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 РОБОТОТЕХНИКА И КОМПЛЕКСНАЯ АВТОМАТИЗАЦИЯ (РК)

КАФЕДРА РК6 «СИСТЕМЫ АВТОМАТИЗИРОВАННОГО ПРОЕКТИРОВАНИЯ»

Отчет по лабораторной работе

Аналитическое исследование эффективности статической балансировки загрузки МВС

Студент ________________ **Абидоков Р. Ш.**
РК6-21М

Преподаватель ________________ **Карпенко А. П.**

2022 г.

Постановка задачи исследования эффективности статистической балансировки загрузки МВС

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 $\Pi = \{X \mid x_i^- \leq x_i \leq x_i^+, i \in [1,n]\}$, где x_i^-, x_i^+ – заданные константы. На вектор X дополнительно наложено некоторое количество функциональных ограничений, формирующих множество $D = \{X \mid g_i(X) \geq 0, j = 1,2, \dots\}$, где $g_i(X)$ – непрерывные ограничивающие функции.

На множестве $D_x = \Pi \cap D$ тем или иным способом (аналитически или алгоритмически) определена вектор-функция $\mathrm{F}(X)$ со значениями в пространстве R^m. Ставится задача поиска значения некоторого функционала $\Phi(\mathrm{F}(X))$.

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 $\mathrm{F}(X)$.

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

Суммарное количество арифметических операций, необходимых для *однократного* определения принадлежности вектора X множеству D_x (т.е. суммарную вычислительную сложность ограничений $x_i^- \leq x_i \leq x_i^+$ и ограничивающих функций $g_i(X)$, обозначим $C_g \geq 0$. *Далее в эксперименте будем полагать* $C_g = 0$.

Неизвестную вычислительную сложность вектор-функции $\mathrm{F}(X)$ обозначим $\mathrm{C}_f(X)$. Подчеркнем зависимость величины C_f от вектора X. Величина $\mathrm{C}_f(X)$ удовлетворяет, во-первых, очевидному ограничению $\mathrm{C}_f(X) \geq 0$. Во-вторых, положим, что известно ограничение сверху на эту величину C_f^{max}, имеющее смысл ограничения на максимально допустимое время вычисления значения $\mathrm{F}(X)$. Вычислительную сложность $\mathrm{C}_f(X_i)$ 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_Ω, где ζ (дзета) - общее количество узлов сетки Ω, принадлежащих множествуD_x.

Далее в эксперименте также будем полагать $\mathrm{C}_f = C_\Omega = 0$.

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:

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

В качестве меры эффективности параллельных вычислений используем ускорение

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

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.

Статическая балансировка загрузки методом равномерной декомпозиции параллелепипеда П

Простейшим методом балансировки загрузки является статический метод на основе декомпозиции параллелепипеда П на N равных подобластей и назначении каждой из этих подобластей своему процессору. Назовем данный метод балансировки методом равномерной декомпозиции параллелепипеда *П*. Для двумерного случая $n = 2$ этот метод балансировки иллюстрирует Рис. 1.

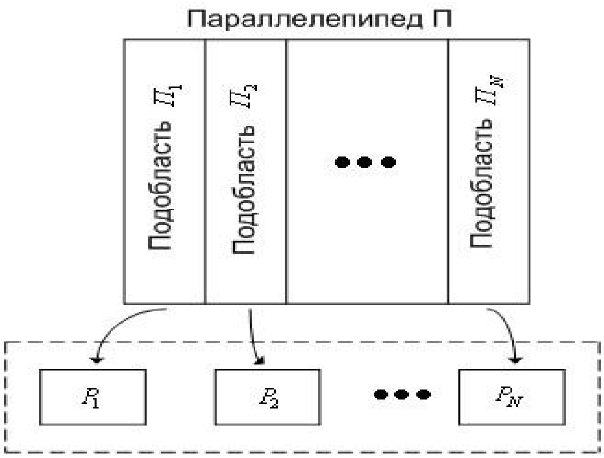

Рис. 1. К балансировке загрузки методом равномерной декомпозиции параллелепипеда *П*

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

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

где $z_i,\ z_i \leq z -$ количество узлов в сетке Ω_i (т.е. попавших в подобласть Π_i); $\zeta_i, \zeta_i \leq z_i -$ количество узлов в сетке Ω_i, попавших в множество D_x.

Время решения всей задачи можно оценить величиной

$$T_i(N) = max_{i\in[1,N]}\tau_i, \qquad (3)$$

а время решение задачи на одном процессоре величиной

$$T(1) = t\zeta C_f. \qquad (4)$$

Таким образом, схема алгоритма для аналитической оценки эффективности рассматриваемого метода балансировки загрузки имеет следующий вид:

- в квадрате Π строим равномерную по каждому из измерений сетку Ω;
- прямыми, параллельными одной из осей координат $0x_1, 0x_2$ разбиваем квадрат Π на N одинаковых подобластей $\Pi_i, i \in [1, N]$;
- для всех подобластей $\Pi_i, i \in [1, n]$ находим количества узлов z_i, ζ_i;
- по формуле (2) вычисляем значение величины τ_i;
- по формуле (3) находим величину $T_i(N)$;
- по формуле (4) определяем значение величины $T(1)$; по формуле (1) находим оценку ускорения.

Примем также, что вычислительная сложность C_f вектор-функции $\mathrm{F}(X)$ одинакова во всей области D_x.

Статическая балансировка загрузки методом равномерной декомпозиции параллелепипеда П

Положим, что из числа Z узлов расчетной сетки Ω множеству D_x принадлежит ζ узлов $\tilde{X}_1, \tilde{X}_2 \dots \tilde{X}_\zeta$. Обозначим $z = \left[\frac{\zeta}{N}\right]$. Тогда идею рассматриваемого метода балансировки загрузки можно представить в следующем виде (рис. 2):

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

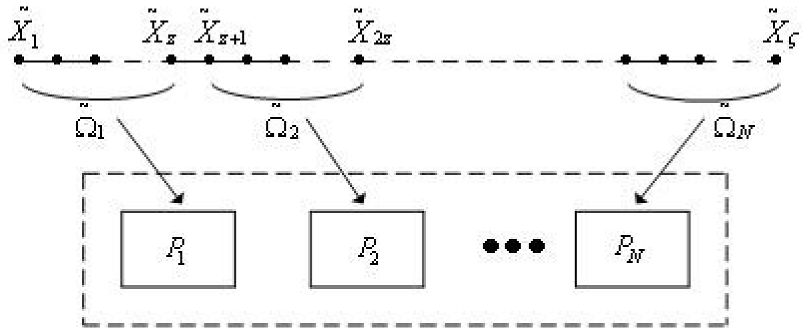

Рис. 2. К балансировке загрузки методом 2.

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

$$\tau_i = \tau = 2t_s + znldt_c + zmldt_c + tzC_f, \tag{5}$$

время параллельного решения всей задачи – величиной

$$T_2(N) = \tau, \tag{6}$$

а время решения задачи на одном процессоре – величиной (4).

Таким образом, схема алгоритма для аналитической оценки эффективности балансировки загрузки методом равномерной декомпозиции расчетных узлов имеет следующий вид:

- в квадрате П строим равномерную по каждому из измерений сетку Ω;
- находим количества узлов ζ, z;
- по формуле (5) вычисляем значение величины τ;
- по формуле (6) находим величину $T_2(N)$;
- по формуле (4) определяем значение величины $T(1)$;
- по формуле (1) находим оценку ускорения.

Экспериментальная часть

Исходные данные:

$N = 2,4,8,16,32,64;$
$C_f = 10^2, 10^3, 10^4$ с;
$a = 1.0;$
$b = -0.1;$
$m = 100;$
$l = 8;$

$t = 10 * 10^{-9}$ с;
$t_s = 50 * 10^{-6}$ с;
$t_c = \frac{1}{80} * 10^{-6}$ с;
$d(N) = 2\sqrt{N} - 1;$
$Z = 256 * 256 = 65536.$

Полученные значения ускорения для методов равномерной декомпозиции параллелепипеда П и равномерной декомпозиции расчетных узлов приведены соответственно в Табл. 1 и Табл. 2.

Табл. 1
Равномерная декомпозиция параллелепипеда П

N	S	Cf
2	0.07	100.00
4	0.08	100.00
8	0.09	100.00
16	0.12	100.00
32	0.14	100.00
64	0.15	100.00
2	0.49	1,000.00
4	0.60	1,000.00
8	0.79	1,000.00
16	1.04	1,000.00
32	1.30	1,000.00
64	1.49	1,000.00
2	1.19	10,000.00
4	1.90	10,000.00
8	3.17	10,000.00
16	5.25	10,000.00
32	8.06	10,000.00
64	11.00	10,000.00

Табл. 2
Равномерная декомпозиция расчетных узлов

N	S	Cf
2	0.10	100.00
4	0.12	100.00
8	0.16	100.00
16	0.19	100.00
32	0.23	100.00
64	0.26	100.00
2	0.70	1,000.00
4	0.97	1,000.00
8	1.33	1,000.00
16	1.75	1,000.00
32	2.19	1,000.00
64	2.50	1,000.00
2	1.69	10,000.00
4	3.05	10,000.00
8	5.32	10,000.00
16	8.83	10,000.00
32	13.56	10,000.00
64	18.52	10,000.00

Графики зависимости ускорения от количества процессоров при заданных значениях C_f для методов равномерной декомпозиции параллелепипеда П и равномерной декомпозиции расчетных узлов приведены соответственно на Рис. 1, 2.

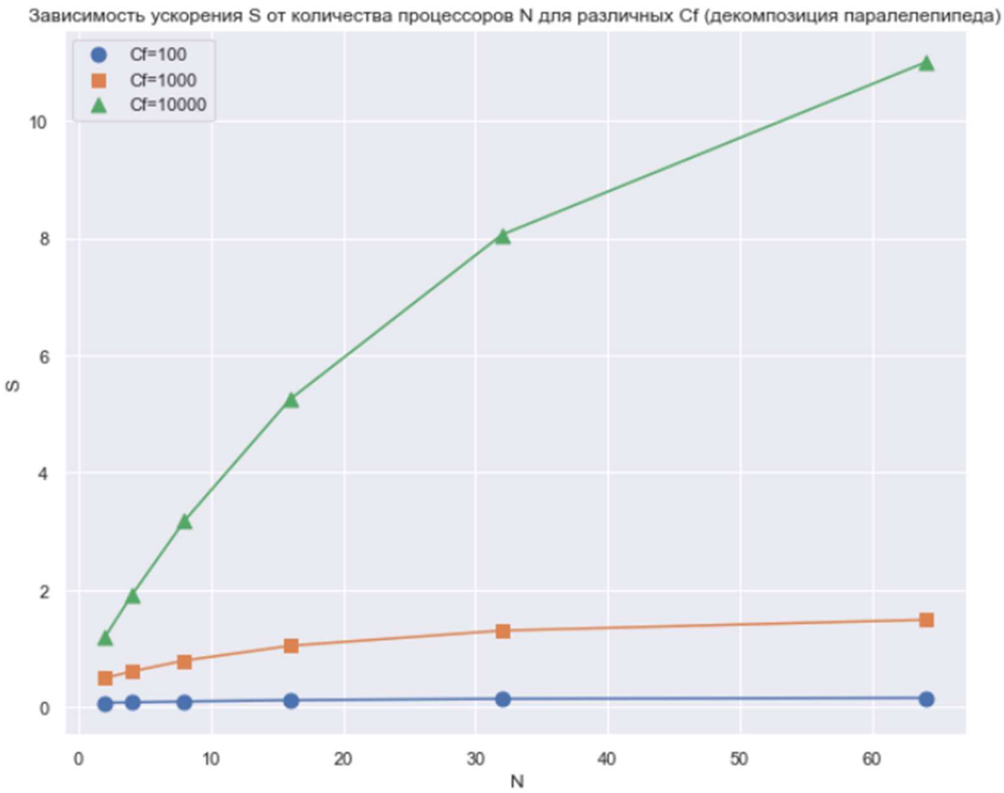

Рис. 1

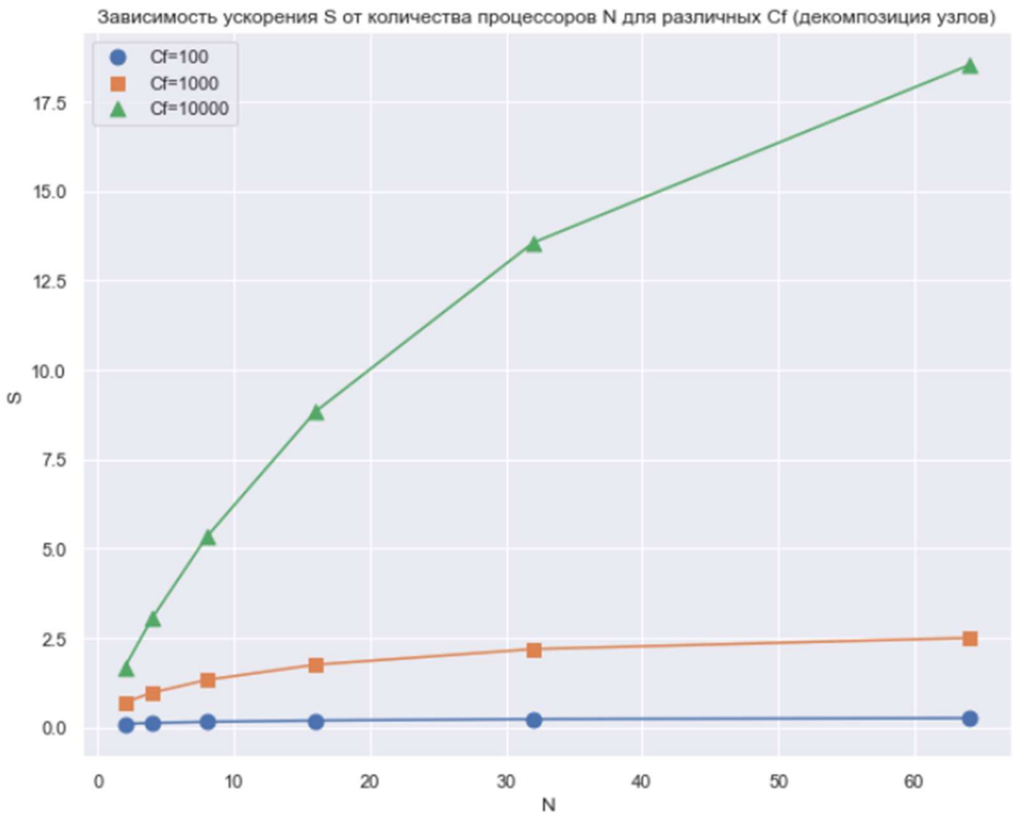

Рис. 2

Ответы на контрольные вопросы

1. *Почему ускорения* $S_1(N)$, $S_2(N)$ *при всех N меньше* N *?*

Появляются издержки на коммуникацию между процессорами; для декомпозиции параллелепипеда П также характерна неравномерная нагрузка на процессоры

2. Почему ускорение $S_1(N)$, *меньше ускорения* $S_2(N)$*?*

Для алгоритма равномерной декомпозиции расчетных узлов характерно более равномерное разделение вычислений по процессорам – как следствие, уменьшается максимальное время выполнения на конкретном процессоре.

3. Почему наблюдается отклонение зависимостей $S_1(N)$, $S_2(N)$ *от линейной зависимости?*

С увеличением количества процессоров N растет доля накладных расходов на коммуникацию – в результате значение ускорения стремится к некоторому асимптотическому значению.

Исходный код программы

Программа выполнена на языке Python 3.8 с использованием библиотек numpy, pandas, seaborn

```
import numpy as np
import pandas as pd
import itertools
import seaborn as sns

a      = 1.0                         # g(X) = x2 - ax1 - b
b      = -0.1                        #
N_list = [2**i for i in range(1,7)]  # Число процессоров
Cf_list = [1.0e2, 1.0e3, 1.0e4]      # Вычислительная сложность вектор-ф-ии F(X)

m  = 100                             #
l  = 8                               # Длина вещ. числа в байтах
t  = 10e-9                           # Время выполнения арифм. оп. с плавающей
точкой
ts = 50e-6                           # Латентность комм. сети
tc = (1/80)*1e-6                     # Время передачи данных между двумя
соседними проц.

def d(N):                            # Диаметр коммуникационной сети
    return 2*np.sqrt(N)-1

Z_side = 256                         # Размерность сетки
Z = Z_side**2                        # Количество узлов в сетке

# Метод равномерной декомпозиции параллелепипеда П
nodes_all = np.array([x for x in itertools.product(np.linspace(0., 1., Z_side),
np.linspace(0., 1., Z_side))])
g = nodes_all[:, 1] - a*nodes_all[:, 0] - b
```

```
# Количество всех узлов в подобласти Пi
def z(i, N):
    z, _ = np.histogram(nodes_all[:, 0], bins=np.linspace(0., 1., N+1))
    return z[i]

# Количество узлов с g > 0 в подобласти Пi
def dzeta(i, N):
    dzeta, _ = np.histogram(nodes_all[g > 0][:, 0], bins=np.linspace(0., 1.,
N+1))
    return dzeta[i]

# Оценка времени решения на процессоре Pi для метода равномерной декомпозиции
параллелепипеда П
def tau(i, N, Cf=Cf):
    return 2*ts + z(i, N)*N*l*d(N)*tc + dzeta(i, N)*m*l*d(N)*tc + t*dzeta(i,
N)*Cf

# Оценка времени параллельного решения
def T_parallel(N, Cf=Cf):
    return max([tau(i, N, Cf) for i in range(N)])

# Оценка времени однопоточного решения
def T_single(Cf=Cf):
    return t*Cf*nodes_all[g > 0].shape[0]

# Оценка ускорения
def S(N, Cf=Cf):
    return T_single(Cf)/T_parallel(N, Cf)

data = {
    'N': [],
    'S': [],
    'Cf': []
}

for Cf in Cf_list:
    data['N'] += N_list
    data['Cf'] += [Cf]*len(N_list)
    data['S'] += [S(N, Cf) for N in N_list]

data = pd.DataFrame(data)

sns.set(rc={'figure.figsize':(11.7,8.27)})
sns.scatterplot(
    data=data,
    x='N',
    y='S',
    hue="Cf",
    palette='deep'
).set_title('Зависимость ускорения S от количества процессоров N для различных
Cf (декомпозиция параллелепипеда)')

# Метод равномерной декомпозиции расчетных узлов

nodes_all = np.array([x for x in itertools.product(np.linspace(0., 1., Z_side),
np.linspace(0., 1., Z_side))])
g = nodes_all[:, 1] - a*nodes_all[:, 0] - b

# Количество узлов с g > 0 на один процессор
def z(N):
    return nodes_all[g > 0].shape[0] // N

# Оценка времени решения на процессоре Pi для метода равномерной декомпозиции
узлов (одинакова для всех процессоров)
```

```
def tau(N, Cf=Cf):
    return 2*ts + z(N)*N*l*d(N)*tc + z(N)*m*l*d(N)*tc + t*z(N)*Cf

# Оценка времени параллельного решения
def T_parallel(N, Cf=Cf):
    return tau(N, Cf)

# Оценка времени однопоточного решения
def T_single(Cf=Cf):
    return t*Cf*nodes_all[g > 0].shape[0]

# Оценка ускорения
def S(N, Cf=Cf):
    return T_single(Cf)/T_parallel(N, Cf)

data = {
    'N': [],
    'S': [],
    'Cf': []
}

for Cf in Cf_list:
    data['N'] += N_list
    data['Cf'] += [Cf]*len(N_list)
    data['S'] += [S(N, Cf) for N in N_list]

data = pd.DataFrame(data)

sns.set(rc={'figure.figsize':(11.7,8.27)})
sns.scatterplot(
    data=data,
    x='N',
    y='S',
    hue="Cf",
    palette='deep'
).set_title('Зависимость ускорения S от количества процессоров N для различных 
Cf (декомпозиция узлов)')
```