Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное

образовательное учреждение высшего образования

«Южно-Уральский государственный университет

(национальный исследовательский университет)»

Высшая школы электроники и компьютерных наук

Кафедра системного программирования

ОТЧЕТ  
о лабораторной работе №7  
по дисциплине «Технологии параллельного программирования»

Выполнил:   
студент группы КЭ-220   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Голенищев А. Б.   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.   
   
Отчет принял:   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Жулев А. Э.  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.

***Задание 30. Проект в среде Visual Studio 2010 с поддержкой MPI и OpenMP***

Настроили проект в QtCreator для комбинированного использования MPI и OPenMP, листнинг 1.

TEMPLATE = app

CONFIG += console c++17

CONFIG -= app\_bundle

CONFIG -= qt

# Флаги компилятора и линковки для OpenMP

QMAKE\_CXXFLAGS += -fopenmp

QMAKE\_LFLAGS += -fopenmp

# Указываем компилятор MPI

QMAKE\_CC = mpicc

QMAKE\_CXX = mpic++

# Путь к заголовочным файлам MPI

INCLUDEPATH += /usr/include/openmpi

# Библиотеки MPI

LIBS += -lmpi\_cxx -lmpi -lpthread -lrt

# Дополнительные флаги компиляции

QMAKE\_CXXFLAGS += -Bsymbolic-functions

SOURCES += \

main.cpp

Листнинг 1. Настройка файла проекта \*.pro в QtCreator

***Задание 31. Программа «I am»***

Разработали программу выводв номеров процессов MPI и их потоков OpenMP, листнинг 2. Представлен результат ее работы, рисунок 1.

#include <mpi.h>

#include <omp.h>

#include <cstdio>

// Golenishchev Artem, KE-220 Task 31

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

int n; // Количество нитей

if (argc != 2) {

printf("Usage: %s <number\_of\_threads>\n", argv[0]);

return 1;

}

n = std::stoi(argv[1]);

MPI\_Init(&argc, &argv);

int world\_size, world\_rank;

MPI\_Comm\_size(MPI\_COMM\_WORLD, &world\_size); // Общее количество процессов

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &world\_rank); // Номер текущего процесса

// Рассчитаем общее количество гибридных нитей

int total\_hybrid\_threads = n \* world\_size;

// Параллельный блок OpenMP

#pragma omp parallel num\_threads(n)

{

int thread\_num = omp\_get\_thread\_num(); // Номер нити

printf("I am %d thread from %d process. Number of hybrid threads = %d\n",

thread\_num, world\_rank, total\_hybrid\_threads);

}

MPI\_Finalize();

return 0;

}

Листнинг 2. Код программы «мастер-рабочие»

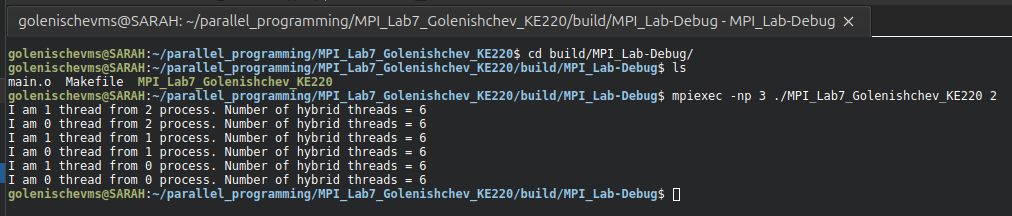


Рисунок 2. Результат выполнения программы «I am»

***Задание 32. Программа «Число 𝜋π»:***

Разработали программу обмена сообщениями между процессами от всех каждому, листнинг 3. Представлен результат ее работы, рисунок 2.

Листнинг 3. Код программы вычисления числа π

#include <mpi.h>

#include <omp.h>

#include <iostream>

#include <iomanip>

#include <cmath>

// Golenishchev Artem, KE-220 Task 32

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

MPI\_Init(&argc, &argv);

int world\_size, world\_rank;

MPI\_Comm\_size(MPI\_COMM\_WORLD, &world\_size); // Количество MPI процессов

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &world\_rank); // Ранг текущего процесса

long long N;

if (world\_rank == 0) {

if (argc != 2) {

std::cerr << "Usage: " << argv[0] << " <number\_of\_intervals>" << std::endl;

MPI\_Abort(MPI\_COMM\_WORLD, 1); }

N = std::stoll(argv[1]); }

MPI\_Bcast(&N, 1, MPI\_LONG\_LONG, 0, MPI\_COMM\_WORLD);

long long chunk\_size = N / world\_size;

long long start = world\_rank \* chunk\_size;

long long end = (world\_rank == world\_size - 1) ? N : start + chunk\_size;

double step = 1.0 / static\_cast<double>(N); // Шаг разбиения

double local\_sum = 0.0;

#pragma omp parallel for reduction(+:local\_sum)

for (long long i = start; i < end; ++i) {

double x = (i + 0.5) \* step;

local\_sum += 4.0 / (1.0 + x \* x);

}

local\_sum \*= step;

double global\_sum = 0.0;

MPI\_Reduce(&local\_sum, &global\_sum, 1, MPI\_DOUBLE, MPI\_SUM, 0, MPI\_COMM\_WORLD);

// Вывод результата на главном процессе

if (world\_rank == 0) {

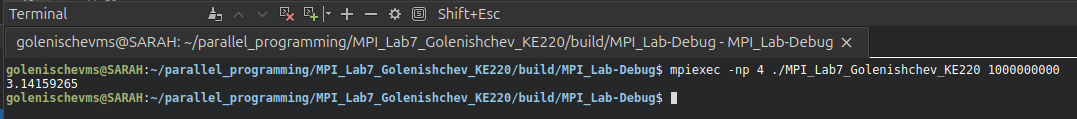
std::cout << std::fixed << std::setprecision(8) << global\_sum << std::endl;

}

MPI\_Finalize();

return 0;

}

Рисунок 2. Результат выполнения программы вычисления числа π

Программа вычисляет число π с использованием метода прямоугольников (метод левых или средних прямоугольников). Сначала общее количество разбиений делится между MPI-процессами, каждый из которых получает свой диапазон индексов. Каждый процесс в свою очередь распараллеливает вычисления внутри этого диапазона с помощью OpenMP, распределяя итерации между потоками. Внутри цикла вычисляется значение функции в точках, равномерно распределённых на отрезке, и суммируется в локальную переменную. Для более точного результата точки берутся в середине каждого интервала. После завершения вычислений локальные суммы от всех процессов объединяются с использованием MPI-операции `MPI\_Reduce`, чтобы получить глобальную сумму, которая на главном процессе преобразуется в значение π и выводится.

***Ответы на вопросы:***

1. Для какого класса архитектур параллельных вычислительных систем предназначена гибридная (MPI + OpenMP) технология программирования?

Гибридная технология MPI + OpenMP предназначена для распределённых систем с общей и распределённой памятью, таких как кластеры с многоядерными узлами.

1. Какими объектами ОС она оперирует?

MPI оперирует процессами, которые взаимодействуют через сообщения, а OpenMP работает с потоками, совместно использующими память в пределах одного процесса.

1. Не противоречит ли ответу на предыдущий вопрос запуск данной программы на Вашем ноутбуке? Какую архитектуру он имеет?

Нет, запуск не противоречит, поскольку ноутбук с многоядерным процессором поддерживает многопоточность и может эмулировать взаимодействие MPI-процессов в пределах одного устройства.

1. Можно ли в коде программы поменять порядок вложения блоков MPI и OpenMP (MPI\_Init-MPI\_Finalize внутри параллельного региона OpenMP)? Почему?

Нельзя, так как MPI требует инициализации перед использованием, а OpenMP-потоки запускаются уже внутри процесса, который должен быть инициализирован через MPI.

1. Прокомментируйте порядок вывода номеров процессов и нитей.

Порядок вывода номеров процессов и нитей не гарантирован, так как выполнение потоков и процессов происходит асинхронно, что приводит к перемешанным результатам.

1. Исходя из каких соображений в Задании № 32 выбран способ распределения итераций по процессам (количество, порядок выбора итераций)?

Итерации распределяются равномерно между процессами для балансировки нагрузки и минимизации межпроцессорного взаимодействия.

1. Исходя из каких соображений в Задании № 32 выбран способ распределения итераций по нитям (статический/динамический, размер чанка)?

Используется статическое распределение, так как работа равномерно разделена и динамическое распределение добавило бы излишние накладные расходы.

1. Замерьте время выполнения расчетов при различном количестве нитей и процессов, приведите результаты в таблице. Сравните со временем выполнения аналогичных заданий из предыдущих разделов. Прокомментируйте результат.

Результаты зависят от конфигурации системы. С увеличением числа нитей и процессов наблюдается ускорение до определённого предела, после которого накладные расходы на синхронизацию и коммуникацию начинают замедлять выполнение.

***Выводы:***

Изучили использование гибридного подхода MPI + OpenMP для эффективного решения задач параллельных вычислений. На практике освоили распределение нагрузки между процессами и потоками, взаимодействие процессов через сообщения, а также параллельное выполнение задач в пределах одного узла. Мы разобрались в особенностях методов распределения работы, оценили производительность на разных конфигурациях, и научились применять эти подходы для задач, требующих высокой точности и производительности, таких как вычисление числа π.