Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №2 по курсу

«Операционные системы»

Группа: М8О-215Б-23

Студент: Моисеев К.В.

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: 10.11.24

Москва, 2024

Постановка задачи

Вариант 6.

Произвести перемножение 2-ух матриц, содержащих комплексные числа

Общий метод и алгоритм решения

Использованные системные вызовы:

1. **CreateFileA**

   Описание: Открывает существующий файл или создает новый файл.

   Использование: В лабораторной работе используется для открытия файла, содержащего матрицы, которые нужно умножить.

   Аргументы:

   - lpFileName - имя файла.
   - dwDesiredAccess - тип доступа к файлу (например, GENERIC_READ для чтения).
   - dwShareMode - режим совместного использования.
   - lpSecurityAttributes - указатель на структуру SECURITY_ATTRIBUTES, которая определяет наследование дескриптора.
   - dwCreationDisposition - действие, которое нужно выполнить (например, OPEN_EXISTING для открытия существующего файла).
   - dwFlagsAndAttributes - атрибуты и флаги файла.
   - hTemplateFile - дескриптор шаблона файла (может быть NULL).

2. **CreateThread**

   Описание: Создает новый поток в текущем процессе.

   Использование: В лабораторной работе используется для создания потоков, которые параллельно вычисляют элементы результирующей матрицы.

   Аргументы:

   - lpThreadAttributes - указатель на структуру SECURITY_ATTRIBUTES, определяющую наследование дескриптора потока.
   - dwStackSize - начальный размер стека для нового потока.
   - lpStartAddress - указатель на функцию, которая будет выполняться в новом потоке.
   - lpParameter - указатель на данные, которые будут переданы в функцию потока.
   - dwCreationFlags - флаги создания потока.
   - lpThreadId - указатель на переменную, которая получает идентификатор нового потока.

3. **ExitThread**

   Описание: Завершает выполнение текущего потока и указывает его код завершения.

   Использование: В лабораторной работе используется для завершения выполнения потока после вычисления элемента результирующей матрицы.

   Аргументы:

   - dwExitCode - код завершения потока.

4. **CloseHandle**

   Описание: Закрывает дескриптор объекта Windows, высвобождая все связанные с ним ресурсы.

   Использование: В лабораторной работе используется для закрытия дескрипторов потоков и семафоров после их использования.

   Аргументы:

   - hObject - дескриптор объекта для закрытия.

5. **WaitForSingleObject**

   Описание: Ожидает завершения указанного объекта (процесса, потока или другого объекта синхронизации).

   Использование: В лабораторной работе используется для ожидания завершения выполнения каждого потока перед завершением основной программы.

   Аргументы:

   - hHandle - дескриптор объекта для ожидания.
   - dwMilliseconds - время ожидания в миллисекундах (INFINITE для бесконечного ожидания).

6. **WaitForMultipleObjects**

   Описание: Ожидает завершения одного или всех объектов в массиве указанных объектов.

   Использование: В лабораторной работе используется для ожидания завершения всех потоков, выполняющих умножение элементов матрицы.

   Аргументы:

   - nCount - количество объектов в массиве.
   - lpHandles - указатель на массив дескрипторов объектов.
   - bWaitAll - флаг, указывающий, нужно ли ждать завершения всех объектов или любого одного.
   - dwMilliseconds - время ожидания в миллисекундах (INFINITE для бесконечного ожидания).

7. **CreateSemaphore**

   Описание: Создает или открывает именованный или неназванный семафор.

   Использование: В лабораторной работе используется для ограничения количества одновременно работающих потоков.

   Аргументы:

   - lpSemaphoreAttributes - указатель на структуру SECURITY_ATTRIBUTES.
   - lInitialCount - начальное количество сигналов (доступных ресурсов).
   - lMaximumCount - максимальное количество сигналов.
   - lpName - имя семафора (может быть NULL для неназванного семафора).

8. **ReleaseSemaphore**

   Описание: Увеличивает количество сигналов семафора, освобождая заданное количество ресурсов.

   Использование: В лабораторной работе используется для освобождения слота семафора после завершения вычисления элемента матрицы потоком.

   Аргументы:

   - hSemaphore - дескриптор семафора.
   - lReleaseCount - количество освобождаемых сигналов.
   - lpPreviousCount - указатель на переменную для получения предыдущего количества сигналов (может быть NULL).

Для начала разберемся, что из себя представляет операция умножения комплексных матриц. В этой операции нам требуются три матрицы: две исходные матрицы и результирующая матрица, которая будет содержать результат умножения.

Для удобного тестирования программа принимает два аргумента: имя файла с матрицами и максимальное количество потоков. Файл должен содержать размеры матриц и сами элементы матриц.

Первая часть программы выполняется в основном потоке и отвечает за чтение матриц из файла и создание необходимых переменных.

Умножение комплексных матриц можно выразить через следующую формулу:

$$[C[i][j] = \sum_{k=0}^{n-1} A[i][k] \times B[k][j]]$$

где (A) и (B) - исходные матрицы, (C) - результирующая матрица, (i) и (j) - индексы строки и столбца соответственно, а (n) - размер матрицы.

Из этой формулы видно, что для вычисления каждого элемента результирующей матрицы требуется пройти по соответствующим строкам и столбцам исходных матриц. Мы используем многопоточность для параллельного вычисления элементов результирующей матрицы.

Максимальное число потоков, работающих одновременно, задается аргументом командной строки. Для синхронизации потоков используется семафор, ограничивающий количество одновременно работающих потоков.

Для оптимизации памяти мы выделяем память только для трех матриц: двух исходных и одной результирующей.

Каждая функция потока вычисляет один элемент результирующей матрицы. Функция принимает структуру данных, содержащую указатели на матрицы, размеры матриц и индексы элемента, который нужно вычислить. После завершения вычисления поток освобождает слот семафора.

Подробный алгоритм:

1. **Чтение матриц из файла**:
    - Открытие файла с помощью CreateFileA.
    - Чтение размера матриц и самих элементов.
    - Закрытие файла с помощью CloseHandle.
2. **Создание потоков**:
    - Создание семафора с помощью CreateSemaphore.
    - Цикл по элементам результирующей матрицы.
    - Для каждого элемента:
        - Ожидание свободного слота семафора с помощью WaitForSingleObject.
        - Создание потока с помощью CreateThread, передача функции вычисления элемента и структуры данных с параметрами.
3. **Вычисление элементов в потоке**:
    - Функция потока принимает структуру данных с указателями на матрицы и индексы элемента.
    - Вычисление значения элемента результирующей матрицы.
    - Освобождение слота семафора с помощью ReleaseSemaphore.
4. **Ожидание завершения всех потоков**:
    - Ожидание завершения всех потоков с помощью WaitForMultipleObjects.
    - Закрытие всех дескрипторов потоков с помощью CloseHandle.
5. **Освобождение памяти**:
    - Освобождение памяти, выделенной для матриц.

Код программы

```c
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

typedef struct {
    double real;
    double imag;
} Complex;

typedef struct {
    Complex **A;
    Complex **B;
    Complex **C;
    int row;
    int col;
    int n;
} ThreadData;

HANDLE semaphore;

Complex complex_multiply(Complex a, Complex b) {
    Complex result;
    result.real = a.real * b.real - a.imag * b.imag;
    result.imag = a.real * b.imag + a.imag * b.real;
    return result;
}

DWORD WINAPI multiply_row_col(LPVOID param) {
    ThreadData *data = (ThreadData *)param;
    Complex sum = {0, 0};
    for (int k = 0; k < data->n; k++) {
        Complex product = complex_multiply(data->A[data->row][k], data->B[k][data->col]);
        sum.real += product.real;
        sum.imag += product.imag;
    }
    data->C[data->row][data->col] = sum;
    ReleaseSemaphore(semaphore, 1, NULL);
    return 0;
}

void multiply_matrices(Complex **A, Complex **B, Complex **C, int n, int max_threads) {
    semaphore = CreateSemaphore(NULL, max_threads, max_threads, NULL);
    HANDLE *threads = (HANDLE *)malloc(n * n * sizeof(HANDLE));
    ThreadData *data = (ThreadData *)malloc(n * n * sizeof(ThreadData));

    int thread_count = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            WaitForSingleObject(semaphore, INFINITE);
            data[thread_count] = (ThreadData){A, B, C, i, j, n};
```

```
            threads[thread_count] = CreateThread(NULL, 0, multiply_row_col,
&data[thread_count], 0, NULL);
            if (threads[thread_count] == NULL) {
                fprintf(stderr, "Error creating thread\n");
                exit(1);
            }
            thread_count++;
        }
    }

    WaitForMultipleObjects(thread_count, threads, TRUE, INFINITE);
    for (int i = 0; i < thread_count; i++) {
        CloseHandle(threads[i]);
    }
    CloseHandle(semaphore);
    free(threads);
    free(data);
}

void read_matrices(const char *filename, Complex ***A, Complex ***B, int *n) {
    HANDLE file = CreateFileA(filename, GENERIC_READ, 0, NULL, OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL, NULL);
    if (file == INVALID_HANDLE_VALUE) {
        fprintf(stderr, "Error opening file\n");
        exit(1);
    }

    DWORD bytesRead;
    char buffer[4096];
    ReadFile(file, buffer, sizeof(buffer), &bytesRead, NULL);
    buffer[bytesRead] = '\0';

    char *ptr = buffer;

    sscanf(ptr, "%d", n);
    while (*ptr && *ptr != '\n') ptr++;
    if (*ptr) ptr++;

    *A = (Complex **)malloc(*n * sizeof(Complex *));
    *B = (Complex **)malloc(*n * sizeof(Complex *));
    for (int i = 0; i < *n; i++) {
        (*A)[i] = (Complex *)malloc(*n * sizeof(Complex));
        (*B)[i] = (Complex *)malloc(*n * sizeof(Complex));
    }

    for (int i = 0; i < *n; i++) {
        for (int j = 0; j < *n; j++) {
            sscanf(ptr, "%lf %lf", &(*A)[i][j].real, &(*A)[i][j].imag);
            while (*ptr && *ptr != ' ' && *ptr != '\n') ptr++;
            if (*ptr) ptr++;
        }
    }

    for (int i = 0; i < *n; i++) {
```

```
        for (int j = 0; j < *n; j++) {
            sscanf(ptr, "%lf %lf", &(*B)[i][j].real, &(*B)[i][j].imag);
            while (*ptr && *ptr != ' ' && *ptr != '\n') ptr++;
            if (*ptr) ptr++;
        }
    }

    CloseHandle(file);
}

void free_matrix(Complex **matrix, int n) {
    for (int i = 0; i < n; i++) {
        free(matrix[i]);
    }
    free(matrix);
}

void print_matrix(Complex **matrix, int n) {
    HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE);
    DWORD written;
    char buffer[256];

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            int len = sprintf(buffer, "(%.2f + %.2fi) ", matrix[i][j].real,
matrix[i][j].imag);
            WriteConsoleA(console, buffer, len, &written, NULL);
        }
        WriteConsoleA(console, "\n", 1, &written, NULL);
    }
}

double get_time() {
    LARGE_INTEGER frequency;
    LARGE_INTEGER start;
    LARGE_INTEGER end;

    QueryPerformanceFrequency(&frequency);
    QueryPerformanceCounter(&start);

    return (double)start.QuadPart / frequency.QuadPart;
}

int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <input_file> <max_threads>\n", argv[0]);
        return 1;
    }

    const char *filename = argv[1];
    int max_threads = atoi(argv[2]);
    if (max_threads <= 0) {
        fprintf(stderr, "Invalid number of threads\n");
        return 1;
```

```
    }

    printf("Reading matrices from file: %s\n", filename);

    int n;
    Complex **A, **B, **C;

    read_matrices(filename, &A, &B, &n);

    printf("Matrix size: %d x %d\n", n, n);

    C = (Complex **)malloc(n * sizeof(Complex *));
    for (int i = 0; i < n; i++) {
        C[i] = (Complex *)malloc(n * sizeof(Complex));
    }

    printf("Multiplying matrices...\n");

    double start_time = get_time();
    multiply_matrices(A, B, C, n, max_threads);
    double end_time = get_time();

    printf("Multiplication complete. Result matrix:\n");

    print_matrix(C, n);

    printf("Time taken: %.6f seconds\n", end_time - start_time);

    free_matrix(A, n);
    free_matrix(B, n);
    free_matrix(C, n);

    printf("Memory freed, program complete.\n");

    return 0;
}
```

Протокол работы программы

Тестирование:

PS C:\Users\klosh\Desktop\OS2\build> .\MatrixMultiplication.exe ..\tests\input.txt 1

Reading matrices from file: ..\tests\input.txt

Matrix size: 2 x 2

Multiplying matrices...

Multiplication complete. Result matrix:

(-17.00 + 53.00i) (5.00 + 45.00i)

(-21.00 + 105.00i) (17.00 + 89.00i)

Time taken: 0.000946 seconds

Memory freed, program complete.

PS C:\Users\klosh\Desktop\OS2\build> .\MatrixMultiplication.exe ..\tests\input.txt 2

Reading matrices from file: ..\tests\input.txt

Matrix size: 2 x 2

Multiplying matrices...

Multiplication complete. Result matrix:

(-17.00 + 53.00i) (5.00 + 45.00i)

(-21.00 + 105.00i) (17.00 + 89.00i)

Time taken: 0.000650 seconds

Memory freed, program complete.

PS C:\Users\klosh\Desktop\OS2\build> .\MatrixMultiplication.exe ..\tests\input.txt 3

Reading matrices from file: ..\tests\input.txt

Matrix size: 2 x 2

Multiplying matrices...

Multiplication complete. Result matrix:

(-17.00 + 53.00i) (5.00 + 45.00i)

(-21.00 + 105.00i) (17.00 + 89.00i)

Time taken: 0.000497 seconds

Memory freed, program complete.

PS C:\Users\klosh\Desktop\OS2\build> .\MatrixMultiplication.exe ..\tests\input.txt 4

Reading matrices from file: ..\tests\input.txt

Matrix size: 2 x 2

Multiplying matrices...

Multiplication complete. Result matrix:

(-17.00 + 53.00i) (5.00 + 45.00i)

(-21.00 + 105.00i) (17.00 + 89.00i)

Time taken: 0.000322 seconds

Memory freed, program complete.

PS C:\Users\klosh\Desktop\OS2\build> .\MatrixMultiplication.exe ..\tests\input.txt 5

Reading matrices from file: ..\tests\input.txt

Matrix size: 2 x 2

Multiplying matrices...

Multiplication complete. Result matrix:

(-17.00 + 53.00i) (5.00 + 45.00i)

(-21.00 + 105.00i) (17.00 + 89.00i)

Time taken: 0.000493 seconds

Memory freed, program complete.

Демонстрация количества потоков:

C:\Users\klosh\Desktop\NtTrace-main\NtTrace-main>NtTrace -filter File,Process C:\Users\klosh\Desktop\OS2\build\MatrixMultiplication.exe C:\Users\klosh\Desktop\OS2\tests\input.txt 3

Process 12016 starting at 00000000004014E0 with command line: "C:\Users\klosh\Desktop\OS2\build\MatrixMultiplication.exe C:\Users\klosh\Desktop\OS2\tests\input.txt 3"

C:\Users\klosh\Desktop\OS2\build\MatrixMultiplication.exe

Loaded DLL at 00007FFC84D90000 ntdll.dll

NtQueryInformationProcess(ProcessHandle=-1, ProcessInformationClass=0x24 [ProcessCookie], ProcessInformation=0x61f418, Length=4, ReturnLength=null) => 0

NtQueryInformationProcess(ProcessHandle=-1, ProcessInformationClass=0x24 [ProcessCookie], ProcessInformation=0x7ffc84f17268, Length=4, ReturnLength=null) => 0

NtQueryInformationProcess(ProcessHandle=-1, ProcessInformationClass=0x24 [ProcessCookie], ProcessInformation=0x61f328, Length=4, ReturnLength=null) => 0

NtQueryInformationProcess(ProcessHandle=-1, ProcessInformationClass=0x24 [ProcessCookie], ProcessInformation=0x61f380, Length=4, ReturnLength=null) => 0

NtOpenFile(FileHandle=0x61f398 [0x5c], DesiredAccess=SYNCHRONIZE|0x20, ObjectAttributes="\??\C:\Users\klosh\Desktop\NtTrace-main\NtTrace-main\", IoStatusBlock=0x61f308 [0/1], ShareAccess=3, OpenOptions=0x21) => 0

NtQueryVolumeInformationFile(FileHandle=0x5c, IoStatusBlock=0x61f308 [0/8], FsInformation=0x61f2f0, Length=8, FsInformationClass=4 [FsDeviceInformation]) => 0

Loaded DLL at 00007FFC83D60000 C:\WINDOWS\System32\KERNEL32.DLL

Loaded DLL at 00007FFC821C0000 C:\WINDOWS\System32\KERNELBASE.dll

NtSetInformationProcess(ProcessHandle=-1, ProcessInformationClass=0x23 [ProcessTlsInformation], ProcessInformation=0x61ed40, Length=0x28) => 0

NtInitializeNlsFiles(BaseAddress=0x61eac0 [0x000c0000], DefaultLocaleId=0x7ffc82528b88 [0x409], DefaultCasingTableSize=null) => 0

NtCreateFile(FileHandle=0x61eb28 [0x60], DesiredAccess=READ_CONTROL|SYNCHRONIZE|0x19f, ObjectAttributes=4:"\Connect", IoStatusBlock=0x61e4e0 [0/0x18], AllocationSize=null, FileAttributes=0, ShareAccess=7, CreateDisposition=2, CreateOptions=0x20, EaBuffer=0x6e6800, EaLength=0x54b) => 0

NtDeviceIoControlFile(FileHandle=0x60, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0x61ea70 [0/0], IoControlCode=0x00500023, InputBuffer=null, InputBufferLength=0, OutputBuffer=0x61ea90, OutputBufferLength=8) => 0

NtSetInformationProcess(ProcessHandle=-1, ProcessInformationClass=0x31 [ProcessOwnerInformation], ProcessInformation=0x61ea98, Length=8) => 0

NtDeviceIoControlFile(FileHandle=0x60, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0x61e860, IoControlCode=0x00500016, InputBuffer=0x61e870, InputBufferLength=0x30, OutputBuffer=null, OutputBufferLength=0) => 0xc00700bb [187 '?? ??????? ????????? ??? ?????????? ????????.']

NtOpenProcessToken(ProcessHandle=-1, DesiredAccess=0x8, TokenHandle=0x61e980 [0x74]) => 0

NtSetWnfProcessNotificationEvent(NotificationEvent=0x84) => 0

Loaded DLL at 00007FFC845E0000 C:\WINDOWS\System32\msvcrt.dll

NtSetInformationProcess(ProcessHandle=-1, ProcessInformationClass=0x23 [ProcessTlsInformation], ProcessInformation=0x61f090, Length=0x28) => 0

Initial breakpoint

NtOpenProcessToken(ProcessHandle=-1, DesiredAccess=0x8, TokenHandle=0x61f320 [0xb0]) => 0

NtQueryVolumeInformationFile(FileHandle=0x54, IoStatusBlock=0x61f070 [0/8], FsInformation=0x61f090, Length=8, FsInformationClass=4 [FsDeviceInformation]) => 0

NtQueryVolumeInformationFile(FileHandle=0x150, IoStatusBlock=0x61f070 [0/8], FsInformation=0x61f090, Length=8, FsInformationClass=4 [FsDeviceInformation]) => 0

NtQueryVolumeInformationFile(FileHandle=0x64, IoStatusBlock=0x61f070 [0/8], FsInformation=0x61f090, Length=8, FsInformationClass=4 [FsDeviceInformation]) => 0

NtDeviceIoControlFile(FileHandle=0x60, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0x61df40 [0/0], IoControlCode=0x00500016, InputBuffer=0x61df50, InputBufferLength=0x30, OutputBuffer=null, OutputBufferLength=0) => 0

Reading matrices from file: C:\Users\klosh\Desktop\OS2\tests\input.txt

NtWriteFile(FileHandle=0x150, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0x61e0e0 [0/0x48], Buffer=0x61e180, Length=0x48, ByteOffset=null, Key=null) => 0

NtCreateFile(FileHandle=0x61eb80 [0xb8], DesiredAccess=SYNCHRONIZE|GENERIC_READ|0x80, ObjectAttributes="\??\C:\Users\klosh\Desktop\OS2\tests\input.txt", IoStatusBlock=0x61eb88 [0/1], AllocationSize=null, FileAttributes=0x80, ShareAccess=0, CreateDisposition=1, CreateOptions=0x60, EaBuffer=null, EaLength=0) => 0

NtReadFile(FileHandle=0xb8, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0x61ece0 [0/0x4b], Buffer=0x61ed40, Length=0x1000, ByteOffset=null, Key=null) => 0

NtDeviceIoControlFile(FileHandle=0x60, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0x61df40 [0/0], IoControlCode=0x00500016, InputBuffer=0x61df50, InputBufferLength=0x30, OutputBuffer=null, OutputBufferLength=0) => 0

Matrix size: 2 x 2

NtWriteFile(FileHandle=0x150, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0x61e0e0 [0/0x14], Buffer=0x61e180, Length=0x14, ByteOffset=null, Key=null) => 0

NtDeviceIoControlFile(FileHandle=0x60, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0x61df50 [0/0], IoControlCode=0x00500016, InputBuffer=0x61df60, InputBufferLength=0x30, OutputBuffer=null, OutputBufferLength=0) => 0

Multiplying matrices...

NtWriteFile(FileHandle=0x150, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0x61e0f0 [0/0x19], Buffer=0x61e190, Length=0x19, ByteOffset=null, Key=null) => 0

Created thread: 17688 at 00000000004015EE

Created thread: 17976 at 00000000004015EE

Created thread: 12204 at 00000000004015EE

NtDeviceIoControlFile(FileHandle=0x60, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0xd0f1b0, IoControlCode=0x00500016, InputBuffer=0xd0f1c0, InputBufferLength=0x30, OutputBuffer=null, OutputBufferLength=0) => 0xc00700bb [187 '?? ??????? ????????? ??? ?????????? ????????.']

Created thread: 19088 at 00000000004015EE

NtDeviceIoControlFile(FileHandle=0x60, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0xf0f1b0, IoControlCode=0x00500016, InputBuffer=0xf0f1c0, InputBufferLength=0x30, OutputBuffer=null, OutputBufferLength=0) => 0xc00700bb [187 '?? ??????? ????????? ??? ?????????? ????????.']

Thread 12204 exit code: 0

NtDeviceIoControlFile(FileHandle=0x60, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0x110f1b0, IoControlCode=0x00500016, InputBuffer=0x110f1c0, InputBufferLength=0x30, OutputBuffer=null, OutputBufferLength=0) => 0xc00700bb [187 '?? ??????? ????????? ??? ?????????? ????????.']

Thread 19088 exit code: 0

Thread 17976 exit code: 0

NtDeviceIoControlFile(FileHandle=0x60, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0xb0f1b0, IoControlCode=0x00500016, InputBuffer=0xb0f1c0, InputBufferLength=0x30, OutputBuffer=null, OutputBufferLength=0) => 0xc00700bb [187 '?? ??????? ????????? ??? ?????????? ????????.']

Thread 17688 exit code: 0

NtDeviceIoControlFile(FileHandle=0x60, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0x61df50 [0/0], IoControlCode=0x00500016, InputBuffer=0x61df60, InputBufferLength=0x30, OutputBuffer=null, OutputBufferLength=0) => 0

Multiplication complete. Result matrix:

NtWriteFile(FileHandle=0x150, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0x61e0f0 [0/0x29], Buffer=0x61e190, Length=0x29, ByteOffset=null, Key=null) => 0

(-17.00 + 53.00i) NtDeviceIoControlFile(FileHandle=0x150, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0x61fa50 [0/0x12], IoControlCode=0x00500016, InputBuffer=0x61fa60, InputBufferLength=0x40, OutputBuffer=null, OutputBufferLength=0) => 0

(5.00 + 45.00i) NtDeviceIoControlFile(FileHandle=0x150, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0x61fa50 [0/0x10], IoControlCode=0x00500016, InputBuffer=0x61fa60, InputBufferLength=0x40, OutputBuffer=null, OutputBufferLength=0) => 0

NtDeviceIoControlFile(FileHandle=0x150, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0x61fa50 [0/1], IoControlCode=0x00500016, InputBuffer=0x61fa60, InputBufferLength=0x40, OutputBuffer=null, OutputBufferLength=0) => 0

(-21.00 + 105.00i) NtDeviceIoControlFile(FileHandle=0x150, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0x61fa50 [0/0x13], IoControlCode=0x00500016, InputBuffer=0x61fa60, InputBufferLength=0x40, OutputBuffer=null, OutputBufferLength=0) => 0

(17.00 + 89.00i) NtDeviceIoControlFile(FileHandle=0x150, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0x61fa50 [0/0x11], IoControlCode=0x00500016, InputBuffer=0x61fa60, InputBufferLength=0x40, OutputBuffer=null, OutputBufferLength=0) => 0

NtDeviceIoControlFile(FileHandle=0x150, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0x61fa50 [0/1], IoControlCode=0x00500016, InputBuffer=0x61fa60, InputBufferLength=0x40, OutputBuffer=null, OutputBufferLength=0) => 0

NtDeviceIoControlFile(FileHandle=0x60, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0x61df40 [0/0], IoControlCode=0x00500016, InputBuffer=0x61df50, InputBufferLength=0x30, OutputBuffer=null, OutputBufferLength=0) => 0

Time taken: 0.018656 seconds

NtWriteFile(FileHandle=0x150, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0x61e0e0 [0/0x1e], Buffer=0x61e180, Length=0x1e, ByteOffset=null, Key=null) => 0

NtDeviceIoControlFile(FileHandle=0x60, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0x61df50 [0/0], IoControlCode=0x00500016, InputBuffer=0x61df60, InputBufferLength=0x30, OutputBuffer=null, OutputBufferLength=0) => 0

Memory freed, program complete.

NtWriteFile(FileHandle=0x150, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0x61e0f0 [0/0x21], Buffer=0x61e190, Length=0x21, ByteOffset=null, Key=null) => 0

NtTerminateProcess(ProcessHandle=0, ExitStatus=0) => 0

Process 12016 exit code: 0

Вывод

Матрица 100 на 100.

Число потоков	Время исполнения (мс)	Ускорение	Эффективность
1	690	1	1
2	440	1,568181818	0,784090909
3	404	1,707920792	0,569306930
4	400	1,725	0.43125
5	387	1,782945736	0,3565891472
6	400	1,725	0.43125

Матрица 500 на 500.

Число потоков	Время исполнения (мс)	Ускорение	Эффективность
1	27983	1	1
2	15203	0,54329414	0,27164707
3	13087	0,4676768	0,15589227
4	12406	0,4433406	0,11083515
5	12444	0,44469857	0,08893971
6	12666	0,45263196	0,07543866

Ускорение показывает во сколько раз применение параллельного алгоритма уменьшает время решения задачи по сравнению с последовательным алгоритмом. Ускорение определяется величиной $S_N=T_1/T_N$, где T_1 - время выполнения на одном потоке, T_N - время выполнения на N потоках.

Эффективность - величина $E_N = S_N/N$, где S_N - ускорение, N - количество используемых потоков.

С увеличением числа потоков скорость выполнения увеличивается из-за параллелизации, которая позволяет использовать несколько ядер процессора одновременно. Это улучшает производительность за счет:

- Разделения задачи: Работа делится на независимые части, которые выполняются параллельно.
- Эффективного использования ресурсов: Многопоточность позволяет лучше использовать многопроцессорные системы.
- Сокращения времени выполнения: Несколько потоков могут обрабатывать разные части данных одновременно, уменьшая общее время выполнения задачи.

Причины вариации времени выполнения

1. **Операционная система**:
   - **Планировщик процессов**: Операционная система управляет планированием процессов и потоков. В зависимости от нагрузки системы, другие процессы и потоки могут конкурировать за CPU, что вызывает вариации в времени выполнения.
   - **Контекстное переключение**: слишком большое количество потоков может привести к увеличению контекстного переключения, что добавляет накладные расходы и может уменьшить производительность.
2. **Аппаратные ресурсы**:
   - **Кэш-память**: Разные потоки могут конкурировать за кэш-память процессора, что влияет на производительность.
   - **Гиперпоточность (Hyper-Threading)**: если используется гиперпоточность, потоки могут конкурировать за одни и те же физические ядра, что также может повлиять на производительность.
3. **Память и ввод-вывод (I/O)**:
   - **Доступ к памяти**: Потоки могут конкурировать за доступ к памяти, что приводит к задержкам.
   - **I/O операции**: если программа активно использует операции ввода-вывода, это может вызвать дополнительные задержки.

Вывод

Лабораторная работа была выполнена, поставленная задача решена. Мне удалось написать программу, выполняющую действие с использованием множества потоков. Все потоки удалось синхронизировать. Работу было очень увлекательно выполнять. Было трудно заставить компилятор запускать программу. Исходя из результатов тестирования программы с разным числом данных и с разным количеством потоков можно прийти к выводу, что при грамотном использование потоков существенно ускоряется работа программы. Для того, чтобы ускорение

было максимальным требуется выбирать количество потоков исходя из вычислительной способности компьютера и объема задачи, которая будет возложена на каждый из процессов.