|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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

[Оглавление 2](#_Toc131539214)

[Цель лабораторной работы 3](#_Toc131539215)

[Постановка задачи 3](#_Toc131539216)

[Статическая балансировка загрузки методом равномерной декомпозиции параллелепипеда П 5](#_Toc131539217)

[Вычислительный эксперимент 10](#_Toc131539218)

[Метод равномерной декомпозиции параллелепипеда 11](#_Toc131539219)

[Метод равномерной декомпозиции расчётных узлов 11](#_Toc131539220)

[Программная реализация 12](#_Toc131539221)

[Результат работы программы 14](#_Toc131539222)

[Заключение 17](#_Toc131539223)

[Список использованных источников 18](#_Toc131539224)

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 – n-мерный вектор параметров задачи. Положим, что , где – n-мерное арифметическое пространство. Параллелепипедом допустимых значенийвектора параметров назовем не пустой параллелепипед *П*, где - заданные константы. На вектор *X* дополнительно наложено некоторое количество функциональных ограничений, формирующих множество , где - 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 *П* некоторой сеткой (равномерной или неравномерной, детерминированной или случайной) с узлами .

*Шаг 2.* В тех узлах сетки , которые принадлежат множеству , вычисляем значения вектор функции .

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

В виде рассмотренной схемы можно представить, например, решение задачи вычисления многомерного определенного интеграла от функции  в области .

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

Неизвестную вычислительную сложность вектор-функции  обозначим . Подчеркнем зависимость величины от вектора . 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 положим равной , а вычислительную сложность конечномерной аппроксимации функционала  - равной , где – общее количество узлов сетки , принадлежащих множеству . Положим, что при данных *n ,m* величины , – известные константы.

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

* – длина вещественного числа в байтах;
* – время выполнения одной арифметической операции с плавающей запятой;
* – латентность коммуникационной сети;
* – время передачи байта данных между двумя соседними процессорами системы без учета времени ;
* – диаметр коммуникационной сети.

В качестве меры эффективности параллельных вычислений используем ускорение

где – время последовательного решения задачи на одном процессоре системы, – время параллельного решения той же задачи на *N* процессорах, - номер метода балансировки. Будем рассматривать также в качестве меры эффективности параллельных вычислений асимптотическое ускорение

где - время параллельного решения задачи на *N* процессорах без учета коммуникационных расходов (когда ).

Статическая балансировка загрузки методом равномерной декомпозиции параллелепипеда П

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

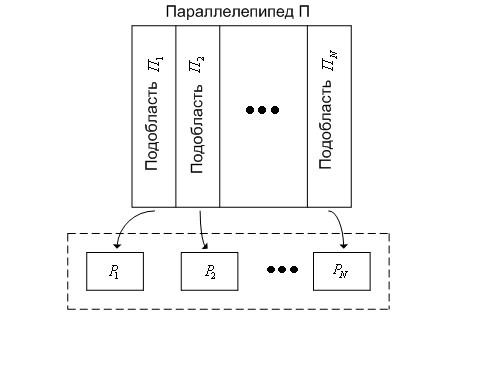


Рисунок 1 – Схема балансировки методом равномерной декомпозиции параллелепипеда *П*.

Введём следующие обозначения:

* – множество узлов сетки , покрывающих подобласть ;
* – количество узлов во множестве , , где символ означает ближайшее большее целое;
* - количество узлов сетки , принадлежащий множеству ;
* – узлы сетки , принадлежащие множеству ;
* – общее количество узлов сетки , принадлежащих множеству , ;

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:

* cтроит cетку ;
* плоскостями, параллельными одной из координатных плоскостей, разбивает ее узлы на множества ;
* передает процессору координаты узлов множества .

*Шаг 2.* Процессор выполняет следующие действия:

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

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

* принимает от процессоров вычисленные ими значения вектор-функции 𝐹(𝑋);
* на основе 𝜁 полученных значений вектор-функции 𝐹(𝑋) вычисляет приближенное значение функционала Ф(𝐹(𝑋)).

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

где сумма представляет собой вычислительную загрузку процессора , а сумма – его коммуникационную нагрузку. Отсюда следует, что время параллельного решения всей задачи равно

Аналогично, время решения задачи на одном процессоре равно

**Статическая балансировка загрузки методом равномерной декомпозиции расчетных узлов**

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

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

Наглядно схема работы метода изображена на рисунке 2.

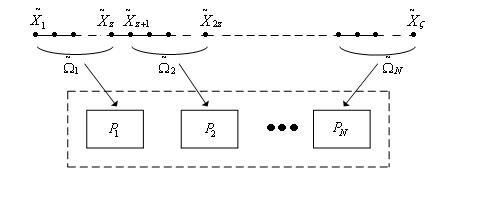


Рисунок 2 – Схема балансировки методом равномерной декомпозиции

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:

* cтроит cетку ;
* среди всех узлов сетки выделяет узлов ;
* разбивает узлы на множеств узлов ;
* передает процессору координаты узлов множества .

*Шаг 2.* Процессор выполняет следующие действия:

* принимает от host-процессора координаты узлов множества ;
* вычисляет в каждом из этих узлов значение вектор-функции 𝐹(𝑋);
* передает *host-*процессору вычисленных векторов и заканчивает вычисления.

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

* принимает от процессоров вычисленные ими значения вектор-функции 𝐹(𝑋);
* на основе полученных значений вектор функций 𝐹(𝑋) вычисляет приближённое значение функционала Ф(F(X)).

Обозначим – узлы сетки , принадлежащие множеству . Тогда при балансировке загрузки методом равномерной декомпозиции расчётных узлов время решения задачи на процессоре можно оценить величиной

где слагаемое представляет собой вычислительную загрузку процессора , а слагаемые – его коммуникационную загрузку. Таким образом, время параллельного решения всей задачи оценивается величиной

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*. Положим, что , , так что область *П* является единичным квадратом (рисунок 3).

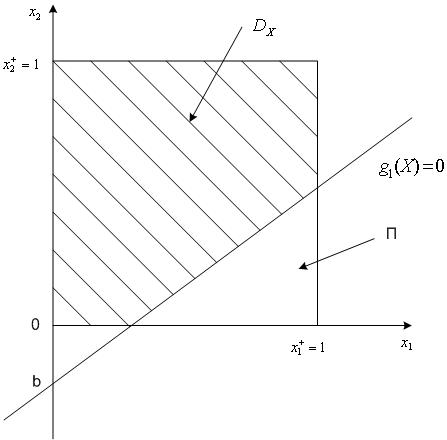


Рисунок 3 – Расчетная область задачи.

Множество *D* формируется с помощью одной ограничивающей функции , т.е. . Примем, что эта функция линейна и проходит через заданную преподавателем точку плоскости с координатами (0, b). Таким образом, уравнение этой функции имеет вид (при этом, очевидно, ).

В качестве сетки используем равномерную детерминированную сетку с количеством узлов по осям и равным 256, т.е. сетку с количеством узлов .

Будем исходить из следующих значений параметров задачи и МВС:

* ;
* ;
* ;
* ;
* ;
* .

Пренебрежем вычислительными затратами на построение сетки , на вычисление значений ограничивающей функции 𝑔1(𝑋), а также на построение приближенного значения функционала Ф(𝐹(𝑋)), т.е. положим 𝐶𝛺 = 0, 𝐶𝑔 = 0, 𝐶Ф = 0. Примем также, что вычислительная сложность 𝐶𝑓 вектор-функции 𝐹(𝑋) одинакова во всей области 𝐷𝑋.

Метод равномерной декомпозиции параллелепипеда

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

время параллельного решения всей задачи – величиной

а время решения задачи на одном процессоре – величиной

Асимптотическое ускорение данного метода равно

Метод равномерной декомпозиции расчётных узлов

Для данного метода время решения задачи на процессоре можно оценить величиной

где слагаемое представляет собой вычислительную загрузку процессора , а слагаемые – его коммуникационную загрузку.

Время параллельного решения всей задачи оценивается величиной

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

Асимптотическое ускорение метода равно

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, 𝐶𝑓 были вычислены ускорения для метода равномерной декомпозиции расчетной области и 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 𝐶𝑓, а также асимптотические ускорения методов

и при .

Таблица 1. Результаты работы программы

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | |  | |  |  |
|  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |
| 16 |  |  |  |  |  |  |
| 32 |  |  |  |  |  |  |
| 64 |  |  |  |  |  |  |

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

На рисунках 4 и 5 графически представлены зависимости , , для 𝐶𝑓 = 0.5∙107 и 𝐶𝑓 = 5∙104.



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 𝐶𝑓 вектор-функции . Полученные результаты свидетельствуют о том, что метод равномерной декомпозиции расчётных узлов имеет большую эффективность. Увеличение вычислительной сложности 𝐶𝑓 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>).