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: Гижевская В.Д.

Группа: 6413-020302D  
  
Преподаватель: Хабибуллин Р.М.  
Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Самара 2022

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc99412906)

[2 Ход работы 4](#_Toc99412907)

[2.1 Описание последовательного алгоритма умножения матриц 4](#_Toc99412908)

[2.2 Перемножение с помощью библиотек 5](#_Toc99412909)

[2.2.1 Библиотека JAMA 5](#_Toc99412910)

[2.2.2 Библиотека MTJ 6](#_Toc99412911)

[2.3 Результаты работы 7](#_Toc99412912)

[Заключение 9](#_Toc99412913)

[Список использованных источников 10](#_Toc99412914)

[Приложение А Реализация последовательного алгоритма 11](#_Toc99412915)

[Приложение Б Реализация алгоритма с помощью библиотеки JAMA 12](#_Toc99412916)

[Приложение В Реализация алгоритма с помощью библиотеки MTJ 13](#_Toc99412917)

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.

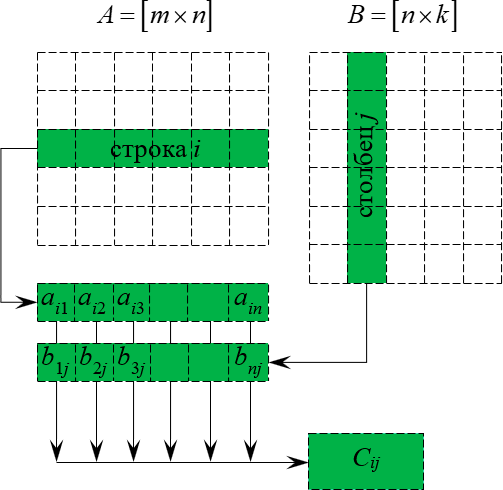
1. Ход работы
   1. Описание последовательного алгоритма умножения матриц

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 Java есть виртуальная библиотека протестированного кода - это уже готовые решения ко многим задачам, которые стоят перед программистами в их ежедневной работе [2].

В этой виртуальной библиотеке Java информация разбита по пакетам ("packages") - это своеобразный аналог полочек в книжном магазине. В каждом пакете протестированный код по какому-то отдельно взятому направлению [2].

Например, есть такие пакеты:

* java.applet
* java.lang – это основной пакет языка Java
* java.util
* java.io
* java.net
* и т.д.

В своей работе я использовала специальные библиотеки для перемножения матриц – библиотека JAMA и библиотека MTJ.

* + 1. Библиотека JAMA

JAMA (англ. Java Matrix Library — библиотека матриц на языке Java) — библиотека функций линейной алгебры. Библиотека создана в NIST и является общественным достоянием [3].

Библиотека существует в двух версиях: на языке Java (собственно JAMA) и библиотека шаблонов на языке C++ (JAMA/C++). Версия на C++ использует Template Numerical Toolkit, разработанный там же. Версия на Java выполняет низкоуровневые операции сама.

Основные операции, выполняемые библиотекой:

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

Поскольку JAMA не содержит ничего, кроме заголовочных файлов с шаблонами, библиотека не требует компиляции. Поскольку все классы используют шаблоны, одинаково легко использовать матрицы и вектора с элементами типа float, double или описанных пользователем типов [3].

Реализация данного алгоритма представлена в приложении Б.

* + 1. Библиотека MTJ

Matrix Toolkit Java (MTJ) — это программная библиотека Java с открытым исходным кодом для выполнения численной линейной алгебры. Библиотека содержит полный набор стандартных операций линейной алгебры для плотных матриц на основе кода BLAS и LAPACK. Частичный набор разреженных операций предоставляется через проект Templates. Библиотеку можно настроить для работы как чистой библиотеки Java или использовать машинно-оптимизированный код BLAS через собственный интерфейс Java [4].

Ниже приводится обзор возможностей MTJ, перечисленных на веб-сайте проекта [4]:

* Структуры данных для плотных и структурированных разреженных матриц.
* Прозрачная поддержка симметричного и треугольного хранения.
* Структуры данных для неструктурированных разреженных матриц.
* Плотные и структурированные разреженные матрицы строятся поверх BLAS и LAPACK и включают различные встроенные операции.
* Неструктурированные разреженные матрицы поддерживают те же операции, что и структурированные, за исключением того, что они не имеют прямых решателей. Однако их методы умножения матриц/векторов оптимизированы для использования в итерационных решателях.
* Матричное разложение плотных и структурированных разреженных матриц.
* Итерационные решатели для неструктурированных разреженных матриц из проекта Templates.
* Подборка алгебраических предобуславливателей.

Реализация данного алгоритма представлена в приложении В.

* 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 | | |
| Последовательный алгоритм | JAMA | MTJ |
| 100 | 0,008 | 0,004 | 0,235 |
| 200 | 0,031 | 0,03 | 0,325 |
| 500 | 0,578 | 0,181 | 0,431 |
| 1000 | 10,273 | 1,438 | 1,204 |
| 2000 | 119,545 | 10,415 | 7,123 |
| 5000 | 1564,712 | 157,878 | 101,041 |

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 процессе выполнения лабораторной работы был разработан последовательный алгоритм умножения матриц, а также были найдены библиотеки, ускоряющие работу программы (библиотеки JAMA и MTJ) на языке Java, проведены вычислительные эксперименты для различных параметров задачи и получены следующие выводы и закономерности:

1. При любых параметрах задачи последовательный алгоритм работает медленнее, чем алгоритм на основе библиотеки JAMA;
2. При малом количестве строк (меньше 200) алгоритм, написанный с помощью библиотеки MTJ работает медленнее, чем последовательный или алгоритм на JAMA. Однако при увеличении количества строк его скорость работы значительно меньше, чем у последовательного и JAMA.
3. При большом количестве строк (больше 500), алгоритмы, написанные с помощью библиотек, дают значительное ускорение (примерно в 10 раз)

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 Java? [Электронный ресурс] // Vertex: [сайт]. URL: https://vertex-academy.com/tutorials/ru/biblioteka-java/ (дата обращения: 27.03.2022).
3. JAMA (библиотека) [Электронный ресурс] // Строительные материалы [сайт]. URL: https://delta-design.ru/stati/17777-jama-biblioteka.html (дата обращения: 27.03.2022).
4. Матрица Инструментарий Java [Электронный ресурс] // HMong: [сайт]. URL: https://www.hmong.press/wiki/Matrix\_Toolkit\_Java (дата обращения: 27.03.2022).

ПРИЛОЖЕНИЕ А  
Реализация последовательного алгоритма

package lab;

import java.time.Duration;

import java.time.Instant;

import java.util.Random;

public class Main {

public static double[][] getMatrix(int size) {

double[][] matrix = new double[size][size];

Random random = new Random();

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

matrix[i][j] = random.nextDouble();

}

}

return matrix;

}

static int size = 200;

public static void main(String[] args) {

double[][] a = getMatrix(size);

double[][] b = getMatrix(size);

double[][] res = new double[size][size];

Instant time1 = Instant.now();

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

double x = 0.0;

for (int k = 0; k < size; k++) {

x += a[i][k] \* b[k][j];

}

res[i][j] = x;

}

}

Instant time2 = Instant.now();

System.out.printf("Duration (milliseconds): %d%n", Duration.between(time1, time2).toMillis());

}

ПРИЛОЖЕНИЕ Б   
Реализация алгоритма с помощью библиотеки JAMA

package lab;

import jama.Matrix;

import java.time.Duration;

import java.time.Instant;

public class MainJAMA {

static int size = 5000;

public static void main(String[] args) {

Matrix a = Matrix.random(size, size);

Matrix b = Matrix.random(size, size);

Instant time1 = Instant.now();

Matrix res = a.times(b);

Instant time2 = Instant.now();

System.out.printf("Duration (milliseconds): %d%n", Duration.between(time1, time2).toMillis());

}

}

ПРИЛОЖЕНИЕ В   
Реализация алгоритма с помощью библиотеки MTJ

package lab;

import no.uib.cipr.matrix.DenseMatrix;

import java.time.Duration;

import java.time.Instant;

import static lab.Main.getMatrix;

public class MainMTJ {

static int size = 5000;

public static void main(String[] args) {

DenseMatrix a = new DenseMatrix(getMatrix(size));

DenseMatrix b = new DenseMatrix(getMatrix(size));

DenseMatrix res = new DenseMatrix(size, size);

Instant time1 = Instant.now();

a.mult(b, res);

Instant time2 = Instant.now();

System.out.printf("Duration (milliseconds): %d%n", Duration.between(time1, time2).toMillis());

}

}