ИИКС

Лабораторная работа №5: «Технология MPI. Введение»

Грущин Илья

Б21-515

2023 г.

Описание используемой рабочей среды

Виртуальная машина: VirtualBox

Модель процессора: 11th Gen Intel(R) Core(TM) i5-11400 @ 2.70GHz, ядер: 6, логических процессоров: 12.

Логических процессоров выделенных под виртуальную машину — 6.

Объем оперативной памяти: 6.0 Гб

Тип оперативной памяти: DDR4

Операционная система: Fedora Linux 38 (Workstation Edition)

Разрядность ОС: 64

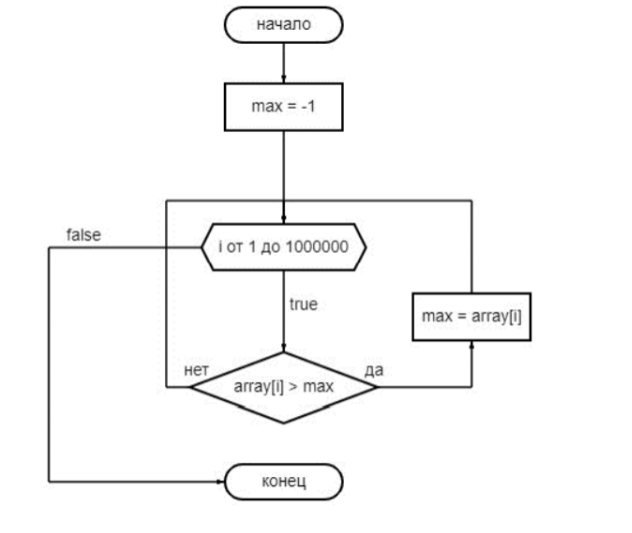
Среда разработки: gcc 13.2.1

Поддерживаемая версия OpenMP: 4.5

Анализ приведенного алгоритма

Используется алгоритм поиска максимума в массиве путем поиска локальных максимумов на подмассивах и поиска максимумов из всех. Количество подмассивов определяется количеством потоков.

Блок-схема алгоритма:



Описание используемых директив и функций OpenMP и MPI

* #pragma omp parallel num\_threads(threads) shared(array, count) reduction(max: max) - для настройки зоны работы openMP
* #pragma omp for - для определения цикла, который будет разделен на потоки
* MPI\_Init(&argc, &argv) - инициализирует MPI
* MPI\_Comm\_size(MPI\_COMM\_WORLD, &size) - узнать сколько всего потоков. MPI\_COMM\_WORLD - глобальный коммуникатор, который имеет доступ ко всем запущенным потокам.
* MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank) — возвращает номер процесса. Нужно для распараллеливания.
* MPI\_Bcast(array, count, MPI\_INTEGER, 0, MPI\_COMM\_WORLD) - отправить всем потокам массив.
* MPI\_Wtime() - астрономическое время в секундах.
* MPI\_Reduce() - Reduce по потокам. В этой программе — найти максимум из локальных максимумов.
* MPI\_Finalise() - завершить не основной поток.

Графики

На всех графиках ось абсцисс — количество потоков.

Заключение

В данной работе был реализован алгоритм нахождения максимума на массиве с помощью двух технологий - MPI и OpenMP. Изучены основные функции MPI для инициализации среды выполнения, пересылки данных между процессами и завершения работы с MPI, а также методы работы с процессами.

В результате измерения времени двух реализаций были построены графики времени работы, ускорения и эффективности. Реализации использующие MPI работали медленнее, чем OpenMP. Причина тому — отсутствие в MPI общей памяти, и данные приходится пересылать между процессами, в то время как в OpenMP возможно использование одних и тех же данных разными потоками.

Затраты на содержание n различных процессов, а также на пересылку данных между процессами внутри ограниченной в ресурсах виртуальной машины выше, нежели содержание n потоков.

Приложение

Код файла openmp.c

#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

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

{

    if (argc != 3)

    {

        printf("Usage: ./lab5 (threads) (seed)\n");

        return 0;

    }

    srand(atoi(argv[2]));

    const int count = 1000000000;

    const int threads = atoi(argv[1]);

    int \*arr = (int \*)malloc(count \* sizeof(int));

    for (int i = 0; i < count; ++i)

    {

        arr[i] = rand();

    }

    int max = arr[0];

    double begin = omp\_get\_wtime();

    #pragma omp parallel num\_threads(threads) shared(arr, count) reduction(max : max) default(none)

    {

        #pragma omp for

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

        {

            max = (arr[i] > max) ? arr[i] : max;

        }

    }

    double end = omp\_get\_wtime();

    printf("max: %d\n", max);

    FILE \*fp = fopen("log.txt", "a");

    fprintf(fp, "%d %lf\n", threads, (end - begin));

    return 0;

}

Код файла mpi.c

#include <stdlib.h>

#include <stdio.h>

#include <mpi.h>

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

{

    if (argc != 2)

    {

        printf("Usage: ./lab5 (seed)\n");

        return 0;

    }

    int size, rank;

    const int count = 100000000;

    const int random\_seed = atoi(argv[1]);

    double begin, end;

    int \*arr, lmax, max;

    FILE \*fp;

    MPI\_Init(&argc, &argv);

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

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

    arr = (int \*)malloc(count \* sizeof(int));

    if (!rank)

    {

        srand(random\_seed);

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

        {

            arr[i] = rand();

        }

    }

    MPI\_Bcast(arr, count, MPI\_INTEGER, 0, MPI\_COMM\_WORLD);

    const int wstart = rank \* count / size;

    const int wend = (rank + 1) \* count / size;

    MPI\_Barrier(MPI\_COMM\_WORLD);

    if (!rank)

    {

        begin = MPI\_Wtime();

    }

    lmax = arr[0];

    for (int i = wstart; i < wend; ++i)

    {

        if (arr[i] > lmax)

        {

            lmax = arr[i];

        }

    }

    MPI\_Reduce(&lmax, &max, 1, MPI\_INTEGER, MPI\_MAX, 0, MPI\_COMM\_WORLD);

    free(arr);

    if (!rank)

    {

        end = MPI\_Wtime();

    }

    MPI\_Finalize();

    if (!rank)

    {

        printf("Maximum is %d\n", max);

        fp = fopen("log1.txt", "a");

        fprintf(fp, "%d %lf\n", size, end - begin);

    }

    return 0;

}

Приложение

Таблицы с теоретическими результатами и результатами вычислительных экспериментов Способ вычисления ускорения: T1 – время работы алгоритма с одним потоком

Tn – время работы алгоритма с n потоками

Sn – ускорение с n потоками, Sn = T1/Tn

En – эффективность с n потоками, En = Sn/n

