Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей
Кафедра информатики
дисциплина архитектура вычислительных систем

ОТЧЁТ

к лабораторной работе №3
на тему

Интерфейс OpenMP

Выполнил: студент группы 053505
Слуцкий Никита Сергеевич

Проверил: ассистент кафедры информатики
Калиновская Анастасия Александровна

Минск 2022

Цель работы
Изучить использование интерфейса OpenMP для программирования простых многопоточных приложений.

Краткие теоретические сведения
Исполняемый процесс в памяти может состоять из нескольких (множества) вычислительных потоков, имеющих общее адресное пространство, но собственный контекст, состоящий из подмножества значений регистров общего назначения и сегментных регистров, а также собственного стека.

В различных операционных системах, реализованных на базе разных аппаратных платформ, многозадачность реализована по-разному: на прикладном программном уровне – в зависимости от предлагаемого операционной системой интерфейса для создания многозадачных приложений, на уровне операционных систем – в зависимости от того, в какой степени и каким образом аппаратура (в частности микропроцессор) позволяет реализовывать параллельное и/или одновременное выполнение нескольких потоков инструкций.

OpenMP – набор спецификаций, определяющих интерфейс прикладного уровня для реализации многопоточных приложений. Данный интерфейс может быть реализован разработчиками компиляторов языков C++ и Fortran как надмножество соответствующего языка. Программа, использующая интерфейс OpenMP, может быть собрана любым компилятором, реализующим поддержку этого интерфейса. Таким образом, интерфейс OpenMP призван обеспечивать кросс-платформенный способ для создания многопоточных приложений.

OpenMP использует модель fork-join. Запущенная программа состоит из единственного потока, называемого потоком-мастером. В некоторый момент выполнение может быть распределено между несколькими потоками, образующими команду потоков во главе с потоком-мастером, называемом в этом контексте мастером команды. Чаще всего предполагается, что после окончания выполнения своей части работы эти потоки будут завершены, и управление будет вновь передано потоку-мастеру, выполнение которого затем снова может быть распределено.

OpenMP спроектирован таким образом, что зачастую переход к его использованию для оптимизации существующего программного кода требует минимальных временных затрат благодаря его простоте и интуитивной понятности.

Ход выполнения лабораторной работы

В рамках лабораторной работы реализована программа, выполняющая операцию матричного умножения для матриц большого размера. Реализация представляется в двух вариантах: синхронном – использующем единственный поток для выполнения операции, и асинхронном – использующем для этого несколько потоков. Асинхронная реализация выполнена с использованием интерфейса OpenMP.

Размер матриц выбирается достаточно большим, чтобы время операции было ощутимым. В примере данной работы – 1200*1200. Такой размер выбран с учётом желаемого времени выполнения операции в синхронном варианте – около 5 секунд.

Так как OpenMP – это интерфейс может быть реализован в рамках компиляторов для C++, для его применения был выбран данный язык программирования.

Был использован установленный на компьютере компилятор из семейства GNU с оболочкой MinGW. Исходные файлы собирались с помощью сборщика CMake, а не с помощью традиционных инструментов от Microsoft Visual Studio. Особый интерес составляет тот факт, что без подключения (предварительной настройки) интерфейса программа запускается без ошибок, но и, соответственно, отрабатывает вычисления лишь в одном потоке. Для активации OpenMP потребовалось видоизменить CMakeLists.txt файл до следующего вида:

```
cmake_minimum_required(VERSION 3.19)
project(LR)

set(CMAKE_CXX_STANDARD 20)

find_package(OpenMP)
if (OPENMP_FOUND)
    set (CMAKE_C_FLAGS "${CMAKE_C_FLAGS} ${OpenMP_C_FLAGS}")
    set (CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${OpenMP_CXX_FLAGS}")
    set (CMAKE_EXE_LINKER_FLAGS "${CMAKE_EXE_LINKER_FLAGS} ${OpenMP_EXE_LINKER_FLAGS}")
endif()

add_executable(LR main.cpp)
```

Асинхронная реализация с применением интерфейса OpenMP:

```
#pragma clang diagnostic push
#pragma ide diagnostic ignored "openmp-use-default-none"
#include <iostream>
#include <omp.h>


const std::size_t kThreadNumber{10};
const std::size_t kRowsPerThread{120};
const constexpr std::size_t kMatrixSize{kThreadNumber * kRowsPerThread};


typedef struct
{
    std::size_t first_row;
    std::size_t last_row;
} TDATA, *PTDATA;


int op1[kMatrixSize][kMatrixSize];
int op2[kMatrixSize][kMatrixSize];
int res[kMatrixSize][kMatrixSize];


void initialize_operands();

void stdoutput_result();

int thread_multiplier(PTDATA data);


int main()
{
    PTDATA pThreadData[kThreadNumber];

    initialize_operands();

    for (std::size_t counter{0}; counter < kThreadNumber; ++counter)
    {
        //initializing TDATA params
        pThreadData[counter] = new TDATA;

        pThreadData[counter]->first_row = kRowsPerThread * counter;
        pThreadData[counter]->last_row = kRowsPerThread * (counter + 1) - 1;
    }

    #pragma omp parallel for
    for (std::size_t i = 0; i < kThreadNumber; ++i)
    {
        std::cout << "\nStarted thread # " << omp_get_thread_num() << '\n';
        thread_multiplier(pThreadData[i]);
        std::cout << "\nFinished thread # " << omp_get_thread_num() << '\n';
    }


    std::cout << "\nmultiplication finished\n";
    //stdoutput_result();

    return 0;
}


int thread_multiplier(PTDATA data)
{
    const auto first_row{data->first_row};
    const auto last_row{data->last_row};

    for (std::size_t i{first_row}; i <= last_row; ++i)
    {
        for (std::size_t j = 0; j < kMatrixSize; ++j)
        {
            res[i][j] = 0;

            for (std::size_t k = 0; k < kMatrixSize; ++k)
                res[i][j] += op1[i][k] * op2[k][j];
        }
    }

    return 0;
}

void initialize_operands()
{
    for (std::size_t row{0}; row < kMatrixSize; ++row)
        for (std::size_t col{0}; col < kMatrixSize; ++col)
            op1[row][col] = op2[row][col] = 1;
}

void stdoutput_result()
```

```
{
    for (std::size_t i{0}; i < kMatrixSize; ++i)
    {
        for (std::size_t j{0}; j < kMatrixSize; ++j)
            std::cout << res[i][j] << ' ';

        std::cout << '\n';
    }
}
```

Вычисление с использованием многопоточности занимает в несколько раз времени меньше, чем вычисление, производимое в одном потоке.

Вывод

В результате проделанной работы был изучен способ реализации многопоточных приложений с использованием интерфейса OpenMP.

Реализована программа, применяющая интерфейс OpenMP для выполнения операции перемножения двух матриц очень большого размера. Получено представление о преимуществах от применения многопоточности для выполнения ресурсоёмких вычислительных задач.