят в функции **static void kernel_fdtd_2d(…).**

Функция **static void print_array(…)** выводит содержимое массивов **ex, ey, hz** в поток сообщений об ошибках stderr.

2.3 Описание параллельной версии программы

Программа была распараллелена с помощью программного обеспечения MPI. Параллельная версия программы находится в директории **/MPI_Version/original_program/.**

Функционирует программа следующим образом:

- Отправка и получение данных между процессами осуществляется при помощь **MPI_Isend**, **MPI_Irecv**, **MPI_Recv**, **MPI_Send**;
- Контроль за процессами осуществляется при помощи **MPI_Waitall** и **MPI_Barrier**;
- Двумерные массивы поделены поровну между всеми процессами по строкам (каждому процессу достается = nx / ProcNum + (nx % ProcNum > ProcRank))

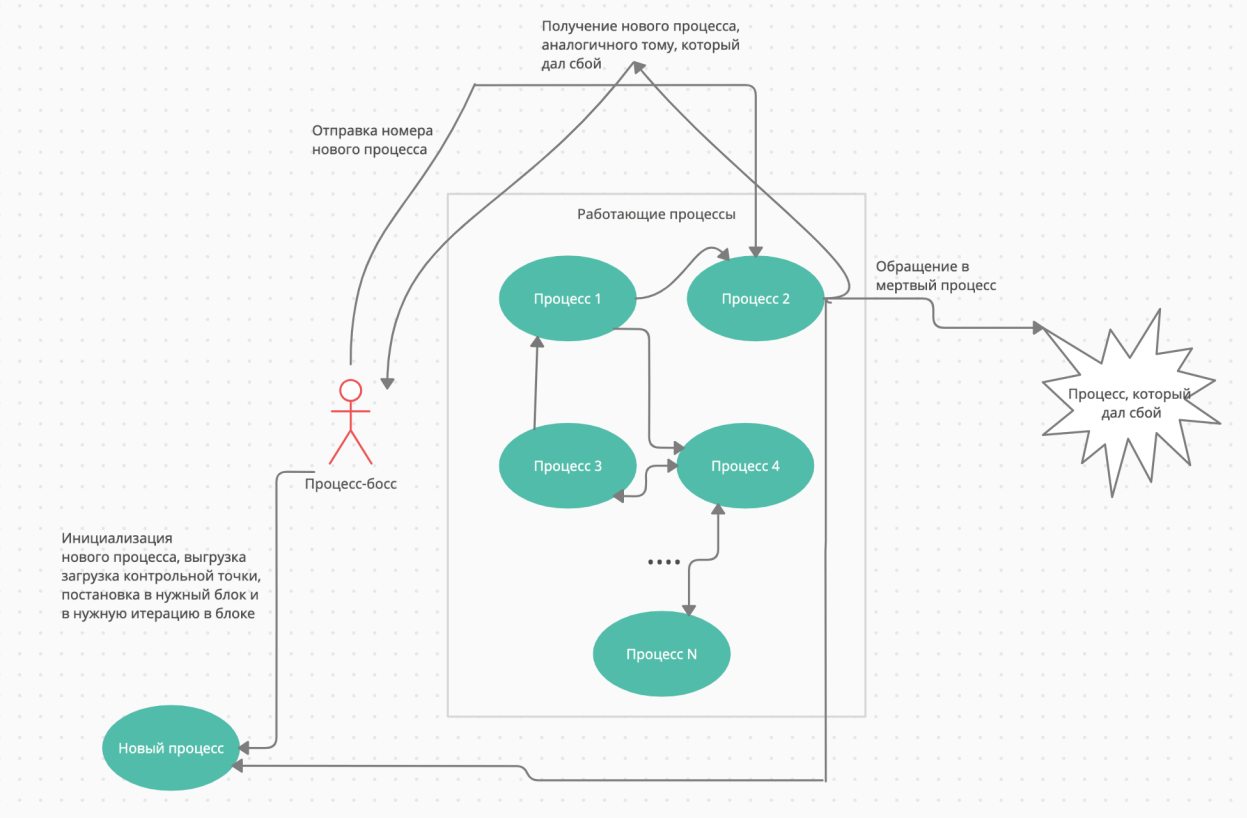
- В процессе с номером 0 происходит инициализация исходных данных в массивах **ex**, **ey**, **hz**;
- Процесс с номером 0 отсылает остальным процессам участки данных из массивов **ex**, **ey**, **hz**, за которые те ответственны;
- Далее запускается функция **kernel_fdtd_2d(…)** в каждом процессе для вычисления результата;
- В функции **kernel_fdtd_2d(…)** процесс с номером 0 посылает данные процессу с номером 1 необходимые для подсчет обновленных данных в массиве **ey** и асинхронно считает результаты для **ey**. Процесс с номером 1 принимает данные от 0-го процесса, отсылает данные 2-му процессу, и считает обновленные данные, и так далее. Массив **ex** в каждом процессе считается независимо, так все данные в каждом процессе есть. Для подсчета **hz** требуются подсчитанные значения первой строчки массива **ey** из следующего процесса, поэтому процесс с номером N передает 1-ю строчку массива **ey** процессу с номером (N - 1) асинхронно. Дальше считаются обновленные данные для массива **hz**;
- После того, как функция **kernel_fdtd_2d(…)** все процессы пересылают посчитанные результаты процессу с номером 0;
- После в процессе с номером 0 осуществляется замер времени и вывод результата через функцию **print_array**.

2.4 Описание устойчивой к сбоям параллельной версии программы

Устойчив к сбоям параллельная версия программы находится в директории /MPI_Version_fault_tolerance/original_program/.

Функционирует программа следующим образом:

- Для обработки сбоя (сбой в одном из процессов) в программе использовались функция из стандарта MPI **MPI_Comm_set_errhandler**(MPI_COMM_WORLD, MPI_ERRORS_RETURN) для того, чтобы функции MPI возвращали код ошибки;
- В качестве основного решения был выбран вариант, когда при запуске программы создается некоторое количество резервных процессов, которые смогут заменить убитые процессы;
- Эта реализация не предполагает, что 0-й или резервные процессы дадут сбой;
- Данные сохраняются для каждого процесса в специальных контрольных точках с помощью функции **save_checkpoint(…);**
- Загрузка посчитанных результатов в резервный процесс осуществляется при помощи функции **load_checkpoint**. Файлы с сохраненными вычислениями называются в следующем формате: “save_point_[**Номер процесса**]_[**Номер блока**]_[**Итерация блока, в которой были последний раз посчитаны результаты**].txt”;
- Программа разделана на вычисляющие блоки, где в конце каждого блока для каждого процесса происходит сохранение посчитанных результатов на жесткий диск, в эти блоки можно попасть с помощью оператора **goto**;
- Программа на вход принимает 1 аргумент **cnt_bad_proces** - количество случайных процессов, которые в случайном месте в коде дадут сбой. Если не передавать программе никаких аргументов, то она будет работать, как будто **cnt_bad_proces == 0**. Важное ограничение на этот параметр: если N - общее число процессов, то N >= **cnt_bad_proces** * 2 + 2;



- В начале программы считается количество активных процессов (**ProcNum**), количество резервных процессов(**ReservProcNum**), вычисляется номер процесса-босса(**ReservProcBoss**), в который будут обращаться работающие процессы в случае обнаружения процесса, который погиб. Также в начале программы рандомным образом выбираются рабочие процессы, которые во время программы дадут сбой и рандомно выбирается блок и итерация, на которой процесс даст сбой;

- Ниже дан схематичный рисунок как все устроено при сбоях. Вместе с резервными процессами резервируется один процесс, чтобы контролировать остальные процессы во время сбоя других, и говорить какому резервному процессу и где занять свое место. Процесс-босс мониторит все свое время сообщения от других процессов (с помощью функций **MPI_Irecv** и **MPI_Test**), если пришел запрос от процесса X о том, что процесс Y сломался, процесс-босс инициализирует один из свободных резервных процессов Z с помощью последней контрольной точки процесса Y, и заменяет номер процесса Y на Z. В следующий раз, если придет сообщение от какого-то процесса о том, что

сломался процесс Y, процесс-босс сразу же вернет номер нового процесса Z. Процесс-босс также отсылает сообщение процессу Z о том, что ему нужно занять определенное место в коде и продолжить работу за место процесса, который дал сбой;

- Правильность полученных данных сравнивалась с оригинальной последовательной программой, а также с распараллеленой версией без обработки ошибок.

2.5 Инструкция по запуску программы

Исходный код доступен в открытом сервисе **github** по ссылке: **https://github.com/arteeemik/ROC_PracticalTask/tree/main/task_2_fault_tolerance**

Есть три папки **MPI_Version**, **MPI_Version_fault_tolerance**, **original_program**. В первой дирректории хранится распараллеленная программа без обработки ошибок, во второй - с обработкой ошибок, и в третьей - оригинальная последовательная программа.

Для запуска оригинальной программы требуется выполнить следующие команды в директории **original_program**:

- **mpicc -o main fdtd-2d.c**
- **mpirun -np 1 main**

Для запуска распараллеленной программы без обработки ошибок требуется выполнить следующие команды в директории **MPI_Version**:

- **mpicc -o main fdtd-2d.c**
- **mpirun -np <нужное количество процессов> main**

Для запуска распараллеленной программы с обработкой ошибок требуется установить пакет **mpich** и выполнить следующие команды в директории **MPI_Version_fault_tolerance**:

- **mpicc -o main fdtd-2d.c**
- **mpirun -np <нужное количество процессов> —disable-auto-cleanup main <количество процессов, которые дадут сбой>**

3. Заключение

В результате данной практической работы были выполнены задачи:

- Алгоритм MPI_Ssend для транспьютерной матрицы
- Доработка MPI-программы, реализованной в рамках курса “Суперкомпьютеры и параллельная обработка данных”. Исправная работа программы при возникновении сбоев в процессах.

Для алгоритма MPI_Ssend для транспьютерной матрицы была также получена временная оценка.

Была доработана MPI-программа, реализованная в рамках курса “Суперкомпьютеры и параллельная обработка данных”, которая стала устойчива к сбоям в процессах.