МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ   
РОССИЙСКОЙ ФЕДЕРАЦИИ  
  
Федеральное государственное автономное образовательное учреждение   
высшего образования   
«Самарский национальный исследовательский университет   
имени академика С.П. Королёва» (Самарский университет)  
  
Институт информатики и кибернетики  
Кафедра программных систем  
  
Дисциплина  
**Суперкомпьютеры и их применение**

ОТЧЕТ  
по лабораторной работе №6  
  
Исследование эффективности использования прикладного программного обеспечения на суперкомпьютерных системах

Студент: Мананников М.А.

Группа: 6403-020302D  
  
Преподаватель: Хабибуллин Р.М.  
Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Самара 2025

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc197813009)

[2 Ход работы 4](#_Toc197813010)

[2.1 Описание последовательного алгоритма умножения матриц 4](#_Toc197813011)

[2.2 Перемножение с помощью библиотек 5](#_Toc197813012)

[2.2.1 Библиотека Gonum 5](#_Toc197813013)

[2.3 Результаты работы 6](#_Toc197813014)

[ЗАКЛЮЧЕНИЕ 8](#_Toc197813015)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 9](#_Toc197813016)

[ПРИЛОЖЕНИЕ А Реализация последовательного алгоритма 10](#_Toc197813017)

[ПРИЛОЖЕНИЕ Б Реализация алгоритма с помощью библиотеки Gonum 11](#_Toc197813018)

1. Постановка задачи

Цель работы: Изучение существующих программных продуктов для кластерных систем.

Порядок выполнения работы:

1. Разработать программу, реализующую параллельный алгоритм матричного умножения C=AB с использованием библиотеки готовых подпрограмм согласно варианту.
2. Измерить время работы программы для различных размеров матриц;
3. Сравнить производительность программы с производительностью матричного умножения с помощью последовательного алгоритма.
4. Объяснить наблюдаемые зависимости.
5. Составить отчёт по результатам работы.

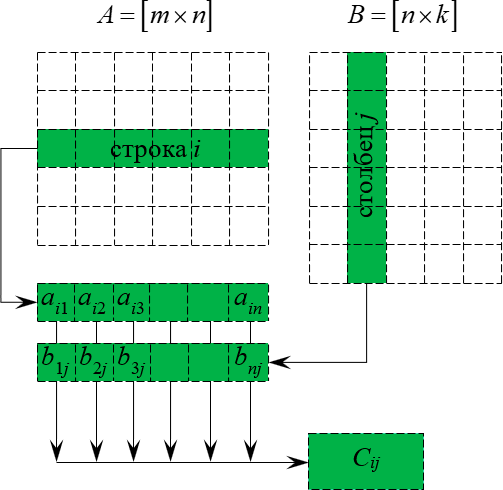
1. Ход работы
   1. Описание последовательного алгоритма умножения матриц

Рассмотрим две согласованные матрицы: и . И определим для них операцию умножения [1].

Произведение двух согласованных матриц и – это новая матрица , элементы которой считаются по формуле:

Обозначается такое произведение стандартно: C = A\*B [1].

Процесс перемножения двух матриц изображён на рисунке 1.

  
Рисунок 1 – Схема перемножения двух матриц

Код программы умножения матриц с помощью последовательного алгоритма приведён в приложении А.

* 1. Перемножение с помощью библиотек

В языке Go также существует обширная стандартная библиотека — это набор тщательно протестированных и готовых к использованию решений, которые помогают программистам справляться с повседневными задачами программирования [2].

Стандартная библиотека Go организована по пакетам (packages), которые можно представить как полки с инструментами, где каждая "полка" отвечает за своё направление. Каждый пакет содержит функции, типы и интерфейсы, решающие конкретную задачу.

Примеры стандартных пакетов в Go:

* fmt — для форматированного ввода и вывода;
* math — базовые математические функции;
* time — работа с временем и таймерами;
* net/http — работа с HTTP;
* os — взаимодействие с операционной системой;
* sync — работа с параллелизмом и потокобезопасностью.

В своей работе я использовал внешние библиотеки для умножения матриц:

* библиотеку Gonum — мощную и популярную библиотеку для численных расчётов;
* собственную реализацию на основе двумерных срезов **(**[][]float64**)**, чтобы сравнить производительность с более простой реализацией.
  + 1. Библиотека Gonum

Gonum — это библиотека для численных вычислений на языке Go, аналогичная JAMA в Java. Она предоставляет функции линейной алгебры и другие инструменты для научных и инженерных расчётов. Библиотека активно развивается сообществом и распространяется с открытым исходным кодом [3].

Gonum не требует отдельной компиляции — всё подключается через пакетный менеджер Go (go get). Она использует типы и структуры Go, что делает её удобной и безопасной в использовании.

Основные операции, поддерживаемые библиотекой:

* LU-разложение;
* обращение матриц;
* вычисление определителей;
* нахождение собственных значений и собственных векторов;
* QR-разложение;
* разложение Холецкого;
* сингулярное разложение (SVD).

Gonum написана на чистом Go и не требует сторонних зависимостей. Благодаря этому она хорошо интегрируется в проекты на Go и позволяет легко обрабатывать матрицы и векторы с типом float64, который является стандартом для чисел с плавающей точкой в Go [3].

Реализация данного алгоритма представлена в приложении Б.

* 1. Результаты работы

Вычислительные эксперименты проводились на квадратных матрицах, заполненных числами от 0 до 1.

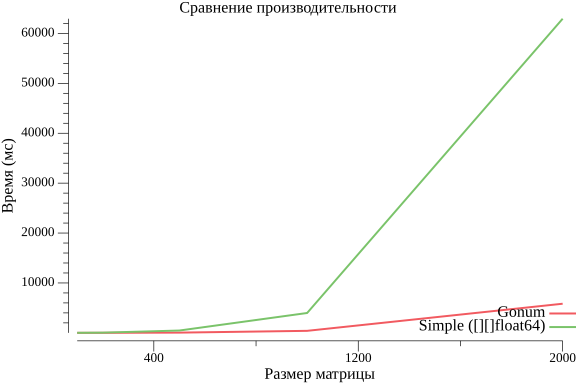
Сравнение результатов последовательного алгоритма и алгоритмов с использованием библиотек представлены в таблице 1.

Таблица 1 – Сравнение результатов работы алгоритма

|  |  |  |
| --- | --- | --- |
| Размерность | Время работы в миллисекундах | |
| math | gonum |
| 1 | 2 | 3 |
| 100 | 7.84 | 1.26 |
| 200 | 40.98 | 7.23 |
| 500 | 466.03 | 46.86 |

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 1000 | 3979.35 | 403.42 |
| 2000 | 62949.16 | 5835.55 |

На рисунке 1 представлен график изменения последовательного алгоритма и алгоритмов с использованием библиотек в зависимости от размерности матрицы.

  
Рисунок 1 – Сравнение времени работы алгоритмов

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы были реализованы два алгоритма умножения матриц на языке Go:  
последовательный алгоритм на основе двумерных срезов ([][]float64),  
и алгоритмы на базе библиотеки линейной алгебры Gonum, включая последовательный и параллельный варианты.

Были проведены вычислительные эксперименты для различных размеров матриц, и получены следующие выводы:

* библиотека Gonum обеспечивает ускорение по сравнению с простой реализацией при больших размерах матриц (от 300×300 и выше);
* при небольших размерах матриц (до 200×200) простая реализация на [][]float64 может работать быстрее из-за меньших накладных расходов;
* параллельное умножение на Gonum даёт заметное преимущество по скорости при больших размерах, особенно при использовании нескольких потоков (goroutine);
* при увеличении размера матрицы разница в производительности становится существенной: Gonum с параллельным вычислением в некоторых случаях в 5–10 раз быстрее.

Таким образом, использование специализированных библиотек в Go позволяет существенно ускорить вычисления, особенно при работе с большими объёмами данных и применении параллелизма.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Умножение матриц [Электронный ресурс] // Павел Бердов. Репетитор по математике: [сайт]. URL: https://www.berdov.com /works/matrix/umnozhenie-matric/ (дата обращения: 26.03.2022).
2. Что такое библиотеки классов Go? [Электронный ресурс] // Vertex: [сайт]. URL: https://vertex-academy.com/tutorials/ru/biblioteka-java/ (дата обращения: 27.03.2022).
3. Gonum (библиотека) [Электронный ресурс] // Строительные материалы [сайт]. URL: https://delta-design.ru/stati/17777-jama-biblioteka.html (дата обращения: 27.03.2022).

ПРИЛОЖЕНИЕ А  
Реализация последовательного алгоритма

package simpleimpl

import (

"math/rand"

)

func RandomMatrix(rows, cols int) [][]float64 {

data := make([][]float64, rows)

for i := range data {

row := make([]float64, cols)

for j := range row {

row[j] = rand.Float64() \* 10

}

data[i] = row

}

return data

}

func Multiply(a, b [][]float64) [][]float64 {

ar, ac := len(a), len(a[0])

br, bc := len(b), len(b[0])

if ac != br {

panic("несовместимые размеры матриц")

}

result := make([][]float64, ar)

for i := range result {

result[i] = make([]float64, bc)

for j := 0; j < bc; j++ {

for k := 0; k < ac; k++ {

result[i][j] += a[i][k] \* b[k][j]

}

}

}

return result

}

ПРИЛОЖЕНИЕ Б   
Реализация алгоритма с помощью библиотеки Gonum

package gonumimpl

import (

"math/rand"

"sync"

"gonum.org/v1/gonum/mat"

)

func RandomMatrix(rows, cols int) \*mat.Dense {

data := make([]float64, rows\*cols)

for i := range data {

data[i] = rand.Float64() \* 10

}

return mat.NewDense(rows, cols, data)

}

func ParallelMultiply(a, b \*mat.Dense, workers int) \*mat.Dense {

ar, ac := a.Dims()

br, bc := b.Dims()

if ac != br {

panic("несовместимые размеры матриц")

}

result := mat.NewDense(ar, bc, nil)

var wg sync.WaitGroup

chunkSize := ar / workers

for w := 0; w < workers; w++ {

start := w \* chunkSize

end := start + chunkSize

if w == workers-1 {

end = ar

}

wg.Add(1)

go func(start, end int) {

defer wg.Done()

for i := start; i < end; i++ {

for j := 0; j < bc; j++ {

var sum float64

for k := 0; k < ac; k++ {

sum += a.At(i, k) \* b.At(k, j)

}

result.Set(i, j, sum)

}

}

}(start, end)

}

wg.Wait()

return result

}