Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

“НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО”

Факультет Программной инженерии и компьютерной техники

Направление подготовки (специальность) Системное и прикладное ПО

ОТЧЕТ

Лабораторная работа №4
по предмету «Параллельные вычисления»

Тема проекта: «Метод доверительных интервалов при измерении времени выполнения параллельной OpenMP-программы».

Обучающийся Ткаченко В.В. (**Фамилия И.О.**) Р4114 (**номер группы**)

Преподаватель Жданов А. Д. (**Фамилия И.О.**)

Санкт-Петербург
2023 г.

Содержание

Описание решаемой задачи

Вариант: (576)

Map: 6 | 5

Merge: 1

Sort: 6

1. В программе, полученной в результате выполнения ЛР-3, так изменить этап Generate, чтобы генерируемый набор случайных чисел не зависел от количества потоков, выполняющих программу. Например, на каждой итерации i перед вызовом rand_r можно вызывать функцию srand(f(i)), где f – произвольно выбранная функция. Можно придумать и использовать любой другой способ.

2. Заменить вызовы функции gettimeofday на omp_get_wtime.

3. Распараллелить вычисления на этапе Sort, для чего выполнить сортировку в два этапа: ˆ Отсортировать первую и вторую половину массива в двух независимых нитях (можно использовать OpenMP-директиву ”parallel sections”); ˆ Объединить отсортированные половины в единый массив.

4. Написать функцию, которая один раз в секунду выводит в консоль сообщение о текущем проценте завершения работы программы. Указанную функцию необходимо запустить в отдельном потоке, параллельно работающем с основным вычислительным циклом.

5. Обеспечить прямую совместимость (forward compatibility) написанной параллельной программы.

Провести эксперименты, варьируя N от min(Nx/2 , N1) до N2, где значения N1 и N2 взять из ЛР-1, а Nx – это такое значение N, 94 при котором накладные расходы на распараллеливание превышают выигрыш от

распараллеливания. Написать отчёт о проделанной работе. Подготовиться к устным вопросам на защите.

7. Необязательное задание на «четвёрку» и «пятёрку». Уменьшить количество итераций основного цикла с 100 до 10 и провести эксперименты, замеряя время выполнения следующими методами:

- Использование минимального из десяти полученных замеров;

- Расчёт по десяти измерениям доверительного интервала с уровнем доверия 95%.

Привести графики параллельного ускорения для обоих методов в одной системе координат, при этом нижнюю и верхнюю границу доверительного интервала следует привести двумя независимыми графиками.

8. Необязательное задание на «пятёрку»: в п.3 задания на этапе Sort выполнить параллельную сортировку не двух частей массива, а k частей в k нитях (тредах), где k – это количество процессоров (ядер) в системе, которое становится известным только на этапе выполнения программы с помощью команды «k = omp_get_num_procs()».

Краткая характеристика «железа»

Имя ОС:	Майкрософт Windows 10 Pro
Версия:	10.0.19045 Сборка 19045
Изготовитель:	LENOVO
Модель:	20BE009ART
Тип:	Компьютер на базе x64
SKU системы:	LENOVO_MT_20BE
Процессор:	Intel(R) Core(TM) i7-4710MQ
Версия BIOS:	LENOVO GMET85WW (2.33), 30.05.2018

Версия SMBIOS: 2.7

Версия встроенного контроллера: 1.14

Режим BIOS: Устаревший

gcc version: 11.3.0 (Ubuntu 11.3.0-1ubuntu1~22.04)

WSL2

Листинг программы lab4.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <float.h>
#include <unistd.h>
#include <sys/time.h>
#include <string.h>
#ifdef _OPENMP
#include <omp.h>
#else
int omp_get_thread_num() { return 0; }
void omp_set_num_threads(int num_threads) {}
void omp_set_dynamic(int dynamic_threads) {}
double omp_get_wtime() { return 0; }
#endif

void parallel_insertion_sort(double arr[], int n)
{
    int i, j;
    double key;

#pragma omp parallel num_threads(2)
    {
        int thread_id = omp_get_thread_num();
        int start = thread_id * n / 2;
        int end = (thread_id + 1) * n / 2;

        for (i = start + 1; i < end; i++)
        {
            key = arr[i];
            j = i - 1;

            while (j >= start && arr[j] > key)
            {
                arr[j + 1] = arr[j];
                j = j - 1;
```

```
            }

            arr[j + 1] = key;
        }
    }
}

void merge_arrays(double arr[], int n, double M2_copy[])
{
    int mid = n / 2;
    double *left = arr;
    double *right = arr + mid;
    int i = 0;

    while (left < arr + mid && right < arr + n)
    {
        if (*left <= *right)
        {
            M2_copy[i] = *left;
            left++;
        }
        else
        {
            M2_copy[i] = *right;
            right++;
        }
        i++;
    }

    while (left < arr + mid)
    {
        M2_copy[i] = *left;
        left++;
        i++;
    }

    while (right < arr + n)
    {
        M2_copy[i] = *right;
        right++;
        i++;
    }

    memcpy(arr, M2_copy, n * sizeof(double));
}

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

    int i, j, N, M;
```

```
    double e = exp(1.0);
    double T1, T2;
    double delta_ms;
    double min = 1;
    double max = 576;
    int progress = 0;
    T1 = omp_get_wtime();
    double min_time = DBL_MAX;

    omp_set_nested(1);
#pragma omp parallel sections num_threads(2) shared(i, progress)
    {
#ifdef _OPENMP
#pragma omp section
        {
            double time = 0;
            while (progress < 1)
            {
                double time_temp = omp_get_wtime();
                if (time_temp - time < 1)
                {
                    usleep(100);
                    continue;
                };
                printf("\rPROGRESS: %d", i);
                fflush(stdout);
                time = time_temp;
            }
        }
#endif
#pragma omp section
        {
            N = atoi(argv[1]);
            M = atoi(argv[2]);

            double *restrict M1 = (double *)malloc(N * sizeof(double));
            double *restrict M2 = (double *)malloc(N / 2 * sizeof(double));
            double *restrict M2_copy = (double *)malloc(N / 2 * sizeof(double));

            unsigned int seed;
            unsigned int *restrict seed1 = &seed;
            unsigned int *restrict seed2 = &seed;

#if defined(_OPENMP)
            omp_set_dynamic(0);
            omp_set_num_threads(M);
#endif

            for (i = 0; i < 100; i++)
            {
```

```
                //------------------GENERATE---------------------//
                double min_nonzero = INFINITY;
                double sum_sin = 0.0;
                seed = i;

                for (j = 0; j < N; j++)
                {
                    M1[j] = ((double)rand_r(seed1) / (RAND_MAX)) * (max - min) +
min;
                }
                for (j = 0; j < N / 2; j++)
                {
                    M2[j] = ((double)rand_r(seed2) / (RAND_MAX)) * (max * 10 -
max) + max;
                }
                //-----------------------------------//

#pragma omp parallel default(none) shared(M1, M2, M2_copy, i, min, max, e, N,
sum_sin, min_nonzero)
                {
#pragma omp for
                    for (j = 0; j < N; j++)
                    {
                        M1[j] = cbrt(M1[j] / e);
                    }

#pragma omp for
                    for (j = 0; j < N / 2; j++)
                    {
                        M2_copy[j] = M2[j];
                    }

#pragma omp for
                    for (j = 1; j < N / 2; j++)
                    {
                        if (j == 0)
                        {
                            M2[0] = log(fabs(tan(M2[0])));
                        }
                        else
                        {
                            M2[j] = log(fabs(tan(M2[j] + M2_copy[j - 1])));
                        }
                    }

#pragma omp for
                    for (j = 0; j < N / 2; j++)
                    {
                        M2[j] = pow(M1[j], M2[j]);
                    }
```

```
                    parallel_insertion_sort(M2, N / 2);

#pragma omp single
                    {
                        int j;
                        for (j = 0; j < N / 2 - 1 && M2[j] == 0; j++)
                            ;
                        min_nonzero = M2[j];
                    }

#pragma omp for reduction(+ : sum_sin)
                    for (j = 0; j < N / 2; j++)
                    {
                        if ((int)(M2[j] / min_nonzero) % 2 == 0)
                        {
                            sum_sin += sin(M2[j]);
                        }
                    }
#pragma omp barrier
                }
            }
            progress = 1;

            free(M1);
            free(M2);
            free(M2_copy);
        }
    }
    T2 = omp_get_wtime();
    delta_ms = (T2 - T1) * 1000;
    printf("\r%f\n", delta_ms);

    return 0;
}
```

Результаты экспериментов

Работа выполнялась с помощью компилятора gcc.

Разделение сортировки на две секции:

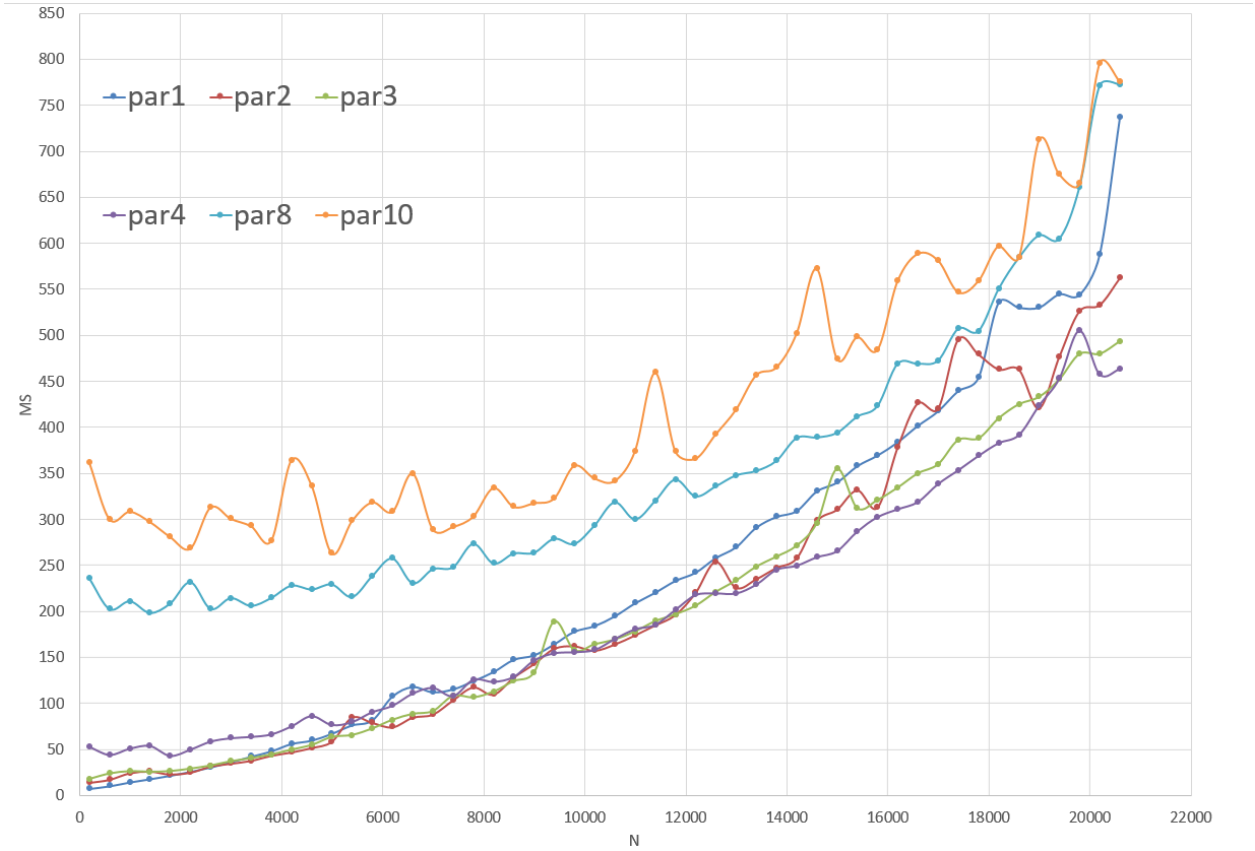

Рис.1 Время выполнения программы

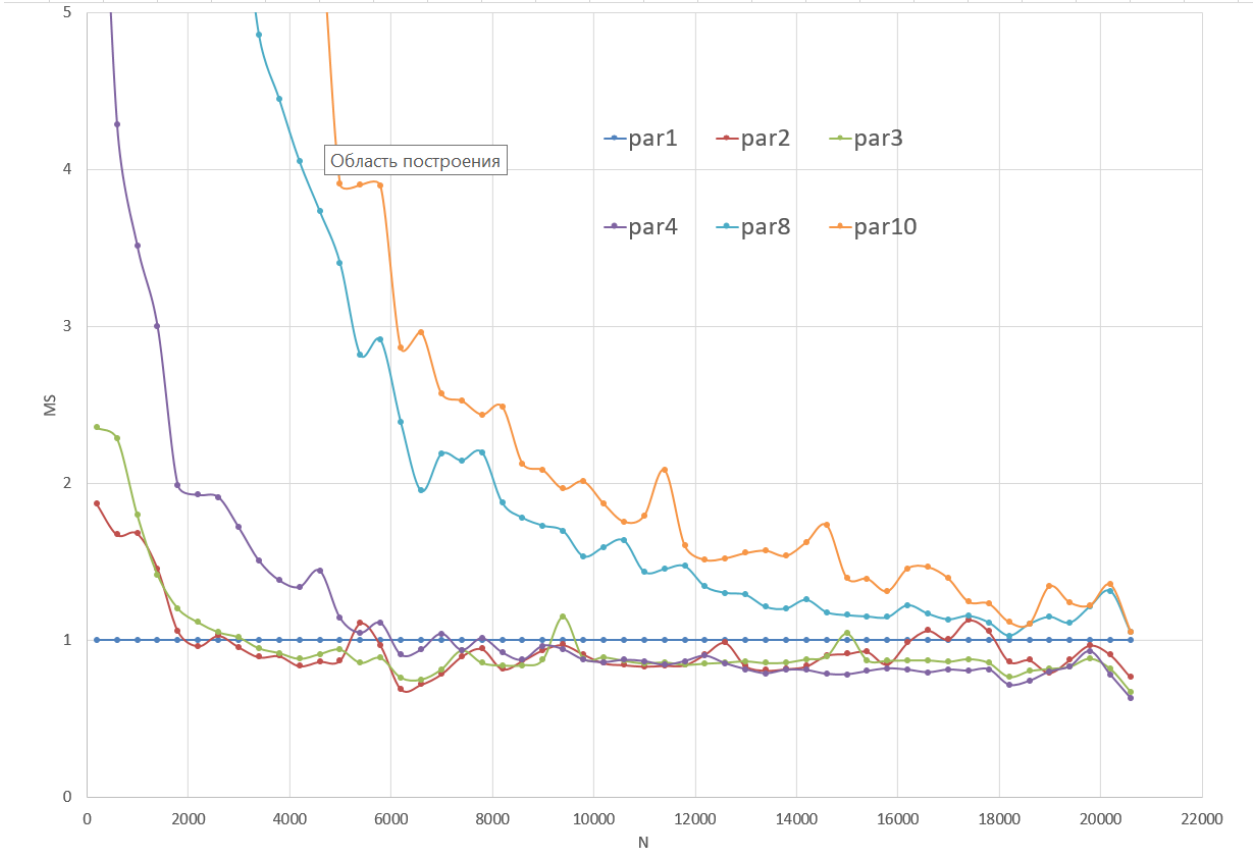

Рис.2 Параллельное ускорение

Самый ощутимый прирост дает разделение сортировки на 2 потока, далее прирост варьируется в пределах погрешности и его можно обусловить разным поведением системы под нагрузкой. Т.к. сортировка использует только 2 секции, то дальнейшее увеличение числа потоков не оказывает существенного влияния.

Далее представлены графики времени выполнения программы с 10 итерациями для методов 1.минимальное из десяти 2.доверительный интервал

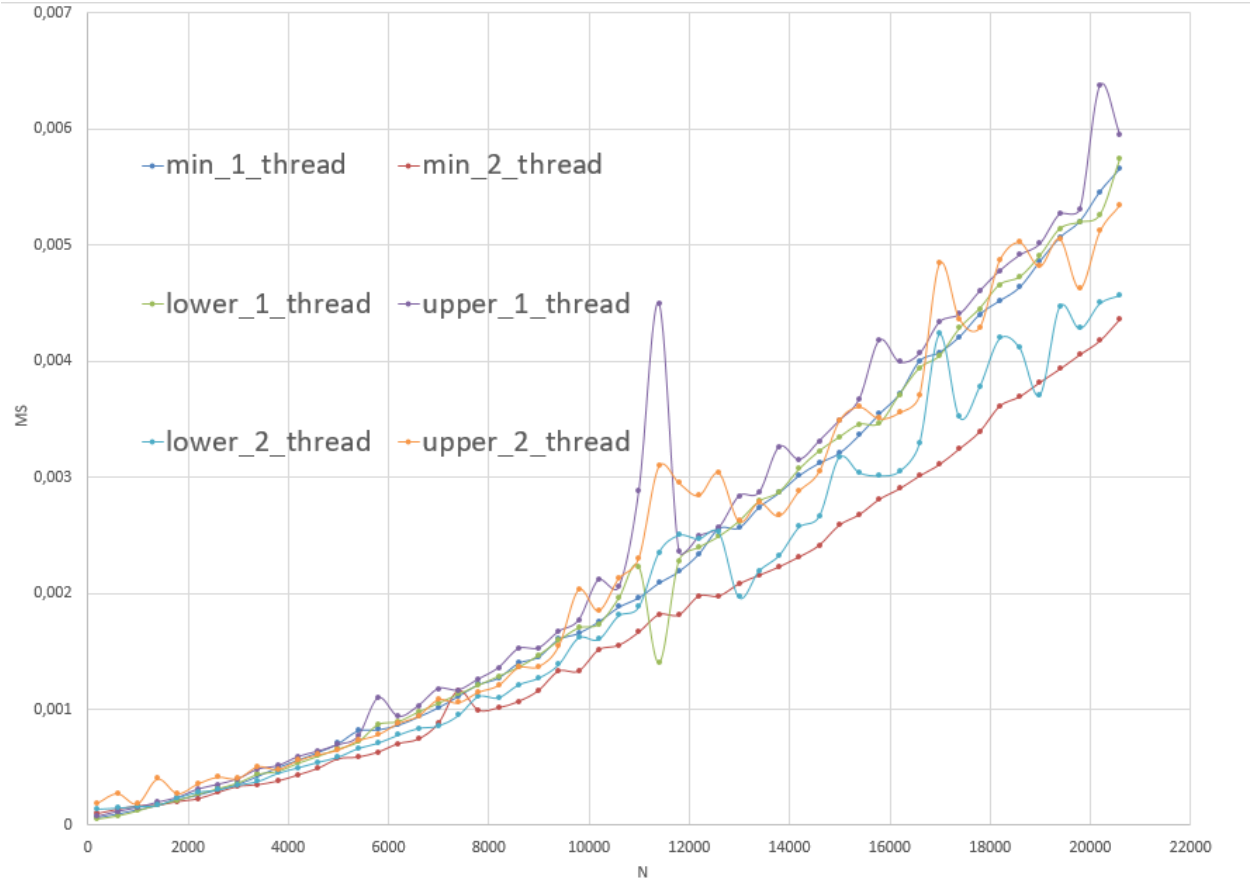

Рис. 3 время выполнения

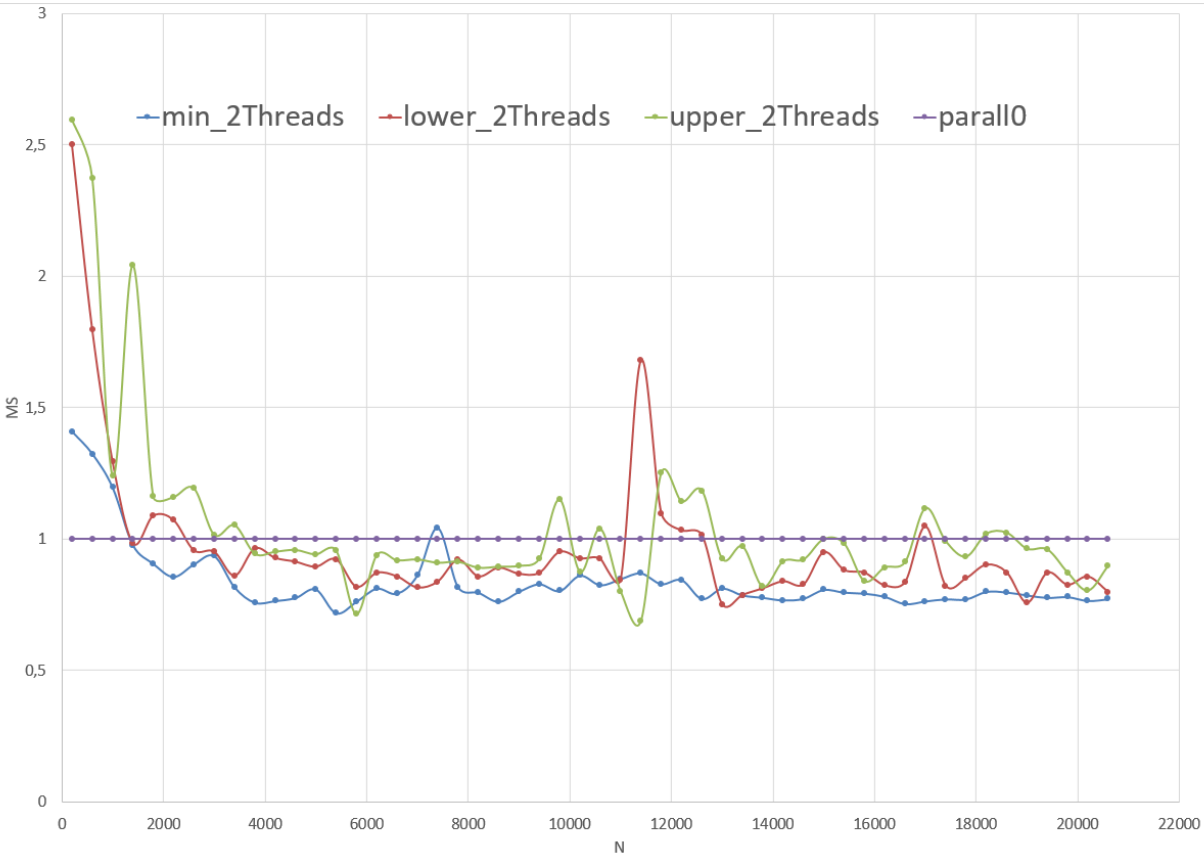

Рис.4 параллельное ускорение

Вывод

Разделение сортировки на две секции дает прирост производительности, однако при указании числа потоков больше количества физических ядер, наблюдается серьезное ухудшение производительности.