МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М.В. Ломоносова
Факультет Вычислительной Математики и Кибернетики

Практикум по учебному курсу
"РАСПРЕДЕЛЁННЫЕ СИСТЕМЫ"

ОТЧЕТ

о выполненном задании

Улучшение параллельной верчии программы транспонирования матриц

студентки 428 учебной группы факультета ВМК МГУ
Шелепнёвой Дарьи Дмитриевны

Москва, 2022г.

Содержание

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: а) продолжить работу программы только на “исправных” процессах; б) вместо процессов, вышедших из строя, создать новые MPI-процессы, которые необходимо использовать для продолжения расчетов; в) при запуске программы на счет сразу запустить некоторое дополнительное количество MPI-процессов, которые использовать в случае сбоя.

2 Алгоритм

2.1 Описание алгоритма

Напомню условие прошлогодней задачи: Разработка параллельной версии программы для транспонирования матриц.
Требуется:

1. Распараллелить программу с помощью технологии OpenMP и MPI.
2. Исследовать масштабируемость полученной программы, построить графики зависимости времени выполнения программ от числа используемых ядер (процессов) и объёма входных данных.

Отчёт и прошлогодний код можно найти тут:
`https://github.com/DariaShel/skipod`

Т. к. новая задача состоит в защите от сбоев, то был удалён цикл с экспериментами, и программа запускалась на матрице размера 10x10 на 64 процессах.

Улучшение:

Я выбрала стратегию, в которой при запуске программы на счет сразу запускается некоторое дополнительное количество MPI-процессов, которые используются в случае сбоя.

При запуске программы, записываем входную матрицу в файл "checkpoint.txt"процессом-мастером и при каждом входе в функцию *transpose()* им же считываем её из файла.

Если хотя бы один из процессов (кроме мастера) во время выполнения очередной итерации упал, то перераспределим работу между оставшимися процессами и начнём вычисления заново.

Чтобы программа не завершалась аварийно при падении процессов, зарегистрируем обработчик таким образом, чтобы при падении процессов функции коммуникаций возвращали ошибки: *MPI_Comm_set_errhandler(comm, MPI_ERRORS_RETURN)*.

Будем смотреть в программе на возвращаемое значение таких функций и в случае неуспеха прерывать выполнение функции *transpose()* и возвращать -1 из неё. Для запуска повторных итераций при возникновении ошибок была написана дополнительная функция *solver()*. После этого будем создавать новый коммуникатор из выживших процессов с помощью *MPIX_Comm_shrink(main_comm, &main_comm)* и перезапускать итерацию.

2.2 Реализация алгоритма

Реализацию алгоритма есть на гитхабе: `https://github.com/DariaShel/skipod`

2.3 Инструкции по запуску

Для запуска необходимо выполнить следующие команды в терминале:

- **mpicc transpose_mpi.c -o transpose_mpi**
- **mpirun -np 64 transpose_mpi –enable-recovery –with-ft ulfm –oversubscribe**

3 Временная оценка

Установка контрольных точек отразится на производительности.

Операции чтения из файла и запись в файл мастером будут занимать некоторое время, зависящее от размеров исходной матрицы A. В это время все остальные процессы будут простаивать в ожидании своего кусочка от мастера. Таким образом, в лучшем случае программа замедлится на $fault\_count * T_{read} + T_{write}$, где

T_{read} — время чтения входной матрицы из файла;

T_{write} — время записи входной матрицы в файл;

$fault\_count$ - количество сбоев программы.

То есть чем больше итераций, на которых падают процессы, тем дольше будет выполняться программа. Таким образом, в худшем случае программа будет выполняться $(nProc - 1) * T_{read} + T_{write}$