



Министерство науки и высшего образования Российской Федерации  
Калужский филиал  
федерального государственного автономного  
образовательного учреждения высшего образования  
**«Московский государственный технический университет имени Н.Э. Баумана**  
(национальный исследовательский университет)  
(КФ МГТУ им. Н.Э. Баумана)

**ФАКУЛЬТЕТ ИУК «Информатика и управление»**

**КАФЕДРА ИУК4 «Программное обеспечение ЭВМ, информационные технологии»**

**ЛАБОРАТОРНАЯ РАБОТА №4**

**«Распараллеливание алгоритмов решения СЛАУ итерационными методами»**

**ДИСЦИПЛИНА: «Параллельные процессы в информационных системах»**

Выполнил: студент гр. ИУК4-31М

\_\_\_\_\_  
(Подпись)

(Герасимова С.В.)  
(Ф.И.О.)

Проверил:

\_\_\_\_\_  
(Подпись)

(Корнюшин Ю.П.)  
(Ф.И.О.)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:

- Оценка:

Калуга, 2025

**Цель работы:** формирование практических навыков распараллеливания алгоритмов решения систем линейных алгебраических уравнений итерационными методами на языках MPI и OpenMP.

### **Задачи**

1. Получение навыков решения СЛАУ методом простой итерации на языках MPI и OpenMP.
2. Получение навыков решения СЛАУ методом сопряженных градиентов на языках MPI и OpenMP.
3. Сравнение временных характеристик двух алгоритмов решения СЛАУ методом простой итерации на языках MPI и OpenMP.

### **Задание**

#### **Вариант 2**

Разработать параллельный алгоритм, написать и отладить параллельную программу решения СЛАУ методом сопряженных градиентов в MPI.

### **Результат выполнения работы**

#### **Листинг программы**

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

void local_matvec(const double *A_local, const double *x_global, double
*y_local,
                  int local_rows, int n) {
    for (int i = 0; i < local_rows; i++) {
        double sum = 0.0;
        for (int j = 0; j < n; j++) {
            sum += A_local[i * n + j] * x_global[j];
        }
        y_local[i] = sum;
    }
}

void build_spd_matrix(double *A, int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i == j) {
                A[i * n + j] = 2.0 * n;
            } else {
                A[i * n + j] = 1.0;
            }
        }
    }
}
```

```

int main(int argc, char **argv) {
    MPI_Init(&argc, &argv);

    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    const int n = 1000;
    const double tolerance = 1e-10;
    const int max_iter = 100;

    int local_n = n / size;
    int remainder = n % size;
    if (rank < remainder) {
        local_n++;
    }

    int *counts = (int*)calloc(size, sizeof(int));
    int *displs = (int*)calloc(size, sizeof(int));
    for (int i = 0; i < size; i++) {
        counts[i] = n / size + (i < remainder ? 1 : 0);
        displs[i] = (i == 0) ? 0 : displs[i-1] + counts[i-1];
    }

    int *mat_counts = (int*)malloc(size * sizeof(int));
    int *mat_displs = (int*)malloc(size * sizeof(int));
    for (int i = 0; i < size; i++) {
        mat_counts[i] = counts[i] * n;
        mat_displs[i] = displs[i] * n;
    }

    double *A_local = (double*)calloc(local_n * n, sizeof(double));
    double *b_local = (double*)calloc(local_n, sizeof(double));
    double *x_local = (double*)calloc(local_n, sizeof(double));
    double *r_local = (double*)calloc(local_n, sizeof(double));
    double *p_local = (double*)calloc(local_n, sizeof(double));
    double *Ap_local = (double*)calloc(local_n, sizeof(double));

    if (rank == 0) {
        double *A_full = (double*)calloc(n * n, sizeof(double));
        double *b_full = (double*)malloc(n * sizeof(double));

        build_spd_matrix(A_full, n);
        for (int i = 0; i < n; i++) {
            b_full[i] = 1.0;
        }

        MPI_Scatterv(A_full, mat_counts, mat_displs, MPI_DOUBLE,
                     A_local, local_n * n, MPI_DOUBLE, 0, MPI_COMM_WORLD);
        MPI_Scatterv(b_full, counts, displs, MPI_DOUBLE,
                     b_local, local_n, MPI_DOUBLE, 0, MPI_COMM_WORLD);

        free(A_full);
        free(b_full);
    } else {
        MPI_Scatterv(NULL, NULL, NULL, MPI_DOUBLE,
                     A_local, local_n * n, MPI_DOUBLE, 0, MPI_COMM_WORLD);
        MPI_Scatterv(NULL, NULL, NULL, MPI_DOUBLE,
                     b_local, local_n, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    }

    double *x_global = (double*)calloc(n, sizeof(double));
    memcpy(r_local, b_local, local_n * sizeof(double));
    memcpy(p_local, b_local, local_n * sizeof(double));
}

```

```

double local_rr = 0.0;
for (int i = 0; i < local_n; i++) {
    local_rr += r_local[i] * r_local[i];
}
double r_norm_sq_old;
MPI_Allreduce(&local_rr, &r_norm_sq_old, 1, MPI_DOUBLE, MPI_SUM,
MPI_COMM_WORLD);

double local_b_norm_sq = 0.0;
for (int i = 0; i < local_n; i++) {
    local_b_norm_sq += b_local[i] * b_local[i];
}
double b_norm_sq;
MPI_Allreduce(&local_b_norm_sq, &b_norm_sq, 1, MPI_DOUBLE, MPI_SUM,
MPI_COMM_WORLD);
double b_norm = sqrt(b_norm_sq);

if (rank == 0) {
    printf("Initial residual norm: %e\n", sqrt(r_norm_sq_old));
}

int iter = 0;
double alpha, beta;

while (iter < max_iter) {
    double rel_residual = sqrt(r_norm_sq_old) / b_norm;
    if (rel_residual <= tolerance) {
        break;
    }

    MPI_Allgatherv(p_local, local_n, MPI_DOUBLE,
                   x_global, counts, displs, MPI_DOUBLE, MPI_COMM_WORLD);

    local_matvec(A_local, x_global, Ap_local, local_n, n);

    double local_pAp = 0.0;
    for (int i = 0; i < local_n; i++) {
        local_pAp += p_local[i] * Ap_local[i];
    }
    double pAp;
    MPI_Allreduce(&local_pAp, &pAp, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);

    if (fabs(pAp) < 1e-14) {
        if (rank == 0) {
            printf("Breakdown: p^T A p is near zero.\n");
        }
        break;
    }

    alpha = r_norm_sq_old / pAp;

    for (int i = 0; i < local_n; i++) {
        x_local[i] += alpha * p_local[i];
        r_local[i] -= alpha * Ap_local[i];
    }

    double local_rr_new = 0.0;
    for (int i = 0; i < local_n; i++) {
        local_rr_new += r_local[i] * r_local[i];
    }
    double r_norm_sq_new;
    MPI_Allreduce(&local_rr_new, &r_norm_sq_new, 1, MPI_DOUBLE, MPI_SUM,
MPI_COMM_WORLD);

    beta = r_norm_sq_new / r_norm_sq_old;
}

```

```

        for (int i = 0; i < local_n; i++) {
            p_local[i] = r_local[i] + beta * p_local[i];
        }

        r_norm_sq_old = r_norm_sq_new;
        iter++;

        if (rank == 0 && iter <= 5) {
            printf("Iteration %d: relative residual = %e\n", iter,
sqrt(r_norm_sq_old) / b_norm);
        }
    }

MPI_Gatherv(x_local, local_n, MPI_DOUBLE,
            x_global, counts, displs, MPI_DOUBLE, 0, MPI_COMM_WORLD);

if (rank == 0) {
    double final_rel_residual = sqrt(r_norm_sq_old) / b_norm;
    if (final_rel_residual <= tolerance) {
        printf("Converged in %d iterations.\n", iter);
    } else {
        printf("Failed to converge within %d iterations.\n", max_iter);
    }
    printf("Final relative residual: %e\n", final_rel_residual);

    printf("First 5 components of solution:\n");
    for (int i = 0; i < 5 && i < n; i++) {
        printf("x[%d] = %e\n", i, x_global[i]);
    }
}

free(A_local);
free(b_local);
free(x_local);
free(r_local);
free(p_local);
free(Ap_local);
free(x_global);
free(counts);
free(displs);
free(mat_counts);
free(mat_displs);

MPI_Finalize();
return 0;
}

```

```

PS E:\Dev\bmstu-magistracy\3rd-term\parallel-processing\lab4> mpiexec -n 4 ./bin/lab4.exe
Initial residual norm: 3.162278e+01
Iteration 1: relative residual = 1.110223e-16
Converged in 1 iterations.
Final relative residual: 1.110223e-16
First 5 components of solution:
x[0] = 3.334445e-04
x[1] = 3.334445e-04
x[2] = 3.334445e-04
x[3] = 3.334445e-04
x[4] = 3.334445e-04

```

**Рисунок 1 – Результат выполнения программы**

**Вывод:** в ходе выполнения лабораторной работы были сформированы практические навыки распараллеливания алгоритмов решения систем линейных алгебраических уравнений итерационными методами на языках MPI и OpenMP.