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 № 1

по курсу   
Параллельное программирование

Группа 6408

Студент Н.В. Носов

(*подпись*)

Преподаватель,

к.ф.-м.н. Е.С. Козлова

(*подпись*)

Самара 2023

ЗАДАНИЕ

**Произвести запуск программ** «Hello world»**, которые используют технологии MPI и OpenMP, на различном количестве процессоров (потоков). В ходе анализа работы программы оценить время ее выполнения на различном количестве исполняющих нитей (процессов), для чего в заготовках необходимо произвести изменения. Здесь и далее для уменьшения влияния сторонних факторов необходимо проводить усреднение времени выполнения не менее чем по 12 запускам.**

Таблица 1 – Исходные данные на ЛР № 1

|  |  |
| --- | --- |
| Количество процессов/потоков | (6, 12) |

ВВЕДЕНИЕ

В современном мире есть много задач, для решения которых необходимо использовать огромные вычислительные мощности. Также постоянно возрастают требования к точности решения и скорости выполнения вычислений. Один из основных способов решения этих проблем – создание параллельных вычислительных систем.

В данной лабораторной работе показаны результаты вычисления времени выполнения программы «Hello World!» при параллельном запуске на определенном количестве процессов с использованием технологий MPI и OpenMP.

ЭКСПЕРИМЕНТАЛЬНЫЕ ИССЛЕДОВАНИЯ

2.1 Характеристики ПК

В таблице 2 представлены характеристики ПК.

Таблица 2 – Характеристики ПК

|  |  |
| --- | --- |
| Процессор | AMD Ryzen 5 4500U with Radeon Graphics 2.38 GHz |
| Количество ядер | 6 |
| Количество потоков | 6 |
| Оперативная память | 8 GB |
| Тип системы | 64-bit operating system, x64-based processor |

2.2 Результаты работы программ

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

На рисунках 1-2 представлены скрины запуска и работы программ.

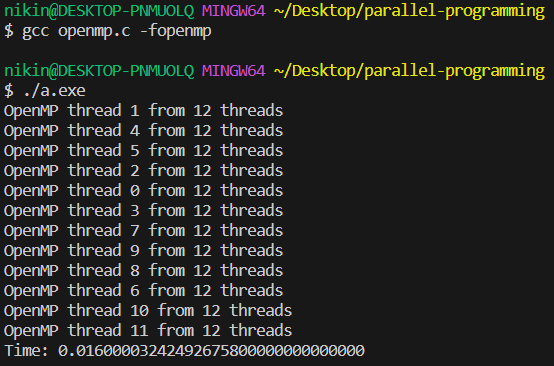


Рисунок 1 – Пример работы программы OpenMP на 12 процессах

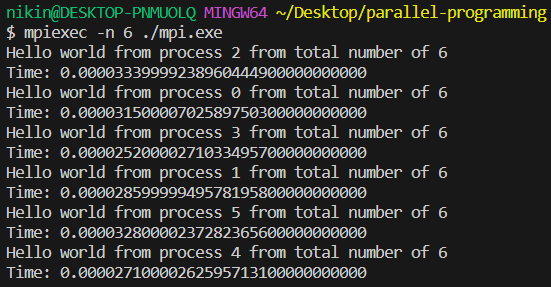


Рисунок 2 – Пример работы программы MPI на 6 процессах

Последовательная программа представляла собой вызовы функции printf, количество которых соответствует количеству нитей в параллельном варианте программы. Время работы последовательной программы составило 0.8 мс для 6 вызовов и 1.5 мс для 12 вызовов.

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

Таблица 3 – Результаты выполнения программ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во процессов, шт | OpenMP | | MPI | |
| Время, мc | Ускорение | Время, мс | Ускорение |
| 6 | 6,3333 | 0,126 | 0,05 | 16 |
| 12 | 5,4166 | 0,278 | 0,04 | 37,5 |

На рисунке 3 приведен график зависимости времени работы программ от количества процессов. На рисунке 4 приведен график зависимости ускорения программ от количества процессов.

Рисунок 3 – Время работы программ

Рисунок 4 – Ускорение программ

**ВЫВОДЫ:**

Из полученных результатов видно, что:

1. При увеличении числа процессоров или нитей (в зависимости от технологии) изменяется время работы программы, ускорение увеличивается по сравнению с последовательным вариантом для технологии MPI, а для OpenMP наоборот, уменьшается. При этом можем заметить, что время выполнения параллельного алгоритма c технологией MPI оказалось меньше, чем у OpenMP, так как OpenMP тратит больше время на создание потоков. Также вывод производится в одну консоль, время уходит на борьбу ресурсов между нитями.
2. Программа с технологией MPI даёт минимальное ускорение равное 16, а максимальное – 37,5. Программа с технологией OpenMP даёт минимальное ускорение равное 0,126, а максимальное – 0,278. Технология OpenMP показала ускорение меньше единицы, что говорит нам о том, что она только замедлила работу программы, так как создание нитей и процессов является довольно ресурсоемким процессом.
3. Программа, которая реализована с помощью MPI, работает примерно в 126 раз быстрее, чем программа, реализованная при помощи OpenMP, что можно объяснить спецификой данных технологий.
4. Полученные результаты требуют дальнейшего изучения и уточнения ввиду, в частности, отсутствия синхронизации процессов для MPI программы, в то время как при выходе из параллельной области происходит автоматическая синхронизация процессов в OpenMP. Также требуется отдельное тестирование функции замеров времени: MPI\_Wtime(), omp\_get\_wtime(), std::chrono::steady\_clock::now(), для получения согласованных результатов.

ЗАКЛЮЧЕНИЕ

Цель лабораторной работы – написать параллельные программы вывода текстового сообщения на экран с использованием технологий MPI и OpenMP и сравнить время выполнения с длительностью последовательной программы достигнута. Показано, что использование параллельных технологий для данного типа программ не требуется, в виду того, что происходят затраты ресурсов на саму реализацию параллелизма, которые увеличивают время выполнения программы. Лучше всего себя показала технология OpenMP. Это можно объяснить тем, что изначально она предназначена для распараллеливания программ на одной машине с несколькими ядрами или процессорами, а MPI в свою очередь предназначен для распараллеливания на кластерах.

В ходе выполнения лабораторной работы я изучил основы MPI и OpenMP, приобрел навыки по написанию параллельных программ с использованием вышеперечисленных технологий. Наиболее сложной частью выполнения лабораторной работы был анализ полученных результатов в виде построения графиков и таблиц. Интерес вызвало изучение основ MPI и OpenMP, также приобретение навыков по написанию параллельных программ с использованием вышеперечисленных технологий.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Ma, W. Parallel multibody separation simulation using MPI and OpenMP with communication optimization [Текст] / W. Ma, X. Hu, X. Liu // Journal of Algorithms & Computational Technology. – 2018. – Vol. 13. – P. 1-17.
2. Козлова, Е.С. Лабораторные работы по курсу «Параллельное программирование»: Методические указания [Текст] / Сост. Е.С. Козлова, А.С. Широканев − Самара, 2019. – 61 с.
3. оеводин, В. В. Параллельные вычисления [Текст] / В. В. Воеводин, Вл. В. Воеводин. — СПб.: БХВ-Петербург, 2002. — 608 с.
4. Богачёв К.Ю. Основы параллельного программирования: учебное пособие, 2-е изд. [Текст] / К. Ю. Богачёв ‑ М. : БИНОМ. Лаборатория знаний, 2013. ‑ 344 с.
5. Гергель, В. П. Теория и практика параллельных вычислений, 2-е изд. [Текст] / B. II. Гергель. — М.: Интуит. 2016. - 500 с.
6. 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.
7. 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 программы с технологией MPI

#include "mpi.h"

#include "stdio.h"

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

int rank, ranksize, i;

MPI\_Init(&argc, &argv);

double start\_time, finish\_time;

start\_time = MPI\_Wtime();

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

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

printf("Hello world from process %d from total number of %d\n", rank, ranksize);

finish\_time = MPI\_Wtime() - start\_time;

printf("Time: %.32f\n", finish\_time);

MPI\_Finalize();

return 0;

}

ПРИЛОЖЕНИЕ Б

Код программы с технологией OpenMP

#include <omp.h>

#include <stdlib.h>

#include <stdio.h>

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

omp\_set\_num\_threads(12);

int nThreads, threadNum;

double start\_time = omp\_get\_wtime();

#pragma omp parallel private(nThreads, threadNum)

{

nThreads = omp\_get\_num\_threads();

threadNum = omp\_get\_thread\_num();

printf("OpenMP thread %d from %d threads \n", threadNum, nThreads);

}

double finish\_time = omp\_get\_wtime();

double result\_time = finish\_time - start\_time;

printf("Time: %.32f\n", result\_time);

return 0;

}

ПРИЛОЖЕНИЕ В

Код последовательной программы

#include <omp.h>

#include <stdio.h>

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

double result\_time = 0;

double start\_time = omp\_get\_wtime();

for (int i = 0; i < 6; ++i)

{

printf("Hello World %d \n", i);

}

double finish\_time = omp\_get\_wtime();

result\_time += finish\_time - start\_time;

printf("Result time = %.32f\n", result\_time);

return 0;

}