###### 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 22204

Соломенникова Николая Александровича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

доцент

А.Ю.Власенко

Новосибирск 2023

**СОДЕРЖАНИЕ**

[ЦЕЛЬ 2](#__RefHeading___1)

[ЗАДАНИЕ 2](#__RefHeading___2)

[ОПИСАНИЕ РАБОТЫ 2](#__RefHeading___3)

[ЗАКЛЮЧЕНИЕ 4](#__RefHeading___4)

[ИСПОЛЬЗУЕМЫЕ КОМАНДЫ 5](#__RefHeading___5)

[ИСХОДНИКИ 5](#__RefHeading___6)

[config.h 5](#__RefHeading___7)

[sequential.cpp 5](#__RefHeading___8)

[parallel.cpp 9](#__RefHeading___9)

# ЦЕЛЬ

Реализовать программу для решения системы линейных алгебраических уравнений с помощью OpenMP

# ЗАДАНИЕ

1. Последовательную программу из предыдущей практической работы, реализующую итерационный алгоритм решения 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

Для выявления, какие параметры для omp parallel for лучше использовать, в ходе одного запуска программы весь алгоритм прокручивался несколько раз (5 раз), и в таблицы результатов заносилось минимальное время работы алгоритма.

Были проведены замеры времени для параллельной программы с использованием разных вариантов schedule. Для каждого цикла for писалась секция omp parallel for schedule(runtime). Размер массива брался N = 1000, а количество потоков 4. Для некоторых вариантов schedule нужен парамерт n, поэтому было проведено несколько эксперементов с разными значениями этого параметра:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Параметр n для schedule | static | dynamic | guided | auto | без schedule |
| - | - | - | - | 18,1 | 19,4 |
| 250 | 17,3 | 18,6 | 18,2 | - | - |
| 100 | 18,0 | 16,2 | 18,4 | - | - |
| 50 | 16,9 | 15,8 | 15,9 | - | - |
| 10 | 17,4 | 15,0 | 15,8 | - | - |

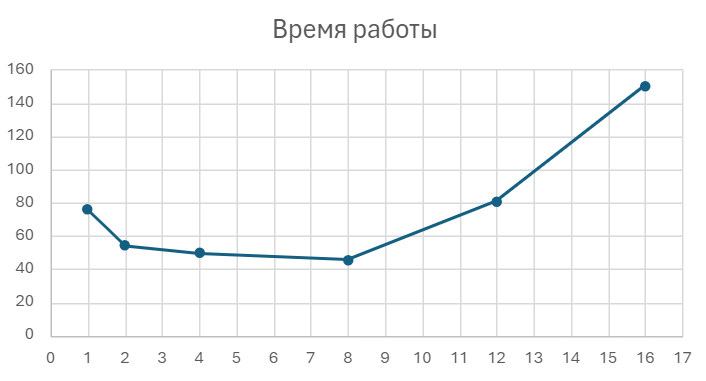
Также для omp parallel for можно указать параметр collapse(2) (если есть 2 вложенных цикла). Было проведено несколько замеров времени с использованием collapse и без. В итоге разницы по времени практически не было.

Далее приведены результаты работы программы на кластере при N=4000.

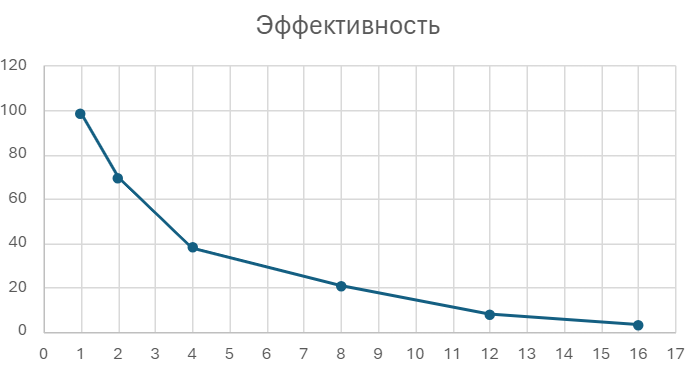
Последовательная программа: T0 = 74,7 сек.

Параллельная программа:

|  |  |
| --- | --- |
| Кол-во потоков | Время Tp |
| 1 | 75,8 |
| 2 | 53,9 |
| 4 | 49,4 |
| 8 | 45,6 |
| 12 | 80,8 |
| 16 | 150,2 |







# ЗАКЛЮЧЕНИЕ

В данной программе в качестве параметров schedule лучше всего показал себя (dynamic, n). Понятно, что указывать слишком болььшой n не имеет смысла, так как иначе теряется смысл динамического распределения, но и слишком маленьким его делать не стоит, так как иначе каждый отдельный поток будет выполнять слишком мало работы, и будет тратиться много времени на управление потоками.

По графикам видно, что при использовании больше 8 потоков, время работы программы значительно увеличивается. Это связано с тем, что на одном узле всего 8 ядер.

# ИСПОЛЬЗУЕМЫЕ КОМАНДЫ

Компиляция:

$g++ prog.cpp -o prog.out -fopenmp

$icpc prog.cpp -o prog.out -qopenmp

Запуск:

$export OMP\_SCHEDULE="dynamic,10"

$export OMP\_NUM\_THREADS=4

$./prog.out

# ИСХОДНИКИ

### config.h

const int N = 1000;                   // Размер векторов

constexpr int MAX\_ITERS = 1e4;        // Максимальное количество итераций

constexpr double EPSILON = 1e-5;      // Точность вычислений

constexpr double TAU = 5e-4;          // Константа, используемая в алгоритме

const double CALCULATION\_ERROR = 0.1; // Погрешность вычислений (нужно для проверки ответа)

### sequential.cpp

#include "config.h"

#include <cmath>

#include <exception>

#include <iostream>

#include <omp.h>

// Класс исключений для отслеживания ошибок

class myException : public std::exception

{

private:

    const char \*msg\_;

public:

    explicit myException(const char \*msg) : msg\_(msg){};

    const char \*what() const noexcept override { return msg\_; }

};

// Инициализация входных данных и вектора для записи результата

inline void initData(double \*&A, double \*&b, double \*&x, double \*&res)

{

    A = (double \*)malloc(sizeof(double) \* (N \* N));

    if (A == nullptr)

        throw myException("Malloc fail");

    for (int i = 0; i < N; ++i)

    {

        for (int j = 0; j < N; ++j)

        {

            if (i == j)

                A[i \* N + j] = 2.0;

            else

                A[i \* N + j] = 1.0;

        }

    }

    b = (double \*)malloc(sizeof(double) \* N);

    if (b == nullptr)

        throw myException("Malloc fail");

    for (int i = 0; i < N; ++i)

        b[i] = N + 1.0;

    x = (double \*)malloc(sizeof(double) \* N);

    if (x == nullptr)

        throw myException("Malloc fail");

    for (int i = 0; i < N; ++i)

        x[i] = sin(i);

    res = (double \*)malloc(sizeof(double) \* N);

    if (res == nullptr)

        throw myException("Malloc fail");

}

// Освобождение ресурсов

inline void deleteData(double \*&A, double \*&b, double \*&x, double \*&res)

{

    free(A);

    free(b);

    free(x);

    free(res);

}

// res = A \* x

inline void mul(const double \*const &A, const double \*const &x, double \*const &res)

{

    for (int i = 0; i < N; ++i)

    {

        res[i] = 0;

        const int iN = i \* N;

        for (int j = 0; j < N; ++j)

            res[i] += A[iN + j] \* x[j];

    }

}

// res = res \* TAU

inline void mulTAU(double \*const &res)

{

    for (int i = 0; i < N; ++i)

        res[i] \*= TAU;

}

// res = res - b

inline void sub(double \*const &res, const double \*const &b)

{

    for (int i = 0; i < N; ++i)

        res[i] -= b[i];

}

// Квадрат модуля вектора

double mod2(const double \*const &x)

{

    double ans = 0;

    for (int i = 0; i < N; ++i)

        ans += x[i] \* x[i];

    return ans;

}

// res = x

inline void copy(const double \*const &x, double \*const &res)

{

    for (int i = 0; i < N; ++i)

        res[i] = x[i];

}

// Модуль для double

inline double abs(const double a)

{

    return a < 0 ? -a : a;

}

// Проверка ответа на корректность

inline void testAns(const double \*const &res)

{

    for (int i = 0; i < N; ++i)

    {

        if (abs(res[i] - 1.0) > CALCULATION\_ERROR)

        {

            fprintf(stderr, "res[%d] = %lf, but expected 1.0\n", i, res[i]);

            throw myException("Wrong result");

        }

    }

}

inline double alg()

{

    double startTime, endTime;

    startTime = omp\_get\_wtime();

    double \*A = nullptr, \*b = nullptr, \*x = nullptr, \*res = nullptr;

    initData(A, b, x, res);

    int iters = 0;

    const double constForCheck = mod2(b) \* EPSILON \* EPSILON;

    while (true)

    {

        mul(A, x, res);

        sub(res, b);

        if (mod2(res) < constForCheck)

        {

            copy(x, res);

            break;

        }

        mulTAU(res);

        sub(x, res);

        if ((++iters) > MAX\_ITERS)

            throw myException("Too many iterations");

    }

    // std::cout << "Iteration: " << iters << "\n";

    testAns(res);

    deleteData(A, b, x, res);

    endTime = omp\_get\_wtime();

    return endTime - startTime;

}

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

{

    try

    {

        const double deltaTime = alg();

        std::cout << "Total time: " << deltaTime << "\n";

    }

    catch (const myException &e)

    {

        std::cerr << e.what() << '\n';

    }

    return 0;

}

### parallel.cpp

#include "config.h"

#include <cmath>

#include <exception>

#include <iostream>

#include <omp.h>

// Для обработки ошибок в параллельной секции

enum whileFlag

{

    CONTINUE\_WHILE,

    TOO\_MANY\_ITERATIONS,

    CORRECT\_RESULT,

    WRONG\_RESULT,

};

// Класс исключений для отслеживания ошибок

class myException : public std::exception

{

private:

    const char \*msg\_;

public:

    explicit myException(const char \*msg) : msg\_(msg){};

    const char \*what() const noexcept override { return msg\_; }

};

// Инициализация буферов (обязательно последовательный код)

inline void initData(double \*&A, double \*&b, double \*&x, double \*&res)

{

    A = (double \*)malloc(sizeof(double) \* (N \* N));

    if (A == nullptr)

        throw myException("Malloc fail");

    b = (double \*)malloc(sizeof(double) \* N);

    if (b == nullptr)

        throw myException("Malloc fail");

    x = (double \*)malloc(sizeof(double) \* N);

    if (x == nullptr)

        throw myException("Malloc fail");

    res = (double \*)malloc(sizeof(double) \* N);

    if (res == nullptr)

        throw myException("Malloc fail");

}

// Заполнение буферов

inline void fillData(double \*&A, double \*&b, double \*&x, double \*&res)

{

#pragma omp for collapse(2) schedule(runtime)

    for (int i = 0; i < N; ++i)

    {

        for (int j = 0; j < N; ++j)

        {

            if (i >= N)

                A[i \* N + j] = 0.0;

            else if (i == j)

                A[i \* N + j] = 2.0;

            else

                A[i \* N + j] = 1.0;

        }

    }

#pragma omp for schedule(runtime)

    for (int i = 0; i < N; ++i)

        b[i] = N + 1.0;

#pragma omp for schedule(runtime)

    for (int i = 0; i < N; ++i)

        x[i] = sin(i);

}

// Освобождение ресурсов

inline void deleteData(double \*&A, double \*&b, double \*&x, double \*&res)

{

    free(A);

    free(b);

    free(x);

    free(res);

}

// res = A \* x

inline void mul(const double \*const &A, const double \*const &x, double \*const &res)

{

#pragma omp for schedule(runtime)

    for (int i = 0; i < N; ++i)

    {

        res[i] = 0;

        for (int j = 0; j < N; ++j)

            res[i] += A[i \* N + j] \* x[j];

    }

}

// res = res \* TAU

inline void mulTAU(double \*const &res)

{

#pragma omp for schedule(runtime)

    for (int i = 0; i < N; ++i)

        res[i] \*= TAU;

}

// res = res - b

inline void sub(double \*const &res, const double \*const &b)

{

#pragma omp for schedule(runtime)

    for (int i = 0; i < N; ++i)

        res[i] -= b[i];

}

// Квадрат модуля вектора (mod2Res - общая переменная, где будет находиться ответ)

void mod2(const double \*const &x, double &mod2Res)

{

#pragma omp single

    mod2Res = 0;

#pragma omp for reduction(+ : mod2Res)

    for (int i = 0; i < N; ++i)

        mod2Res += x[i] \* x[i];

}

// res = x

inline void copy(const double \*const &x, double \*const &res)

{

#pragma omp for schedule(runtime)

    for (int i = 0; i < N; ++i)

        res[i] = x[i];

}

// Модуль для double

inline double abs(const double a)

{

    return a < 0 ? -a : a;

}

// Проверка ответа на корректность

void testAns(const double \*const &res, whileFlag &flag)

{

#pragma omp for schedule(runtime)

    for (int i = 0; i < N; ++i)

    {

        if (abs(res[i] - 1.0) > CALCULATION\_ERROR)

        {

#pragma omp critical

            fprintf(stderr, "res[%d] = %lf, but expected 1.0\n", i, res[i]);

            flag = WRONG\_RESULT;

        }

    }

}

// Алгоритм для нахождения решения системы линейных алгебраических уравнений. Возвращает время работы алгоритма

inline double alg()

{

    double startTime, endTime;

    startTime = omp\_get\_wtime();

    double \*A = nullptr, \*b = nullptr, \*x = nullptr, \*res = nullptr;

    initData(A, b, x, res);

    int iters = 0;                   // Кол-во итерации цикла while

    whileFlag flag = CONTINUE\_WHILE; // Для работы цикла while

    double mod2Res;                  // Переменная для вычисления функции mod2

    double constForCheck = 0;        // Константа для проверки на выход из цикла

#pragma omp parallel firstprivate(iters, flag) shared(mod2Res, constForCheck) default(shared)

    {

        fillData(A, b, x, res);

        mod2(b, mod2Res);

#pragma omp single

        constForCheck = mod2Res \* EPSILON \* EPSILON;

        while (flag == CONTINUE\_WHILE)

        {

            mul(A, x, res);

            sub(res, b);

            mod2(res, mod2Res);

            if (mod2Res < constForCheck)

            {

                flag = CORRECT\_RESULT;

                copy(x, res);

                testAns(res, flag);

            }

            if (flag == CONTINUE\_WHILE)

            {

                mulTAU(res);

                sub(x, res);

                if ((++iters) > MAX\_ITERS)

                    flag = TOO\_MANY\_ITERATIONS;

            }

        }

    }

    deleteData(A, b, x, res);

    if (flag == TOO\_MANY\_ITERATIONS)

        std::cerr << "Too many iterations\n";

    else if (flag == WRONG\_RESULT)

        std::cerr << "Wrong result\n";

    endTime = omp\_get\_wtime();

    return endTime - startTime;

}

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

{

    try

    {

        const double deltaTime = alg();

        std::cout << "Total time: " << deltaTime << "\n";

    }

    catch (const myException &e)

    {

        std::cerr << e.what() << '\n';

    }

    return 0;

}