МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ имени М.В.ЛОМОНОСОВА
Факультет Вычислительной Математики и Кибернетики

Практикум по курсу

"Суперкомпьютеры и параллельная обработка данных"

Разработка параллельной версии программы для решения определенного интеграла методом Симпсона

ОТЧЕТ

о выполненном задании

Выполнил студент 313 группы

Хаметов Марк Владиминович

edu-cmc-skpod22-313-7

Москва, 2021 г.

Оглавление

1 Постановка задачи

Ставится задача нахождения значения интеграла на интервале.

Дана функция π/`2 * |sin X|` , дан интервал [A;B] = [0; 10π], требуется получить значение

$$\int_0^{10\pi} \frac{1}{2}\pi\,|\sin(x)|\,dx = 10\pi \approx 31.416$$

.

Требуется:

1. Реализовать параллельный алгоритм с помощью библиотеки параллельного программирования OpenMP.
2. Исследовать масштабируемость программы построив график времени выполнения программы от числа используемых потоков и при разном количестве интервалов разбиения функции.

2 Описание

2.1 Основа: последовательный алгоритм

По следующей формуле была написана программа:

$$\int_a^b f(x)dx \approx \int_a^b p_2(x)dx = \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right),$$

Алгоритм имеет вид:

```
double f(double x) {  return 1.57 * (fabs(sin(x)));}//pi/2 * |sin|  ->  f(k*pi) = k*pi

int main ()
{
    int segnum = 200000000;
    int i;
    double otvet = 0.0;
    double x_l, x_r, x_mid;//точка (слева, справа, центр) на интервале
    double h = (B - A) / (double) segnum;// шаг
    {
        for (i = 0; i < segnum; i++)
        {
            x_l =   A + h * (double) i;
            x_mid = A + h * (double) (i + 0.5);
            x_r =   A + h * (double) (i + 1);
            otvet +=  h/6.0 * (f(x_l) + 4.0 * f(x_mid) + f(x_r));
        }
    }
    printf("%.20f  ~~ pi*10\n", otvet);
```

Этот алгоритм имеет сложность O(nm), где m сложность функции, а n сложность метода Симпсона.

2.2 Параллельный алгоритм

```
    start = omp_get_wtime();
//    omp_set_num_threads(1);
    #pragma omp parallel default (none) private (i, x_l, x_r, x_mid) shared (segnum, h,treads_used) reduction(+: otvet)
    {
    treads_used = omp_get_num_threads();
        #pragma omp for schedule (static)
        for (i = 0; i < segnum; i++)
        {
            x_l =   A + h * (double) i;
            x_mid = A + h * (double) (i + 0.5);
            x_r =   A + h * (double) (i + 1);
            otvet +=  h/6.0 * (f(x_l) + 4.0 * f(x_mid) + f(x_r));
        }
    }
    end = omp_get_wtime();
    printf("%d  threads\n", treads_used);
    printf("%f seconds\n", end - start);
    printf("%.20f  ~~ pi*10\n", otvet);
```

omp_get_wtime() используется дважды для получения времени исполнения подсчетов

omp_set_num_threads(кол-во); используется для ограничения количества нитей исполняемого сегмента программы (альтернатива OMP_NUM_THREADS=кол-во в командной строке)

omp_get_num_threads() используется для подсчета кол-ва нитей выделенных на данный момент

default(none) возвращает ошибку при компиляции, если в параллельном сегменте есть переменные, которые не описаны в строке #pragma …

private(имя_переменной) указывает, что каждый поток должен иметь свою копию переменной

shared(имя_переменной) указывает, что переменная общая и доступна каждому потоку

reduction(операция: имя_переменной) указывает, что переменная private и подлежит указанной операции в конце работы параллельного региона. В нашем случае она складывает результаты работы каждой нити (переменная otvet) перед завершением многопоточного сегмента.

3 Результаты замеров времени выполнения

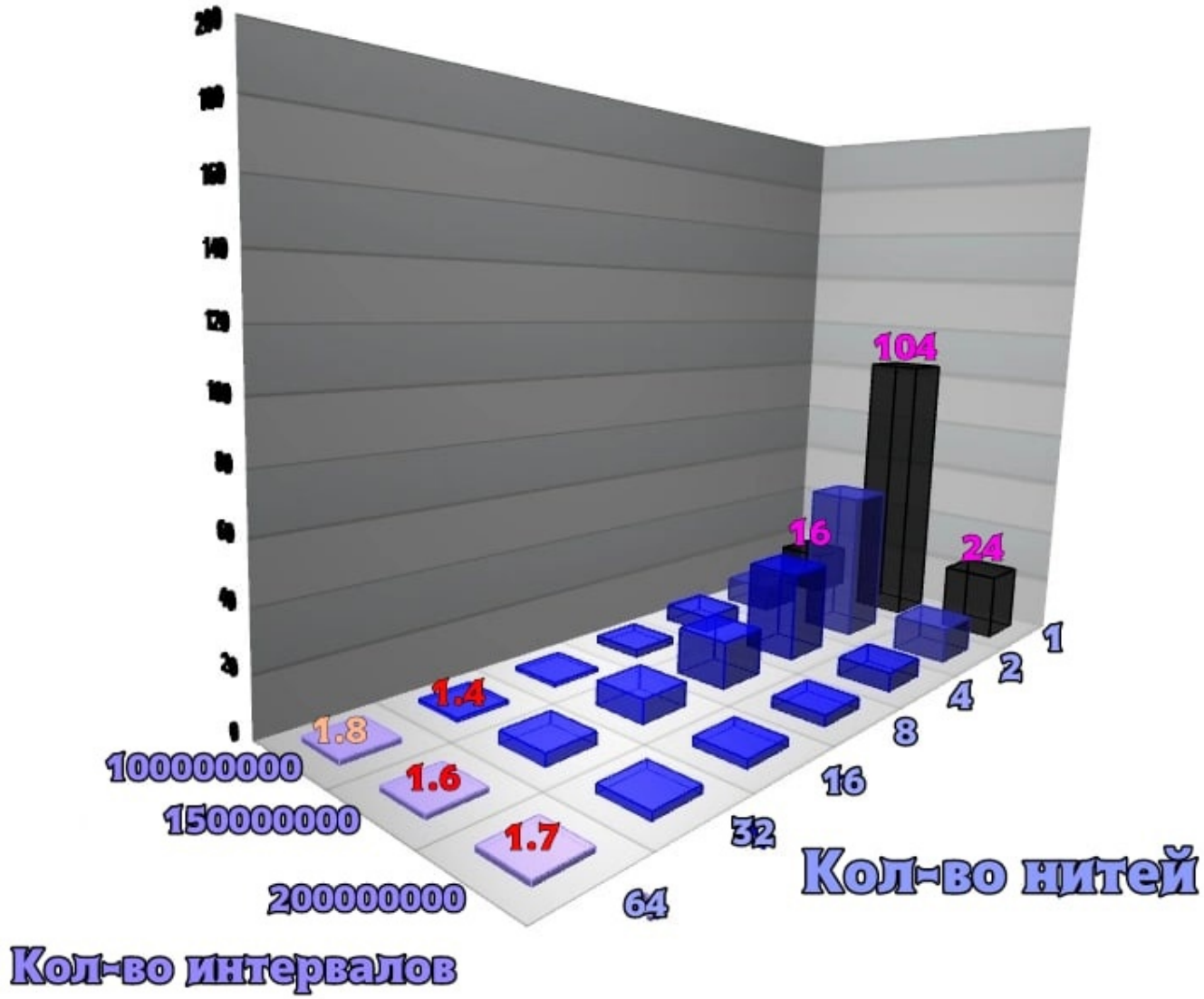
Программа запускалась со следующими параметрами:

- Количество интервалов {100000000, 150000000, 200000000}
- Количество потоков {1,2,4,8,16,32.64}

3.1 Таблица

	1	2	4	8	16	32	64
100000000	16.438371	9.604401	6.211589	2.942551	1.553967	1.395267	1.782826
150000000	103.976037	54.349175	32.007572	16.559334	8.378800	4.331188	1.596898
200000000	24.257653	12.677348	6.730313	3.956137	3.138084	2.870995	1.739337

3.2 График



104/1.6 = 65.0

16/1.4 ~~ 11.42857

24/1.7 ~~ 14.1176

4 Анализ результатов

1) В случае 100 000 000 интервалов 32 нити оказалось ближе к оптимальному количеству нитей чем 64. Это связано с тем, что управление нитями, то есть, например, выделение памяти на приватные переменные, тоже занимает время. При увеличении количества интервалов эта проблема исчезает.
2) Тестирование в случае 150 000 000 интервалов проходило на следующее утро. Наблюдается резкий скачок в количестве времени на исполнение программы во всех случаях, кроме использования 64 нитей. Вероятно проблема может заключаться в одной из следующих причин: выборе интервалов неудобного для деления размера или единоразовой заминкой при подсчете. Не могу опровергнуть эти предположение без дальнейших тестов, что подчеркивает необходимость проводить тесты по несколько раз.
3) Были найдены значения количества нитей и интервалов близкие к оптимальным, что можно предположить из 2 следующих фактов. Tmin100 000 000 < Tmin150 000 000 < Tmin200 000 000. Во всех случаях было получено примерно правильное значение интеграла.

5 Выводы

Изучена библиотека OpenMP для написания параллельных программ. Проанализировано время исполнения на суперкомпьютере Полюс. Были получены приблизительно оптимальные условия подсчета задачи со сложностью O(mn), что может помочь в определении условий подсчета задач с подобной сложностью, а может и не помочь, так как надо больше тестировать. Вероятно не поможет так как из-за разных начальных условий системные вызовы оказываются менее/более значительными.

Использование OpenMP дает значительный выигрыш во времени исполнения тяжелых программ. Выигрыш достигал от 11 до 65 раз.