Министерство науки и высшего образования Российской Федерации

Новосибирский Государственный технический университет

Кафедра автоматизированных систем управления



**Отчет по лабораторной работе №5**

**по дисциплине «Параллельное программирование»**

**«Знакомство с MPI»**

Вариант - 7

Выполнили

студенты группы АВТ-813:

Кинчаров Данил

Пайхаев Алексей

Чернаков Кирилл

Преподаватель:

Ландовский Владимир Владимирович,

к.т.н., доцент кафедры АСУ

г. Новосибирск

2020 г.

Содержание

[1. Постановка задачи. 3](#_Toc54807100)

[2. Описание алгоритмов с учетом взаимодействия процессов и с описанием структур данных, использующихся в ходе взаимодействия. 3](#_Toc54807101)

[3. Примеры работы программы. 8](#_Toc54807102)

[4. Описание процесса работы программы с использованием диаграммы последовательности. 8](#_Toc54807103)

[5. Результаты работы программы. **Ошибка! Закладка не определена.**](#_Toc54807104)

[6. Выводы. 10](#_Toc54807105)

# **1. Постановка задачи:**

Реализовать параллельный алгоритм численного интегрирования методом трапеций с помощью MPI.

# **2. Задание:**

# Нулевой процесс отправляет границы интервалов остальным процессам, использовать рассылку (Scatter);

# Для получения окончательного результата выполняется сборка данных (Gather) и суммирование нулевым процессом.

# **3. Пример работы программы:**

В нашем случае MPI с помощью нескольких процессов считает определённый интеграл методом трапеций, и нулевой процесс рассылает границы интервалов остальным процессам, использовать рассылку (Scatter), а для получения окончательного результата выполняется сборка данных (Gather) и суммирование нулевым процессом.

Подынтегральная функция:

Пределы интегрирования:

Шаг:

В таблице на рисунке 1 приведены результаты выполнения программы, выполнявшейся с использованием процессора 4/8 (Intel core i7-7700HQ), а также график зависимости времени от количества процессов.

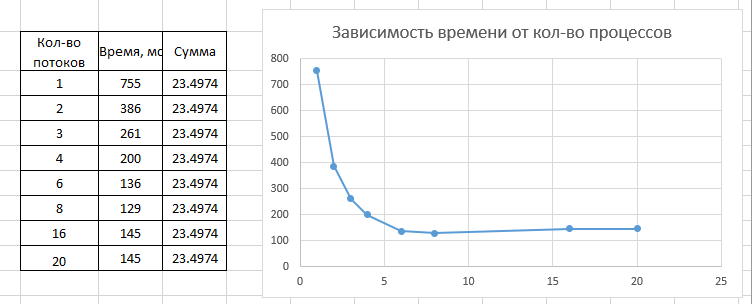


Рисунок 1 – Таблица с полученными данными и график зависимости времени от количества процессов

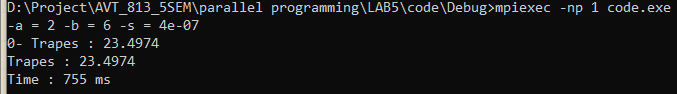


Рисунок 2 – Результат работы программы при использовании 1 процесса

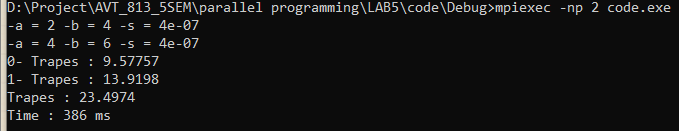


Рисунок 3 – Результат работы программы при использовании 2 процесса

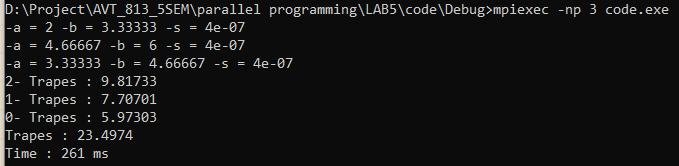


Рисунок 4 – Результат работы программы при использовании 3 процесса

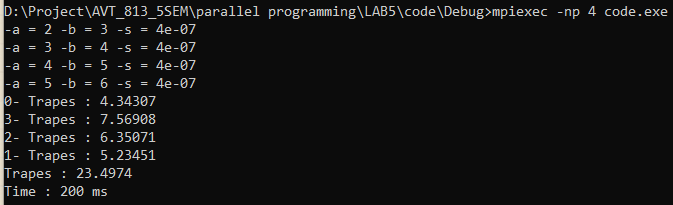


Рисунок 5 – Результат работы программы при использовании 4 процесса

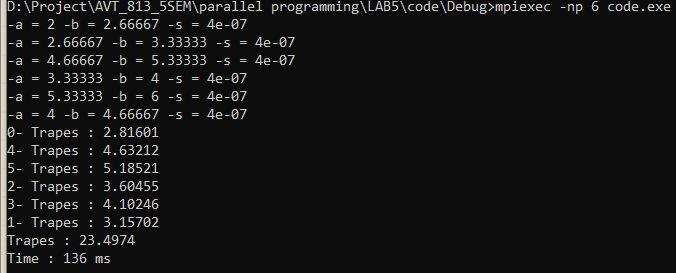


Рисунок 6 – Результат работы программы при использовании 6 процессов

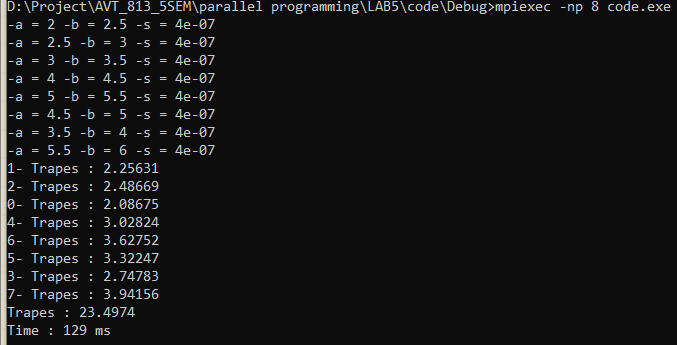


Рисунок 7 – Результат работы программы при использовании 8 процессов

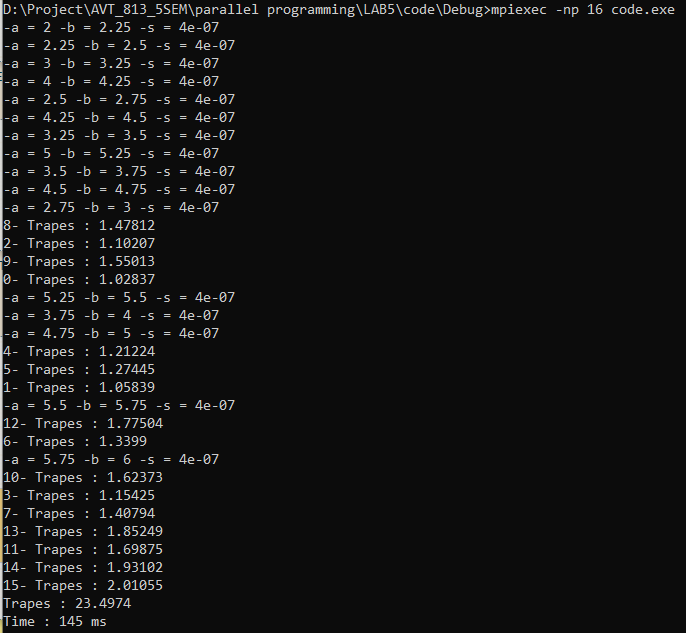


Рисунок 8 – Результат работы программы при использовании 16 процессов

# 

# **4. Листинг программы:**

#include <mpi.h>

#include <cmath>

#include <iostream>

#include <ctime>

double Fx(double *x*)

{

   return (sqrt(x) \* x) / log(x);

}

double trapesIntegr(double *a*, double *b*, double *shift*)

{

   std::cout << "-a = " << a << " -b = " << b << " -s = " << shift << std::endl;

   double C0 = 3. / 8.;

   double h = (b - a) / shift;

   double result = 0;

   double x1 = a;

   double S = C0 \* shift \* result;

   for (int i = 0; i < h / 3; ++i)

   {

      result += (Fx(x1) + 3 \* Fx(x1 + shift) + 3 \* Fx(x1 + shift \* 2) + Fx(x1 + shift \* 3));

      x1 = x1 + shift \* 3;

   }

   S = C0 \* shift \* result;

   return S;

}

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

{

   MPI\_Init(&argc, &argv);

   double a = 2, b = 6;

   int rank;

   int numTasks;

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

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

   const int numArgs = 3;

   double\* sendBuf = new double[numArgs \* numTasks];

   double recvBuffer[numArgs];

   double step = (b - a) / numTasks;

   double shift = (b - a) / 10000000.;

   clock\_t time = clock();

   if (rank == 0) {

      for (size\_t i = 0; i < numTasks \* 3; i += 3) {

         sendBuf[i] = a + i / 3 \* step;

         sendBuf[i + 1] = sendBuf[i] + step;

         sendBuf[i + 2] = shift;

      }

   }

   MPI\_Scatter(sendBuf, 3, MPI\_DOUBLE, recvBuffer, 3, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

   double res = trapesIntegr(recvBuffer[0], recvBuffer[1], recvBuffer[2]);

   std::cout << rank << "- Trapes : " << res << std::endl;

   double \*curRes = new double[numTasks + 1];

    MPI\_Gather(&res, 1, MPI\_DOUBLE, curRes, 1, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

    if (rank == 0)

    {

       time = clock() - time;

       res = 0;

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

          res += curRes[i];

        std::cout << "Trapes : " << res << std::endl;

        std::cout << "Time : " << time << " ms" << std::endl;

     }

     delete[] curRes;

   MPI\_Finalize();

}

# **6. Вывод:**

В ходе лабораторной работы была написана программа, с помощью которой осуществляется подсчёт определённого интеграла при помощи MPI с множеством процессов и метода трапеций.

Исходя из результатов тестирования, можно сделать вывод о том, что в результате работы программы при увеличении числа процессов уменьшается время работы программы, а после превышения числа доступных для процессора процессов идёт увеличение времени работы программы, так как процессы ожидают освобождения вычислительной мощности процессора.