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**

Студент: Андреева А.А.

Группа:  6413-020302D   
  
Преподаватель: Оплачко Д.С.  
Подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Самара 2021

Содержание

[Постановка задачи 3](#_Toc87540285)

[Описание вычислительной системы, на которой производились эксперименты 5](#_Toc87540286)

[1 Разработка последовательного алгоритма 6](#_Toc87540287)

[1.1 Стохастические методы 6](#_Toc87540288)

[1.2 Метод иглы Бюффона 6](#_Toc87540289)

[2 Параллельный алгоритм 9](#_Toc87540290)

[2.1 OpenMP 9](#_Toc87540291)

[2.2 MPI 9](#_Toc87540292)

[Результаты вычислительных экспериментов 11](#_Toc87540293)

[Выводы 15](#_Toc87540294)

[Список использованных источников 16](#_Toc87540295)

[Приложение А.](#_Toc87540296) [Последовательный алгоритм вычисления числа π 17](#_Toc87540297)

[Приложение Б.](#_Toc87540298) [Параллельный алгоритм вычисления числа π с использованием OpenMP 18](#_Toc87540299)

[Приложение В.](#_Toc87540300) [Параллельный алгоритм вычисления числа π с использованием MPI 19](#_Toc87540301)

# Постановка задачи

Цель работы: Изучение OpenMP – технологии параллельного программирования для вычислительных систем с общей памятью. Изучение MPI – широко распространенной технологии параллельного программирования для распределенных вычислительных систем.

1. Разработать последовательный алгоритм вычисления числа методом иглы Бюффона. Метод заключается в моделировании процесса бросания иглы длины l на плоскость, разлинованную параллельными бесконечно длинными прямыми, равноудалёнными друг от друга на расстояние l. При каждом бросании игла либо не пересекает прямые, либо пересекает ровно одну из них. Можно показать, что отношение числа пересечений иглы с произвольными линиями (m) к общему числу бросков (n) стремится к 2\ при увеличении числа бросков до бесконечности:

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 с точностью до 12 знаков после запятой.

3. Измерить время работы программы для следующих значений n: 1000, 10000, 100000, 1000000, 10 000000, 100 000000, 1 000 000 000.

4. Результаты измерений записать в таблицу.

5. На основе последовательного алгоритма вычисления числа 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 технологии OpenMP и библиотеки MPI, реализующую указанный алгоритм. Программа должна предоставлять пользователю возможность задавать общее количество бросков (n) перед запуском вычислений и должна отображать вычисленное значение с точностью до 12 знаков после запятой.

7. Измерить время работы программы для следующих значений n и количества процессов p:

n = {1000, 10000, 100000, 1000000, 10 000000, 100 000 000, 1 000 000 000}

p = {2, 4, 8}

8. Результаты измерений записать в таблицы.

9. Составить отчет по результатам работы.

# Описание вычислительной системы, на которой производились эксперименты

Модель процессора: Intel(R) Core(TM) i5-4200U CPU @ 1.60GHz 2.30 GHz

Количество физических ядер: 2

Поддержка hyperthreading: да

1. Разработка последовательного алгоритма

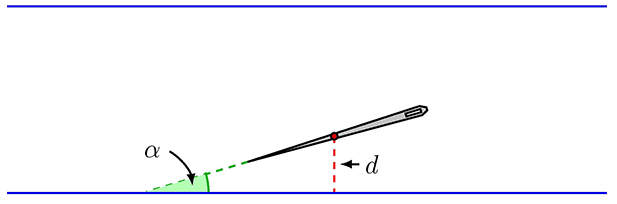
# Стохастические методы

Для решения широкого круга научных и инженерных задач применяются стохастические вычислительные алгоритмы, основанные на операциях с последовательностями псевдослучайных чисел. Вследствие относительно слабой сходимости, обусловленной статистическими свойствами алгоритмов, получение достоверного результата требует больших объемов вычислений, что, в свою очередь, делает обоснованным применение параллельных алгоритмов и соответствующих им методов и технологий параллельного программирования.

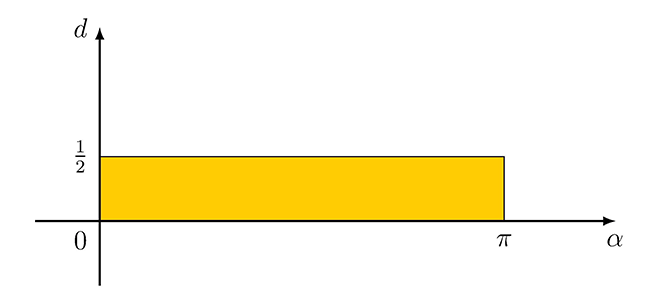
* 1. Метод иглы Бюффона

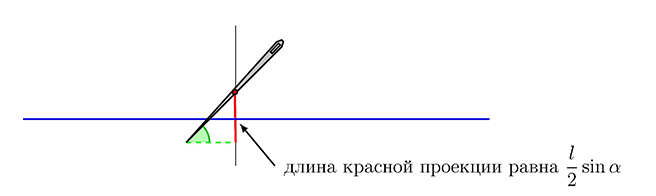
Метод иглы Бюффона заключается в моделировании процесса бросания иглы длины l на плоскость, разлинованную параллельными бесконечно длинными прямыми, равноудалёнными друг от друга на расстояние l. При каждом бросании игла либо не пересекает прямые, либо пересекает ровно одну из них. При каждом бросании игла либо не пересекает прямые, либо пересекает ровно одну из них. Можно показать, что отношение числа пересечений иглы с произвольными линиями (m) к общему числу бросков (n) стремится к 2\ при увеличении числа бросков до бесконечности:

Для решения задачи иглы Бюффона нужно ввести параметры, которые бы определяли положение иголки и позволяли бы описать все случаи, когда она пересекает линии. Возьмем угол наклона α иголки к прямым и расстояние *d* от середины иголки до ближайшей прямой (рисунок 1).

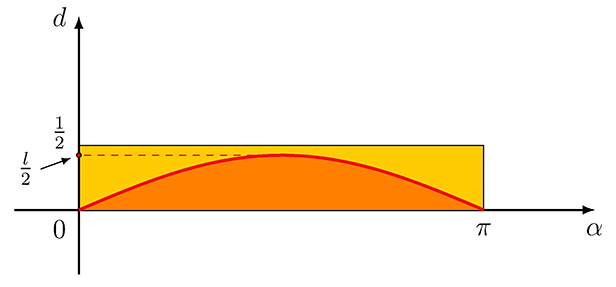
  
Рисунок 1 – Моделирование бросания иглы

[Радианная](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B4%D0%B8%D0%B0%D0%BD) мера угла α меняется от 0 до π, а *d* принимает значения от 0 (если середина иголки попала на прямую) до 1/2 (дальше середина иголки от прямых быть не может). На плоскости с координатами (α, *d*) эти ограничения задают прямоугольник (рисунок 2).

   
Рисунок 2 – Прямоугольник с координатами (α, *d*)

   
Рисунок 3 – Условие, при котором иголка пересекает хотя бы одну прямую

Из рисунка 3 видно, при каком условии на α и d иголка пересекает хотя бы одну прямую: проекция половины иголки на направление, перпендикулярное прямым, должна быть больше *d*. То есть должно выполняться неравенство . Осталось вычислить площадь под графиком синусоиды и разделить её на площадь всего прямоугольника, которая равна π/2 (рисунок 4).

   
Рисунок 4 – График синусоиды и прямоугольника

Как известно, площадь под графиком функции равна определённому интегралу от этой функции на нужном промежутке:https://elementy.ru/images/problems/needle_form3_617.gif.

В итоге получаем, что искомая вероятность равна .

Последовательный алгоритм представляет из себя цикл от 0 до n, где n –количество бросков, в котором генерируются значения α и *d*. После этого определяется, произошло ли пересечение. Если пересечение произошло, то увеличивается m. После завершения цикла число вычисляется как .

Текст программы с реализацией последовательного алгоритма приведен в приложении А.

1. Параллельный алгоритм

Согласно поставленной задаче, на основе последовательного алгоритма вычисления числа методом иглы Бюффона, нужно разработать параллельный алгоритм решения той же задачи. В последовательном алгоритме итерации цикла никак не зависят друг от друга, что позволяет распараллелить алгоритм. В параллельном варианте алгоритма количество бросков n разделяется на несколько процессов, после чего каждый процесс моделирует броски полученное количество раз.

* 1. OpenMP

Параллельность в данной программе достигается с помощью директивы #pragma omp parallel for shared(n) reduction(+:m) num\_threads(p). Опция shared указывает, что переменная n является общей. Параметр reduction (+:m) позволяет собрать вместе в главном потоке результаты вычислений частичных сумм, разностей и т. п. из параллельных потоков последующего параллельного структурного блока. Таким образом можно избежать ошибку конкурентности потоков при сложении значений в общий счетчик. Директива num\_threads указывает количество используемых потоков.

График ускорения S параллельных алгоритмов с помощью технологии OpenMP и MPI представлен на рисунках 3 – 5.

Текст программы с реализацией параллельного алгоритма с использованием OpenMP приведен в приложении Б.

* 1. MPI

Под параллельной программой в рамках MPI понимается множество одновременно выполняемых процессов. Основу MPI составляют операции передачи сообщений.

При вычислении числа каждый процесс получает число n1 – количество итераций цикла, которые он должен выполнить. Число n1 высчитывается в нулевом процессе как n/size, где size – число процессов. Рассылка данного значения по всем процессам производится с помощью операции MPI\_Bcast (). Каждый процесс выполняет n1 итерацию: генерирует значения и при необходимости инкрементирует m. С помощью операции редукции MPI\_Reduce () значения m собираются со всех процессов в нулевой и затем суммируются (в операцию MPI\_Reduce передается значение MPI\_SUM). Главный процесс вычисляет число и измеряет общую длительность выполнения программы с помощью функции MPI\_Wtime ().

Текст программы с реализацией параллельного алгоритма с использованием MPI приведен в приложении В.

# Результаты вычислительных экспериментов

В таблице 1 приведены результаты вычислений для последовательного и параллельного с использованием OpenMP алгоритмов.

На рисунке 5 приведены графики ускорений в зависимости от количества моделируемых бросков и количества процессов при использовании OpenMP.

Рисунок 5 – Графики ускорений для OpenMP

По результатам вычислений видно, что при использовании OpenMP ускорение равномерно растет до 100000 бросков. При 1000000 бросков ускорение резко возрастает. Максимальное ускорение, которого удалось добиться, примерно равно 4,7.

В таблице 2 приведены результаты вычислений для последовательного и параллельного с использованием MPI алгоритмов.

На рисунке 6 приведены графики ускорений в зависимости от количества моделируемых бросков и количества процессов при использовании MPI.

Рисунок 6 – Графики ускорений для MPI

По результатам вычислений видно, что при небольшом количестве моделируемых бросков последовательный алгоритм работает быстрее, чем параллельный при 8 процессах, так как на организацию параллельных вычислений тратится довольно много времени. Начиная с 100000 бросков, параллельные алгоритмы начинают давать преимущество по времени. При 1000000 бросков ускорение резко возрастает. Максимальное ускорение, которого удалось добиться, примерно равно 3,3.

Таблица 1 – Результаты вычислений для последовательного и параллельного с использованием OpenMP алгоритмов

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Количество моделируемых бросаний иглы, n | 1000 | 10000 | 100000 | 1000000 | 10000000 | 100000000 | 1000000000 |
| tпослед, сек | 0.000524 | 0.004092 | 0.030331 | 0.408646 | 3.148669 | 31.380927 | 317.238130 |
| Число pi | 3.110419906687 | 3.087849312954 | 3.134992789517 | 3.138598960810 | 3.140737055609 | 3.141289648298 | 3.141608081671 |
| p = 2 | | | | | | | |
| tпаралл, сек | 0.003053 | 0.002995 | 0.010999 | 0.103022 | 0.838574 | 9.589523 | 84.122901 |
| Ускорение S | 0.01716 | 1.36628 | 2.75761 | 3.96659 | 3.75479 | 3.27242 | 3,77113 |
| π | 3.067484662577 | 3.121098626717 | 3.132243312661 | 3.131958821005 | 3.140407624910 | 3.141298825286 | 3.141527468019 |
| p = 4 | | | | | | | |
| tпаралл, сек | 0.004728 | 0.005719 | 0.019835 | 0.092945 | 0.792749 | 7.829820 | 67.490502 |
| Ускорение S | 0.11082 | 0.71550 | 1.52916 | 4.39679 | 3.97183 | 4.00787 | 4.70049 |
| π | 3.125000000000 | 3.119151590767 | 3.127932436659 | 3.137058066945 | 3.140107843864 | 3.140441945602 | 3.141362335463 |
| p = 8 | | | | | | | |
| tпаралл, сек | 0.013072 | 0.010752 | 0.029870 | 0.108188 | 0.859002 | 8.671418 | 78.122728 |
| Ускорение S | 0.04008 | 0.38058 | 1.01543 | 3.77718 | 3.66550 | 3.61889 | 4.06077 |
| π | 3.125000000000 | 3.094059405941 | 3.105590062112 | 3.132321802213 | 3.140711408823 | 3.140469165994 | 3.141369282661 |

Таблица 2 – Результаты вычислений для последовательного и параллельного с использованием MPI алгоритмов

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Количество моделируемых бросаний иглы, n | 1000 | 10000 | 100000 | 1000000 | 10000000 | 100000000 | 1000000000 |
| tпослед, сек | 0.000524 | 0.004092 | 0.030331 | 0.408646 | 3.148669 | 31.380927 | 317.238130 |
| π | 3.110419906687 | 3.087849312954 | 3.134992789517 | 3.138598960810 | 3.140737055609 | 3.141289648298 | 3.141608081671 |
| p = 2 | | | | | | | |
| tпаралл, сек | 0.001688 | 0.003224 | 0.022550 | 0.175832 | 2.002987 | 19.009417 | 183.020391 |
| Ускорение S | 0.31042 | 1.26923 | 1.34505 | 2.32407 | 1.57198 | 1.65080 | 1.73334 |
| π | 3.067484662577 | 3.121098626717 | 3.1322433126  61 | 3.131958821005 | 3.140407624910 | 3.1412988252  86 | 3.1415274680  19 |
| p = 4 | | | | | | | |
| tпаралл, сек | 0.002053 | 0.003143 | 0.024242 | 0.122915 | 1.245770 | 12.239490 | 119.976614 |
| Ускорение S | 0.25523 | 1.30194 | 1.25117 | 3.32462 | 2.52748 | 2.56390 | 2.64416 |
| π | 3.1250000000  00 | 3.1191515907  67 | 3.1279324366  59 | 3.1370580669  45 | 3.140107843864 | 3.1404419456  02 | 3.1413623354  63 |
| p = 8 | | | | | | | |
| tпаралл, сек | 0.003065 | 0.008080 | 0.026209 | 0.140154 | 1.252223 | 12.287460 | 117.638218 |
| Ускорение S | 0.176096 | 0.50643 | 1.15727 | 2.91569 | 2.51446 | 2.55389 | 2.69672 |
| π | 3.1250000000  00 | 3.0940594059  41 | 3.1055900621  12 | 3.1323218022  13 | 3.140711408823 | 3.1404691659  94 | 3.1413692826  61 |

# Выводы

Из приведенных в таблице 1 результатов видно, что параллельные алгоритмы с OpenMP могут давать значительное ускорение – около 4.4 раз. Распараллеливание эффективно при количестве бросков от 10000 при всех количествах потоков. При 1000000 бросков алгоритма резко вырастает. Максимальные значения ускорения, которых удалось добиться:

* при р = 2: в ~3.7 раз;
* при р = 4: в ~4.7 раз;
* при р = 8: в ~4.1 раз.

Из приведенных в таблице 2 результатов видно, что параллельные алгоритмы с MPI могут давать ускорение – около 3.3 раз. Распараллеливание эффективно при количестве бросков от 100000 при всех количествах потоков. При 1000000 бросков эффективность алгоритма резко возрастает. Эффективно распараллелить программу с применением MPI позволила функция редукции MPI\_Reduce (). Максимальные значения ускорения, которых удалось добиться:

* при р = 2: в ~2.3 раз;
* при р = 4: в ~3.3 раз;
* при р = 8: в ~2.9 раз.

Результаты показали, что распараллеливание не эффективно при любом количестве процессов, при количестве бросков до 10000 для алгоритма с OpenMP и до 100000 для алгоритма с MPI.

Алгоритм с OpenMP показал результаты слегка выше, чем алгоритм с MPI. Это связано с тем, что MPI тратит большое количество времени на передачу данных между различными процессами и на подготовку передаваемых сообщений.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Жидченко В.В. Лекционный материал по курсу «Параллельное программирование» [Электронный ресурс]. URL: http://virtual6.ssau.ru/Moodle/course/view.php?id=973 (дата обращения: 09.11.2021).
2. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления - СПб.: БХВ-Петербург, 2002. - 608 с. (дата обращения: 09.11.2021).
3. Р.В. Жалнин, Е.Н. Панюшкина, Е. Е. Пескова, П.А. Шаманаев. Основы параллельного программирования с использованием технологий MPI И OPENMP - Саранск: Изд-во СВМО, 2013. – 78 с. (дата обращения: 09.11.2021).
4. Учебник по OpenMP [Электронный ресурс]. URL: https://pro-prof.com/archives/4335#page\_1 (дата обращения: 09.11.2021).

приложение а

Последовательный алгоритм вычисления числа π

#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

#define \_USE\_MATH\_DEFINES

#include <math.h>

//последовательный алгоритм

void main() {

int n;

int m = 0;

double t1, t2;

printf("Enter n = ");

scanf\_s("%d", &n);

t1 = omp\_get\_wtime();//время начала работы алгоритма

for (int i = 0; i < n; i++) {

double x = (double)rand() / (double)RAND\_MAX;

double dgr = ((double)rand() / (double)RAND\_MAX) \* M\_PI;

if (x < 0.5 \* sin(dgr) || (1 - x) < 0.5 \* sin(dgr)) {

m++;

}

}

double pi = 2 \* (double)n / (double)m;

printf("pi = %.12f\n", pi);

t2 = omp\_get\_wtime();//время окончания работы алгоритма

m = 0;

printf("t = %f\n", t2 - t1);

}

# приложение б

# Параллельный алгоритм вычисления числа π с использованием OpenMP

#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

#define \_USE\_MATH\_DEFINES

#include <math.h>

void main() {

int n;

double pi;

int m = 0, i;

double t1, t2;

printf("Enter n = ");

scanf\_s("%d", &n);

t1 = omp\_get\_wtime();//время начала работы алгоритма

m = 0;

#pragma omp parallel for private(i) reduction(+ : m) shared(n) num\_threads(8)

for (i = 0; i < n; i++)

{

double x = (double)rand() / (double)RAND\_MAX;

double dgr = ((double)rand() / (double)RAND\_MAX) \* M\_PI;

if (x < 0.5 \* sin(dgr) || (1 - x) < 0.5 \* sin(dgr)) {

m++;

}

}

pi = 2 \* (double)n / (double)m;

printf("pi = %.12f\n", pi);

t2 = omp\_get\_wtime();//время окончания работы алгоритма

printf("t = %f\n", t2 - t1);

}

Приложение В

Параллельный алгоритм вычисления числа π с использованием MPI

#include <stdio.h>

#include <stdlib.h>

#define \_USE\_MATH\_DEFINES

#include <math.h>

#include "mpi.h"

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

int n = 1000000000;

int rank, size;

MPI\_Status status;

MPI\_Init(&argc, &argv);

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

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

int n1 = n / size;

double duration = MPI\_Wtime();

MPI\_Bcast(&n1, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

int m = 0;

for (int i = 0; i < n1; i++) {

double x = (double)rand() / (double)RAND\_MAX;

double dgr = ((double)rand() / (double)RAND\_MAX) \* M\_PI;

if (x < 0.5 \* sin(dgr) || (1 - x) < 0.5 \* sin(dgr)) {

m++; } }

int m\_fin = 0;

MPI\_Reduce(&m, &m\_fin, 1, MPI\_INT, MPI\_SUM, 0, MPI\_COMM\_WORLD);

if (rank == 0) {

double pi = 2 \* (double)n / (double)m\_fin;

printf("n = %d\npi = %.12f\n", n, pi);

duration = MPI\_Wtime() - duration;

printf("t = %f", duration); }

MPI\_Finalize();

return 0;

}