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.

Челябинск
2024

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»

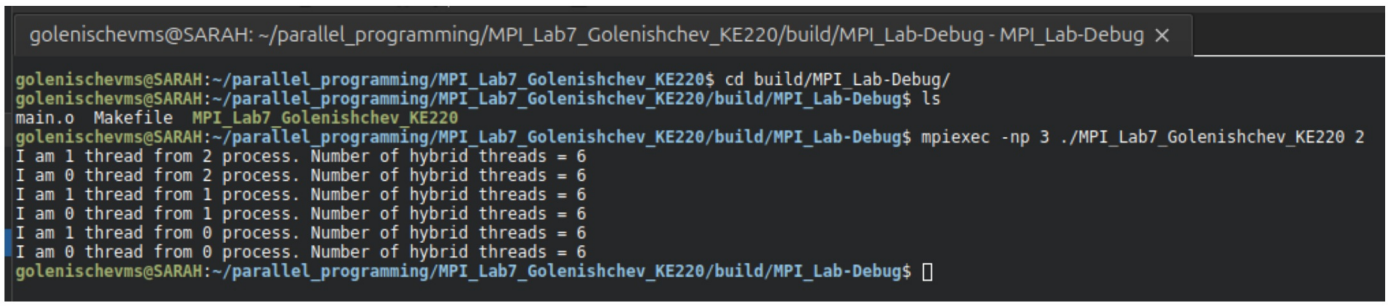

Рисунок 2. Результат выполнения программы «I am»

Задание 32. Программа «Число π»:

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

```
#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;
```

Листнинг 3. Код программы вычисления числа π

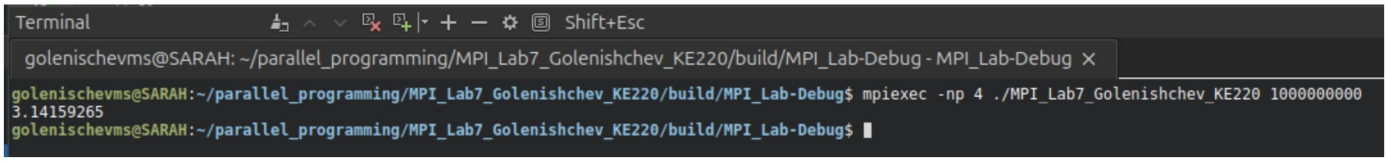

Рисунок 2. Результат выполнения программы вычисления числа π

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.

2. Какими объектами ОС она оперирует?

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

3. 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.

4. 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.

5. 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.

6. 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.

7. 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.

8. 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.