Новосибирский Государственный Университет

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

Курс “Основы параллельного программирования”

Лабораторная работа №4

«**Решение уравнения Пуассона в MPI**»

Выполнил:
Пятаев Егор, гр. 15206
Преподаватель:
Городничев Максим Александрович

Новосибирск 2017

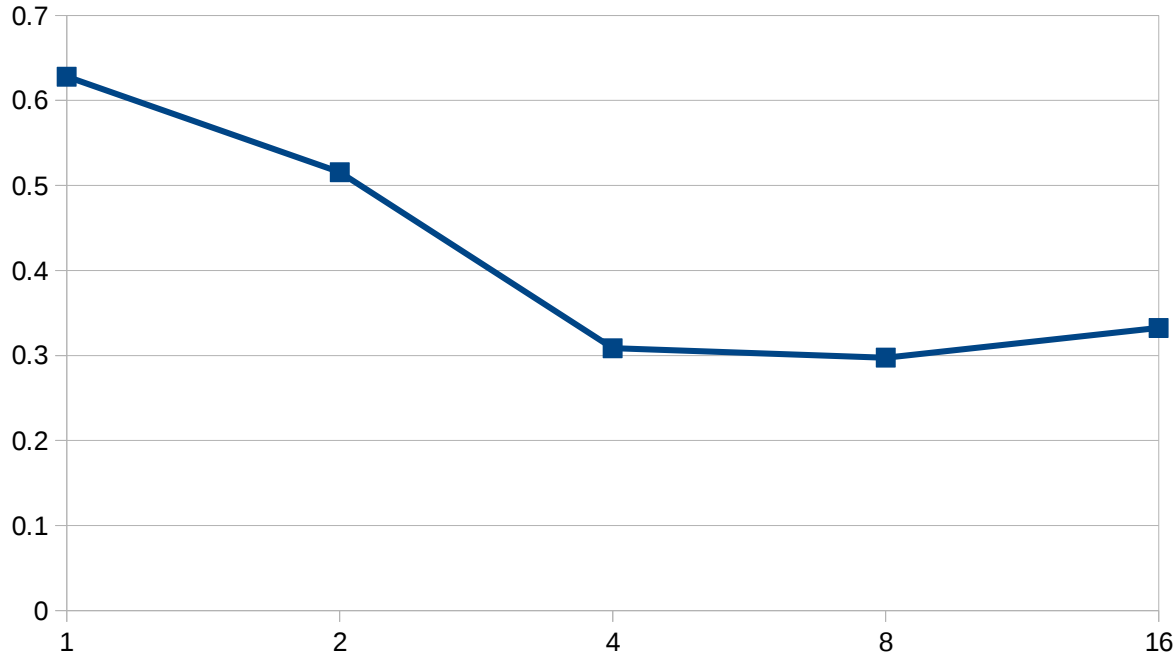
Цели работы

1. Написать параллельную программу на языке C/C++ с использованием MPI, реализующую решение уравнение Пуассона.
2. Замерить время работы программы при использовании различного числа процессорных ядер: 1 .. 16. Построить графики зависимости времени работы программы, ускорения и эффективности распараллеливания от числа используемых ядер.
3. Выполнить профилирование программы с помощью MPE.

Графики времени работы программы в зависимости от размера матрицы

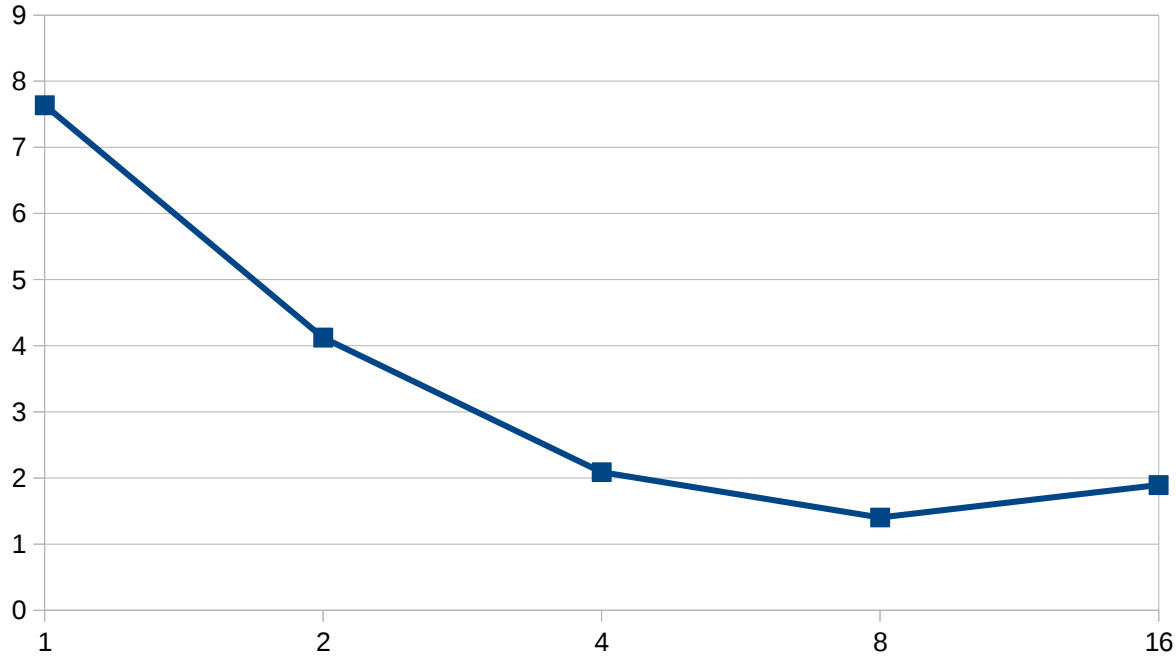
Матрица 64

Ядра	Время, сек	Ускорение	Эффективность
1	0.627990	1	1
2	0.515580	1.22	0.61
4	0.308516	2.04	0.51
8	0.297456	2.11	0.26
16	0.332391	1.89	0.12



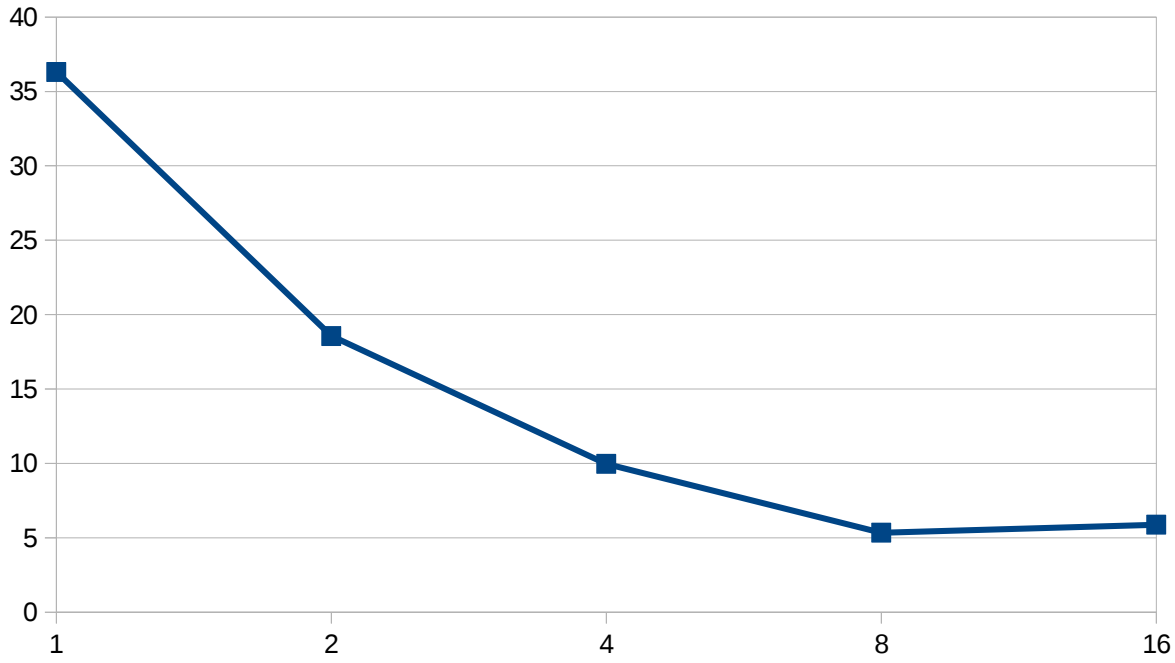
Матрица 128

Ядра	Время, сек	Ускорение	Эффективность
1	7.636695	1	1
2	4.123884	1.85	0.93
4	2.088577	3.66	0.91
8	1.403572	5.44	0.68
16	1.894186	4.03	0.25



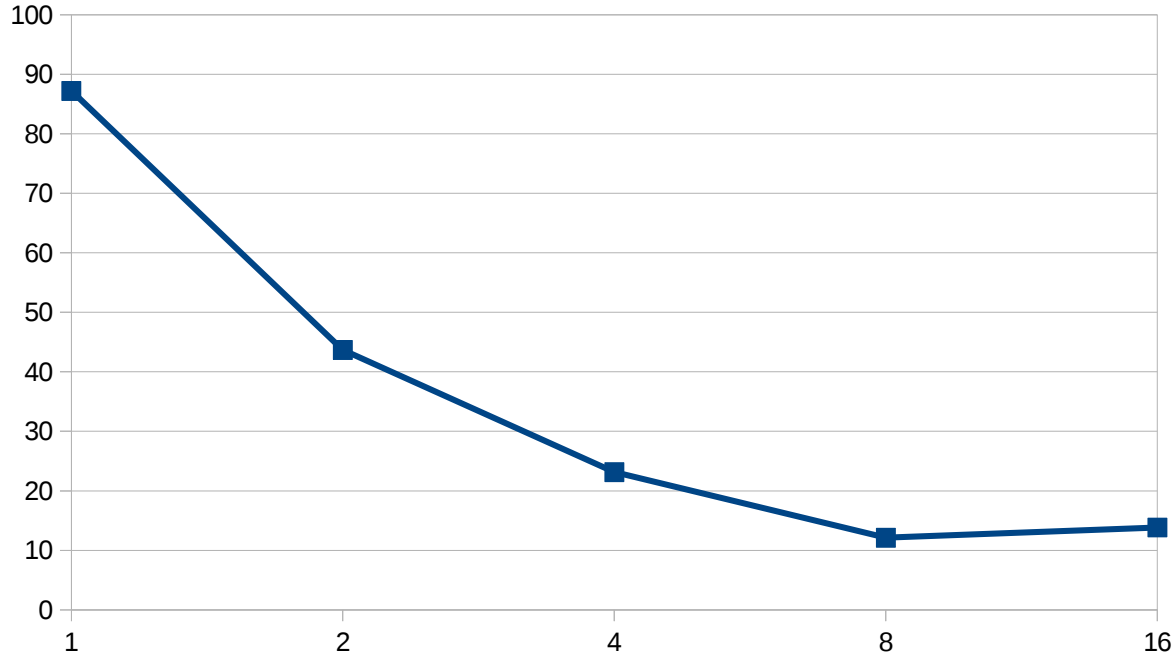
Матрица 192

Ядра	Время, сек	Ускорение	Эффективность
1	36.309083	1	1
2	18.563979	1.96	0.98
4	9.970704	3.64	0.91
8	5.339991	6.8	0.85
16	5.886346	6.17	0.39

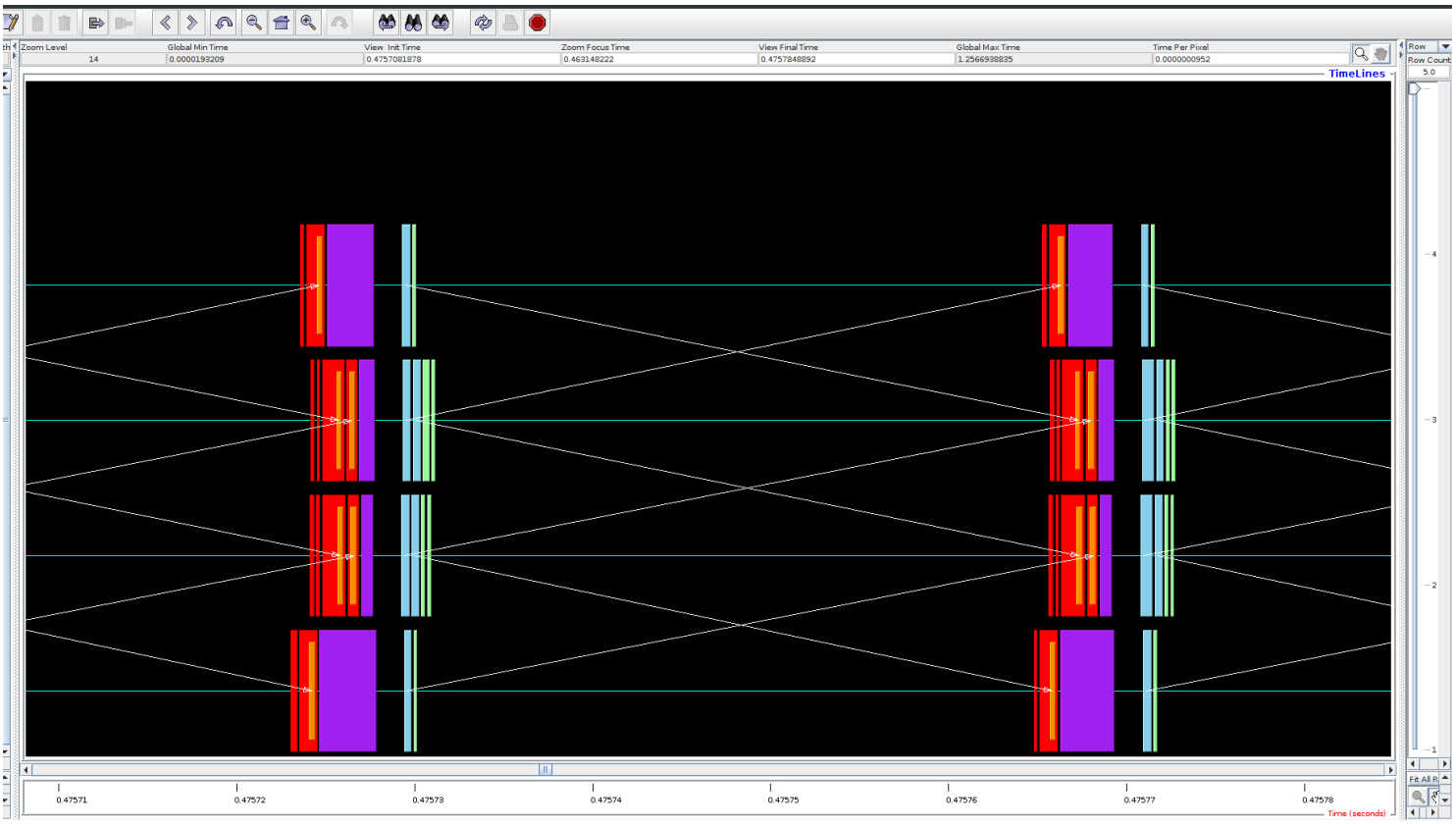


Матрица 240

Ядра	Время, сек	Ускорение	Эффективность
1	87.190779	1	1
2	43.665858	2	1
4	23.143270	3.77	0.94
8	12.120076	7.19	0.9
16	13.834582	6.3	0.39



Профилирование



Код программы

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <mpi.h>

const double EPS = 0.00001;

int main(int argc, char *argv[]){
  int N = 64;
  int m;
  int size, rank;
  int i;
  int j;
  double t1, t2, tmax;
  double prev;
  double locmax = 0;
  double globmax;
  double lmd = 0;
  double gmd;
  double tmp;

  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  MPI_Request req[4];
  MPI_Status st[4];

  m = N / size;
  double A[m + 2][N + 2];
  double B[m][N];

  for(i = 0; i < m; i++){
    for(j = 0; j < N; j++){
      B[i][j] = 0;
    }
  }

  for(i = 0; i < m + 2; i++){
    for(j = 0; j < N + 2; j++){
      if(i == 0 && rank == 0) A[i][j] = j;
      else if(i == (m + 1) && rank == size - 1) A[i][j] = (N + 1) + j;
      else if(j == 0) A[i][j] = (i + m * rank);
      else if(j == N + 1) A[i][j] = (i + m * rank) + (N + 1);
      else A[i][j] = 0;
    }
  }

  t1 = MPI_Wtime();

  do {
    for(i = 1; i <= N; i++){
      prev = B[0][i - 1];
      B[0][i - 1] = 0.25 * (A[0][i] + A[2][i] + A[1][i + 1] + A[1][i - 1]);
      if(fabs(B[0][i - 1] - prev) > locmax) locmax = fabs(B[0][i - 1] - prev);
      prev = B[m - 1][i - 1];
      B[m - 1][i - 1] = 0.25 * (A[m][i - 1] + A[m][i + 1] + A[m - 1][i] + A[m + 1][i]);
      if(fabs(B[m - 1][i - 1] - prev) > locmax) locmax = fabs(B[m - 1][i - 1] - prev);
    }

    if(rank != 0) MPI_Isend(&B, N, MPI_DOUBLE, rank - 1, 0, MPI_COMM_WORLD, &req[0]);
    if(rank != size - 1) MPI_Isend(&B[m - 1][0], N, MPI_DOUBLE, rank + 1, 0, MPI_COMM_WORLD, &req[1]);

    if(rank != 0) MPI_Irecv(&A[0][1], N, MPI_DOUBLE, rank - 1, 0, MPI_COMM_WORLD, &req[2]);
    if(rank != size - 1) MPI_Irecv(&A[m + 1][1], N, MPI_DOUBLE, rank + 1, 0, MPI_COMM_WORLD, &req[3]);

    for(i = 2; i <= m - 1; i++){
```

```
    for(j = 1; j <= N; j++){
      B[i - 1][j - 1] = 0.25 * (A[i][j - 1] + A[i][j + 1] + A[i - 1][j] + A[i + 1][j]);
    }
   }

   for(i = 1; i <= m; i++){
    for(j = 1; j <= N; j++){
      prev = A[i][j];
      A[i][j] = B[i - 1][j - 1];
      if(fabs(A[i][j] - prev) > locmax) locmax = fabs(A[i][j] - prev);
    }
   }

   for(i = 0; i < 4; i++){
    if((rank == 0 && (i == 0 || i == 2)) || (rank == size - 1 && (i == 3 || i == 1))) continue;
    else MPI_Wait(&req[i], &st[i]);
   }

   MPI_Allreduce(&locmax, &globmax, 1, MPI_DOUBLE, MPI_MAX, MPI_COMM_WORLD);

   locmax = 0;

  } while(globmax > EPS);

  t2 = MPI_Wtime();
  t2 -= t1;

  for(i = 0; i < m; i++){
   for(j = 0; j < N; j++){
    if((tmp = fabs((i + m * rank + j) - B[i][j] + 2)) > lmd) lmd = tmp;
   }
  }
  MPI_Allreduce(&lmd, &gmd, 1, MPI_DOUBLE, MPI_MAX, MPI_COMM_WORLD);

  if(rank == 0) printf("Max Differece = %f\n", gmd);

  MPI_Allreduce(&t2, &tmax, 1, MPI_DOUBLE, MPI_MAX, MPI_COMM_WORLD);

  if(rank == 0) printf("N = %d\nThreads = %d\nTime = %f\n", N, size, tmax);

  MPI_Finalize();

  return 0;
}
```

Выводы

Для достижения поставленных целей была реализована программа решения уравнения Пуассона, выполнены замеры времени работы при размерах матрицы 64, 128, 192, 240 и при количестве ядер 1 .. 16, выполнено профилирование.

Из результатов замеров видно, что эффективность и скорость работы программы улучшается при увеличении размеров матриц и количества процессов.