|  |  |
| --- | --- |
|  | **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 -мерный вектор параметров задачи. Положим, что  
 , где -мерное арифметическое пространство. Параллелепипедом допустимых значений вектора параметров назовем не пустой параллелепипед  
где заданные константы. На вектор дополнительно наложено некоторое количество функциональных ограничений, формирующих множество непрерывные ограничивающие функции.

На множестве тем или иным способом (аналитически или алгоритмически) определена вектор-функция со значениями в пространстве . Ставится задача поиска значения некоторого функционала .

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

*Шаг 1.*Покрываем параллелепипед некоторой сеткой Ω (равномерной или неравномерной, детерминированной или случайной) с узлами *Шаг 2.* В тех узлах сетки Ω, которые принадлежат множеству **,** вычисляем значения вектор функции .

*Шаг 3.* На основе вычисленных значений вектор функции находим приближенное значение функционала.

Суммарное количество арифметических операций, необходимых для *однократного* определения принадлежности вектора множеству (т.е. суммарную вычислительную сложность ограничений и ограничивающих функций , обозначим . *Далее в эксперименте будем полагать .*

Неизвестную вычислительную сложность вектор-функции обозначим . Подчеркнем зависимость величины  от вектора *X*. Величина удовлетворяет, во-первых, очевидному ограничению . Во-вторых, положим, что известно ограничение сверху на эту величину , имеющее смысл ограничения на максимально допустимое время вычисления значения . Вычислительную сложность назовем вычислительной сложностью узла.

Вычислительную сложность генерации сетки Ω положим равной , а вычислительную сложность конечномерной аппроксимации функционала - равной, где (дзета) - общее количество узлов сетки Ω, принадлежащих множеству**.**

*Далее в эксперименте также будем полагать .*

В качестве вычислительной системы рассмотрим однородную МВС с распределенной памятью, состоящую из процессоров и *host*-процессора, имеющих следующие параметры:

* – время выполнения одной арифметической операции с плавающей запятой;
* диаметр коммуникационной сети;
* длина вещественного числа в байтах;
* – латентность коммуникационной сети;

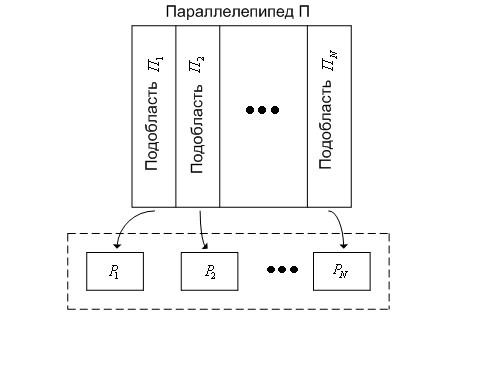
* – время передачи байта данных между двумя соседними процессорами системы без учета времени .

В качестве меры эффективности параллельных вычислений используем ускорение

где время последовательного решения задачи на одном процессоре cистемы, время параллельного решения той же задачи на *N* процессорах, номер метода балансировки.

**Статическая балансировка загрузки методом равномерной декомпозиции параллелепипеда П**

Простейшим методом балансировки загрузки является статический метод на основе декомпозиции параллелепипеда на равных подобластей и назначении каждой из этих подобластей своему процессору. Назовем данный метод балансировки методом равномерной декомпозиции параллелепипеда *П*. Для двумерного случая 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 *П* время решения задачи на процессоре можно оценить величиной

где количество узлов в сетке (т.е. попавших в подобласть ); количество узлов в сетке , попавших в множество .

Время решения всей задачи можно оценить величиной

а время решение задачи на одном процессоре величиной

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

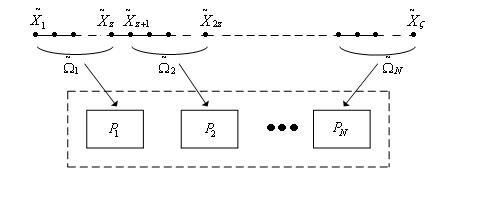
* в квадрате П строим равномерную по каждому из измерений сетку Ω;
* прямыми, параллельными одной из осей координат разбиваем квадрат на  одинаковых подобластей  
  ;
* для всех подобластей находим количества узлов;
* по формуле (2) вычисляем значение величины;
* по формуле (3) находим величину ;
* по формуле (4) определяем значение величины ; по формуле (1) находим оценку ускорения.

Примем также, что вычислительная сложность вектор-функции одинакова во всей области .

**Статическая балансировка загрузки методом равномерной декомпозиции параллелепипеда П**

Положим, что из числа узлов расчетной сетки Ω множеству принадлежит узлов. Обозначим . Тогда идею рассматриваемого метода балансировки загрузки можно представить в следующем виде (рис. 2):

* среди всех узлов сетки Ω выделяем узлов ;
* разбиваем узлы на множеств , где множество содержит узлы , множество - узлы и т.д.
* назначаем для обработки процессору  множеств узлов , .



**Рис. 2**. К балансировке загрузки методом 2.

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

время параллельного решения всей задачи – величиной

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 строим равномерную по каждому из измерений сетку Ω;
* находим количества узлов ;
* по формуле (5) вычисляем значение величины ;
* по формуле (6) находим величину ;
* по формуле (4) определяем значение величины ;
* по формуле (1) находим оценку ускорения.

**Экспериментальная часть**

Исходные данные:

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



Табл. 2

Равномерная декомпозиция расчетных узлов



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

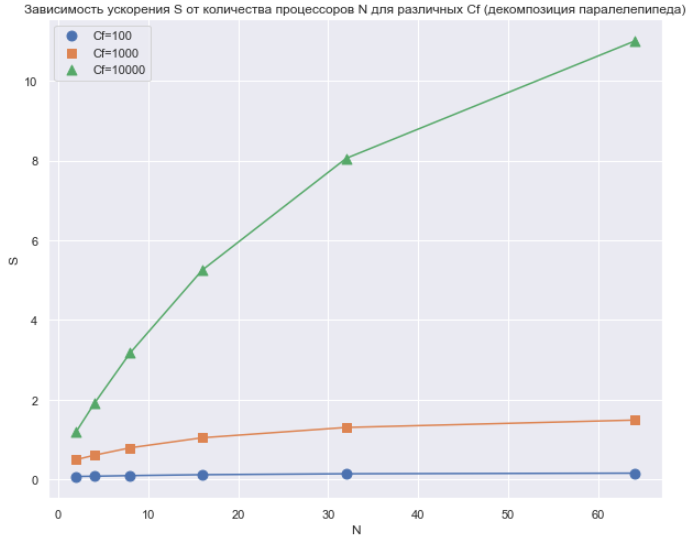


Рис. 1

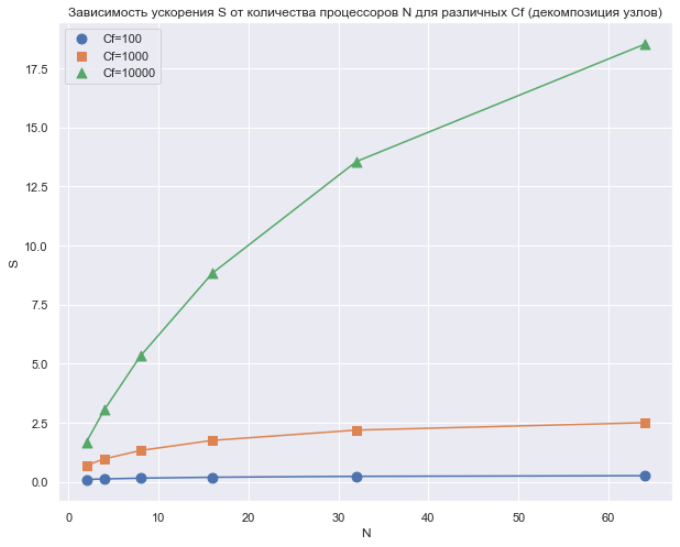


Рис. 2

**Ответы на контрольные вопросы**

1. *Почему ускорения   при всех N меньше ?*

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

1. *Почему ускорение  меньше ускорения ?*

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

1. *Почему наблюдается отклонение зависимостей   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)')