

Содержание

[Введение 3](#_Toc90280094)

[Вычисление интеграла 5](#_Toc90280095)

[Вычисление числа Пи 6](#_Toc90280096)

[Решение одномерной задачи теплопроводности 7](#_Toc90280097)

[Заключение 9](#_Toc90280098)

[Список использованной литературы 10](#_Toc90280099)

[Приложение 11](#_Toc90280100)

[1.Код программы «вычисления интеграла» на языке С++ 11](#_Toc90280101)

[2.Код программы «вычисления числа Пи» на языке С++ 13](#_Toc90280102)

[3.Код программы «одномерная задача теплопроводности» на языке С++ 15](#_Toc90280103)

# Введение

Идея распараллеливания программ состоит в том, чтобы ускорить их работу, не повлияв на точность получаемых результатов. Достигается распараллеливание путем использования двух и более процессоров/ядер в комбинации для решения одной задачи. Теоретически, при использовании двух процессоров/ядер скорость расчетов должна возрасти в два раза, но из-за затрат на время пересылки результатов это не так.

Цель данной работы заключается в ознакомлении с процессом распараллеливания на простых задачах с использованием специального интерфейса MPI [1].

Рассматриваются следующие задачи:

* Вычисление интеграла;
* Вычисление числа Пи;
* Решение одномерной или двумерной задачи теплопроводности.

Для простоты задачи будут разделены на равные части в зависимости от количества процессоров.

Основные функции, использованные для реализации распараллеливания:

|  |  |
| --- | --- |
| MPI\_Init(NULL, NULL); | Инициализация среды MPI |
| MPI\_Comm\_rank(MPI\_COMM\_WORLD, &my\_rank); | Определение номера процессора (my\_rank) |
| MPI\_Comm\_size(MPI\_COMM\_WORLD, &my\_size); | Количество задействованных процессоров (my\_size) |
| MPI\_Send(&Rezult, 1, MPI\_FLOAT, 0, 0, MPI\_COMM\_WORLD); | Функция, позволяющая отправлять полученный результат (Rezult) 1 процессору |
| MPI\_Recv(&prom, 1, MPI\_FLOAT, i, 0, MPI\_COMM\_WORLD, &status); | Функция, позволяющая принимать сообщения (prom) от других процессоров (i) |
| MPI\_Finalize(); | Деактивация среды MPI |

Код программ написан с использованием языка программирования C++ [2].

Для рассмотрения эффективности распараллеливания будет рассчитываться коэффициент распараллеливания. Коэффициент распараллеливания рассчитывается по формуле:

где t1 – время, затрачиваемое на расчет, при расчете одним процессором, мс; - время, затрачиваемое на расчет, при нескольких процессорах, мс; n – количество процессоров

# Вычисление интеграла

Дано:

* функция:
* промежуток: [0; 2]

Задание: найти решение интеграла с использованием распараллеливания процесса [3].

Результаты:

|  |  |  |  |
| --- | --- | --- | --- |
| Количество разбиений интеграла | Количество процессоров | Результат вычисления | Коэф. распараллеливания |
| 1000000 | 1 | 2.66 |  |
| 2 | 2.64 |  |
| 3 | 2.66 |  |
| 4 | 2.66 |  |
| 5 | 2.66 |  |
| 6 | 2.67 |  |
| 7 | 2.67 |  |
| 8 | 2.67 |  |
| 9 | 2.67 |  |
| 10 | 2.67 |  |

Количество ядер

# Вычисление числа Пи

Задание: найти значение числа Пи.

Решение: для решения этой задачи было рассмотрено решение методом Монте-Карло. В ходе решения случайным образом ставятся точки в квадрат, в который вписан круг.

При данном методе число Пи рассчитывается по формуле:

где Nk – число точек попавшие в круг; N – общее число точек.

Результаты:

|  |  |  |
| --- | --- | --- |
| Количество процессоров | Результат вычисления | Коэф. распараллеливания |
| 1 | 3.13911 | 1.0000 |
| 2 | 3.13969 | 0.8931 |
| 3 | 3.13953 | 0.8371 |
| 4 | 3.13997 | 0.8029 |
| 5 | 3.13882 | 0.7854 |
| 6 | 3.14003 | 0.7524 |
| 7 | 3.13958 | 0.6821 |
| 8 | 3.13954 | 0.6052 |
| 9 | 3.13957 | 0.5300 |
| 10 | 3.13902 | 0.4841 |

# Решение одномерной задачи теплопроводности

Задание: решить краевую задачу на основе одномерного уравнения теплопроводности с граничными условиями I рода [3].

Решение: будем решать задачу методом конечных разностей. Согласно методу, непрерывные производные заменяются на конечно-разностные производные:

|  |  |
| --- | --- |
|  | (2) |
|  | (3) |

Подставим формулы (2), (3) в выражение (1). В результате получится рекуррентное соотношение, с помощью которого решаем задачу:

Переходя к решению задачи, обозначим условия:

Результаты расчета программы представляет из себя матрицу чисел. Представить данную матрицу в работе не представляется возможным.

Результаты:

|  |  |
| --- | --- |
| Количество процессоров | Коэф. распараллеливания |
| 1 | 1.0000 |
| 2 | 0.3462 |
| 3 | 0.2165 |
| 4 | 0.1559 |
| 5 | 0.1156 |
| 6 | 0.0946 |
| 7 | 0.0826 |
| 8 | 0.0656 |
| 9 | 0.0565 |
| 10 | 0.0460 |

# Заключение

Таким образом, в ходе выполнения работы был приобретен навык распараллеливания процесса в двух случаях: в одном случае вычисление на каждом процессоре происходит независимо друг от друга, в другом – происходит «обмен» данными для вычисления своего участка. Было подтверждено следующее: при применении двухпроцессорного алгоритма время работы не сокращается в двое, что связанно с передачей, принятием и сбором данных. Так же замечена низкая эффективность применения распараллеливания там, где в ходе решения будут требоваться данные с других процессоров (ядер), так как будет требоваться большое количество передач данных.

# Список использованной литературы

1. Абрамян М. Э. Параллельное программирование на основе технологии MPI – Издательство Южного федерального университета, 2018
2. Клюшин Д. А. Полный курс С++. Профессиональная работа. – М.: Издательский дом «Вильямс», 2004. – 672 с.: ил. – ISBN 5-8459-0536-2.
3. Самарский А. А., Гулин А. В. Численные методы. – Москва: Наука, 1989.

# Приложение

## 1.Код программы «вычисления интеграла» на языке С++

#include <mpi.h>

#include <stdio.h>

#include <string>

#include <math.h>

#include <Windows.h>

using namespace std;

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

MPI\_Init(NULL, NULL);

// Initialize the MPI environment

int my\_rank;

int my\_size;

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

// Get the rank of the process

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

double start = GetTickCount();

float a;

float b;

float a\_0 = 0.0;

float b\_0 = 2.0;

MPI\_Status status;

a = a\_0 + my\_rank \* (b\_0 - a\_0) / my\_size;

int n = 1000000;

int n\_p = 10000000 / my\_size;

int n\_g = n\_p \* my\_size;

float Sum = 0;

float dx = (b\_0 - a\_0) / n\_g;

float x\_i;

for (int i = 0; i < n\_p; i++)

{

x\_i = a + i \* dx;

Sum += x\_i \*x\_i\*dx;

}

if (my\_rank != 0)

{

MPI\_Send(&Sum, 1, MPI\_FLOAT, 0, 0, MPI\_COMM\_WORLD);

}

else

{

float prom;

float rez = Sum;

for (int i = 1; i < my\_size; i++)

{

MPI\_Recv(&prom, 1, MPI\_FLOAT, i, 0, MPI\_COMM\_WORLD, &status);

rez += prom;

}

double finish = GetTickCount();

double time = finish - start;

printf("%.6f", rez);

}

MPI\_Finalize();

}

## 2.Код программы «вычисления числа Пи» на языке С++

#include <mpi.h>

#include <stdio.h>

#include <string>

#include <math.h>

#include <Windows.h>

#include <random>

using namespace std;

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

MPI\_Init(NULL, NULL);

// Initialize the MPI environment

int my\_rank;

int my\_size;

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

// Get the rank of the process

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

double start = GetTickCount();

float a;

float x;

float y;

//float a\_0 = 0.0;

// float b\_0 = 2.0;

MPI\_Status status;

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<int> dist(0, 1000);

a = 1.000;

int n = 1000000;

int n\_p = 10000000 / my\_size;

int n\_g = n\_p \* my\_size;

float Sum = 0;

// float dx = (b\_0 - a\_0) / n\_g;

// float x\_i;

for (int i = 0; i < n\_p; i++)

{

x = dist(gen) / 1000.00000;

y = dist(gen) / 1000.00000;

if ((x \* x + y \* y) <= a \* a)

Sum += 1;

}

if (my\_rank != 0)

{

MPI\_Send(&Sum, 1, MPI\_FLOAT, 0, 0, MPI\_COMM\_WORLD);

}

else

{

float prom;

float rez = Sum;

for (int i = 1; i < my\_size; i++)

{

MPI\_Recv(&prom, 1, MPI\_FLOAT, i, 0, MPI\_COMM\_WORLD, &status);

rez += prom;

}

float pi;

pi = 4 \* rez / n\_g;

double finish = GetTickCount();

double time = finish - start;

printf("%.6f\n", time);

printf("%.6f", pi);

}

MPI\_Finalize();

}

## 3.Код программы «одномерная задача теплопроводности» на языке С++

#include <mpi.h>

#include <stdio.h>

#include <string>

#include<math.h>

#include <random>

#include<windows.h>

#include <iostream>

using namespace std;

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

// Initialize the MPI environment

MPI\_Init(NULL, NULL);

// Get the rank of the process

double start = GetTickCount();

int my\_rank;

int my\_size;

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

// Print the message

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

int nPoint, nPointmax, nPointProts;

nPointmax = 10000000;

nPointProts = nPointmax / my\_size;

nPoint = nPointProts \* my\_size;

float ak;

float bk;

float a = 0;

float b = 10;

int n = 9999;

float dx;

float xi;

float dt;

MPI\_Status status;

int n\_p = n / my\_size;

int n\_g = n\_p \* my\_size;

ak = a + (b - a) / my\_size \* my\_rank;

bk = a + (b - a) / my\_size \* (my\_rank + 1);

dx = (a + b) / n\_g;

//s = 0;

dt = 1.00/ (n\_g\*1.00);

double\* points = new double[n\_p + 1];

double\* point\_rasch = new double[n\_p + 1];

double t\_0;

double t\_n;

double \*\*sum = new double\*[10001];

for (int count = 0; count < 10001; count++)

sum[count] = new double[10001];

for (int t = 0; t <= n\_g; t++)

{

if (t == 0)

{

if (my\_rank != 0)

{

point\_rasch[0] = 100\*sin(3.14\*ak/b);

}

else point\_rasch[0] = 0;

// if (my\_rank==(my\_size-1))

// printf("%.g\n", point\_rasch[0]);

//point\_rasch[n\_p] = 100 - 10 \* bk;

for (int i = 1;i < n\_p + 1;i++)

{

xi = ak + i \* dx;

point\_rasch[i] = 100 \* sin(3.14 \* xi / b);

// printf("%.g\n", point\_rasch[i]);

}

points = point\_rasch;

}

else

{

if (my\_rank == 0)

{

for (int i = 1;i < n\_p;i++)

{

point\_rasch[i] = dt/dx/dx\*(points[i - 1] - 2 \* points[i] + points[i + 1]) + points[i];

}

MPI\_Send(&points[n\_p], 1, MPI\_DOUBLE, 1, 0, MPI\_COMM\_WORLD);

MPI\_Recv(&t\_n, 1, MPI\_DOUBLE, 1, 0, MPI\_COMM\_WORLD, &status);

points = point\_rasch;

points[0] = 0;

points[n\_p] = t\_n;

// points[1] = 100;

}

else if (my\_rank == my\_size - 1 && my\_size != 1)

{

for (int i = 1;i < n\_p;i++)

{

point\_rasch[i] = dt / dx / dx \* (points[i - 1] - 2 \* points[i] + points[i + 1]) + points[i];

}

MPI\_Send(&points[0], 1, MPI\_DOUBLE, my\_rank - 1, 0, MPI\_COMM\_WORLD);

MPI\_Recv(&t\_0, 1, MPI\_DOUBLE, my\_rank - 1, 0, MPI\_COMM\_WORLD, &status);

points = point\_rasch;

points[0] = t\_0;

points[n\_p] = 0;

}

else

{

for (int i = 1;i < n\_p;i++)

{

point\_rasch[i] = dt / dx / dx \* (points[i - 1] - 2 \* points[i] + points[i + 1]) + points[i];

}

MPI\_Send(&points[0], 1, MPI\_DOUBLE, my\_rank - 1, 0, MPI\_COMM\_WORLD);

MPI\_Send(&points[n\_p], 1, MPI\_DOUBLE, my\_rank + 1, 0, MPI\_COMM\_WORLD);

MPI\_Recv(&t\_0, 1, MPI\_DOUBLE, my\_rank - 1, 0, MPI\_COMM\_WORLD, &status);

MPI\_Recv(&t\_n, 1, MPI\_DOUBLE, my\_rank + 1, 0, MPI\_COMM\_WORLD, &status);

points = point\_rasch;

points[0] = t\_0;

points[n\_p] = t\_n;

}

}

if (my\_rank != 0)

{

MPI\_Send(points, n\_p + 1, MPI\_DOUBLE, 0, my\_rank, MPI\_COMM\_WORLD);

}

else

{

double\* prom = new double[n\_p + 1];

for (int i = 0;i < n\_p + 1;i++)

{

sum[t][i] = points[i];

}

for (int i = 1; i < my\_size;i++)

{

MPI\_Recv(prom, n\_p + 1, MPI\_DOUBLE, i, i, MPI\_COMM\_WORLD, &status);

for (int j = 0;j < n\_p + 1;j++)

{

sum[t][i \* n\_p + j] = prom[j];

}

}

}

}

if (my\_rank == 0)

{

//sumT += t;

for (int i = 0;i <= 10;i++)

{

for (int j = 0; j <= n\_g;j++)

{

std::cout << " " << sum[i][j] << " ";

}

std::cout << "\n";

}

double finish = GetTickCount();

double t = finish - start;

printf("%d", my\_rank);

printf("%g\n", t);

}

MPI\_Finalize();

}