Задание 1
Многопоточное вычисление числа π с помощью библиотеки pthreads

1. Задание

Задача: Реализовать параллельный алгоритм с использованием интерфейса POSIX Threads, вычисляющий число π, как интеграл:

$$\pi = \int_0^1 \frac{4}{1+x^2}\, dx$$

методом прямоугольников.

Чтобы повысить скорость вычислений, многопоточность используется для распределения вычислительных задач между различными процессорными ядрами, тестирования производительности при различном количестве потоков и разделах и расчета коэффициента ускорения по времени.

2. Экспериментальная среда

Эксперимент проводился на следующих системах：

- Operation System：Ubuntu 20.04.5 LTS
- Architecture: x86_64
- Model name：Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
- CPU(s):1
- Memory:8 GB
- gcc (Ubuntu 9.4.0-1ubuntu1~20.04.2) 9.4

Эксперимент реализован на языке C и использует библиотеку POSIX threading (pthread) для многопоточных вычислений.

3. Экспериментальный проект

Мы используем интервалы разбиения $1*10^8$, $2*10^8$, $3*10^8$, $4*10^8$, $5*10^8$, $6*10^8$, $7*10^8$, $8*10^8$, $9*10^8$и 10^9 для вычислений и используем от 1 до 20 потоков соответственно, чтобы проверить точность вычисления π и время выполнения программы в каждом конкретном случае. Результаты расчетов отображаются в табличной форме, и одновременно записывается время выполнения каждого эксперимента.

Чтобы оценить производительность параллельных алгоритмов, мы вычисляем коэффициент ускорения：

$$S_p = \frac{T_1}{Tp}$$

где *T1* — время работы программы на одной нити, *Tp* — время работы на p нитях.

Сначала я отслеживала ситуацию с одним вводом (код сохраняется в файле *calculate_pi.c*), вручную введите интервал разделения и количество потоков в интервале [0, 1].

```
collapsor@ubuntu:~/Desktop/tspp_shihui/sem01$ gcc -Wall -Werror -pthread -o calc
ulate_pi calculate_pi.c
collapsor@ubuntu:~/Desktop/tspp_shihui/sem01$ ./calculate_pi 100 4
Estimated value of pi: 3.141600986923125
Elapsed time: 0.000123 seconds
```

```
collapsor@ubuntu:~/Desktop/tspp_shihui/sem01$ ./calculate_pi 1000000 5
Estimated value of pi: 3.141592653589920
Elapsed time: 0.006852 seconds
collapsor@ubuntu:~/Desktop/tspp_shihui/sem01$ ./calculate_pi 100000000 1
Estimated value of pi: 3.141592653590426
Elapsed time: 1.199259 seconds
collapsor@ubuntu:~/Desktop/tspp_shihui/sem01$ ./calculate_pi 100000000 2
Estimated value of pi: 3.141592653590022
Elapsed time: 1.195440 seconds
collapsor@ubuntu:~/Desktop/tspp_shihui/sem01$ ./calculate_pi 100000000 3
Estimated value of pi: 3.141592653590154
Elapsed time: 1.216145 seconds
collapsor@ubuntu:~/Desktop/tspp_shihui/sem01$ ./calculate_pi 100000000 4
Estimated value of pi: 3.141592653590217
Elapsed time: 1.235401 seconds
collapsor@ubuntu:~/Desktop/tspp_shihui/sem01$ ./calculate_pi 100000000 5
Estimated value of pi: 3.141592653589644
Elapsed time: 1.836632 seconds
collapsor@ubuntu:~/Desktop/tspp_shihui/sem01$ ./calculate_pi 100000000 6
Estimated value of pi: 3.141592653589672
Elapsed time: 1.675764 seconds
```

Чтобы сделать данные более наглядными, я улучшила вторую строку кода. В файле calculate_pi2.c , я использовала интервалы разбиения $1*10^8$, $2*10^8$,

$3*10^8$, $4*10^8$, $5*10^8$, $6*10^8$, $7*10^8$, $8*10^8$, $9*10^8$и 10^9 для вычислений и используем от 1 до 20 потоков соответственно и сохранила результаты в файле *pi_results.csv*.

4. Результаты

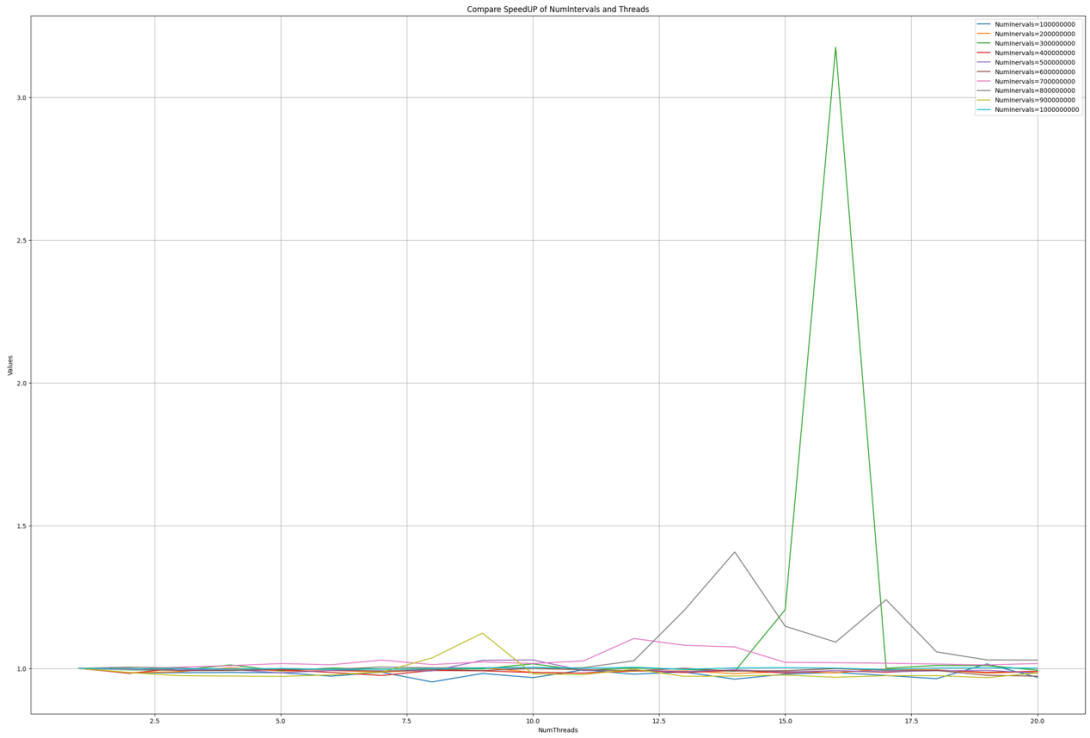
значение	**этикетки**
Количество интервалов	NumIntervals
Количество потоков	NumThreads
Значение π	PiValue
Время выполнения	ElaspsedTime
Ускорение	SpeedUp

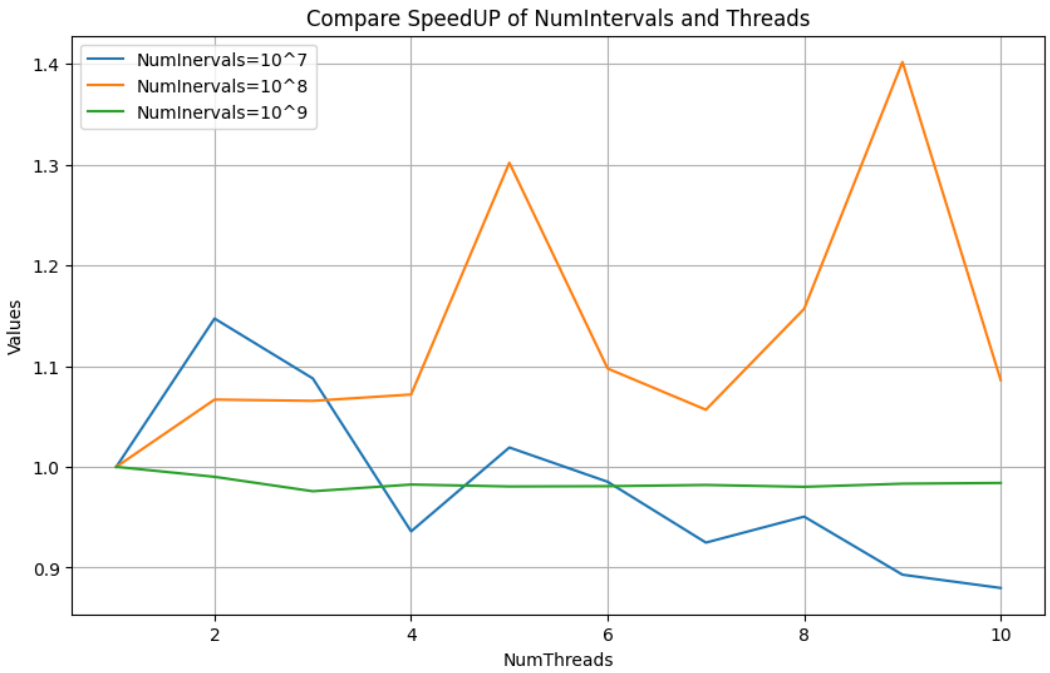
Неполные результаты приведены в таблице ниже.

NumInterval	NumThread	PiValue	ElaspsedTime(s)	SpeedUp
100000000	1	3.14159265	0.612365	1
100000000	2	3.14159265	0.602738	0.98427898
100000000	3	3.14159265	0.603098	0.98486686
100000000	4	3.14159265	0.603474	0.98548088
100000000	5	3.14159265	0.603041	0.98477378
100000000	6	3.14159265	0.595701	0.97278747
100000000	7	3.14159265	0.604082	0.98647375
100000000	8	3.14159265	0.583299	0.95253484
100000000	9	3.14159265	0.60151	0.98227364
100000000	10	3.14159265	0.592518	0.96758959
100000000	11	3.14159265	0.609038	0.99456696
100000000	12	3.14159265	0.59992	0.97967715
100000000	13	3.14159265	0.604511	0.98717431
100000000	14	3.14159265	0.589162	0.9621092
100000000	15	3.14159265	0.600096	0.97996456
100000000	16	3.14159265	0.603637	0.98574706
100000000	17	3.14159265	0.597109	0.97508675
100000000	18	3.14159265	0.590187	0.96378303
100000000	19	3.14159265	0.621878	1.01553485
100000000	20	3.14159265	0.59304	0.96844202

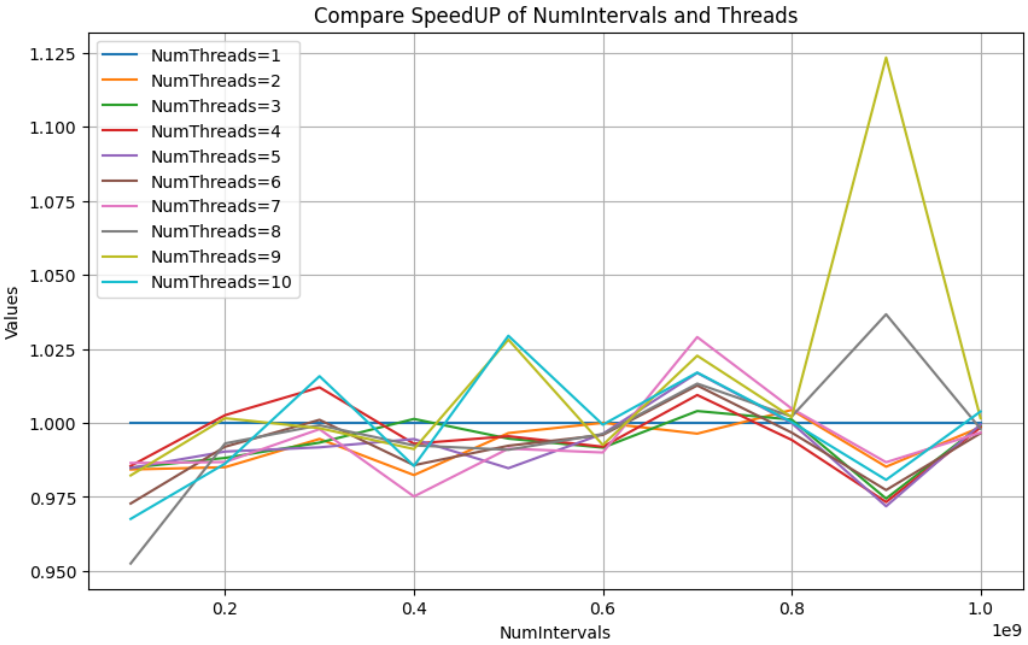
Мы выводим результаты в виде линейной диаграммы.

На следующем рисунке показано изменение времени выполнения при увеличении количества потоков с одинаковым интервалом разделения. Данный график отображает ускорение вычислений числа π с разным количеством потоков для различных значений количества интервалов. На оси абсцисс представлено количество потоков (от 1 до 20), а на оси ординат – ускорение.





Данный график показывает ускорение вычислений при фиксированном количестве потоков (от 1 до 10), но с разным количеством интервалов для вычисления числа π. На оси абсцисс представлены значения интервалов, а на оси ординат — ускорение вычислений.



5. Анализ результатов

Для большинства случаев увеличение количества потоков не приводит к значительному ускорению. Линии графика практически остаются на уровне близком к 1, что свидетельствует о слабом эффекте распараллеливания.

Единственное заметное отклонение представлено на графике зелёной линией ($9*10^8$ интервалов), где при использовании 15 потоков наблюдается значительный всплеск ускорения до значения выше 3. Однако затем ускорение быстро возвращается к прежним значениям при увеличении числа потоков до 20.

В некоторых случаях, при большем числе потоков, графики показывают небольшие колебания, что может свидетельствовать о накладных расходах на управление потоками или недостаточном разделении задач между потоками.

Для данного алгоритма вычисления числа π, увеличение числа потоков не всегда приводит к ожидаемому ускорению. В большинстве случаев ускорение

находится в пределах значений, близких к единице, что указывает на ограниченность распараллеливания. Возможные причины включают низкую нагрузку на процессор или высокие накладные расходы на создание и управление потоками.

6. Итог

В ходе эксперимента удалось успешно реализовать метод вычисления значения числа π с помощью многопоточности. Однако результаты эксперимента показали, что связь между количеством потоков и временем выполнения программы не является строго линейной. Видно, что увеличение количества потоков действительно сокращает время выполнения программы, но когда количество потоков превышает количество физических ядер процессора, время выполнения начинает увеличиваться.

При использовании большего количества интервалов (для задач, требующих значительных вычислительных ресурсов), параллельные вычисления с использованием потоков обеспечивают хорошее ускорение. Это свидетельствует о том, что многопоточность может эффективно повысить производительность в крупномасштабных вычислениях, особенно когда количество потоков соответствует числу ядер.

Тем не менее, для метода вычисления числа π этот эффект ускорения не был достаточно значительным. В дальнейшем можно рассмотреть другие методы вычислений для более точной оценки эффективности многопоточности и провести дополнительные проверки.

Также стоит отметить, что накладные расходы на управление потоками могут влиять на эффективность параллельных вычислений. Это особенно заметно при превышении числа потоков по сравнению с количеством доступных физических ядер, что приводит к конкурентной борьбе за ресурсы процессора. Для оптимизации подобного рода вычислений рекомендуется адаптировать количество потоков к числу физических и логических ядер

системы, а также применять дополнительные оптимизации для снижения накладных расходов.