Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Робототехники и комплексной автоматизации

КАФЕДРА Системы автоматизированного проектирования (РК-6)

ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

по дисциплине: «Параллельные методы и алгоритмы»

Студент	Макаров Тимофей Геннадьевич
Группа	РК6-22М
Тип задания	Лабораторная работа
Тема	Аналитическое исследование эффективности статической балансировки загрузки МВС

Студент _____ **Макаров Т.Г.**
подпись, дата *фамилия, и.о.*

Преподаватель _____ **Карпенко А.П.**
подпись, дата *фамилия, и.о.*

Оценка _____

Москва, 2023 г.

Оглавление

Цель лабораторной работы

Цель выполнения лабораторной работы – изучение двух методов статической балансировки загрузки многопроцессорной вычислительной системы (МВС):

1) балансировка загрузки на основе геометрической схемы декомпозиции области решения;

2) балансировка загрузки на основе равномерной декомпозиции узлов расчетной сетки.

Постановка задачи

Пусть X – n-мерный вектор параметров задачи. Положим, что $X \in R^n$, где R^n – n-мерное арифметическое пространство. Параллелепипедом допустимых значений вектора параметров назовем не пустой параллелепипед $П = \{X \mid x_i^- \le x_i \le x_i^+, i \in [1:n]\}$, где x_i^-, x_i^+ - заданные константы. На вектор X дополнительно наложено некоторое количество функциональных ограничений, формирующих множество $D = \{X \mid g_j(X) \ge 0, j = 1,2,\dots\}$, где $g_j(X)$ - непрерывные ограничивающие функции. На множестве $D_X = \Pi \cap D$ тем или иным способом (аналитически или алгоритмически) определена вектор-функция $F(X)$ со значениями в пространстве R^m. Ставится задача поиска значения некоторого функционала $\Phi(F(X))$.

Положим, что приближенное решение поставленной задачи может быть найдено по следующей схеме.

Шаг 1. Покрываем параллелепипед $П$ некоторой сеткой Ω (равномерной или неравномерной, детерминированной или случайной) с узлами $X_1, X_2, \dots, X_z$.

Шаг 2. В тех узлах сетки Ω, которые принадлежат множеству D_X, вычисляем значения вектор функции $F(X)$.

Шаг 3. На основе вычисленных значений вектор функции $F(X)$ находим приближенное значение функционала $\Phi(F(X))$.

В виде рассмотренной схемы можно представить, например, решение задачи вычисления многомерного определенного интеграла от функции $F(\mathrm{X})$ в области D_X.

Суммарное количество арифметических операций, необходимых для *однократного* определения принадлежности вектора X множеству D_X (т.е. суммарную вычислительную сложность ограничений $x_i^- \le x_i \le x_i^+$ и ограничивающих функций $g_j(X)$), обозначим $C_g \ge 0$. Вообще говоря, величина C_g зависит от вектора X . Мы, однако, пренебрежем этой зависимостью, и будем полагать, что имеет место равенство $C_g = const$. Заметим, что до начала вычислений величина C_g, как правило, неизвестна. Однако в процессе первого же определения принадлежности некоторого узла сетки Ω множеству D_X, эту величину можно легко определить (с учетом предположения о независимости этой величины от вектора *X*). Поэтому будем полагать величину C_g известной.

Неизвестную вычислительную сложность вектор-функции $F(\mathrm{X})$ обозначим $C_f(X)$. Подчеркнем зависимость величины C_f от вектора X . Величина $C_f(X)$ удовлетворяет, во-первых, очевидному ограничению $C_f(X) \ge 0$. Во-вторых, положим, что известно ограничение сверху на эту величину C_f^{max} , имеющее смысл ограничения на максимально допустимое время вычисления значения $F(\mathrm{X})$. Вычислительную сложность $C_f(X_i)$ назовем вычислительной сложностью узла $X_i, i \in [1:Z]$.

Вычислительную сложность генерации сетки Ω положим равной ZC_Ω, а вычислительную сложность конечномерной аппроксимации функционала $\Phi(\mathrm{F(X)})$ - равной ζC_Φ, где ζ – общее количество узлов сетки Ω, принадлежащих множеству D_X. Положим, что при данных n ,m величины C_Ω, C_Φ – известные константы.

В качестве вычислительной системы рассмотрим однородную МВС с распределенной памятью, состоящую из процессоров $P_1, P_2, \dots, P_N$ и host-процессора, имеющих следующие параметры:

- l – длина вещественного числа в байтах;
- t – время выполнения одной арифметической операции с плавающей запятой;
- t_s – латентность коммуникационной сети;
- t_c – время передачи байта данных между двумя соседними процессорами системы без учета времени t_s;
- $d(N)$ – диаметр коммуникационной сети.

В качестве меры эффективности параллельных вычислений используем ускорение

$$S_i(N) = \frac{T(1)}{T_i(N)},$$

где $T(1)$ – время последовательного решения задачи на одном процессоре системы, $T_i(N)$ – время параллельного решения той же задачи на N процессорах, $i = 1,2$ - номер метода балансировки. Будем рассматривать также в качестве меры эффективности параллельных вычислений асимптотическое ускорение

$$S_i^{\infty}(N) = \frac{T(1)}{T_i^{\infty}(N)},$$

где $T_i^{\infty}(N)$ - время параллельного решения задачи на N процессорах без учета коммуникационных расходов (когда $t_s = t_c = 0$).

Статическая балансировка загрузки методом равномерной декомпозиции параллелепипеда П

Данный метод основан на декомпозиции параллелепипеда Π на N равных подобластей и назначении каждой из этих подобластей своему процессору. Для двумерного случая этот метод иллюстрирует схема, изображенная на рисунке 1.

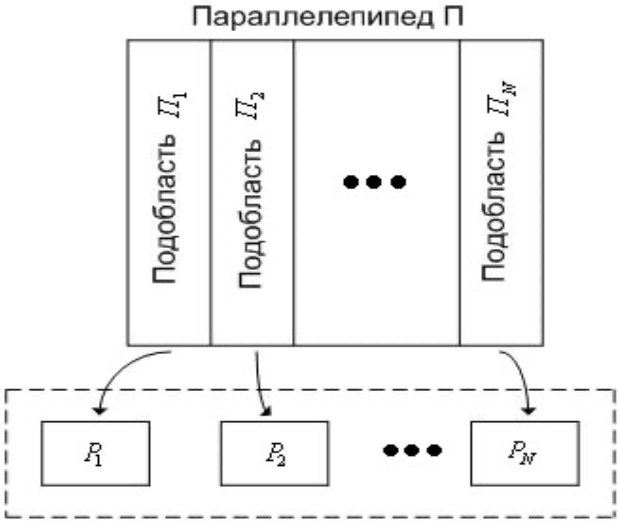

Рисунок 1 – Схема балансировки методом равномерной декомпозиции параллелепипеда *П*.

Введём следующие обозначения:

- Ω_i – множество узлов сетки Ω, покрывающих подобласть $\Pi_i, i \in [1:N]$;
- $z_i \leq z$ – количество узлов во множестве Ω_i, $z = \left\lceil \frac{Z}{N} \right\rceil$, где символ $\lceil\ \rceil$ означает ближайшее большее целое;
- $\zeta_i \leq z_i$ - количество узлов сетки Ω_i, принадлежащий множеству D_X;
- $X_{i,j} \in \{X_1, X_2, \dots, X_Z\}, i \in [1:N],\ j \in [1:\zeta_i]$ – узлы сетки Ω_i, принадлежащие множеству D_X;
- ζ – общее количество узлов сетки Ω, принадлежащих множеству $D_X, \zeta = \sum_{i=1}^{N} \zeta_i$;

В этих обозначениях схему параллельных вычислений при решении рассматриваемой задачи с использованием балансировки загрузки методом равномерной декомпозиции параллелепипеда *П* можно представить в следующем виде.

Шаг 1. Host-процессор выполняет следующие действия:

- строит сетку Ω;

- плоскостями, параллельными одной из координатных плоскостей, разбивает ее узлы на множества $\Omega_i, i \in [1:N]$;
- передает процессору P_i координаты узлов множества Ω_i.

Шаг 2. Процессор P_i выполняет следующие действия:

- принимает от host-процессора координаты узлов множества Ω_i;
- последовательно для всех z_i узлов этого множества определяет их принадлежность множеству D_X;
- вычисляет в каждом из ζ_i узлов множества Ω_i значение вектор-функции $F(X)$;
- передает *host*-процессору ζ_i вычисленных значений и заканчивает вычисления.

Шаг 3. *Host*-процессор выполняет следующие действия:

- принимает от процессоров $P_i, i \in [1:N]$ вычисленные ими значения вектор-функции $F(X)$;
- на основе ζ полученных значений вектор-функции $F(X)$ вычисляет приближенное значение функционала $\Phi(F(X))$.

В соответствии с изложенной схемой балансировки загрузки методом равномерной декомпозиции параллелепипеда $П$, время решения задачи на процессоре P_i можно оценить величиной

$$\tau_i = 2t_s + z_i nldt_c + \zeta mldt_c + tz_i C_g + t\sum_j C_f(X_{i,j}), j \in [1:\zeta_i],$$

где сумма $tz_i C_g + t\sum_j C_f(X_{i,j})$ представляет собой вычислительную загрузку процессора P_i, а сумма $2t_s + z_i nldt_c + \zeta mldt_c$ – его коммуникационную нагрузку. Отсюда следует, что время параллельного решения всей задачи равно

$$T_1(N) = \max_{i\in[1:N]} \tau_i + tZC_\Omega + t\zeta C_\Phi.$$

Аналогично, время решения задачи на одном процессоре равно

$$T(1) = tzC_g + t\sum_{j=1}^{\zeta} C_f(X_j) + tZC_\Omega + t\zeta C_\Phi. \quad (1)$$

Статическая балансировка загрузки методом равномерной декомпозиции расчетных узлов

Положим, что из числа Z узлов расчетной сетки Ω множеству D_X принадлежит ξ узлов $\widetilde{X_1}, \widetilde{X_2}, \dots, \widetilde{X_\zeta}$. Обозначим $z = \left\lceil \frac{\zeta}{N} \right\rceil$. Тогда идею рассматриваемого метода балансировки загрузки можно представить в следующем виде:

- среди всех узлов $\mathrm{X}_1, \mathrm{X}_2, \dots, X_N$ сетки Ω выделяем ζ узлов $\widetilde{X_1}, \widetilde{X_2}, \dots, \widetilde{X_\zeta}$;
- разбиваем узлы $\widetilde{X_1}, \widetilde{X_2}, \dots, \widetilde{X_\zeta}$ на N множеств $\widetilde{\Omega}_i$ $i \in [1:N]$, где множество $\widetilde{\Omega}_1$ содержит узлы $\widetilde{X_1}, \widetilde{X_2}, \dots, \widetilde{X_z}$, множество $\widetilde{\Omega}_2$ – узлы $\tilde{X}_{z+1}, \tilde{X}_{z+2}, \dots, \tilde{X}_{2z}$ и т.д.
- назначаем для обработки процессору P_i множеств узлов $\widetilde{\Omega}_i$ $i \in [1:N]$.

Наглядно схема работы метода изображена на рисунке 2.

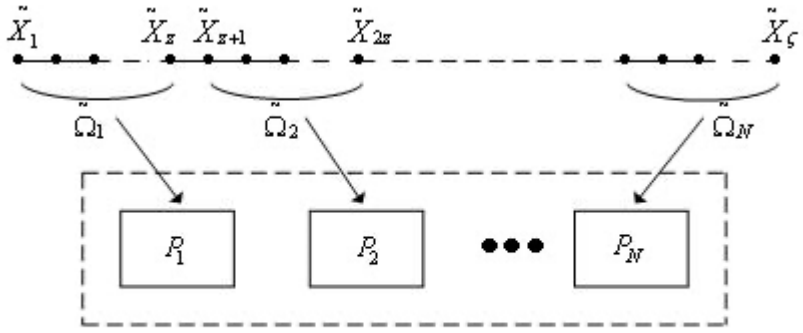

Рисунок 2 – Схема балансировки методом равномерной декомпозиции расчётных узлов.

Схему параллельных вычислений при балансировке загрузки методом равномерной декомпозиции расчетных узлов можно представить в следующем виде.

Шаг 1. Host-процессор выполняет следующие действия:

- строит сетку Ω;
- среди всех узлов $X_1, X_2, \dots, X_z$ сетки Ω выделяет ζ узлов $\widetilde{X_1}, \widetilde{X_2}, \dots, \widetilde{X_\zeta}$;
- разбивает узлы $\widetilde{X_1}, \widetilde{X_2}, \dots, \widetilde{X_\zeta}$ на N множеств узлов $\widetilde{\Omega}_i$ $i \in [1:N]$;
- передает процессору P_i координаты узлов множества $\widetilde{\Omega}_i$.

Шаг 2. Процессор P_i выполняет следующие действия:

- принимает от host-процессора координаты z узлов множества $\widetilde{\Omega}_i$;
- вычисляет в каждом из этих узлов значение вектор-функции $F(X)$;
- передает *host*-процессору z вычисленных векторов и заканчивает вычисления.

Шаг 3. *Host*-процессор выполняет следующие действия:

- принимает от процессоров $P_i, i \in [1:N]$ вычисленные ими значения вектор-функции $F(X)$;
- на основе ζ полученных значений вектор функций $F(X)$ вычисляет приближённое значение функционала Ф(F(X)).

Обозначим $\widetilde{\mathrm{X}}_{i,j}, i \in [1:N], j \in [1:z]$ – узлы сетки $\widetilde{X_1}, \widetilde{X_2}, \dots, \widetilde{X_\zeta}$, принадлежащие множеству $\widetilde{\Omega}_i$. Тогда при балансировке загрузки методом равномерной декомпозиции расчётных узлов время решения задачи на процессоре P_i можно оценить величиной

$$\tau_i = 2t_s + znldt_c + zmldt_c + t\sum_{j=1}^{z} C_f(\widetilde{\mathrm{X}}_{i,j}),$$

где слагаемое $t\sum_{j=1}^{z} C_f(\widetilde{\mathrm{X}}_{i,j})$ представляет собой вычислительную загрузку процессора P_i, а слагаемые $2t_s + znldt_c + zmldt_c$ – его коммуникационную загрузку. Таким образом, время параллельного решения всей задачи оценивается величиной

$$T_2(N) = \max_{i\in[1:N]} \tau_i + tZC_g + t\zeta C_\Phi.$$

Время решения задачи на одном процессоре определяется формулой (1).

Вычислительный эксперимент

Рассмотрим двумерную задачу (n=2). Параллелепипед $П$ в этом случае представляет собой прямоугольник $П = \{X | x_i^- \le x_i \le x_i^+, i \in [1,2]\}$. Положим, что $x_1^- = x_2^- = 0$, $x_1^+ = x_2^+ = 1$, так что область $П$ является единичным квадратом (рисунок 3).

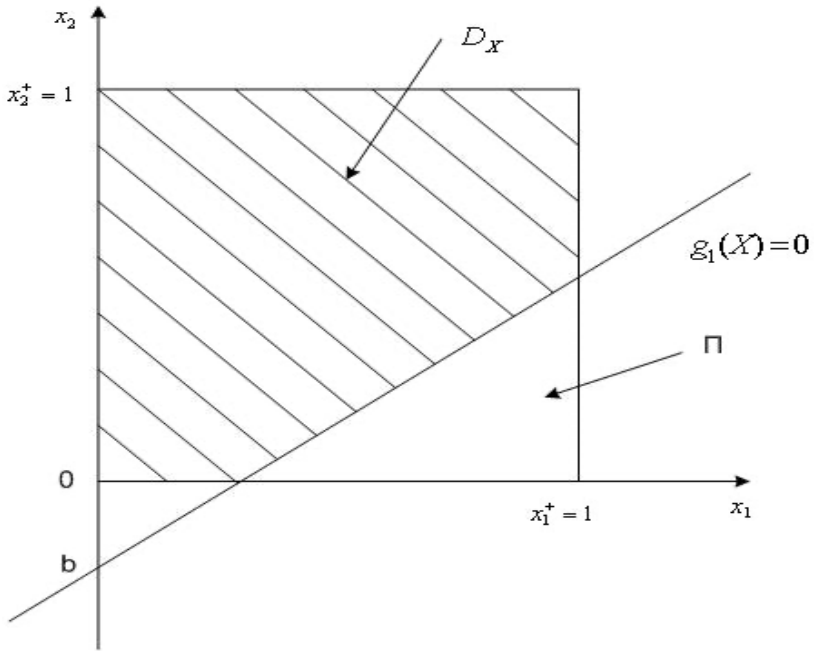

Рисунок 3 – Расчетная область задачи.

Множество D формируется с помощью одной ограничивающей функции $g_1(X) \ge 0$, т.е. $D = \{X | g_1(X) \ge 0\}$. Примем, что эта функция линейна и проходит через заданную преподавателем точку плоскости $0x_1x_2$ с координатами (0, b). Таким образом, уравнение этой функции имеет вид $x_2 = ax_1 + b, a > 0$ (при этом, очевидно, $g_1(X) = g_1(x_1, x_2) = x_2 - ax_1 - b$).

В качестве сетки Ω используем равномерную детерминированную сетку с количеством узлов по осям $0x_1$ и $0x_2$ равным 256, т.е. сетку с количеством узлов $Z = 256 * 256 = 65536$.

Будем исходить из следующих значений параметров задачи и МВС:

- $m = 100$;
- $l = 8$;
- $t = 10 * 10^{-9}[c]$;
- $t_s = 50 * 10^{-6}[c]$;

- $t_c = \frac{1}{80} * 10^{-6}[c]$;
- $d(N) = [2sqrt(N) - 1]$.

Пренебрежем вычислительными затратами на построение сетки Ω, на вычисление значений ограничивающей функции $g_1(X)$, а также на построение приближенного значения функционала $\Phi(F(X))$, т.е. положим $C_\Omega = 0$, $C_g = 0$, $C_\Phi = 0$. Примем также, что вычислительная сложность C_f вектор-функции $F(X)$ одинакова во всей области D_X.

Метод равномерной декомпозиции параллелепипеда

В сделанных предположениях при использовании балансировки загрузки методом равномерной декомпозиции параллелепипеда $П$ время решения задачи на процессоре P_i можно оценить величиной

$$\tau_i = 2t_s + z_i nldt_c + \zeta_i mldt_c + t\zeta_i C_f, \quad (2)$$

время параллельного решения всей задачи – величиной

$$T_1(N) = \max_{i \in [1:N]} \tau_i, \quad (3)$$

а время решения задачи на одном процессоре – величиной

$$T(N) = t\zeta C_f. \quad (4)$$

Асимптотическое ускорение данного метода равно

$$S_1^\infty(N) = \frac{Z\left(1 - \frac{a}{2}\right)}{\frac{Z}{N}\left(1 - \frac{a}{2N}\right)} = \frac{N\left(1 - \frac{a}{2}\right)}{1 - \frac{a}{2N}}. \quad (5)$$

Метод равномерной декомпозиции расчётных узлов

Для данного метода время решения задачи на процессоре P_i можно оценить величиной

$$\tau_i = 2t_s + znldt_c + zmldt_c + tzC_f, \quad (6)$$

где слагаемое tzC_f представляет собой вычислительную загрузку процессора P_i, а слагаемые $2t_s + znldt_c + zmldt_c$ – его коммуникационную загрузку.

Время параллельного решения всей задачи оценивается величиной

$$T_2(N) = \tau, \tag{7}$$

а время решения задачи на одном процессоре определяется формулой (3).

Асимптотическое ускорение метода равно

$$S_2^{\infty}(N) = \frac{t\zeta C_f}{tzC_f} = \frac{\zeta}{z}. \tag{8}$$

Программная реализация

Для написания программной реализации был использован язык программирования C11 и интегрированная среда разработки CLion. Код программы приведён в листинге 1.

Листинг 1. Программная реализация.

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

const int m = 100;
const int l = 8;
const double t = 10e-9;
const double t_s = 50e-6;
const double t_c = 1./80. * 1e-6;
const double c_omega = 0;
const double c_g = 0;
const double c_fi = 0;
const double c_f_1 = 5e6;
const double c_f_2 = 5e4;
const int stepsX1 = 256;
const int stepsX2 = 256;
const int Z = stepsX1 * stepsX2;

const double a = 0.5;
const double b = 0.0;

double d(int n) {
    return ceil(2 * sqrt(n) - 1);
}

double g(double x1, double x2) {
    return x2 - a*x1 - b;
}

double s1(int n, double c_f) {
    double stepSizeX1 = 1./(stepsX1-1);
    double stepSizeX2 = 1./(stepsX2-1);
    int z_i = Z / n;
    int blockWidth = z_i / stepsX1;
    int zeta = 0;
    int maxZeta = 0;
    for (int i = 0; i < n; i++) {
        int zeta_i = 0;
```

```
        for (int j = 0; j < stepsX2; j++) {
            for (int k = i * blockWidth; k < (i + 1) * blockWidth; k++) {
                if (g(k * stepSizeX1, j * stepSizeX2) >= 0) {
                    zeta_i++;
                }
            }
        }
        zeta += zeta_i;
        if (i == 0) {
            maxZeta = zeta_i;
        }
    }
    double T_1 =
            2 * t_s +
            z_i * n * l * d(n) * t_c +
            maxZeta * m * l * d(n) * t_c +
            t * maxZeta * c_f;
    double T_single = t * zeta * c_f;
    return T_single / T_1;
}

double s1_asimp(int n) {
    return n * (1. - 1. / 2) / (1 - 1. / (2 * n));
}

double s2(int n, double c_f) {
    double stepSizeX1 = 1./(stepsX1-1);
    double stepSizeX2 = 1./(stepsX2-1);
    int zeta = 0;
    for (int i = 0; i < stepsX1; i++) { // groups
        for (int j = 0; j < stepsX2; j++) { // row
            if (g(i * stepSizeX1, j * stepSizeX2) >= 0) {
                zeta++;
            }
        }
    }
    int z = zeta / n;
    if (zeta % n != 0) {
        z++;
    }
    double T_2 =
            2 * t_s +
            z * n * l * d(n) * t_c +
            z * m * l * d(n) * t_c +
            t * z * c_f;

    double T_single = t * zeta * c_f;
    return T_single / T_2;
}

double s2_asimp(int n) {
    double stepSizeX1 = 1./(stepsX1-1);
    double stepSizeX2 = 1./(stepsX2-1);
    int zeta = 0;
    for (int i = 0; i < stepsX1; i++) { // groups
        for (int j = 0; j < stepsX2; j++) { // row
            if (g(i * stepSizeX1, j * stepSizeX2) >= 0) {
                zeta++;
            }
        }
```

```
    }
    int z = zeta / n;
    if (zeta % n != 0) {
        z++;
    }
    return (double)zeta / z;
}

int main() {
    int n[6] = {2,4,8,16,32, 64};
    printf("S1 a=%f c_f=%f\n", a, c_f_1);
    for (int i = 0; i < 6; i++) {
        printf("%d %.2f\n", n[i], s1(n[i], c_f_1));
    }
    printf("\n");
    printf("S1 asimp a=1 c_f=%f\n", c_f_1);
    for (int i = 0; i < 6; i++) {
        printf("%d %.2f\n", n[i], s1_asimp(n[i]));
    }
    printf("\n");
    printf("S1 a=%f c_f=%f\n", a, c_f_2);
    for (int i = 0; i < 6; i++) {
        printf("%d %.2f\n", n[i], s1(n[i], c_f_2));
    }
    printf("\n");
    printf("S1 asimp a=1 c_f=%f\n", c_f_2);
    for (int i = 0; i < 6; i++) {
        printf("%d %.2f\n", n[i], s1_asimp(n[i]));
    }
    printf("\n");
    printf("S2 a=%f c_f=%f\n", a, c_f_1);
    for (int i = 0; i < 6; i++) {
        printf("%d %.2f\n", n[i], s2(n[i], c_f_1));
    }
    printf("\n");
    printf("S2 asimp c_f=%f\n", c_f_1);
    for (int i = 0; i < 6; i++) {
        printf("%d %.2f\n", n[i], s2_asimp(n[i]));
    }

    printf("S2 a=%f c_f=%f\n", a, c_f_2);
    for (int i = 0; i < 6; i++) {
        printf("%d %.2f\n", n[i], s2(n[i], c_f_2));
    }
    printf("\n");
    printf("S2 asimp c_f=%f\n", c_f_1);
    for (int i = 0; i < 6; i++) {
        printf("%d %.2f\n", n[i], s2_asimp(n[i]));
    }
    return 0;
}
```

Результат работы программы

С помощью разработанной программы для заданных величин a, b, C_f были вычислены ускорения $S_1(N)$ для метода равномерной декомпозиции расчетной области и $S_2(N)$ для метода равномерной декомпозиции расчетных

узлов при $N = 2, 4, 8, 16, 32, 64$. В таблице 1 представлены результаты вычислений оценки эффективности используемых методов балансировки загрузки при разном количестве процессоров N и разных значениях вычислительной сложности C_f, а также асимптотические ускорения методов $S_1^{\infty}(N)$ и $S_2^{\infty}(N)$ при $a = 1, b = 0$.

Таблица 1. Результаты работы программы

	$S_1(N)$		$S_2(N)$		$S_1^{\infty}(N)$	$S_2^{\infty}(N)$
C_f / N	$0.5 * 10^7$	$5 * 10^4$	$0.5 * 10^7$	$5 * 10^4$	$\to \infty$	$\to \infty$
2	1.71	1.65	2.00	1.92	1.33	2.00
4	3.20	3.01	2.00	3.77	2.29	4.00
8	6.19	5.59	7.99	7.22	4.27	8.00
16	12.17	10.48	15.97	13.76	8.26	16.00
32	24.12	18.74	31.91	24.80	16.25	32.00
64	47.95	32.28	63.69	42.89	32.25	64.00

Из результатов, записанных в таблице 1, можно сделать вывод, что метод равномерной декомпозиции узлов более эффективен.

На рисунках 4 и 5 графически представлены зависимости $S(N) = N$, $S_2(N), S_2(N)$ для $C_f = 0.5 \cdot 10^7$ и $C_f = 5 \cdot 10^4$.

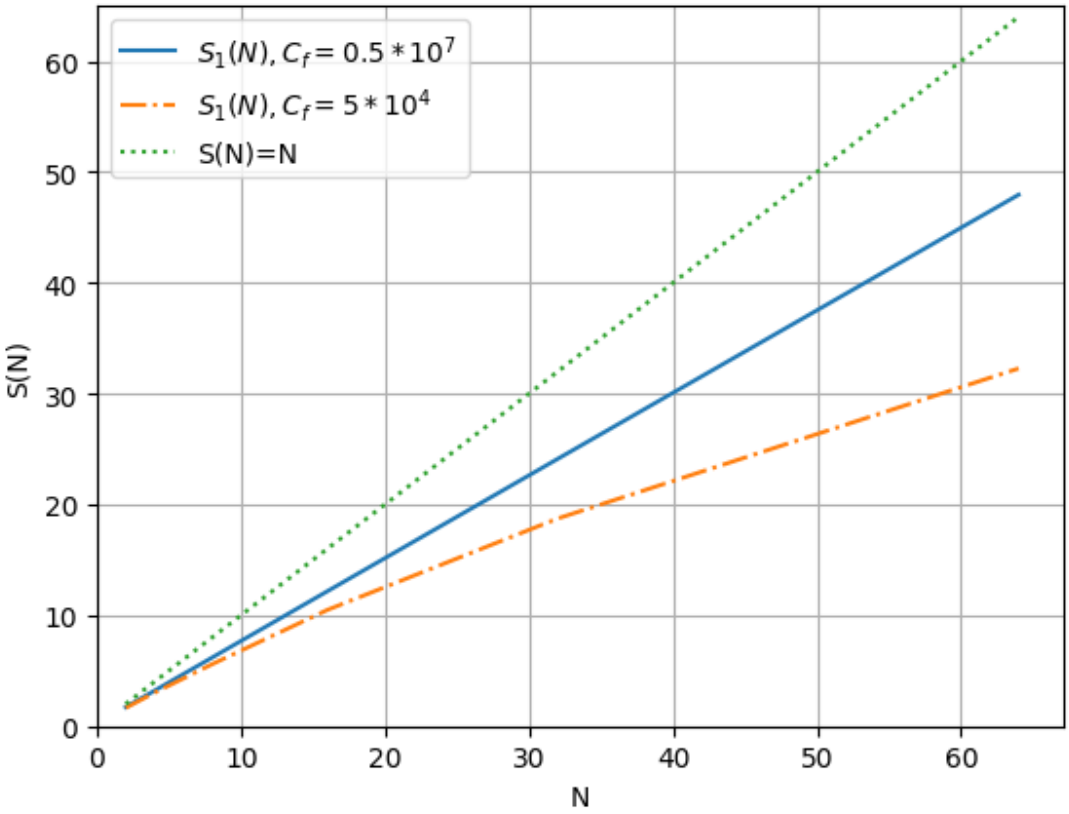

Рисунок 4 – Графики оценки первого метода балансировки нагрузки.

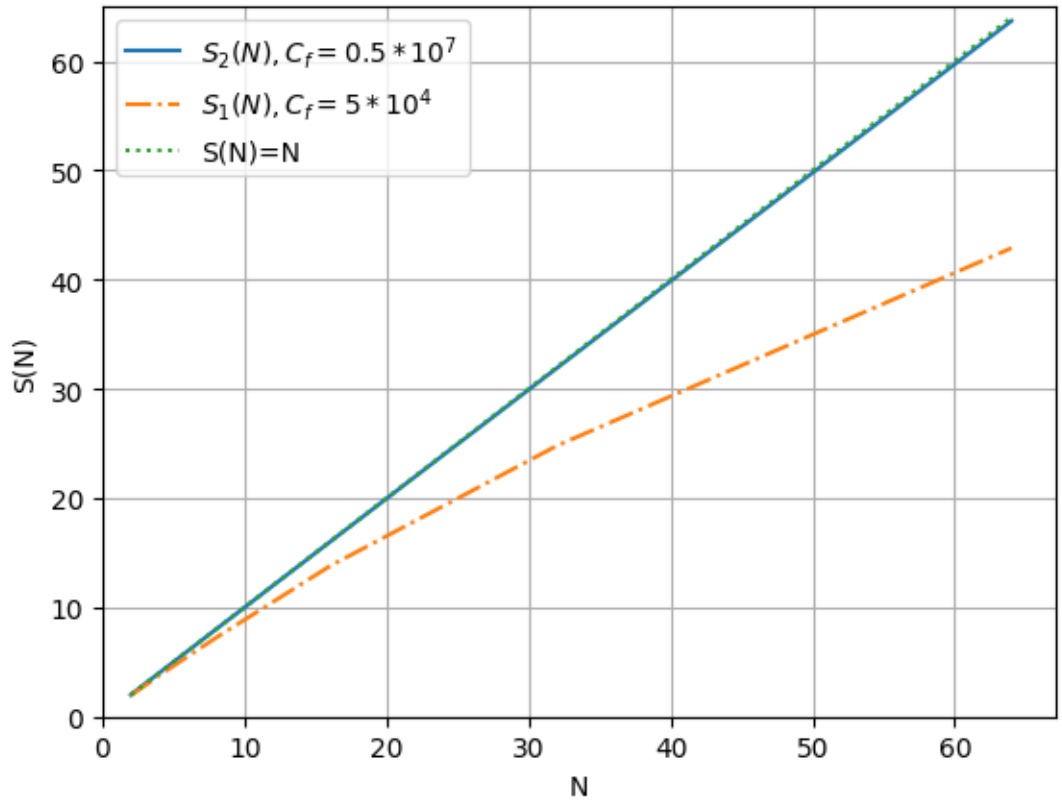

Рисунок 5 – Графики оценки второго метода балансировки нагрузки.

Заключение

В процессе выполнения лабораторной работы были изучены два метода статической балансировки загрузки многопроцессорной вычислительной системы:

- балансировка загрузки на основе геометрической схемы декомпозиции области решения;
- балансировка загрузки на основе равномерной декомпозиции узлов расчетной сетки.

Был проведён сравнительный анализ эффективности методов при варьировании количества используемых процессоров N и вычислительной сложности C_f вектор-функции $F(X)$. Полученные результаты свидетельствуют о том, что метод равномерной декомпозиции расчётных узлов имеет большую эффективность. Увеличение вычислительной сложности C_f приводит к увеличению эффективности обоих методов.

Список использованных источников

1. Карпенко А.П., Федорук Е.В. Параллельные вычисления. Учебнометодическое пособие к лабораторным работам по курсу «Параллельные вычисления». М.: НУК ИУ МГТУ им. Н.Э. Баумана, 2010. 72 с.
2. Карпенко, А.П. Параллельные вычисления: учебное пособие [Электронный ресурс] / А.П.Карпенко // (http://bigor.bmstu.ru/?cnt/?doc=Parallel/base.cou).