МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное   
образовательное учреждение высшего образования  
«Самарский национальный исследовательский университет   
имени академика С.П. Королева»

(Самарский университет)

Институт информатики и кибернетики

Кафедра технической кибернетики

ЛАБОРАТОРНАЯ РАБОТА № 6

по курсу   
Параллельное программирование

Группа 6408

Студент Н.В. Носов

(*подпись*)

Преподаватель,

к.ф.-м.н. Е.С. Козлова

(*подпись*)

Самара 2023

ЗАДАНИЕ

**Реализовать блочно-строчное умножение матриц на MPI на C++. Каждый блок хранится на отдельном процессоре. Размерности матриц кратны количеству процессоров.**

**В ходе анализа работы программы оценить время ее выполнения на различном количестве процессов.**

**Реализовать последовательный вариант программы. Оценить время ее выполнения. Рассчитать ускорение параллельных программ относительно последовательного варианта.**

Таблица 1 – Исходные данные на ЛР № 6

|  |  |
| --- | --- |
| Тип | int |
| (ROWS\_A, COLS\_A, COLS\_B) | [(512, 512, 512), (1024, 1024, 1024), (2048, 2048, 2048)] |
| Количество процессов | 2, 4, 8 |

ВВЕДЕНИЕ

Одним из важных методов программирования является параллельное программирование. Оно позволяет разделить процесс решения целой задачи на несколько отдельных подзадач, которые могут выполняться одновременно на нескольких ядрах компьютера. Этот подход считается эффективным в сокращении времени работы программы [1].

В данной работе применяется технология параллельного программирования MPI. Она используются для распараллеливания программ.

В данной лабораторной работе представлены результаты вычисления времени выполнения программы по блочно-строчному умножению матриц с использованием технологии MPI на различном количестве процессов.

ЭКСПЕРИМЕНТАЛЬНЫЕ ИССЛЕДОВАНИЯ

2.1 Результаты работы программ

В ходе исследования времени работы программ здесь и далее проводилось усреднение не менее чем по 12 запускам.

На рисунке 1 представлен скрин запуска и работы программы.

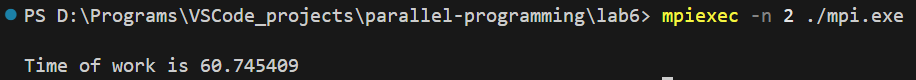


Рисунок 1 – Пример работы параллельной программы при размерностях (2048, 2048, 2048) на 2 процессах.

Последовательная программа представляла собой алгоритм для отыскания произведения двух матриц.

В таблицах 2-3 представлено время выполнения параллельных программ и их ускорение по сравнению с последовательным вариантом.

Таблица 2 – Время работы программ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| (ROWS\_A, COLS\_A, COLS\_B) | | (512, 512, 512) | (1024, 1024, 1024) | (2048, 2048, 2048) |
| Время последовательного, с | | 0.598725 | 7.949293 | 88.914739 |
| Количество процессов, шт | 2 | 0.487257 | 4.663067 | 60.745409 |
| 4 | 0.235367 | 3.198288 | 37.827790 |
| 8 | 0.143540 | 1.641536 | 18.522006 |

Таблица 4 – Ускорение параллельных программ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| (ROWS\_A, COLS\_A, COLS\_B) | | (512, 512, 512) | (1024, 1024, 1024) | (2048, 2048, 2048) |
| Количество процессов, шт | 2 | 1.228766 | 1.704735 | 1.463728 |
| 4 | 2.543793 | 2.485484 | 2.350514 |
| 8 | 4.171137 | 4.842594 | 4.800492 |

На рисунке 2 приведен график зависимости времени работы программ от размерности матрицы при различном количестве процессов. На рисунке 3 приведен график зависимости ускорения программ от размерности матрицы при различном количестве процессов.

Рисунок 2 – Время работы программ

Рисунок 3 – Ускорение программ

**ВЫВОДЫ:**

Из полученных результатов видно, что:

1. Для технологии MPI минимальное время работы составило 0,143540 при размерностях (512, 512, 512) на 8 процессах. Максимальное время работы на MPI составило 60,745409 при размерностях (2048, 2048, 2048) на 2 процессах. Это можно объяснить тем, что чем больше процессов, тем быстрее они справятся с вычислениями. Также чем меньше объем вычислений, тем меньше время выполнения программы.
2. Для технологии MPI минимальное ускорение составило 1,228766 при размерностях (512, 512, 512) на 2 процессах. Максимальное ускорение на MPI составило 4,842594 при размерностях (1024, 1024, 1024) на 8 процессах. С увеличением процессов увеличивается и ускорение для каждой размерности. Увеличение происходит кратно (в среднем).
3. Время работы программ с использованием MPI меньше времени выполнения последовательных программ. Так, например, при размерности (512, 512, 512) время выполнения последовательной программы равно 0,598725, а время выполнения параллельной программы на 2 процессах равно 0,487257, на 4 процессах – 0,235367, на 8 процессах - 0,143540. При размерности (1024, 1024, 1024) время выполнения последовательной программы равно 7,949293, а время выполнения параллельной программы на 2 процессах равно 4,663067, на 4 процессах – 3,198288, на 8 процессах - 1,641536. При размерности (2048, 2048, 2048) время выполнения последовательной программы равно 88,914739, а время выполнения параллельной программы на 2 процессах равно 60,745409, на 4 процессах – 37,827790, на 8 процессах – 18,522006.

ЗАКЛЮЧЕНИЕ

Цель лабораторной работы – **реализовать блочно-строчное умножение матриц на MPI на C++, где каждый блок хранится на отдельном процессоре достигнута.**

Показано, что использование параллельных технологий для данного типа программ обосновано, в виду того, что с помощью MPI было получено ускорение от 1,228766 до 4,842594. Данное ускорение является значительным.

В ходе выполнения лабораторной работы я изучил блочно-строчный алгоритм умножения матриц, а также как реализовать топологию “процессорное кольцо” в MPI. Наиболее сложной частью выполнения лабораторной работы было реализация топологии “процессорное кольцо” в MPI. Интерес вызвали полученные результаты.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Козлова, Е.С. Лабораторные работы по курсу «Параллельное программирование»: Методические указания [Текст] / Сост. Е.С. Козлова, А.С. Широканев − Самара, 2019. – 61 с.
2. оеводин, В. В. Параллельные вычисления [Текст] / В. В. Воеводин, Вл. В. Воеводин. — СПб.: БХВ-Петербург, 2002. — 608 с.
3. Богачёв К.Ю. Основы параллельного программирования: учебное пособие, 2-е изд. [Текст] / К. Ю. Богачёв ‑ М. : БИНОМ. Лаборатория знаний, 2013. ‑ 344 с.
4. Гергель, В. П. Теория и практика параллельных вычислений, 2-е изд. [Текст] / B. II. Гергель. — М.: Интуит. 2016. - 500 с.
5. Боресков А.В. Параллельные вычисления на GPU. Архитектура и программная модель CUDА Учеб. пособие [Текст] / А.В. Боресков ‑ М.: Издательство Московского университета, 2012. - 336 с.
6. Библиографическое описание документа. Общие требования и правила составления [Электронный ресурс] / сост.: В.С. Крылова, С.М. Григорьевская, Е.Ю. Кичигина // Официальный интернет-сайт научной библиотеки Томского государственного университета. – Электрон. дан. – Томск, [2010]. – <http://www.lib.tsu.ru/win/produkzija/metodichka/metodich.html> (дата обращения: 10.09.2019).

ПРИЛОЖЕНИЕ А

Код параллельной программы

#include "mpi.h"

#include <cstdio>

#include <cstdlib>

#include <algorithm>

#define ROWS\_A 2048

#define COLS\_A 2048

#define ROWS\_B COLS\_A

#define COLS\_B 2048

void printMatrix(int\* matrix, int rows, int cols) {

    for (int i = 0; i < rows; ++i) {

        printf("\n");

        for (int j = 0; j < cols; ++j) {

            printf("%d  ", matrix[i \* cols + j]);

        }

    }

    printf("\n");

}

int main(int argc, char\* argv[]) {

    MPI\_Status Status;

    int ProcRank, ProcNum;

    MPI\_Init(&argc, &argv);

    MPI\_Comm\_size(MPI\_COMM\_WORLD, &ProcNum);

    MPI\_Comm\_rank(MPI\_COMM\_WORLD, &ProcRank);

    MPI\_Request request;

    int \*blockA, \*blockB, \*blockC, \*A, \*B, \*C;

    double startTime, endTime;

    blockA = (int\*)malloc(ROWS\_A / ProcNum \* COLS\_A \* sizeof(int));

    blockB = (int\*)malloc(ROWS\_B / ProcNum \* COLS\_B \* sizeof(int));

    blockC = (int\*)malloc(ROWS\_A / ProcNum \* COLS\_B \* sizeof(int));

    for (int i = 0; i < ROWS\_A / ProcNum \* COLS\_B; ++i) {

        blockC[i] = 0;

    }

    if (ProcRank == 0) {

        A = (int\*)malloc(ROWS\_A \* COLS\_A \* sizeof(int));

        B = (int\*)malloc(ROWS\_B \* COLS\_B \* sizeof(int));

        C = (int\*)malloc(ROWS\_A \* COLS\_B \* sizeof(int));

        for (int i = 0; i < ROWS\_A \* COLS\_A; ++i) {

            A[i] = 1;

        }

        for (int i = 0; i < ROWS\_B \* COLS\_B; ++i) {

            B[i] = 2;

        }

    }

    int blockSizeA = ROWS\_A / ProcNum \* COLS\_A;

    int blockSizeB = ROWS\_B / ProcNum \* COLS\_B;

    int rowCountInBlockA = ROWS\_A / ProcNum;

    int rowCountInBlockB = ROWS\_B / ProcNum;

    startTime = MPI\_Wtime();

    MPI\_Scatter(A, blockSizeA, MPI\_INT, blockA, blockSizeA, MPI\_INT, 0, MPI\_COMM\_WORLD);

    MPI\_Scatter(B, blockSizeB, MPI\_INT, blockB, blockSizeB, MPI\_INT, 0, MPI\_COMM\_WORLD);

    for (int i = 0; i < ProcNum; ++i) {

        for (int blockCRowIndex = 0; blockCRowIndex < rowCountInBlockA; ++blockCRowIndex) {

            for (int blockCColIndex = 0; blockCColIndex < COLS\_B; ++blockCColIndex) {

                for (int k = 0; k < rowCountInBlockB; ++k) {

                    int startColInA = (i + ProcRank) % ProcNum \* rowCountInBlockB;

                    blockC[COLS\_B \* blockCRowIndex + blockCColIndex] += blockA[COLS\_A \* blockCRowIndex + startColInA + k] \* blockB[COLS\_B \* k + blockCColIndex];

                }

            }

        }

        if (ProcRank == 0) {

            printMatrix(blockC, rowCountInBlockA, COLS\_B);

        }

        MPI\_Isend(blockB, blockSizeB, MPI\_INT, (ProcRank + 1) % ProcNum, 0, MPI\_COMM\_WORLD, &request);

        MPI\_Recv(blockB, blockSizeB, MPI\_INT, (ProcNum + ProcRank - 1) % ProcNum, 0, MPI\_COMM\_WORLD, &Status);

    }

    MPI\_Gather(blockC, ROWS\_A / ProcNum \* COLS\_B, MPI\_INT, C, ROWS\_A / ProcNum \* COLS\_B, MPI\_INT, 0, MPI\_COMM\_WORLD);

    MPI\_Barrier(MPI\_COMM\_WORLD);

    endTime = MPI\_Wtime();

    double total\_time = endTime - startTime;

    if (ProcRank == 0) {

        printf("\nTime of work is %f", total\_time);

        // printMatrix(C, ROWS\_A, COLS\_B);

        free(A);

        free(B);

        free(C);

    }

    free(blockA);

    free(blockB);

    free(blockC);

    MPI\_Finalize();

    return 0;

}

ПРИЛОЖЕНИЕ Б

Код последовательной программы

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <iostream>

#define ROWS\_A 2048

#define COLS\_A 2048

#define ROWS\_B COLS\_A

#define COLS\_B 2048

using namespace std;

int main(int argc, char\* argv[]) {

    int\*\* A = new int\*[ROWS\_A];

    for (int i = 0; i < ROWS\_A; ++i) {

        A[i] = new int[COLS\_A];

        for (int j = 0; j < COLS\_A; ++j) {

            A[i][j] = 1;

        }

    }

    int\*\* B = new int\*[ROWS\_B];

    for (int i = 0; i < ROWS\_B; ++i) {

        B[i] = new int[COLS\_B];

        for (int j = 0; j < COLS\_B; ++j) {

            B[i][j] = 2;

        }

    }

    int\*\* C = new int\*[ROWS\_A];

    for (int i = 0; i < ROWS\_A; ++i) {

        C[i] = new int[COLS\_B];

        for (int j = 0; j < COLS\_B; ++j) {

            C[i][j] = 0;

        }

    }

    clock\_t start\_time = clock();

    for (int i = 0; i < ROWS\_A; i++) {

        for (int j = 0; j < COLS\_B; j++) {

            for (int k = 0; k < ROWS\_B; k++) {

                C[i][j] += A[i][k] \* B[k][j];

            }

        }

    }

    clock\_t end\_time = clock();

    double total\_time = double(end\_time - start\_time) / CLOCKS\_PER\_SEC;

    cout << "Time of work is " << total\_time << " seconds" << endl;

    // for (int i = 0; i < ROWS\_A; ++i) {

    //     for (int j = 0; j < COLS\_B; ++j) {

    //         cout << C[i][j] << " ";

    //     }

    //     cout << endl;

    // }

    for (int i = 0; i < ROWS\_A; ++i) {

        delete[] A[i];

    }

    delete[] A;

    for (int i = 0; i < ROWS\_B; ++i) {

        delete[] B[i];

    }

    delete[] B;

    for (int i = 0; i < ROWS\_A; ++i) {

        delete[] C[i];

    }

    delete[] C;

    return 0;

}