Практикум по курсу

“Распределенные системы”

Передача сообщения на транспьютерной матрице с использованием неблокирующих операций MPI.

Добавление контрольных точек и возможности восстановления работоспособности программы после сбоя.

Отчет

о выполненном задании

Студента 428 учебной группы факультета ВМК МГУ

Лебедева Никиты Алексеевича

Москва, 2020

Оглавление

Задача 1

Постановка задачи

Требуется разработать и реализовать алгоритм передачи очень длинного сообщения на транспьютерной матрице (8x8) от процесса (0,0) процессу (4,4) с использованием неблокирующих операций 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.

Краткий обзор файлов решения и команд запуска

Код доступен по ссылке.

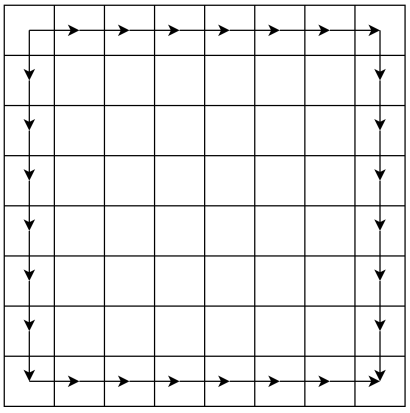
Решение задачи находится в файле 1.c

Для компиляции и запуска и запуска использовать bash-скрипт **./run 1.c 64**

Разработка и тестирование производились на MacOS

Описание работы алгоритма

Используем конвейер. Чтобы ускорить передачу данных, разделим поток на 2 части.



Расчет времени выполнения

Пусть L - длина сообщения, Ts = 100, Tb = 10
Пусть N - количество чанков, на которые мы поделили сообщение.

Время передачи сообщения одним конвейером

Сначала посчитаем минимальное время, за которое сообщение длины L передастся одним конвейером.

T_{chunk} - время передачи одного чанка между соседними клетками транспьютерной матрицы.

Для простоты расчетов положим, что длина сообщения кратна длине чанка.

Нам нужно передать сообщение 14 раз.

Поскольку мы используем конвейер, времени нам потребуется как на передачу (N + 14 - 1) чанков в соседнюю клетку.

$$
\begin{aligned}
t &= T_{chunk} * (N + 13) = (T_S + T_B) * (N + 13) = \\
&= (100 + 1 * L/N) * (N + 13) = \\
&= N * 100 + 13 * 100 + 1 * L + 13 * 1 * L/N = \\
&= N * 100 + 1300 + L + 13 * L/N
\end{aligned}
$$

Найдем N такое, чтобы t было минимальным
Для этого найдем производную t по N и приравняем ее к 0:

$$
\begin{aligned}
dt/dN &= 100 - 13 * L/N^2 = 0 \\
N^2 &= 13L/100 \\
N &= \sqrt{13L} / 10 \text{ - это точка минимума значения t}
\end{aligned}
$$

Таким образом, время, которое займет передача сообщения длины L при оптимальном его разбиении на чанки составит

$$
\begin{aligned}
t_{optimal}(L) &= (\sqrt{13L} / 10) * 100 + 1300 + L + 13 * L/(\sqrt{13L} / 10) = \\
&= 10\sqrt{13L} + 1300 + L + 10\sqrt{13L} = L + 20\sqrt{13L} + 1300
\end{aligned}
$$

Например, при $L = 1\,000\,000$, получим $t_{optimal} = 1\,073\,000$

Время передачи сообщения двумя конвейерами

Поскольку мы используем два конвейера, то поделим длину сообщения пополам пополам:

$$t_{optimal-2}(L) = t_{optimal}(L/2) = L/2 + 20\sqrt{7.5L} + 1300$$

Теперь сообщение длины L = 1 000 000 при оптимальном разбиении передастся за время $t_{optimal-2} = 556\,072$

Вывод

Если бы мы передавали сообщение длины $L = 1\,000\,000$ между ячейками транспьютерной матрицы целиком, время передачи составило бы $t_{not-optimal}(L) = 14 * (T_s + T_b * L) = 14 * 1\,000\,100 = 14\,001\,400$

Таким образом, при $L = 1\,000\,000$ мы ускорили передачу сообщения от ячейке (0,0) в ячейку (7,7) в $t_{not-optimal} / t_{optimal-2} = 14\,001\,400 / 556\,072 = 25.18$ раз при разбиении сообщения на чанки оптимального размера.

Задача 2

Постановка задачи

Необходимо доработать программу, реализованную в рамках предыдущего курса, добавив контрольные точки и возможность восстановления работоспособности программы в случае сбоя.

Краткий обзор файлов решения и команд запуска

Код доступен по ссылке.

Решение задачи находится в файле 2.c

Для компиляции и запуска и запуска использовать bash-скрипт **./run 2.c 6**

Разработка и тестирование производились на MacOS

Описание работы алгоритма

Каждый процесс хранит у себя чекпоинт - полное состояние всех матриц на момент последней синхронизации.

Синхронизация происходит раз в 30 итераций (константа может быть изменена).

У каждого процесса есть массив **procsAlive**, в котором для каждого процесса с рангом **rank** в **procsAlive[rank]** хранится 1, если он жив, и 0 иначе.

В начале каждой итерации процессы вызывают функцию **sync_error_status(status)**, где **status = 0**, если в процессе произошла ошибка, и **1**, если ошибка не произошла.

Функция **sync_error_status** отправляет другим процессам данные о статусе процесса и получает данные от других процессов, обновляя информацию о них в массиве **procsAlive**.

Функция **sync_error_status** возвращает 1, если хотя бы в одном процессе произошла ошибка, и 0 иначе.

Если ошибка произошла в текущем процессе, он делает return и больше не участвует в вычислениях.

Если ошибка произошла в другом процессе, то текущий процесс вызывает функцию **reinit()**.

Функция **reinit()** откатывает текущий процесс к контрольной точке и выделяет ему ту часть матриц, которая соответствует его новому рангу среди оставшихся в живых процессов.

Чтобы процессы были взаимозаменяемыми, вместо **rank-1** и **rank+1** используются переменные **prevRank** и **nextRank**, также обновляемые при вызове **reinit()**

Выводы

Даже при допущении такой условности как то, что процесс уведомляет все другие процессы об ошибке в нём, для добавления возможности восстановления с контрольной точки в рантайме потребовалась значительная переработка изначальной программы. Кажется, что для решения поставленной задачи всё-таки стоило использовать версию MPI с сайта fault-tolerance.