**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет информационных технологий**

**Кафедра параллельных вычислений**

# ОТЧЕТ

## О ВЫПОЛНЕНИИ ПРАКТИЧЕСКОЙ РАБОТЫ

«Параллельная реализация решения системы линейных алгебраических уравнений с помощью OpenMP»

студента 2 курса, группы 21206

**Мельникова Никиты Сергеевича**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
к.т.н., доцент
А.Ю. Власенко

Новосибирск 2023

# СОДЕРЖАНИЕ

## ЦЕЛЬ

1. Распараллелить программу на языке Си средствами OpenMP.
2. Провести исследование на определение оптимальных параметров `#pragma omp for schedule(...)`

## ЗАДАНИЕ

1. Последовательную программу из предыдущей практической работы, реализующую итерационный алгоритм решения методом минимальных невязок (приложение 1) системы линейных алгебраических уравнений вида Ax=b, распараллелить с помощью OpenMP.

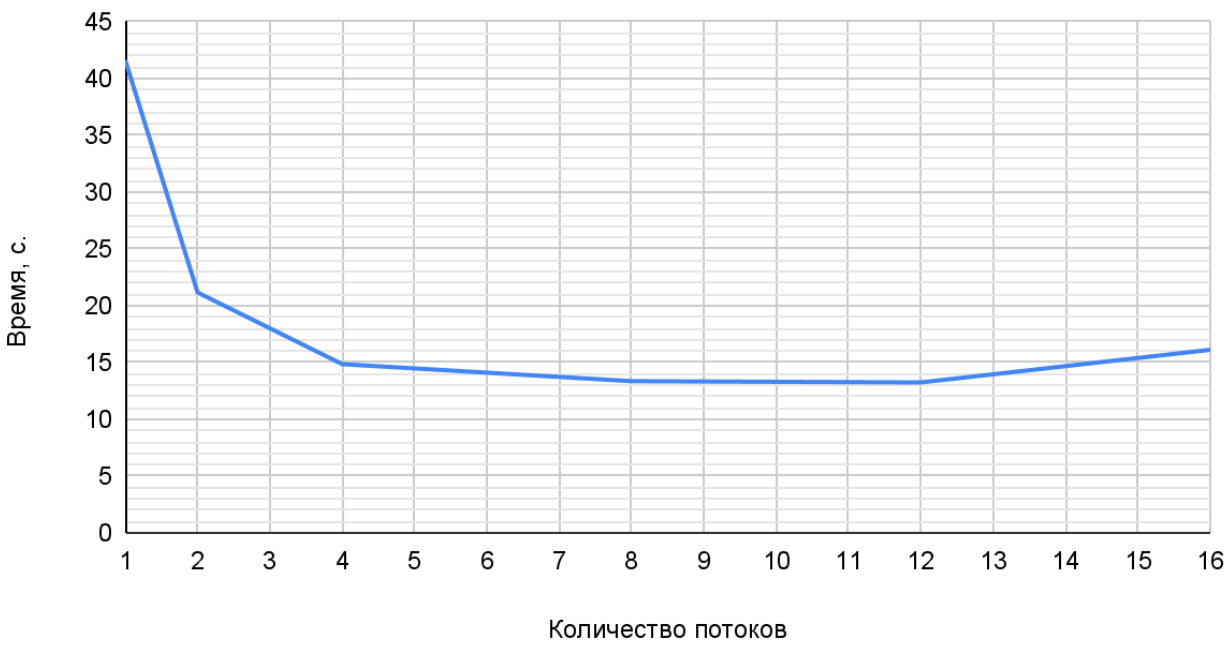
   ОБЯЗАТЕЛЬНОЕ УСЛОВИЕ: создается одна параллельная секция `#pragma omp parallel`, охватывающая весь итерационный алгоритм.

2. Замерить время работы программы на кластере НГУ на 1, 2, 4, 8, 12, 16 потоках. Построить графики зависимости времени работы программы, ускорения и эффективности распараллеливания от числа используемых ядер. Исходные данные и параметры задачи подобрать таким образом, чтобы решение задачи на одном ядре занимало не менее 30 секунд.

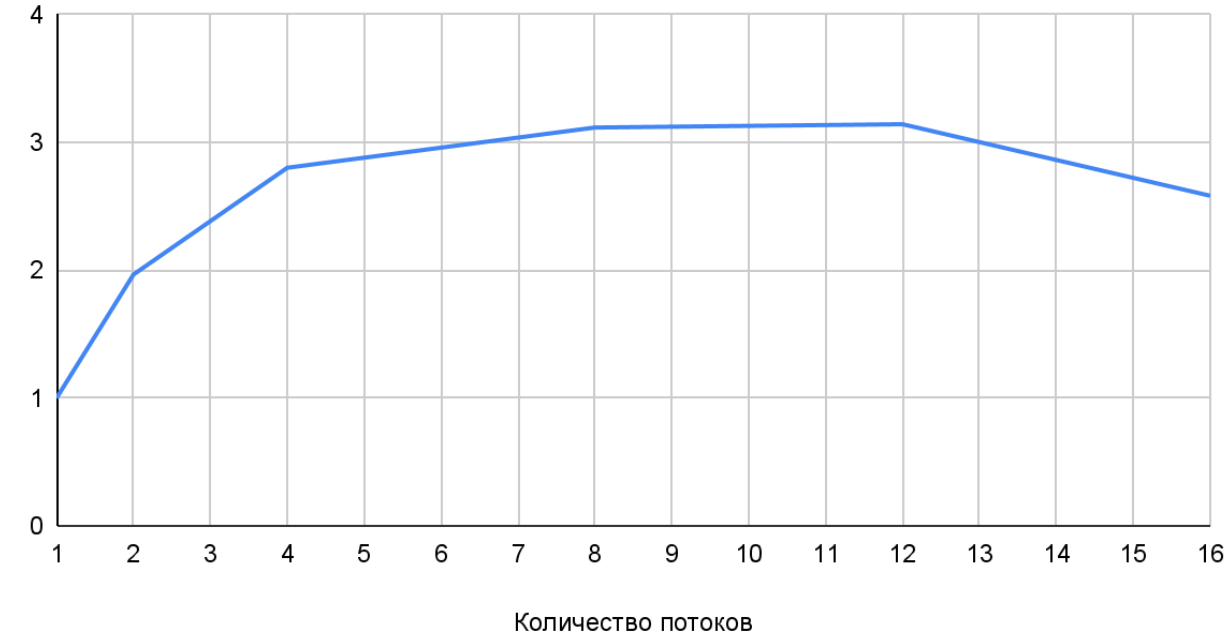
3. Провести исследование на определение оптимальных параметров `#pragma omp for schedule(...)` при некотором фиксированном размере задачи и количестве потоков.

# ОПИСАНИЕ РАБОТЫ

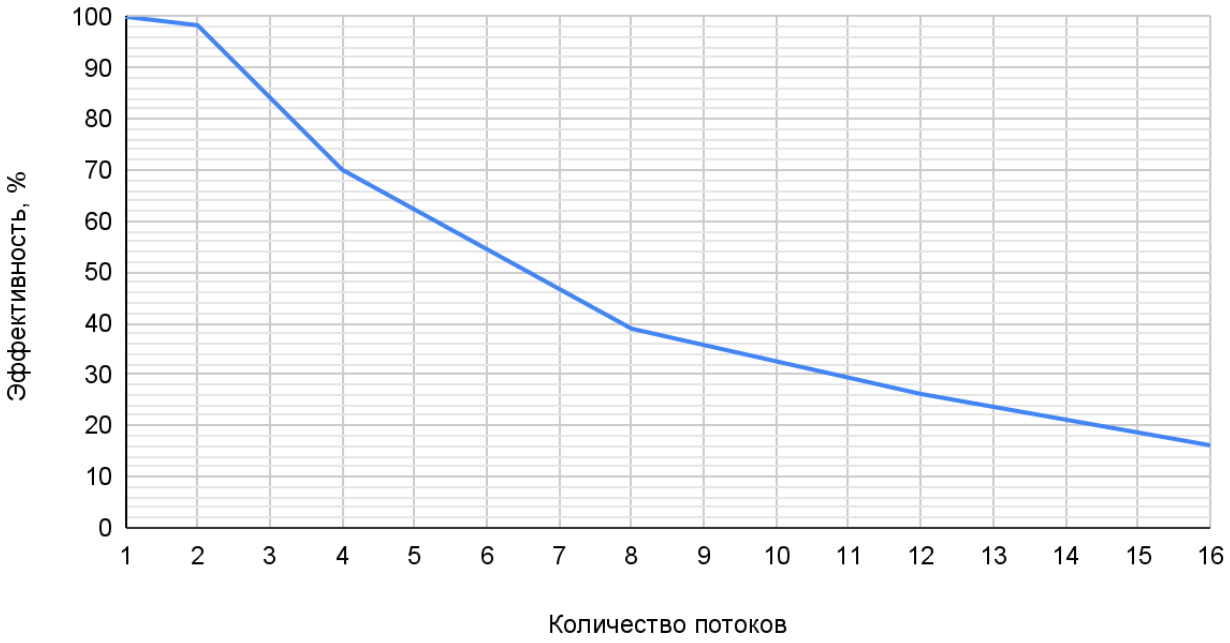
1. С помощью OpenMP была распараллелена программа на языке C (листинг 1), реализующая итерационный алгоритм решения системы линейных алгебраических уравнений вида Ax=b с использованием метода минимальных невязок (приложение 1). Здесь A– матрица размером N×N, x и b – векторы длины N. Тип элементов – double.
2. Было замерено время работы программы на кластере НГУ на 1, 2, 4, 8, 12, 16 потоках. Данные выбраны таким образом, чтобы решение задачи на одном ядре занимало не менее 30 сек (взята симметричная матрица со случайными элементами на отрезке [-1;1] с утяжеленной главной диагональю).
3. Были построены графики ускорения и эффективности, где **Ускорение**: Sp = T1 / Tp, где T1 - время работы ПОСЛЕДОВАТЕЛЬНОЙ программы. Tp - время работы параллельной программы на p процессах/потоках.
   **Эффективность**: Ep = Sp / p * 100%



## График ускорения

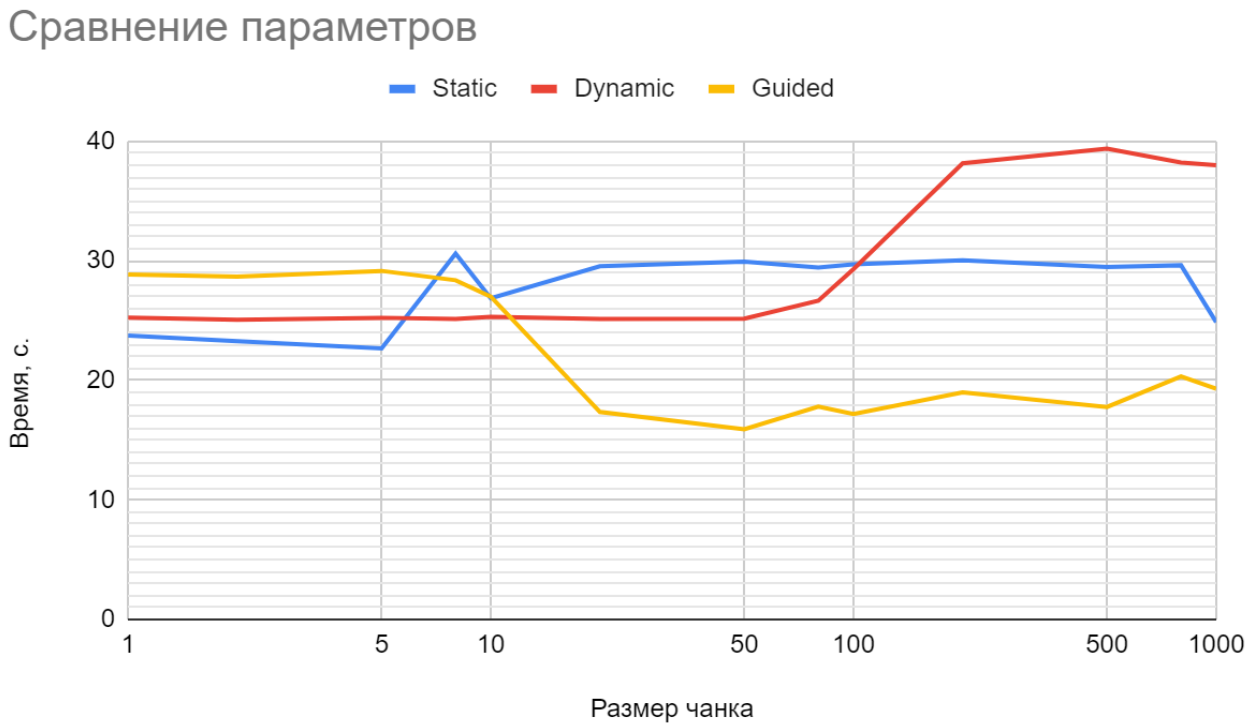



## График эффективности



4. Было проведено исследование на определение оптимальных параметров `#pragma omp for schedule(...)` при N=10000 и 4 потоках.

| Chunk | Static | Dynamic | Guided |
|---|---|---|---|
| 1 | 23,74 | 25,24 | 28,85 |
| 2 | 23,27 | 25,07 | 28,68 |
| 5 | 22,66 | 25,23 | 29,15 |
| 8 | 30,6 | 25,12 | 28,37 |
| 10 | 26,86 | 25,31 | 27 |
| 20 | 29,54 | 25,14 | 17,35 |
| 50 | 29,92 | 25,16 | 15,91 |
| 80 | 29,44 | 26,66 | 17,79 |
| 100 | 29,69 | 29,3 | 17,18 |
| 200 | 30,03 | 38,15 | 18,98 |
| 500 | 29,47 | 39,37 | 17,77 |
| 800 | 29,62 | 38,21 | 20,33 |
| 1000 | 24,86 | 38 | 19,3 |



# ЗАКЛЮЧЕНИЕ

В ходе практической работы удалось выявить существенное повышение производительности при использовании параллельных программ по сравнению с последовательными. При увеличении количества потоков время работы программы уменьшается.

В процессе исследования на определение оптимальных параметров `#pragma omp for schedule(...)`:

1. наибольшее быстродействие показала программа при использовании параметра guided с размером чанка 50 (время работы такой программы составило ≈16 секунд).
2. dynamic более эффективен при малом размере чанка.

## **Приложение 1.** *Метод минимальных невязок*

В методе минимальных невязок преобразование решения на каждом шаге задается следующими формулами:

$$y^n = Ax^n - b,$$

$$\tau^n = \frac{(y^n, Ay^n)}{(Ay^n, Ay^n)},$$

$$x^{n+1} = x^n - \tau^n y^n,$$

Критерий завершения счета:

$$\frac{\|Ax^n - b\|_2}{\|b\|_2} < \varepsilon$$

**Приложение 2.** *Листинг параллельной программы на языке Си*

*Код компиляции:* `icpc -openmp parallel.c -o parallel`

*Листинг 1 — Исходный код программы*

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

#define N                   10000
#define EPSILON             0.000001
#define MAX_ITERATION_COUNT 1000
#define DIAGONAL_DOMINANCE  120
#define MEASURES            3

double *fill_matrix(double *A) {
    srand(N);

    for (size_t i = 0; i < N; ++i) {
        for (size_t j = 0; j < i; ++j) {
            A[i * N + j] = A[j * N + i];
        }
        for (size_t j = i; j < N; ++j) {
              A[i * N + j] = (double) rand() / RAND_MAX * 2.0 - 1.0;  //in
[-1; 1]
            if (i == j) A[i * N + j] += DIAGONAL_DOMINANCE;
        }
    }
    return A;
}

double *fill_vect(double *vect) {
    for (size_t i = 0; i < N; ++i)
        vect[i] = (double) rand() / RAND_MAX * (double) rand();
    return vect;
}

void solve(double *A, double *x, double *b) {
    double *y_n  = (double *) malloc(N * sizeof(double));
    double *Ay_n = (double *) malloc(N * sizeof(double));
    double tau             = 0;
    double tau_numerator   = 0;
    double tau_denominator = 0;
    double check      = 0;
    size_t iterations = 0;
```

```
    double epsilonb_norm_squared = 0;

#pragma omp parallel shared(iterations, check, epsilonb_norm_squared)
{
    //calculate epsilon*|b|^2
#pragma omp for reduction(+:epsilonb_norm_squared)
    for (size_t i = 0; i < N; ++i)
        epsilonb_norm_squared += b[i] * b[i];
#pragma omp single
    epsilonb_norm_squared *= EPSILON * EPSILON;

    //calculate y_n
#pragma omp for
    for (size_t i = 0; i < N; ++i) {
        double tmp = 0;
        for (size_t j = 0; j < N; j++) {
            tmp += A[i * N + j] * x[j];
        }
        y_n[i] = tmp - b[i];
    }

    while (iterations < MAX_ITERATION_COUNT) {
        //Ay_n = A * y_n
#pragma omp for
        for (size_t i = 0; i < N; ++i) {
            double tmp = 0;
            for (size_t j = 0; j < N; j++) {
                tmp += A[i * N + j] * y_n[j];
            }
            Ay_n[i] = tmp;
        }
        //calculate tau
#pragma omp single
{
        tau             = 0;
        tau_numerator   = 0;
        tau_denominator = 0;
}
#pragma omp for reduction(+:tau_numerator, tau_denominator)
        for (size_t i = 0; i < N; ++i) {
            tau_numerator += y_n[i] * Ay_n[i];
            tau_denominator += Ay_n[i] * Ay_n[i];
        }
        tau = tau_numerator / tau_denominator;
        //calculate x_{n+1}
#pragma omp for
```

```
        for (size_t i = 0; i < N; ++i)
            x[i] -= tau * y_n[i];
        //calc y_{n+1}
#pragma omp for
        for (size_t i = 0; i < N; ++i) {
            double tmp = 0;
            for (size_t j = 0; j < N; j++) {
                tmp += A[i * N + j] * x[j];
            }
            y_n[i] = tmp - b[i];
        }

#pragma omp single
{
        ++iterations;
        check = 0;
}
#pragma omp for reduction(+:check)
        for (size_t i = 0; i < N; ++i)
            check += y_n[i] * y_n[i];
        if (check < epsilonb_norm_squared) break;
    }
}
    free(y_n);
    free(Ay_n);
}

int main(int argc, char *argv[]) {
    double time_start = 0;
    double time_end   = 0;
    double min_time   = ULLONG_MAX;
    double *A         = (double *) malloc(N * N * sizeof(double));
    double *x         = (double *) malloc(N * sizeof(double));
    double *b         = (double *) malloc(N * sizeof(double));

    for (int i = 0; i < MEASURES; ++i) {
        fill_matrix(A);
        fill_vect(x);
        fill_vect(b);
        time_start = omp_get_wtime();
        solve(A, x, b);
        time_end   = omp_get_wtime();
             if (time_end - time_start < min_time) min_time = time_end -
time_start;
    }
    printf("Time taken: %f sec.\n", min_time);
```

```
    free(A);
    free(x);
    free(b);
    return EXIT_SUCCESS;
}
```

*Листинг 2 — Скрипт для запуска на кластере НГУ*

```
#!/bin/sh
#PBS -l walltime=00:30:00
#PBS -l select=1:ncpus=8:ompthreads=16

cd $PBS_O_WORKDIR

for i in 1 2 4 8 12 16
do
    export OMP_NUM_THREADS=$i
    echo "OMP_NUM_THREADS = $OMP_NUM_THREADS"
    ./parallel
    echo
done
```

**Приложение 3.** *Листинг параллельной программы на языке Си для определения оптимальных параметров #pragma omp for schedule(...)*

*Код компиляции:* `icpc -openmp sched.c -o sched`

*Листинг 3 — Исходный код программы*

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

#define N                   10000
#define EPSILON             0.000001
#define MAX_ITERATION_COUNT 1000
#define DIAGONAL_DOMINANCE  120
#define MEASURES            3

double *fill_matrix(double *A) {
    srand(N);

    for (size_t i = 0; i < N; ++i) {
        for (size_t j = 0; j < i; ++j) {
            A[i * N + j] = A[j * N + i];
        }
        for (size_t j = i; j < N; ++j) {
               A[i * N + j] = (double) rand() / RAND_MAX * 2.0 - 1.0;  //in
[-1; 1]
            if (i == j) A[i * N + j] += DIAGONAL_DOMINANCE;
        }
    }
    return A;
}

double *fill_vect(double *vect) {
    for (size_t i = 0; i < N; ++i)
        vect[i] = (double) rand() / RAND_MAX * (double) rand();
    return vect;
}

void solve(double *A, double *x, double *b) {
    double *y_n  = (double *) malloc(N * sizeof(double));
    double *Ay_n = (double *) malloc(N * sizeof(double));
    double tau             = 0;
    double tau_numerator   = 0;
    double tau_denominator = 0;
    double check      = 0;
```

```
    size_t iterations = 0;
    double epsilonb_norm_squared = 0;

#pragma omp parallel shared(iterations, check, epsilonb_norm_squared)
{
    //calculate epsilon*|b|^2
#pragma omp for reduction(+:epsilonb_norm_squared) schedule(runtime)
    for (size_t i = 0; i < N; ++i)
        epsilonb_norm_squared += b[i] * b[i];
#pragma omp single
    epsilonb_norm_squared *= EPSILON * EPSILON;

    //calculate y_n
#pragma omp for schedule(runtime)
    for (size_t i = 0; i < N; ++i) {
        double tmp = 0;
        for (size_t j = 0; j < N; j++) {
            tmp += A[i * N + j] * x[j];
        }
        y_n[i] = tmp - b[i];
    }

    while (iterations < MAX_ITERATION_COUNT) {
        //Ay_n = A * y_n
#pragma omp for schedule(runtime)
        for (size_t i = 0; i < N; ++i) {
            double tmp = 0;
            for (size_t j = 0; j < N; j++) {
                tmp += A[i * N + j] * y_n[j];
            }
            Ay_n[i] = tmp;
        }
        //calculate tau
#pragma omp single
{
        tau             = 0;
        tau_numerator   = 0;
        tau_denominator = 0;
}
#pragma     omp     for     reduction(+:tau_numerator,     tau_denominator)
schedule(runtime)
        for (size_t i = 0; i < N; ++i) {
            tau_numerator += y_n[i] * Ay_n[i];
            tau_denominator += Ay_n[i] * Ay_n[i];
        }
        tau = tau_numerator / tau_denominator;
```

```
        //calculate x_{n+1}
#pragma omp for schedule(runtime)
        for (size_t i = 0; i < N; ++i)
            x[i] -= tau * y_n[i];
        //calc y_{n+1}
#pragma omp for schedule(runtime)
        for (size_t i = 0; i < N; ++i) {
            double tmp = 0;
            for (size_t j = 0; j < N; j++) {
                tmp += A[i * N + j] * x[j];
            }
            y_n[i] = tmp - b[i];
        }

#pragma omp single
{
        ++iterations;
        check = 0;
}
#pragma omp for reduction(+:check) schedule(runtime)
        for (size_t i = 0; i < N; ++i)
            check += y_n[i] * y_n[i];
        if (check < epsilonb_norm_squared) break;
    }
}
    free(y_n);
    free(Ay_n);
}

int main(int argc, char *argv[]) {
    double time_start = 0;
    double time_end   = 0;
    double min_time   = ULLONG_MAX;
    double *A         = (double *) malloc(N * N * sizeof(double));
    double *x         = (double *) malloc(N * sizeof(double));
    double *b         = (double *) malloc(N * sizeof(double));

    for (int i = 0; i < MEASURES; ++i) {
        fill_matrix(A);
        fill_vect(x);
        fill_vect(b);
        time_start = omp_get_wtime();
        solve(A, x, b);
        time_end   = omp_get_wtime();
             if (time_end - time_start < min_time) min_time = time_end -
time_start;
```

```
    }
    printf("%f\n", min_time);

    free(A);
    free(x);
    free(b);
    return EXIT_SUCCESS;
}
```

*Листинг 4 — Скрипт для запуска на кластере НГУ*

```
#!/bin/sh
#PBS -l walltime=00:60:00
#PBS -l select=1:ncpus=4:ompthreads=4

cd $PBS_O_WORKDIR
echo "OMP_NUM_THREADS = $OMP_NUM_THREADS"
echo

for SCHEDULE_TYPE in static dynamic guided
do
    echo "$SCHEDULE_TYPE"
    echo
    for CHUNK in 1 2 5 8 10 20 50 80 100 200 500 800 1000
    do
        echo -ne "$CHUNK\t"
        export OMP_SCHEDULE="$SCHEDULE_TYPE,$CHUNK"
        ./sched
    done
done
```