МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение
высшего образования
«Самарский национальный исследовательский университет
имени академика С.П. Королёва» (Самарский университет)

Факультет информатики
Кафедра программных систем

Дисциплина
Параллельное программирование

ОТЧЕТ

по лабораторной работе №2

Параллельный алгоритм вычисления числа пи стохастическим методом – реализация на OpenMP и MPI

Студент: Гижевская В.Д.
Группа: 6413-020302D

Преподаватель: Оплачко Д.С.
Оценка: _____

Дата: _____

Самара 2021

Содержание

Постановка задачи

Цель работы: Разработка программы, реализующей последовательный алгоритм вычисления числа ПИ стохастическим методом. Изучение OpenMP – технологии параллельного программирования для вычислительных систем с общей памятью. Изучение MPI – широко распространённой технологии параллельного программирования для распределённых вычислительных систем.

1 Разработать последовательный алгоритм вычисления числа ПИ методом иглы Бюффона.

2 Разработать программу на языке C, реализующую указанный алгоритм. Программа должна предоставлять пользователю возможность задавать общее количество бросков (n) перед запуском вычислений и должна отображать вычисленное значение pi с точностью до 12 знаков после запятой.

3 Измерить время работы программы для следующих значений n: 1000, 10000, 100000, 1 000 000, 10 000 000, 100 000 000, 1 000 000 000

4 Результаты измерений записать в таблицу.

5 На основе последовательного алгоритма вычисления числа ПИ методом иглы Бюффона, разработать параллельный алгоритм решения той же задачи. Алгоритм должен допускать параллельное выполнение произвольного количества фрагментов алгоритма N <= n, где n – общее количество моделируемых бросаний иглы.

6 Разработать программу на языке C с использованием библиотеки MPI, реализующую указанный алгоритм. Программа должна предоставлять пользователю возможность задавать общее количество бросков (n) перед запуском вычислений и должна отображать вычисленное значение pi с точностью до 12 знаков после запятой.

7 Измерить время работы программы для следующих значений n и количества процессов p: $n \in \{1000, 10000, 100000, 1000000, 10\,000000, 100\,000\,000, 1\,000\,000\,000\}$ $p \in \{2, 4, 8\}$.

8 Результаты измерений записать в таблицы.

9 Разработать программу на языке C с использованием технологии OpenMP, реализующую указанный алгоритм. Программа должна предоставлять пользователю возможность задавать общее количество бросков (n) перед запуском вычислений и должна отображать вычисленное значение pi с точностью до 12 знаков после запятой.

10 Измерить время работы программы для следующих значений n и количества потоков p: $n \in \{1000, 10000, 100000, 1000000, 10\ 000000, 100\ 000000, 1\ 000\ 000\ 000\}$ $p \in \{2, 4, 8\}$

11 Результаты измерений записать в таблицы.

12 Составить отчёт по результатам работы

Описание работы алгоритмов

В рамках данной лабораторной работы было реализовано три алгоритма вычисления числа π стохастическим методом: последовательный, параллельный с использованием технологии OpenMP и параллельный с использованием технологии MPI.

1. Последовательный алгоритм

С целью реализации последовательного алгоритма необходимо было реализовать стохастический алгоритм вычисления числа π.

В качестве примера применения статистического метода анализа стохастической модели может служить способ определения числа π, который был предложен Бюффоном ещё в 1777 году. Он придумал натурную модель реализации случайного события, вероятность которого теоретически выражалась через число π, и, экспериментируя с моделью, по теореме Бернулли статистически оценил эту вероятность. В результате ему удалось статистическим методом приближённо получить значение числа π [1].

Натурная модель реализации случайного события выражалась в многократном бросании иглы длиной l на плоскость, расчерченную параллельными прямыми линиями, расположенными друг от друга на расстоянии, равном длине иглы [1].

Характеристики положения иглы относительно линии выразим парой (x, φ), которую будем рассматривать как точку на координатной плоскости с координатами (x, φ), где:

– $x\epsilon[0,\frac{l}{2}]$ – расстояние от середины иглы до точки пересечения линии;

– $\varphi\epsilon[-\frac{\pi}{2},\frac{\pi}{2}]$ – острый угол между иглой и пересечённой линией;

Острый угол слева от пересекаемой линии - отрицательный, справа - положительный. Множество таких точек будет равно декартовому произведению - (x, φ)$\in \left[0,\frac{l}{2}\right]\times\left[-\frac{\pi}{2},\frac{\pi}{2}\right]$. Очевидно, что расстояние x от середины иглы до пересечения иглой линии будет меньше или равно катету прямоугольного треугольника, гипотенузу которого составляет половина

иглы. Поэтому событие A пересечения иглой некоторой линии возникает, если точка (x, φ) будет удовлетворять условию, выраженному неравенством: $x \leq \frac{l}{2} | \sin \varphi |$ [1].

Положим, что в процессе бросания иглы реализации случайных величин x и φ являются независимыми и подчиняются равномерному закону распределения. Тогда для числа π будет справедлива оценка $\pi \approx \frac{2N}{m}$, где N – общее число бросков, m – количество пересечений иглой линий. Точность вычисления π зависит от числа испытаний N. При достаточно большом числе испытаний N можно получить приемлемый результат оценки числа π [1].

Программная реализация алгоритма состоит в следующем:

– Программа генерирует в цикле на N итераций параметры x и φ.

– Для каждой сгенерированной пары проверяется выполнение неравенства $x \leq \frac{l}{2} | \sin \varphi |$.

– Если неравенство выполняется, то число пересечений m инкрементируется, иначе остаётся неизменным.

– После завершения цикла производится расчёт числа π по формуле $\pi \approx \frac{2N}{m}$.

В приложении А приведён код последовательного алгоритма.

2. Параллельные алгоритмы

2.1. Параллельный алгоритм с использованием OpenMP

Для распараллеливания с помощью OpenMP используются директивы #pragma omp parallel for с дополнительными параметрами:

– Shared\private, указывающие на то, какие переменные являются общими для всех потоков, а какие переменные копируются в память каждого потока.

– Default, определяющий поведение переменных без области видимости в параллельной области. Значение none в данном случае означает, что, если в параллельной области присутствуют переменные, не размеченные

как private, shared, reduction, firstprivate, lastprivate, компилятор будет выдавать ошибку [2].

Schedule, определяющий, как разделяются задачи между потоками:

– Static – итерации равномерно распределяются по потокам. Устанавливается по умолчанию, если параметр schedule опущен при написании кода.

– Dynamic – работа распределяется пакетами заданного размера (по умолчанию размер равен 1) между потоками. Как только какой-либо из потоков заканчивает обработку своей порции данных, он захватывает следующую.

– Guided – данный тип распределения работы аналогичен предыдущему, за тем исключением, сто размер блока изменяется динамически в зависимости от того, сколько необработанных итераций осталось. Размер блока постепенно уменьшается вплоть до указанного значения.

– Runtime – тип распределения определяется в момент выполнения программы.

Таким образом распараллеливается та область алгоритма, в которой выполняется цикл на N итераций. Остальная часть программы остаётся неизменной относительно последовательной реализации.

Для определения числа потоков была использована функция omp_set_num_threads [3].

При распараллеливании данного алгоритма могут возникать состояния гонки (race conditions), при котором сразу несколько потоков пытаются обратиться к переменной, хранящей количество пересечений. В результате могут возникать некорректные значения числа пересечений, и, как следствие, число π может быть вычислено неправильно.

Для предотвращения такой ошибки добавляется директива reduction (оператор : общая переменная), где operator – операция из списка ("+", "*", "-",

"&", "|", "^", "&&", "||"), а общая переменная – это переменная, в данному случае хранящая количество пересечений иглы с линиями.

Принцип работы:

– Для каждой переменной создаются локальные копии в каждом потоке.

– Локальные копии инициализируются соответственно типу оператора. Для аддитивных операций – 0 или его аналоги, для мультипликативных операций – 1 или её аналоги.

– Над локальными копиями переменных после выполнения всех операторов параллельной области выполняется заданный оператор. Порядок выполнения операторов не определён [4].

Таким образом в переменной, хранящей количество пересечений, после окончания работы параллельной секции программы окажется корректное значение.

С помощью функции omp_get_wtime() фиксируется время начала и окончания выполнения алгоритма [3].

В приложении Б приведён код параллельного (OpenMP) алгоритма.

2.2. Параллельный алгоритм с использованием MPI

Реализация алгоритма при помощи MPI заключается в следующем:

Производится инициализация с помощью функции MPI_Init. Все остальные MPI-процедуры могут быть вызваны только после вызова MPI_Init [5].

1 Определяется количество процессов и ранг текущего процесса с помощью функций MPI_Comm_size и MPI_Comm_rank [6].

2 В мастер-процессе фиксируется время начала выполнения алгоритма при помощи функции MPI_Wtime [6].

3 Количество итераций рассылается всем рабочим процессам с помощью функции MPI_Bcast [6].

4 Рабочие процессы производят вычисление количества пересечений иглы с линиями, при этом i-й рабочий процесс берет на себя i-й шаг цикла и последующие с шагом, равным количеству рабочих процессов.

5 Результаты работы всех рабочих процессов суммируются функцией MPI_Reduce: все «локальные» количества пересечений суммируются в одно «глобальное» значение [7].

6 В мастер-процессе фиксируется время окончания выполнения алгоритма, вычисляется число π и на консоль выводится информация о результатах выполнения.

В приложении В приведён код параллельного (MPI) алгоритма.

Результаты вычислительных экспериментов

График сравнения времени параллельных алгоритмов для p=8 с помощью технологии OpenMP и MPI представлен на рисунке 1.

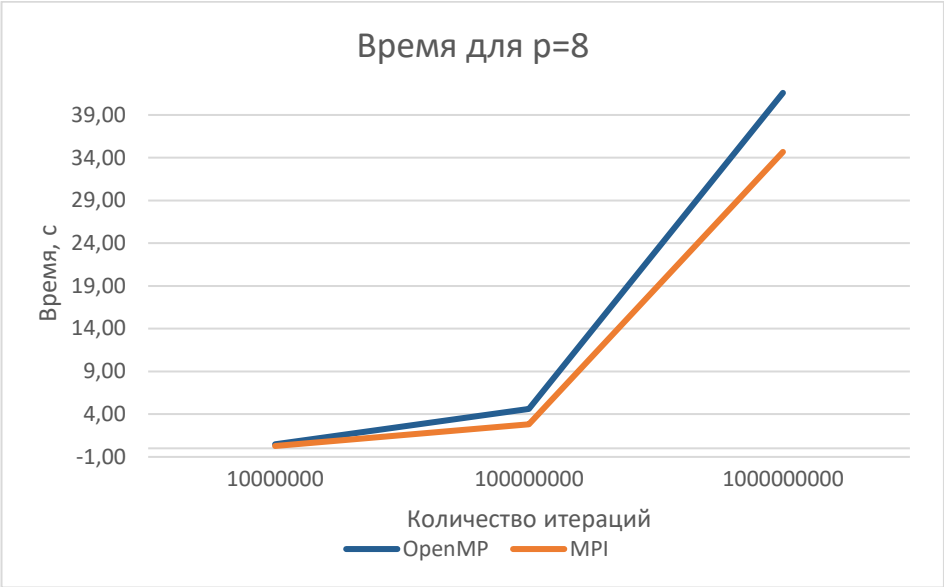

Рисунок 1 – График времени работы программы OMP и MPI

В таблице 1 приведены результаты измерений работы алгоритма OpenMP. На рисунке 2 представлены графики ускорения OpenMP алгоритма по сравнению с последовательной для различного количества потоков.

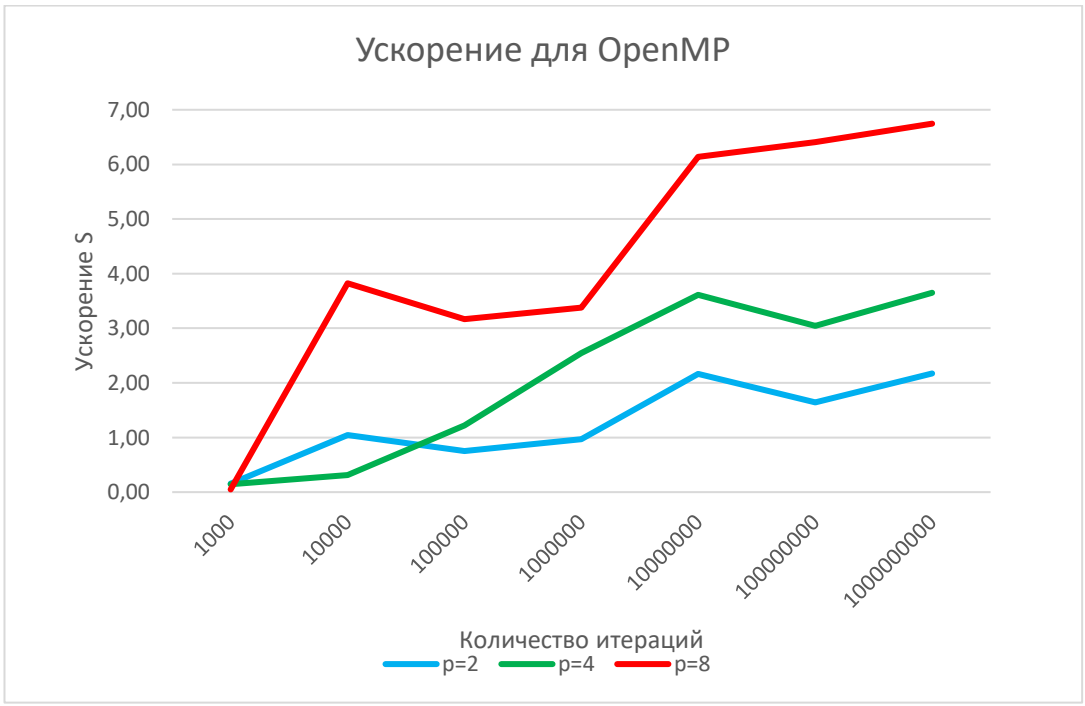

Рисунок 2 – Графики ускорения OpenMP алгоритма

Результаты измерений работы алгоритма MPI представлены в таблице 2. На рисунке 3 представлены графики ускорения MPI алгоритма по сравнению с последовательной для различного количества процессов.

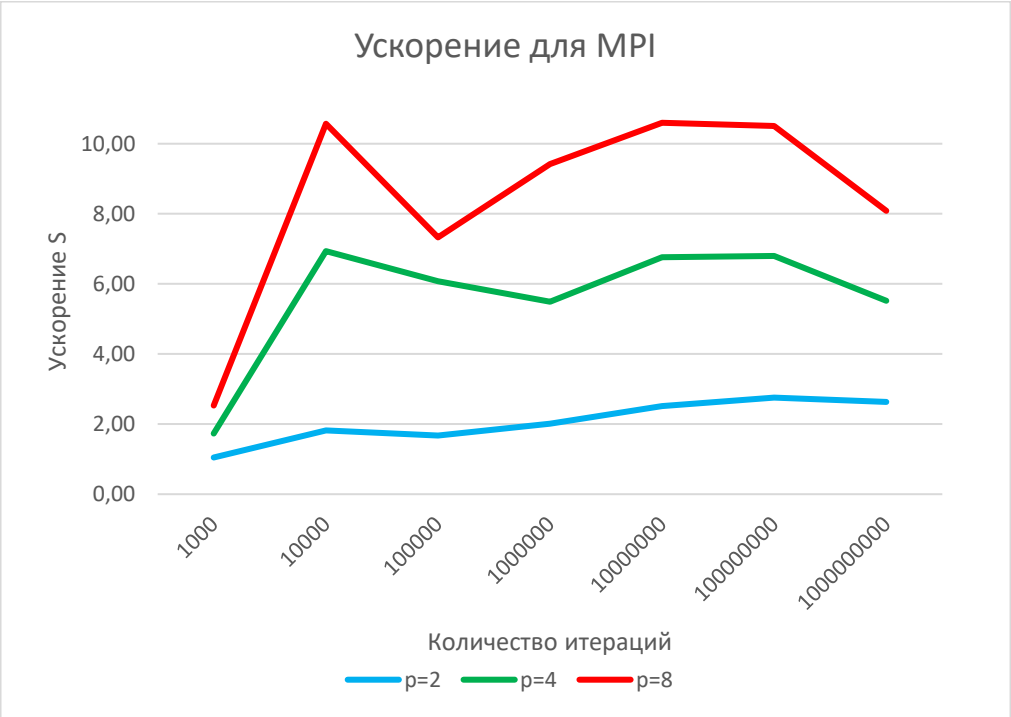

Рисунок 3 – Графики ускорения MPI алгоритма

Графики практически не отличаются для алгоритмов, реализованных с помощью OpenMP и MPI, а значит заметных отличий в эффективности их применения для решения данной задачи нет.

Таблица 1 – Сравнение результатов последовательного алгоритма и параллельного с технологией OpenMP

Размерность матриц, n	1000	10000	100000	1000000	10000000	100000000	1000000000
Время работы последовательного алгоритма t,c	0,0003340	0,002906	0,024991	0,255881	2,916232	29,428691	280,518051
π	3,1120000	3,1112	3,1398	3,1399	3,1409728	3,14138548	3,141537224
Время работы алгоритма с помощью OpenMP p=2 t,c	0,002152	0,002779	0,033116	0,264353	1,348613	17,883213	129,130694
π	3,1200000	3,0992	3,13528	3,14204	3,1412184	3,1413496	3,141474512
Ускорение S	0,1552044610	1,0456998920	0,7546503201	0,9679519430	2,1623935110	1,6456042323	2,1723576503
Время работы алгоритма с помощью OpenMP p=4 t,c	0,002312	0,000793	0,008083	0,073329	0,720303	6,810234	66,454477
π	3,1520000	3,0688	3,1304	3,142592	3,1394592	3,14155968	3,141514176
Ускорение S	0,1444636678	0,3102	1,2198	2,5456	3,6117	3,0454	3,6495
Время работы алгоритма с помощью OpenMP p=8 t,c	0,006864	0,00076	0,00789	0,07572	0,475048	4,592821	41,58462
π	3,1680000	3,0688	3,12	3,135712	3,140848	3,14094688	3,141462592
Ускорение S	0,0486596737	3,8236842105	3,1674271229	3,3793053354	6,1388154460	6,4075414653	6,7457163490

Таблица 2 – Сравнение результатов последовательного алгоритма и параллельного с технологией MPI

Размерность матриц, n	1000	10000	100000	1000000	10000000	100000000	1000000000
Время работы последовательного алгоритма t,c	0,0003340	0,0029060	0,0249910	0,2558810	2,9162320	29,4286910	280,5180510
π	3,1120000	3,1112000	3,1398000	3,1399000	3,1409728	3,1413855	3,1415372
Время работы алгоритма с помощью MPI p=2 t,c	0,000319	0,001598	0,014944	0,12702	1,160839	10,67676	106,425206
π	3,1160000	3,0988	3,13528	3,142036	3,1412176	3,1413496	3,141474508
Ускорение S	1,047022	1,818523	1,672310	2,014494	2,512176	2,756332	2,635823
Время работы алгоритма с помощью MPI p=4 t,c	0,000193	0,000419	0,00411	0,046623	0,431468	4,329809	50,835063
π	3,1520000	3,0684	3,13032	3,142588	3,1394596	3,1415596	3,141514172
Ускорение S	1,730570	6,935561	6,080535	5,488300	6,758860	6,796764	5,518200
Время работы алгоритма с помощью MPI p=8 t,c	0,000132	0,000275	0,00341	0,027162	0,275172	2,801817	34,674031
π	3,1640000	3,0684	3,12004	3,135708	3,1408472	3,14094684	3,141462584
Ускорение S	2,530303	10,567273	7,328739	9,420551	10,597852	10,503431	8,090148

Выводы

Из приведённых выше результатов видно, что алгоритмы на при малом количестве итераций (меньше 10000) показывают маленькое ускорение. Это связано с затратами на параллелизм. Однако при большом количестве итераций параллельные версии программ дают значительное ускорение – примерно в 8 раз.

Алгоритм подсчёта числа π методом иглы Бюффона не является достаточно эффективным. Как видно из таблиц, максимально достигнутая при точность вычислений при количестве итераций n = 100000000 равна 10^{-4}. Это связано с непосредственно стохастичностью алгоритма – полученное значение числа π может оказаться как больше, так и меньше реального.

Список использованных источников

1 Баландин А.В. Моделирование информационных процессов и систем: Учеб. пособие/ Самарский ун-т. - Самара, 2021. 37с.

2 OpenMP Directives [Электронный ресурс]. URL: https://docs.microsoft.com/en-us/cpp/parallel/openmp/reference/openmp-directives?view=msvc-160 (дата обращения: 05.11.2021).

3 OpenMP 4.5 API C/C++ Syntax Reference Guide [Электронный ресурс]. URL: https://www.openmp.org/wp-content/uploads/OpenMP-4.5-1115-CPP-web.pdf (дата обращения: 05.11.2021).

4 Параллельные заметки N5 — продолжаем знакомиться с конструкциями OpenMP [Электронный ресурс]. URL: https://habr.com/ru/company/intel/blog/88574/ (дата обращения 05.11.2021).

5 Краткий справочник по функциям MPI [Электронный ресурс]. URL: https://ssd.sscc.ru/ru/content/краткий-справочник-по-функциям-mpi (дата обращения 05.11.2021).

6 MPICH [Электронный ресурс]. URL: https://www.mpich.org/static/docs/v3.3/www3/ (дата обращения: 05.11.2021).

7 Функция Reduce [Электронный ресурс]. URL: https://www.opennet.ru/docs/RUS/mpi-1/node81.html (дата обращения: 05.11.2021).

8 Пи (число) [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Пи_(число) (дата обращения: 05.11.2021).

ПРИЛОЖЕНИЕ А
Листинг последовательной программы

```
#include <stdio.h>
#include <omp.h>
#include <cstdlib>
#define _CRT_SECURE_NO_WARNINGS



int main()
{
	int n = 1000;
	for (int i = 0; i < 7; i++) {
		double t1, t2;
		int m = 0;
		double x, y, pi;
		t1 = omp_get_wtime();
		for (int i = 0; i < n; i++) {
			x = (double)rand() / (RAND_MAX);
			y = (double)rand() / (RAND_MAX);
			if (x * x + y * y < 1) {
				m++;
			}

		}
		t2 = omp_get_wtime();
		pi = (double)m * 4.0 / n;
		printf("N= %d,  pi = %.12f, Time=%lf \n", n, pi, t2 - t1);
		n = n * 10;
	}
	system("pause");
}
```

ПРИЛОЖЕНИЕ В
Листинг параллельной программы с помощью OpenMP

```
#include <stdio.h>
#include <omp.h>
#include <cstdlib>

using namespace std;
int main()
{
	omp_set_num_threads(8);
	int n = 1000;
	for (int i = 0; i < 7; i++) {
		double t1, t2;
		int m = 0;
		double x, y, pi;
		t1 = omp_get_wtime();
#pragma omp parallel for private(x,y) shared(n) reduction(+:m)
		for (int i = 0; i < n; i++) {
			x = (double)rand() / (RAND_MAX);
			y = (double)rand() / (RAND_MAX);
			if (x * x + y * y < 1) {
				m++;
			}

		}
		t2 = omp_get_wtime();
		pi = (double)m * 4.0 / n;
		printf("N= %d,  pi = %.12f, Time=%lf \n", n, pi, t2 - t1);
		n = n * 10;
	}
	system("pause");
}
```

ПРИЛОЖЕНИЕ С
Листинг параллельной программы с использованием MPI

```
#include <stdio.h>
#include <cstdlib>
#include <mpi.h>
#define _CRT_SECURE_NO_WARNINGS

int main(int* argc, char** argv) {
	int n = 1000;
	int rank, s;
	double t1, t2;
	double x, y, pi, locpi;
	MPI_Init(argc, &argv);
	MPI_Comm_rank(MPI_COMM_WORLD, &rank);
	MPI_Comm_size(MPI_COMM_WORLD, &s);
	for (int i = 0; i < 7; i++) {
		int m = 0;
		MPI_Barrier(MPI_COMM_WORLD);
		t1 = MPI_Wtime();
		for (int i = rank + 1; i < n; i += s) {
			x = (double)rand() / (RAND_MAX);
			y = (double)rand() / (RAND_MAX);
			if (x * x + y * y < 1) {
				m++;
			}

		}
		locpi = (double)4.0 * m / n;
		MPI_Reduce(&locpi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
		MPI_Barrier(MPI_COMM_WORLD);
		t2 = MPI_Wtime();
		if (rank == 0) {
			printf("N= %d,  pi = %.12f, Time=%lf \n", n, pi, t2 - t1);
		}
		n = n * 10;
	}
	MPI_Finalize();
	system("pause");

}
```