МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФГБОУ ВО «АЛТАЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт цифровых технологий, электроники и физики (ИЦТЭФ)
Кафедра вычислительной техники и электроники (ВТиЭ)

Разработка и отладка параллельной MPI-программы для метода Якоби сеточного решения уравнения Лапласа

Отчет по лабораторной работе №2

Выполнил: студент гр. 5.306М
_____ А. В. Лаптев
Проверил: ст. преп. кафедры ВТиЭ
_____ И. А. Шмаков
«____» _____ 2024г.

Барнаул, 2024г.

СОДЕРЖАНИЕ

Цель работы

Изучить метод Якоби, проанализировать приведенный текст параллельной программы этого метода, который необходимо дополнить недостающими блоками для того, чтобы программа отрабатывала без ошибок и корректно решала поставленную задачу.

Задание

1. изучить метод Якоби, изложенный в Приложении (имя файла «Метод_итераций_Якоби.doc»). Проанализировать ниже приведенный текст параллельной программы этого метода, который необходимо дополнить недостающими блоками для того, чтобы программа отрабатывала без ошибок и корректно решала поставленную задачу;
2. после доработки и успешной отладки программы определяются Граничные условия на искомую функцию поверхности (решения уравнения Лапласа), заданные на сторонах прямоугольной сеточной области. Образец задания показан ниже в примере выполнения программы.

Выполнение работы

После изучения метода Якоби был рассмотрен текст программы, в который были внесены некоторые изменения и дополнения. Далее будут описаны те изменения, которые были внесены.

Изменения исходного кода

Первым шагом было добавлено считывание размера сетки и количества итераций для обмена границ внутри сетки, а также добавлен расчет размера полосы, который будет обрабатываться одним потоком/процессором.

Следующим шагом был добавлен вывод результатов для рассчитанной погрешности в переменную *maxdiff*.

После чего были инициализированы матрицы, содержащие граничные условия, матрицы были инициализированы нулями. Также было добавлено определение соседей справа и слева от текущего значения внутренней сетки (исключая границы).

В самом конце была вычислена собственная погрешность расчетов *mydiff*, которая вычисляется как максимальное значение между текущим ее значением и модулем разности между новым и старым значением конкретной ячейки в сетке.

Обработка результатов работы программы

В результате работы программы был сформирован текстовый файл со всеми необходимыми исходными данными для построения графика.

Ниже будет представлено несколько графиков, которые отображают зависимость различных параметров, используемых в расчетах друг от друга.

На данном графике приведено «седло», которое получается при тестовых Граничных Условиях, до получения персональных, а именно функция решения уравнения Лапласа удовлетворяет Граничным Условиям $\Phi = 1 = const$ вдоль одной размерности и условию $\Phi = 0 = const$ вдоль другой размерности.

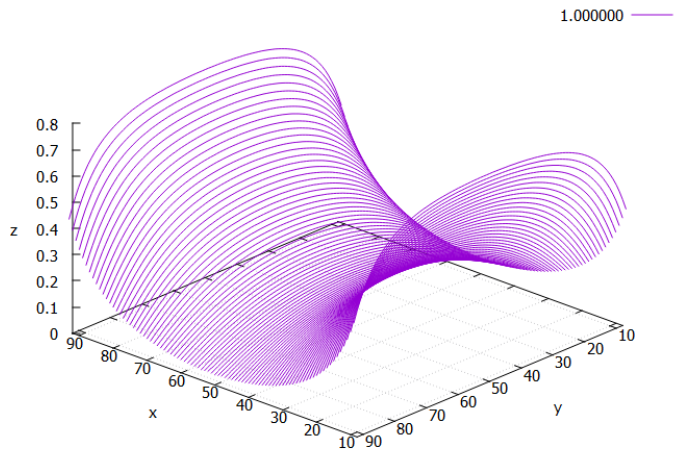

Рис. 1 Трехмерное изображение поверхности , удовлетворяющее уравнению Лапласа и Граничным Условиям.

Ниже представлен график для персональных Граничных Условий, а именно: $\Phi_L = const = 12$, $\Phi_U = const = 11$, $\Phi_R = const = 4$, $\Phi_D = const = -1$.

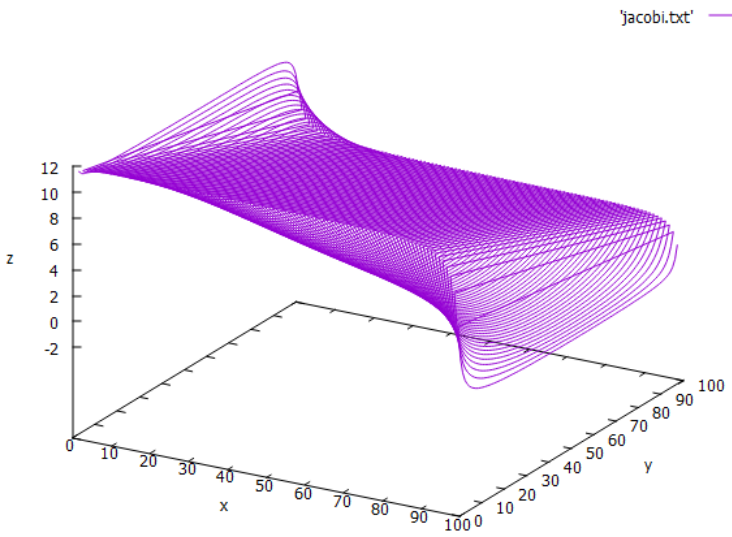

Рис. 2 Трехмерное изображение поверхности, удовлетворяющее уравнению Лапласа и персональным Граничным Условиям.

Вывод

В ходе выполнения работы было осуществлено знакомство с методом Якоби для решения сеточного уравнения Лапласа и изучена его программная реализация. Были внесены необходимые изменения и дополнения в код и построены графики с различными Граничными Условиями.

ПРИЛОЖЕНИЕ

Листинг 1 Исходный код для расчета методом Якоби

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define MAXGRID 258 //максимальный размер сетки с границами
#define COORDINATOR 0//номер управляющего процесса
#define TAG 0// не используется

static void Coordinator (int,int,int);
static void Worker (int,int,int,int,int,int);

void main (int argc, char *argv[])
{
    int myid, numIters, mode;
    int numWorkers, gridSize;//предполагается, что
    int stripSize;//gridSize кратно numWorkers

    MPI_Init (&argc, &argv);//инициализация MPI
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);
    MPI_Comm_size(MPI_COMM_WORLD, &numWorkers);
    numWorkers--;//один управляющий, остальные - рабочие
//???????   прочитать gridSize и numIters; вычислить stripSize;
    mode = atoi(argv[1]);
    gridSize = atoi(argv[2]);
    numIters = atoi(argv[3]);
    stripSize = gridSize / numWorkers;

    if (myid == COORDINATOR)
        Coordinator(numWorkers, stripSize, gridSize);
    else
```

```
31          Worker (myid, numWorkers, stripSize, gridSize, numIters,
             ↪  mode);
32      MPI_Finalize();//окончание работы
33  }
34
35  static void Coordinator (int numWorkers, int stripSize, int
     ↪  gridSize)
36  {
37      double grid[MAXGRID][MAXGRID];
38      double mydiff = 0.0, maxdiff = 0.0;
39      int i, worker, startrow, endrow;
40      MPI_Status status;
41  //получить окончательные значения в сетке от Workers
42      for (worker=1;worker <= numWorkers; worker ++)
43      {
44          startrow = (worker-1)*stripSize + 1;
45          endrow = startrow +stripSize  -1;
46              for (i=startrow; i<=endrow;i++)
47                MPI_Recv(&grid[i][1], gridSize, MPI_DOUBLE, worker,
48                  TAG, MPI_COMM_WORLD, &status);
49      }
50  //редукция погрешностей от Workers
51      MPI_Reduce(&mydiff, &maxdiff, 1 ,MPI_DOUBLE,
52          MPI_MAX, COORDINATOR, MPI_COMM_WORLD);
53  //??????? вывести результаты;
54      printf("%.16f\n", maxdiff);
55      fflush( stdout );
56  }
57
58  static void Worker (int myid, int numWorkers,
59                      int stripSize, int gridSize, int numIters, int
                         ↪  mode)
60  {
61      double grid [2][MAXGRID][MAXGRID];
62      double mydiff, maxdiff;
```

```
63      int i, j, iters;
64      int current = 0, next = 1;//текущая и следующая сетки
65      int left, right;
66      MPI_Status status;
67  //????? инициализировать матрицы; определить соседей left и right;
68      if (mode == 1) {
69          for (int i = 0; i < gridSize; i++)
70          {
71              for (int j = 0; j < gridSize; j++)
72              {
73                  if ((i == 0) || (i == gridSize - 1)) {
74                      grid[current][i][j] = 1.0;
75                      grid[next][i][j] = 1.0;
76                  }
77                  else if ((j == 0) || (j == gridSize - 1)) {
78                      grid[current][i][j] = 0.0;
79                      grid[next][i][j] = 0.0;
80                  }
81                  else
82                      grid[current][i][j] = 0.0;
83              }
84          }
85      }
86      else if (mode == 2) {
87          for (int i = 0; i < gridSize; i++)
88          {
89              for (int j = 0; j < gridSize; j++)
90              {
91                  if (i == 0) {
92                      grid[current][i][j] = 11.0;
93                      grid[next][i][j] = 11.0;
94                  }
95                  else if (i == gridSize - 1) {
96                      grid[current][i][j] = -1.0;
97                      grid[next][i][j] = -1.0;
```

```
98                  }
99                  else if (j == 0) {
100                     grid[current][i][j] = 12.0;
101                     grid[next][i][j] = 12.0;
102                 }
103                 else if (j == gridSize - 1) {
104                     grid[current][i][j] = 4.0;
105                     grid[next][i][j] = 4.0;
106                 }
107                 else
108                     grid[current][i][j] = 0.0;
109             }
110         }
111     }
112
113     for (int i = 1; i < gridSize - 1; i++)
114     {
115         for (int j = 1; j < gridSize - 1; j++)
116         {
117             if (i != gridSize)
118                 right = grid[current][i + 1][j];
119             else if (i != 1)
120                 left = grid[current][i - 1][j];
121
122             for (iters = 1; iters<=numIters; iters++)
123             {//обмен границами с соседями
124               if (right !=0) MPI_Send (&grid[next][stripSize][1],
125                     gridSize, MPI_DOUBLE, right, TAG,
                       ↪  MPI_COMM_WORLD);
126               if (left !=0) MPI_Send (&grid[next][1][1],gridSize,
127                     MPI_DOUBLE, left, TAG, MPI_COMM_WORLD);
128               if (left !=0) MPI_Recv(&grid[next][0][1], gridSize,
129                  MPI_DOUBLE, left, TAG, MPI_COMM_WORLD, &status);
130                if (right !=0)
                   ↪  MPI_Recv(&grid[next][stripSize+1][1],
```

```
131                     gridSize, MPI_DOUBLE, right, TAG,
132                     MPI_COMM_WORLD, &status);
133         // обновить свои точки
134                 for (i = 1; i<stripSize - 1; i++)
135                     for (j = 1; j<gridSize - 1; j++)
136                        grid[next][i][j] = (grid[current][i-1][j]+
137                         grid[current][i+1][j] +
                         ↪  grid[current][i][j-1] +
138                         grid[current][i][j+1]) * 0.25;
139                 current = next; next = 1-next;//поменять местами
                     ↪  сетки
140             }
141         }
142     }
143
144
145 //отправить строки окончательной сетки управляющему процессу
146      for (i=1;i<=stripSize;i++)
147     {
148         MPI_Send(&grid[current][i][1], gridSize, MPI_DOUBLE,
149         COORDINATOR, TAG, MPI_COMM_WORLD);
150     }
151 //?????? вычислить  mydiff;
152     for (int i = 1; i < gridSize; i++)
153     {
154         for(int j = 1; j < gridSize; j++)
155         {
156             mydiff = fmax(mydiff, abs(grid[current][i][j] -
                 ↪  grid[next][i][j]));
157         }
158     }
159 //редукция mydiff в управляющем процессе
160     MPI_Reduce(&mydiff, &maxdiff, 1, MPI_DOUBLE,
161                 MPI_MAX, COORDINATOR, MPI_COMM_WORLD);
162
```

```
163     FILE *graph;
164     graph = fopen("jacobi.txt", "w");
165     for (int i = 1; i < gridSize - 1; i++)
166     {
167         for (int j = 1; j < gridSize - 1; j++)
168         {
169            fprintf(graph, "%d\t%d\t%f\n", i, j, grid[next][i][j]);
170         }
171     }
172     fclose(graph);
173 }
```