Санкт-Петербургский политехнический университет Петра Великого

Физико-механический институт

**Высшая школа «Теоретическая механика»**

**КУРСОВАЯ РАБОТА**

по дисциплине «Проектирование по компьютерным технологиям в механике»

Выполнил

студент гр.5040103/00301 <*подпись*> И. С. Кравченко

Руководитель

<*подпись*> А. А. Ле-Захаров

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 202\_\_ г.

Санкт-Петербург

2021

Содержание

[Введение 3](#_Toc87648712)

[Вычисление интеграла 4](#_Toc87648713)

[Вычисление числа Пи 6](#_Toc87648714)

[Решение одномерной задачи теплопроводности 9](#_Toc87648715)

[Заключение 13](#_Toc87648716)

[Список литературы 14](#_Toc87648717)

# 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 на самом деле это не так.

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

# Вычисление интеграла

Дано:

* функция:
* промежуток: [0; 2]

Задание: найти решение интеграла с использованием распараллеливания процесса.

Реализация

#include <mpi.h>

#include <stdio.h>

#include <string>

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

MPI\_Init(NULL, NULL);

int my\_rank;

int my\_size;

float a;

float b;

float a\_0 = 0.0;

float b\_0 = 2.0;

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

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

MPI\_Status status;

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

b = a\_0 + (my\_rank + 1) \* (b\_0 - a\_0) / my\_size;

int n = 20000000/my\_size;

float Rezult = 0;

float dx = (b\_0 - a\_0) / (my\_size \* n);

float x\_i;

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

{

x\_i = a + i \* dx;

Rezult += x\_i \* dx;

}

if (my\_rank != 0)

{

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

}

else

{

float prom;

float rez\_1 = Rezult;

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

{

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

rez\_1 += prom;

}

printf("%.6f", rez\_1);

}

MPI\_Finalize();

}

Результаты:

|  |  |  |  |
| --- | --- | --- | --- |
| Количество разбиений интеграла | Количество процессоров | Результат вычисления | Коэф. распараллеливания |
| 20000000 | 1 | 1.990 | 1.0000 |
| 2 | 1.990 | 0.8849 |
| 3 | 1.989 | 0.7952 |
| 4 | 1.990 | 0.7401 |
| 5 | 1.990 | 0.6887 |
| 6 | 1.989 | 0.6292 |
| 7 | 1.989 | 0.6124 |
| 8 | 1.990 | 0.5386 |
| 9 | 1.989 | 0.3451 |
| 10 | 1.990 | 0.3253 |

# Вычисление числа Пи

Задание: найти значение числа Пи.

Решение: для решения этой задачи было рассмотрено отношение площадей квадрата и вписанного в него круга.

Реализация:

#include <mpi.h>

#include <stdio.h>

#include <string>

#include <random>

#include <math.h>

#include <iostream>

#include <iomanip>

#include <windows.h>

using namespace std;

int random\_points(float right\_upper\_line, float left\_bottom\_line,

float a, float b, int number\_point\_for\_one\_cpu, float radius)

{

float x, y = 0;

int number\_points\_in\_circle = 0;

for (int i = 0; i < number\_point\_for\_one\_cpu; i++)

{

x = ((float)rand() / RAND\_MAX) \* (b - a) + a;

y = ((float)rand() / RAND\_MAX) \* (right\_upper\_line - left\_bottom\_line) + left\_bottom\_line;

if (sqrt(pow(x, 2) + pow(y, 2)) < radius)

{

number\_points\_in\_circle += 1;

}

}

return (number\_points\_in\_circle);

}

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

double start = GetTickCount();

int my\_rank;

int my\_size;

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

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

int number\_points = 1000000;

int number\_point\_for\_one\_cpu = floor(number\_points / my\_size);

float left\_bottom\_line = -0.5;

float right\_upper\_line = 0.5;

float radius = 0.5;

float pi;

float dx = (right\_upper\_line - left\_bottom\_line) / my\_size;

MPI\_Status status;

srand((unsigned)time(NULL) + my\_rank);

if (my\_rank == 0)

{

float a = left\_bottom\_line + dx \* my\_rank;

float b = left\_bottom\_line + dx \* (my\_rank + 1);

int number\_points\_in\_circle = random\_points(right\_upper\_line, left\_bottom\_line, a,

b, number\_point\_for\_one\_cpu, radius);

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

{

int massage\_about\_number\_points;

MPI\_Recv(&massage\_about\_number\_points, 1, MPI\_FLOAT, i, 0, MPI\_COMM\_WORLD, &status);

number\_points\_in\_circle += massage\_about\_number\_points;

}

pi = (float) 4 \* number\_points\_in\_circle / number\_points;

std::cout << "pi " << pi << "\n";

}

else

{

float a = left\_bottom\_line + dx \* my\_rank;

float b = left\_bottom\_line + dx \* (my\_rank + 1);

int number\_points\_in\_circle = random\_points(right\_upper\_line, left\_bottom\_line, a,

b, number\_point\_for\_one\_cpu, radius);

MPI\_Send(&number\_points\_in\_circle, 1, MPI\_FLOAT, 0, 0, MPI\_COMM\_WORLD);

}

MPI\_Finalize();

}

Результаты:

|  |  |  |
| --- | --- | --- |
| Количество процессоров | Результат вычисления | Коэф. распараллеливания |
| 1 | 3.14004 | 1.0000 |
| 2 | 3.14175 | 0.9261 |
| 3 | 3.14314 | 0.8282 |
| 4 | 3.14303 | 0.7620 |
| 5 | 3.14002 | 0.7067 |
| 6 | 3.13934 | 0.6542 |
| 7 | 3.14342 | 0.6368 |
| 8 | 3.14229 | 0.5492 |
| 9 | 3.14255 | 0.4279 |
| 10 | 3.13963 | 0.3868 |

# Решение одномерной задачи теплопроводности

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

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:

Реализация:

#include <mpi.h>

#include <stdio.h>

#include <string>

#include <ctime>

#include <iostream>

#include <ctime>

#include <windows.h>

#include <fstream>

using namespace std;

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

int my\_rank;

int my\_size;

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

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

double x\_0\_rank;

double x\_n\_rank;

double t\_0\_rank = 0;

double t\_n\_rank = 0.01;

double dt = t\_n\_rank / 10;

double x\_0 = 0.0;

double x\_n = 0.6;

MPI\_Status status;

x\_0\_rank = x\_0 + my\_rank \* (x\_n - x\_0) / my\_size;

x\_n\_rank = x\_0 + (my\_rank + 1) \* (x\_n - x\_0) / my\_size;

int n = 10/my\_size;

int n\_global = n \* my\_size;

double dx = (x\_n - x\_0) / n\_global;

double kof = dt / (dx \* dx);

double \*t\_now = new double[n\_global];

double \*t\_follow = new double[n\_global];

double t\_recv;

double \*t\_recv\_vector = new double[n\_global];

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

{

t\_now[i] = sin(0.55 \* i \* dx + 0.03);

}

t\_now[0] = 0.03;

t\_now[n\_global - 1] = 0.354;

std::cout << my\_rank << "\n";

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

{

std::cout << t\_now[i] << " ";

}

std::cout << "\n";

for (int time\_step = 0; time\_step < 10; time\_step++)

{

if (my\_rank == 0)

{

t\_follow[0] = t\_now[0];

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

{

MPI\_Send(&t\_now[n - 1], 1, MPI\_DOUBLE, my\_rank + 1, 0, MPI\_COMM\_WORLD);

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

t\_now[n] = t\_recv;

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

{

t\_follow[i] = kof \* (t\_now[i - 1] - 2 \* t\_now[i] + t\_now[i + 1]) + t\_now[i];

}

}

else

{

for (int i = 1; i < n - 1; i++)

{

t\_follow[i] = kof \* (t\_now[i - 1] - 2 \* t\_now[i] + t\_now[i + 1]) + t\_now[i];

}

t\_follow[n - 1] = t\_now[n - 1];

}

}

else if (my\_rank == my\_size - 1 && my\_rank != 0)

{

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

t\_now[n \* my\_rank - 1] = t\_recv;

for (int i = n \* my\_rank; i < n \* (my\_rank + 1) - 1; i++)

{

t\_follow[i] = kof \* (t\_now[i - 1] - 2 \* t\_now[i] + t\_now[i + 1]) + t\_now[i];

}

t\_follow[n\_global - 1] = t\_now[n\_global - 1];

MPI\_Send(&t\_follow[n \* my\_rank], 1, MPI\_DOUBLE, my\_rank - 1, 0, MPI\_COMM\_WORLD);

}

else

{

MPI\_Send(&t\_now[n \* my\_rank], 1, MPI\_DOUBLE, my\_rank - 1, 0, MPI\_COMM\_WORLD);

MPI\_Send(&t\_now[n \* (my\_rank + 1) - 1], 1, MPI\_DOUBLE, my\_rank + 1, 0, MPI\_COMM\_WORLD);

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

t\_now[n \* my\_rank - 1] = t\_recv;

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

t\_now[n \* (my\_rank + 1)] = t\_recv;

for (int i = n \* my\_rank; i < n \* (my\_rank + 1); i++)

{

t\_follow[i] = kof \* (t\_now[i - 1] - 2 \* t\_now[i] + t\_now[i + 1]) + t\_now[i];

}

}

if (my\_rank != 0) { MPI\_Send(t\_now, n\_global, MPI\_DOUBLE, 0, 0, MPI\_COMM\_WORLD); }

else

{

double\* t\_rezult = new double[n\_global];

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

{

t\_rezult[j] = t\_now[j];

}

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

{

MPI\_Recv(t\_recv\_vector, n\_global, MPI\_DOUBLE, i, 0, MPI\_COMM\_WORLD, &status);

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

{

t\_rezult[j] = t\_recv\_vector[j];

}

}

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

{

std::cout << t\_rezult[i] << " ";

}

std::cout << "\n";

}

t\_now = t\_follow;

}

MPI\_Finalize();

}

Результаты:

|  |  |
| --- | --- |
|  |  |
| Распределение температур во времени и пространстве при использовании 1 процессора | Распределение температур во времени и пространстве при использовании 3 процессоров |

|  |  |
| --- | --- |
| 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.

# Список 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.