МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное  
образовательное учреждение высшего образования  
«Самарский национальный исследовательский университет  
имени академика С.П. Королева»  
(Самарский университет)

Институт информатики, математики и электроники

Факультет информатики

Кафедра программных систем

**ОТЧЁТ**

по лабораторной работе №4 по дисциплине

«Моделирование и анализ параллельных алгоритмов»

Студент группы №6132-020402D В.А. Артамонов

Проверил Д.С. Оплачко

Самара 2021

СОДЕРЖАНИЕ

[1 Постановка задачи 4](#_Toc88593032)

[2 Описание работы параллельного алгоритма 7](#_Toc88593033)

[3 Текст различных вариантов программы 8](#_Toc88593034)

[3.1 Параллельный алгоритм 8](#_Toc88593035)

[3.2 Последовательный алгоритм 10](#_Toc88593036)

[4 Результаты вычислительных экспериментов 11](#_Toc88593037)

[5 Заключение 13](#_Toc88593038)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 14](#_Toc88593039)

[ПРИЛОЖЕНИЕ А 15](#_Toc88593040)

[Последовательный алгоритм 15](#_Toc88593041)

[А.1 Код класса App 15](#_Toc88593042)

[А.2 Код класса AppSettings 16](#_Toc88593043)

[А.3 Код класса AppSettingsParser 16](#_Toc88593044)

[А.4 package nbody 18](#_Toc88593045)

[А.4.1 package nbody.exceptions 18](#_Toc88593046)

[А.4.1.1 Код класса BodiesNumOutOfBoundsException 18](#_Toc88593047)

[А.4.1.2 Код класса BodyMassOutOfBoundsException 18](#_Toc88593048)

[А.4.1.3 Код класса DeltaTimeOutOfBoundsException 18](#_Toc88593049)

[А.4.1.4 Код класса ErrorDistanceOutOfBoundsException 18](#_Toc88593050)

[А.4.1.5 Код класса ThreadsNumOutOfBoundsException 18](#_Toc88593051)

[А.4.2 Код класса Bodies 18](#_Toc88593052)

[А.4.3 Код класса Body 18](#_Toc88593053)

[А.4.4 Код класса Coords 19](#_Toc88593054)

[А.4.5 Код класса Helpers 20](#_Toc88593055)

[А.4.6 Код класса NbodySettings 20](#_Toc88593056)

[А.4.7 Код класса NbodySolver 21](#_Toc88593057)

[А.4.8 Код класса NbodySolvers 22](#_Toc88593058)

[А.5 package nbodygui 23](#_Toc88593059)

[А.5.1 package nbodygui.exceptions 23](#_Toc88593060)

[А.5.1.1 Код класса DurationMillisOutOfBoundsException 23](#_Toc88593061)

[А.5.1.2 Код класса HeightOutOfBoundsException 23](#_Toc88593062)

[А.5.1.3 Код класса WidthOutOfBoundsException 23](#_Toc88593063)

[А.5.2 Код класса Frame 24](#_Toc88593064)

[А.5.3 Код класса Frames 24](#_Toc88593065)

[А.5.4 Код класса Panel 24](#_Toc88593066)

[А.5.5 Код класса Panels 25](#_Toc88593067)

[ПРИЛОЖЕНИЕ Б 26](#_Toc88593068)

[Параллельный алгоритм 26](#_Toc88593069)

[Б.1 Код класса App 26](#_Toc88593070)

[Б.2 Код класса AppSettings 26](#_Toc88593071)

[Б.3 Код класса AppSettingsParser 27](#_Toc88593072)

[Б.4 package nbody 29](#_Toc88593073)

[Б.4.1 package nbody.exceptions 29](#_Toc88593074)

[Б.4.1.1 Код класса BodiesNumOutOfBoundsException 29](#_Toc88593075)

[Б.4.1.2 Код класса BodyMassOutOfBoundsException 29](#_Toc88593076)

[Б.4.1.3 Код класса DeltaTimeOutOfBoundsException 29](#_Toc88593077)

[Б.4.1.4 Код класса ErrorDistanceOutOfBoundsException 29](#_Toc88593078)

[Б.4.1.5 Код класса ThreadsNumOutOfBoundsException 30](#_Toc88593079)

[Б.4.2 Bodies 30](#_Toc88593080)

[Б.4.3 Body 30](#_Toc88593081)

[Б.4.4 Coords 31](#_Toc88593082)

[Б.4.5 Helpers 31](#_Toc88593083)

[Б.4.6 NbodySettings 32](#_Toc88593084)

[Б.4.7 NbodySolver 33](#_Toc88593085)

[Б.4.8 NbodySolvers 36](#_Toc88593086)

[Б.5 package nbodygui 37](#_Toc88593087)

[Б.5.1 package nbodygui.exceptions 37](#_Toc88593088)

[Б.5.1.1 Код класса DurationMillisOutOfBoundsException 37](#_Toc88593089)

[Б.5.1.2 Код класса HeightOutOfBoundsException 37](#_Toc88593090)

[Б.5.1.3 Код класса WidthOutOfBoundsException 37](#_Toc88593091)

[Б.5.2 Код класса Frame 37](#_Toc88593092)

[Б.5.3 Код класса Frames 38](#_Toc88593093)

[Б.5.4 Код класса Panel 38](#_Toc88593094)

[Б.5.5 Код класса Panels 39](#_Toc88593095)

1. Постановка задачи
2. С использованием одной из моделей взаимодействия параллельных процессов разработать параллельную программу моделирования гравитационного взаимодействия *N* тел.

Каждое тело имеет массу, начальное положение и скорость. Гравитация вызывает перемещение и ускорение тел. Движение системы *N* тел имитируется пошагово с помощью дискретных отрезков времени. На каждом временном шаге вычисляются силы, действующие на каждое тело, и обновляются скорости и положения тел. Имитация гравитационного взаимодействия имеет следующую структуру:

// инициализировать тела

for ( t = t\_start; t < t\_finish; t = t + dt ) {

// вычислить силы

// переместить тела

}

Значение dt является временным шагом.

Величина силы гравитации между двумя телами *i* и *j* вычисляется по формуле:

*F* = *G\*mi\*mj /r2*, (1)

где *mi* и *mj* – массы тел, а *r* – расстояние между ними. Гравитация является чрезвычайно слабым взаимодействием, поэтому значение гравитационной постоянной *G* очень мало: 6.67\*10-11.

Предположим, что все тела расположены на одной плоскости, т.е. задача является двумерной. В этом случае силы, ускорения, скорости и положения тел являются двумерными векторами, то есть их значения задаются парами чисел, каждое из которых определяет проекцию вектора на соответствующую координатную ось. Например, ускорение тела *i* **a***i*=[*aix*, *aiy*].

Направление силы, действующей на тело *i* со стороны тела *j*, задается единичным вектором, направленным от тела *i* в сторону тела *j*, а силы воздействия тела *i* на тело *j* – противоположным вектором. Величины сил, действующих между любыми двумя телами, равны, а направления противоположны. После вычисления модуля силы по формуле (1) необходимо вычислить величину проекций вектора силы на координатные оси, исходя из направления действия силы. Вычисленные значения будут компонентами вектора силы F= [*Fx, Fy*].

Общая сила, действующая на тело, равна векторной сумме сил воздействия со стороны всех остальных тел.

Гравитационные силы, действующие на тело, вызывают его ускорение и перемещение. Ускорение тела i равно отношению общей силы *F*, действующей на тело, к массе тела:

*a = F/mi*

Если за малый интервал времени *dt* ускорение ai тела i остается практически постоянным, то изменение скорости приблизительно равно:

*dvi = ai\*dt*

Изменение положения тела есть интеграл его скорости и ускорения на интервале времени *dt*, который приблизительно равен:

*dpi = vi\*dt + (ai /2)\*dt2 = (vi + dvi /2)\*dt*

Эта формула устроена по так называемой скачкообразной (leapfrog) схеме, в которой одна половина изменения в положении тела обусловлена предыдущей скоростью, а другая – новой.

При использовании приведенных выше формул следует различать векторные и скалярные величины. Компоненты соответствующих векторов следует вычислять отдельно или использовать операции над векторами.

1. Программа должна визуально изображать изменение взаимного положения тел в течение некоторого интервала времени (нескольких дискретных шагов времени) под действием гравитационных сил. Размерами тел можно пренебречь, представив их точками.
2. Измерить длительность работы программы для различного количества тел и различного количества потоков. Количество тел должно задаваться пользователем. Количество потоков взять от 1 до 64, используя до 10 различных значений. Длительность работы последовательной программы должна быть не менее одной минуты.
3. Объяснить наблюдаемые изменения длительности работы программы.
4. Составить отчет по результатам работы.
5. Описание работы параллельного алгоритма

Основанию написания параллельного алгоритма послужила парадигма параллельного программирования, описанная в задании, «взаимодействующие равные» [1].

Распределение вычислений сил притяжения тел и их координат фиксировано заранее, вычисления распределены между потоками. Реализация алогритма представлена в виде двух двумерных массивов recalcingCallables и movingCallables класса NbodySolver. Количество строк массивов соответствует номеру потока, количество столбцов фиксировано – 2 столбца.

Первый столбец – соответствует началу диапазона тел, второй – его концу, иначе говоря, каждая строка содержит диапазон номеров тел, который будет обрабатывать поток, соответствующий строке массива [3].

Задачи делятся поровну между потоками, участвующими в вычислениях. Разбиение на задачи происходит в конструкторе класса NbodySolver.

* recalcingCallables - одна задача имеет границы (диапазон) – номера тел, для которых вычисляются силы. В цикле for для каждого тела вычисляется сила его взаимодействия с другими телами: вычисляется на основе расстояний между телами distance, модуля силы взаимодействия тел с другим телом magnitude, также вычисляется направления суммарной силы (представлены в виде координат).

Первое тело притягивает второе, второе тело притягивает первое с той же величиной magnitude, но в противоположном направлении. Поэтому в цикле for вычисляем силу для второго тела (b[l]) – вычитание происходит в критической секции (захватывается монитор объекта NbodySolver), поскольку другой поток может изменить силу этого тела.

* movingCallables - одна задача имеет границы (диапазон) – номера тел, для которых вычисляются их координаты. В цикле for для каждого тела вычисляется его координата при помощи формул, описанных в задании. Координата вычисляется, используя текущее значения силы, координат и скорости тела, и вычисление происходит после пересчёта вектора силы тела – значит никакой другой поток не может изменить значение координаты тела (любой другой поток вычисляется значения своего диапазона тел) – критическая секция не требуется [4].

1. Текст различных вариантов программы
   1. Параллельный алгоритм (recalcingCallables)

private class RecalcingCallable implements Callable<Void> {

private final int leftIndex;

private final int rightIndex;

public RecalcingCallable(int leftIndex, int rightIndex) {

this.leftIndex = leftIndex;

this.rightIndex = rightIndex;

}

@Override

public Void call() {

double distance;

double magnitude;

Coords direction;

for (int k = leftIndex; k <= rightIndex; k++) {

for (int l = k + 1; l < b.length; l++) {

distance = distance(b[k], b[l]);

magnitude = (distance < errorDistance) ? 0.0 : magnitude(b[k], b[l], distance);

direction = direction(b[k], b[l]);

b[k].setF(

b[k].f().x() + magnitude \* direction.x() / distance,

b[k].f().y() + magnitude \* direction.y() / distance

);

synchronized (NbodySolver.this) {

b[l].setF(

b[l].f().x() - magnitude \* direction.x() / distance,

b[l].f().y() - magnitude \* direction.y() / distance

);

}

}

}

return null;

}

}

* 1. Параллельный алгоритм (movingCallables)

private class MovingCallable implements Callable<Void> {

private final int leftIndex;

private final int rightIndex;

public MovingCallable(int rangeStart, int rangeEnd) {

this.leftIndex = rangeStart - 1;

this.rightIndex = rangeEnd - 1;

}

@Override

public Void call() {

Coords dv; // dv = f/m \* dt

Coords dp; // dp = (v + dv/2) \* dt

for (int i = leftIndex; i <= rightIndex; i++) {

dv = dv(b[i], dt);

dp = dp(b[i], dt, dv);

b[i].setV(

b[i].v().x() + dv.x(),

b[i].v().y() + dv.y()

);

b[i].setP(

b[i].p().x() + dp.x(),

b[i].p().y() + dp.y()

);

b[i].setF(0.0, 0.0);

}

return null;

}

}

* 1. Параллельный алгоритм (Panel)

public void start() {

for (int t = 0; t <= durationMillis; t = t + solver.dt()) {

solver.recalcBodiesCoords();

repaint();

}

solver.stop();

}

* 1. Последовательный алгоритм

import java.util.ArrayList;  
  
public class Posled {  
  
 public static ArrayList<Integer> primes(int lastNumber) {  
  
 if (lastNumber < 2) {  
 throw new IndexOutOfBoundsException();  
 }  
  
 ArrayList<Integer> primes = new ArrayList<>(lastNumber / 2);  
  
 long startMillis = System.*currentTimeMillis*();  
 primes.add(2);  
 for (int i = 3; i <= lastNumber; i += 2) {  
 if (*isPrime*(i, primes)) {  
 primes.add(i);  
 }  
 }  
 long finishMillis = System.*currentTimeMillis*();  
 long consumedMillis = finishMillis - startMillis;  
 System.*out*.println("Consumed millis: " + consumedMillis);  
 return primes;  
 }  
 private static boolean isPrime(int number, ArrayList<Integer> primesBeforeNumber) {  
 double sqrtedNumber = Math.*sqrt*(number);  
 for (int prime : primesBeforeNumber) {  
 if (prime > sqrtedNumber) {  
 return true;  
 }  
 if (number % prime == 0) {  
 return false;  
 }  
 }  
 return true;  
 }  
}

1. Графический интерфейс

Графический интерфейс программы представлен в виде окна, на котором отрисовываются тела. Зеленая точка соответствуют координате тела в определённый момент времени, через заданный промежуток времени.

В файле settings.xml пользователь может задать настройки симуляции гравитационного взаимодействия тел и размеры окна. На рисунке приставлен скриншот содержимого файла settings.xml

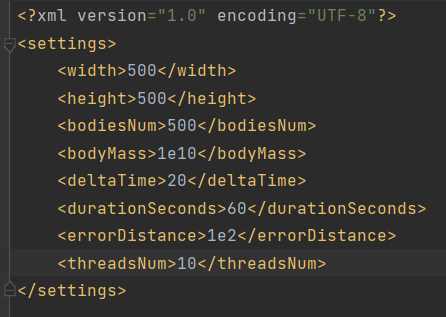


Рисунок 1 – Код файла «settings.xml»

* width – ширина окна
* height – высота окна
* bodiesNum – количество тел
* bodyMass – масса тела (одинаковая для каждого тела)
* deltaTime – промежуток времени, через который происходят пересчёты сил и координат тел
* durationSeconds – длительность симуляции (соответствует длительности симуляции, а не реальном времени, см. t\_finish в задании)
* errorDistance – расстояние обнуления силы взаимодействия (необходимо, иначе при сближении тела будут упруго отталкиваться с колоссальной силой)
* threadsNum – количество потоков.

1. Результаты вычислительных экспериментов

Вычислительные эксперименты, представленные в данном разделе, проводились на вычислительной системе с процессором, обладающим характеристиками, представленными в таблице 1.

Таблица 1 – Характеристики вычислительной системы

|  |  |
| --- | --- |
| Модель процессора | Intel(R) Core(TM) i3-6006U |
| CPU | 2.00GHz |
| Число ядер | 4 |
| Hyper-Threading (multithreading) | Есть |

В ходе выполнения лабораторной работы были проведены измерения времени работы параллельного алгоритмов. Результаты представлены в таблице 2.

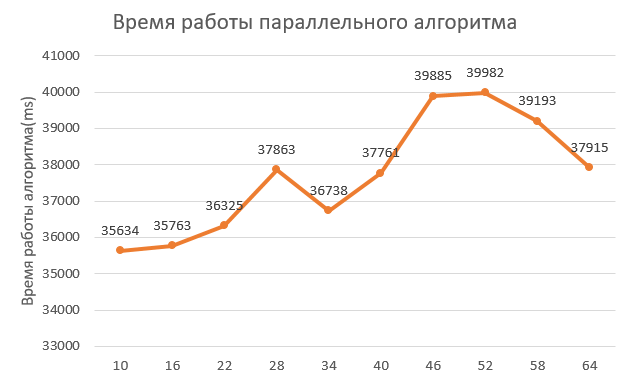
Таблица 2 – Результаты работы алгоритмов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во потоков | Кол-во  тел | Время работы последова-тельного алгоритма 𝑇П, мс | Время работы паралле-льного алгоритма 𝑇||, мс |  |
| 10 | 5\*10^2 | 6\*10^4 | 35634 | 1,684 |
| 16 | 35763 | 1,678 |
| 22 | 36325 | 1,652 |
| 28 | 37863 | 1,585 |
| 34 | 36738 | 1,633 |
| 40 | 37761 | 1,589 |
| 46 | 39885 | 1,504 |
| 52 | 39982 | 1,5 |
| 58 | 39193 | 1,531 |
| 64 | 37915 | 1,582 |

Полученные данные можно представить в виде графиков.

На рисунке 2 представлен график зависимости времени работы параллельного алгоритма от размера файла и количества строк.

Рисунок 2 – Зависимость времени работы параллельного алгоритма от размера файла и количества строк



На рисунке 3 представлен график ускорения работы программы.



Рисунок 3 – Ускорение работы программы

1. Заключение

По полученным данным видно, что затраченное время на работу алгоритма почти не различается в параллельном алгоритме. Из полученных данных видно, что параллельный алгоритм во всех случаях работает быстрее. Наибольшего ускорения программа достигает, когда количество потоков равно 10.

Параллельная программа работает быстрее чем последовательная, за счет того, что в параллельном алгоритме идет распределение подсчета координат (movingCallables) и сил (recalcingCallables) (разделенных на потоки), которое в свою очередь, происходит в реальном времени, за счет изменения данных (пока тела движутся и взаимодействуют друг с другом).

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Многопоточность в Java/Хабр. [Электронный ресурс] URL: https://habr.com/ru/post/164487/ (дата обращения: 23.11.2021).
2. Поиск и проверка простых чисел на Java. [Электронный ресурс] URL: https://javascopes.com/find-check-prime-numbers-in-java-81de4b4e/ (дата обращения: 12.11.2021).
3. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления СПб.:БХВ-Петербург, 2002. - 608 с.
4. Грегори Р. Эндрюс Основы многопоточного, параллельного и распределенного программирования - Вильямс, 2002. - 512 с.

ПРИЛОЖЕНИЕ А

Последовательный алгоритм

А.1 Код класса App

import nbody.\*;  
import nbodygui.Frame;  
import nbodygui.Frames;  
import nbodygui.Panel;  
import nbodygui.Panels;  
import java.awt.\*;  
import java.io.IOException;  
import java.util.Random;  
public class App {  
 public static void main(String[] args) throws IOException {  
 final AppSettingsParser parser = new AppSettingsParser();  
 final int parsedWidth = (parser.width() == null) ? Frames.*DEFAULT\_WIDTH* : parser.width();  
 final int parsedHeight = (parser.height() == null) ? Frames.*DEFAULT\_HEIGHT* : parser.height();  
 final int parsedBodiesNum = (parser.bodiesNum() == null) ? NbodySolvers.*DEFAULT\_BODIES\_NUM* : parser.bodiesNum();  
 final double parsedBodyMass = (parser.bodyMass() == null) ? Bodies.*DEFAULT\_BODY\_MASS* : parser.bodyMass();  
 final int parsedDeltaTime = (parser.deltaTime() == null) ? NbodySolvers.*DEFAULT\_DELTA\_TIME* : parser.deltaTime();  
 final double parsedErrorDistance = (parser.errorDistance() == null) ? NbodySolvers.*DEFAULT\_ERROR\_DISTANCE* : parser.errorDistance();  
 final int parsedDurationMillis = (parser.durationMillis() == null) ? Panels.*DEFAULT\_DURATION\_MILLIS* : parser.durationMillis();  
 final Dimension coordsBounds = new Dimension(parsedWidth - 100, parsedHeight - 100);  
 final Coords[] randomCoordsArr = *randomCoordsArr*(parsedBodiesNum, coordsBounds);  
 final NbodySolver solver = new NbodySolver(randomCoordsArr, parsedBodyMass, parsedDeltaTime, parsedErrorDistance);  
 final Panel panel = new Panel(solver, parsedDurationMillis);  
 final Frame frame = new Frame(parsedWidth, parsedHeight, panel);  
 frame.setVisible(true);  
 }  
 public static Coords[] randomCoordsArr(int num, Dimension coordsBounds) {  
 final Coords[] randomCoordsArr = new Coords[num];  
 int x, y;  
 final Random random = new Random();  
 for (int i = 0; i < randomCoordsArr.length; i++) {  
 x = Math.*abs*(random.nextInt()) % coordsBounds.width;  
 y = Math.*abs*(random.nextInt()) % coordsBounds.height;  
 randomCoordsArr[i] = new Coords(x, y);  
 }  
 return randomCoordsArr;  
 }  
  
 public static Body[] pseudoEarthAndMoon(double x, double y) {  
 Body earth = new Body(new Coords(x, y), 6e10);  
 Body moon = new Body(new Coords(earth.p().x(), earth.p().y() + 300), 6e4, new Coords(0.02, 0.0));  
 return new Body[]{earth, moon};  
 }  
}

А.2 Код класса AppSettings

public class AppSettings {  
 public final int width;  
 public final int height;  
 public final int bodiesNum;  
 public final double bodyMass;  
 public final int deltaTime;  
 public final double errorDistance;  
 public final int durationMillis;  
 public final int threadsNum;  
 public AppSettings(  
 int width,  
 int height,  
 int bodiesNum,  
 double bodyMass,  
 int deltaTime,  
 double errorDistance,  
 int durationMillis,  
 int threadsNum  
 ) {  
 this.width = width;  
 this.height = height;  
 this.bodiesNum = bodiesNum;  
 this.bodyMass = bodyMass;  
 this.deltaTime = deltaTime;  
 this.errorDistance = errorDistance;  
 this.durationMillis = durationMillis;  
 this.threadsNum = threadsNum;  
 }  
}

А.3 Код класса AppSettingsParser

import java.io.IOException;  
import java.nio.file.Files;  
import java.nio.file.Path;  
public class AppSettingsParser {  
 private static final String *WIDTH\_TAG\_NAME* = "width";  
 private static final String *HEIGHT\_TAG\_NAME* = "height";  
 private static final String *BODIES\_NUM\_TAG\_NAME* = "bodiesNum";  
 private static final String *BODY\_MASS\_TAG\_NAME* = "bodyMass";  
 private static final String *DELTA\_TIME\_TAG\_NAME* = "deltaTime";  
 private static final String *ERROR\_DISTANCE\_TAG\_NAME* = "errorDistance";  
 private static final String *DURATION\_SECONDS\_TAG\_NAME* = "durationSeconds";  
 private final String settingsXml;  
 public AppSettingsParser() throws IOException {  
 settingsXml = new String(Files.*readAllBytes*(Path.*of*("settings.xml")));  
 }  
 public Integer width() {  
 Integer width;  
 try {  
 width = Integer.*parseInt*(value(*WIDTH\_TAG\_NAME*));  
 } catch (NumberFormatException e) {  
 width = null;  
 }  
 return width;  
 }  
  
 public Integer height() {  
 Integer height;  
 try {  
 height = Integer.*parseInt*(value(*HEIGHT\_TAG\_NAME*));  
 } catch (NumberFormatException e) {  
 height = null;  
 }  
 return height;  
 }  
 public Integer bodiesNum() {  
 Integer bodiesNum;  
 try {  
 bodiesNum = Integer.*parseInt*(value(*BODIES\_NUM\_TAG\_NAME*));  
 } catch (NumberFormatException e) {  
 bodiesNum = null;  
 }  
 return bodiesNum;  
 }  
 public Double bodyMass() {  
 Double bodyMass;  
 try {  
 bodyMass = Double.*parseDouble*(value(*BODY\_MASS\_TAG\_NAME*));  
 } catch (NumberFormatException | NullPointerException e) {  
 bodyMass = null;  
 }  
 return bodyMass;  
 }  
 public Integer deltaTime() {  
 Integer deltaTime;  
 try {  
 deltaTime = Integer.*parseInt*(value(*DELTA\_TIME\_TAG\_NAME*));  
 } catch (NumberFormatException e) {  
 deltaTime = null;  
 }  
 return deltaTime;  
 }  
 public Double errorDistance() {  
 Double errorDistance;  
 try {  
 errorDistance = Double.*parseDouble*(value(*ERROR\_DISTANCE\_TAG\_NAME*));  
 } catch (NumberFormatException | NullPointerException e) {  
 errorDistance = null;  
 }  
 return errorDistance;  
 }  
  
 public Integer durationMillis() {  
 Integer durationMillis;  
 try {  
 durationMillis = Integer.*parseInt*(value(*DURATION\_SECONDS\_TAG\_NAME*)) \* 1000;  
 } catch (NumberFormatException e) {  
 durationMillis = null;  
 }  
 return durationMillis;  
 }  
  
 private String value(String tagName) {  
 String[] left = settingsXml.split("<" + tagName + ">");  
 if (left.length < 2) {  
 return null;  
 }  
  
 String[] leftAndRight = left[1].split("</" + tagName + ">");  
 if (leftAndRight.length < 1) {  
 return null;  
 }  
  
 return leftAndRight[0];  
 }  
}

А.4 package nbody

А.4.1 package nbody.exceptions

А.4.1.1 Код класса BodiesNumOutOfBoundsException

package nbody.exceptions;  
public class BodiesNumOutOfBoundsException extends IndexOutOfBoundsException {  
}

А.4.1.2 Код класса BodyMassOutOfBoundsException

package nbody.exceptions;  
public class BodyMassOutOfBoundsException extends IndexOutOfBoundsException {  
}

А.4.1.3 Код класса DeltaTimeOutOfBoundsException

package nbody.exceptions;  
public class DeltaTimeOutOfBoundsException extends IndexOutOfBoundsException {  
}

А.4.1.4 Код класса ErrorDistanceOutOfBoundsException

package nbody.exceptions;  
public class ErrorDistanceOutOfBoundsException extends IndexOutOfBoundsException {  
}

А.4.1.5 Код класса ThreadsNumOutOfBoundsException

package nbody.exceptions;  
public class ThreadsNumOutOfBoundsException extends IndexOutOfBoundsException {  
}

А.4.2 Код класса Bodies

package nbody;  
public class Bodies {  
 public static final double G = 6.67e-11;  
 public static final double MIN\_BODY\_MASS = 1e4;  
 public static final double MAX\_BODY\_MASS = 9e15;  
 public static final double DEFAULT\_BODY\_MASS = 1e10;  
}

А.4.3 Код класса Body

package nbody;  
import nbody.exceptions.BodyMassOutOfBoundsException;  
import static nbody.Bodies.MAX\_BODY\_MASS;  
import static nbody.Bodies.MIN\_BODY\_MASS;  
  
public class Body {  
  
 private final double m;  
 private final Coords p;  
 private final Coords v;  
 private final Coords f;  
  
 public Body(Coords xyCoords, double mass) {  
 if (mass < MIN\_BODY\_MASS || mass > MAX\_BODY\_MASS) {  
 throw new BodyMassOutOfBoundsException();  
 }  
 m = mass;  
 p = xyCoords.clone();  
 v = new Coords(0.0, 0.0);  
 f = new Coords(0.0, 0.0);  
 }  
 public Body(Coords xyCoords, double mass, Coords v) {  
 if (mass < MIN\_BODY\_MASS || mass > MAX\_BODY\_MASS) {  
 throw new BodyMassOutOfBoundsException();  
 }  
 m = mass;  
 p = xyCoords.clone();  
 this.v = v.clone();  
 f = new Coords(0.0, 0.0);  
 }  
 public Coords p() {  
 return p;  
 }  
 public Coords v() {  
 return v;  
 }  
 public Coords f() {  
 return f;  
 }  
 public void setP(double x, double y) {  
 p.set(x, y);  
 }  
 public void setV(double x, double y) {  
 v.set(x, y);  
 }  
 public void setF(double x, double y) {  
 f.set(x, y);  
 }  
 public double m() {  
 return m;  
 }  
}

А.4.4 Код класса Coords

package nbody;  
  
public class Coords {  
 private double x;  
 private double y;  
 public Coords(double x, double y) {  
 this.x = x;  
 this.y = y;  
 }  
 public double x() {  
 return x;  
 }  
 public double y() {  
 return y;  
 }  
 public void set(double x, double y) {  
 this.x = x;  
 this.y = y;  
 }  
 public Coords clone() {  
 return new Coords(x, y);  
 }  
}

А.4.5 Код класса Helpers

package nbody;  
  
public class Helpers {  
 public static int[][] ranges(int startNum, int endNum, int rangesNum) {  
 float lowLimit = startNum;  
 float partition = (endNum - lowLimit) / rangesNum; // n is the number partitions of the range [a,b]  
 if (partition < 1.0) {  
 throw new IllegalArgumentException();  
 }  
 int[][] ranges = new int[rangesNum][2];  
 ranges[0][0] = Math.round(lowLimit);  
 for (int rowIndex = 0; rowIndex < rangesNum - 1; rowIndex++) {  
 lowLimit += partition;  
 ranges[rowIndex][1] = Math.round(lowLimit);  
 ranges[rowIndex + 1][0] = ranges[rowIndex][1] + 1;  
 }  
 ranges[rangesNum - 1][1] = endNum;  
 return ranges;  
 }  
}

А.4.6 Код класса NbodySettings

package nbody;  
  
import nbody.exceptions.BodyMassOutOfBoundsException;  
import nbody.exceptions.DeltaTimeOutOfBoundsException;  
import nbody.exceptions.ErrorDistanceOutOfBoundsException;  
import nbody.exceptions.ThreadsNumOutOfBoundsException;  
import static nbody.NbodySolvers.\*;

public class NbodySettings {  
 public final double bodyMass;  
 public final int deltaTime;  
 public final double errorDistance;  
 public final int threadsNum;  
 public NbodySettings(  
 double bodyMass,  
 int deltaTime,  
 double errorDistance,  
 int threadsNum  
 ) {  
  
 if (bodyMass < Bodies.MIN\_BODY\_MASS || bodyMass > Bodies.MAX\_BODY\_MASS) {  
 throw new BodyMassOutOfBoundsException();  
 }  
 if (deltaTime < MIN\_DELTA\_TIME || deltaTime > MAX\_DELTA\_TIME) {  
 throw new DeltaTimeOutOfBoundsException();  
 }  
 if (errorDistance < MIN\_ERROR\_DISTANCE || errorDistance > MAX\_ERROR\_DISTANCE) {  
 throw new ErrorDistanceOutOfBoundsException();  
 }  
 this.bodyMass = bodyMass;  
 this.deltaTime = deltaTime;  
 this.errorDistance = errorDistance;  
 this.threadsNum = threadsNum;  
 }  
  
 public NbodySettings(  
 int deltaTime,  
 double errorDistance,  
 int threadsNum  
 ) {  
 bodyMass = Bodies.DEFAULT\_BODY\_MASS;  
  
 if (deltaTime < MIN\_DELTA\_TIME || deltaTime > MAX\_DELTA\_TIME) {  
 throw new DeltaTimeOutOfBoundsException();  
 }  
 if (errorDistance < MIN\_ERROR\_DISTANCE || errorDistance > MAX\_ERROR\_DISTANCE) {  
 throw new ErrorDistanceOutOfBoundsException();  
 }  
 this.deltaTime = deltaTime;  
 this.errorDistance = errorDistance;  
 this.threadsNum = threadsNum;  
 }  
}

А.4.7 Код класса NbodySolver

package nbody;  
  
import nbody.exceptions.BodiesNumOutOfBoundsException;  
import nbody.exceptions.DeltaTimeOutOfBoundsException;  
import nbody.exceptions.ErrorDistanceOutOfBoundsException;  
import static nbody.NbodySolvers.\*;  
  
public class NbodySolver {  
 private final Body[] b;  
 private final int dt;  
 private final double errorDistance;  
 public NbodySolver(Coords[] bodiesCoords, double bodyMass, int deltaTime, double errorDistance) {  
  
 if (bodiesCoords.length < MIN\_BODIES\_NUM || bodiesCoords.length > MAX\_BODIES\_NUM) {  
 throw new BodiesNumOutOfBoundsException();  
 }  
 if (deltaTime < MIN\_DELTA\_TIME || deltaTime > MAX\_DELTA\_TIME) {  
 throw new DeltaTimeOutOfBoundsException();  
 }  
 if (errorDistance < MIN\_ERROR\_DISTANCE || errorDistance > MAX\_ERROR\_DISTANCE) {  
 throw new ErrorDistanceOutOfBoundsException();  
 }  
 b = new Body[bodiesCoords.length];  
 for (int i = 0; i < b.length; i++) {  
 b[i] = new Body(bodiesCoords[i], bodyMass);  
 }  
 dt = deltaTime;  
 this.errorDistance = errorDistance;  
 }  
 public NbodySolver(Body[] b, int deltaTime, double errorDistance) {  
  
 if (b.length < MIN\_BODIES\_NUM || b.length > MAX\_BODIES\_NUM) {  
 throw new BodiesNumOutOfBoundsException();  
 }  
 if (deltaTime < MIN\_DELTA\_TIME || deltaTime > MAX\_DELTA\_TIME) {  
 throw new DeltaTimeOutOfBoundsException();  
 }  
 if (errorDistance < MIN\_ERROR\_DISTANCE || errorDistance > MAX\_ERROR\_DISTANCE) {  
 throw new ErrorDistanceOutOfBoundsException();  
 }  
 this.b = b.clone();  
  
 dt = deltaTime;  
 this.errorDistance = errorDistance;  
 }  
 public int n() {  
 return b.length;  
 }  
 public int dt() {  
 return dt;  
 }  
 public int bodyX(int index) {  
 return (int) b[index].p().x();  
 }  
 public int bodyY(int index) {  
 return (int) b[index].p().y();  
 }  
 public void recalcBodiesCoords() {  
 recalcBodiesForces();  
 moveNBodies();  
 }  
 private void recalcBodiesForces() {  
 double distance;  
 double magnitude;  
 Coords direction;  
  
 final int n = b.length;  
 for (int k = 0; k < n - 1; k++) {  
 for (int l = k + 1; l < n; l++) {  
 distance = distance(b[k], b[l]);  
 magnitude = (distance < errorDistance) ? 0.0 : magnitude(b[k], b[l], distance);  
 direction = direction(b[k], b[l]);  
 b[k].setF(  
 b[k].f().x() + magnitude \* direction.x() / distance,  
 b[k].f().y() + magnitude \* direction.y() / distance  
 );  
 b[l].setF(  
 b[l].f().x() - magnitude \* direction.x() / distance,  
 b[l].f().y() - magnitude \* direction.y() / distance  
 );  
 }  
 }  
 }  
 private void moveNBodies() {  
 Coords dv; // dv = f/m \* dt  
 Coords dp; // dp = (v + dv/2) \* dt  
 for (Body body : b) {  
 dv = dv(body, dt);  
 dp = dp(body, dt, dv);  
 body.setV(  
 body.v().x() + dv.x(),  
 body.v().y() + dv.y()  
 );  
 body.setP(  
 body.p().x() + dp.x(),  
 body.p().y() + dp.y()  
 );  
 body.setF(0.0, 0.0);  
 }  
 }  
}

А.4.8 Код класса NbodySolvers

package nbody;  
  
import static nbody.Bodies.G;  
public class NbodySolvers {  
  
 public static final int MIN\_BODIES\_NUM = 1;  
 public static final int MAX\_BODIES\_NUM = 1024;  
 public static final int DEFAULT\_BODIES\_NUM = 4;  
 public static final int MIN\_DELTA\_TIME = 16;  
 public static final int MAX\_DELTA\_TIME = 128;  
 public static final int DEFAULT\_DELTA\_TIME = MIN\_DELTA\_TIME  
 public static final double MIN\_ERROR\_DISTANCE = 1e2;  
 public static final double MAX\_ERROR\_DISTANCE = 1e3;  
 public static final double DEFAULT\_ERROR\_DISTANCE = MIN\_ERROR\_DISTANCE;  
 private static final Coords optdMemAllocDirection = new Coords(0.0, 0.0);  
 private static final Coords optdMemAllocDv = new Coords(0.0, 0.0);  
 private static final Coords optdMemAllocDp = new Coords(0.0, 0.0);  
 public static double distance(Body b1, Body b2) {  
 return Math.sqrt(  
 Math.pow(b1.p().x() - b2.p().x(), 2) + Math.pow(b1.p().y() - b2.p().y(), 2)  
 );  
 }  
 public static double magnitude(Body b1, Body b2, double b1b2distance) {  
 return G \* b1.m() \* b2.m() / Math.pow(b1b2distance, 2);  
 }  
 public static Coords direction(Body b1, Body b2) {  
 optdMemAllocDirection.set(  
 b2.p().x() - b1.p().x(),  
 b2.p().y() - b1.p().y());  
 return optdMemAllocDirection;  
 }  
 public static Coords dv(Body b, long dt) {  
 optdMemAllocDv.set(  
 b.f().x() / b.m() \* dt,  
 b.f().y() / b.m() \* dt);  
 return optdMemAllocDv;  
 }  
 public static Coords dp(Body b, long dt, Coords dv) {  
 optdMemAllocDp.set(  
 (b.v().x() + dv.x() / 2) \* dt,  
 (b.v().y() + dv.y() / 2) \* dt  
 );  
  
 return optdMemAllocDp;  
 }  
}

А.5 package nbodygui

А.5.1 package nbodygui.exceptions

А.5.1.1 Код класса DurationMillisOutOfBoundsException

package nbodygui.exceptions;  
public class DurationMillisOutOfBoundsException extends IndexOutOfBoundsException {  
}

А.5.1.2 Код класса HeightOutOfBoundsException

package nbodygui.exceptions;  
public class HeightOutOfBoundsException extends IndexOutOfBoundsException {  
}

А.5.1.3 Код класса WidthOutOfBoundsException

package nbodygui.exceptions;  
public class WidthOutOfBoundsException extends IndexOutOfBoundsException {  
}

А.5.2 Код класса Frame

package nbodygui;

import nbodygui.exceptions.HeightOutOfBoundsException;  
import nbodygui.exceptions.WidthOutOfBoundsException;  
import javax.swing.\*;  
import java.awt.event.WindowAdapter;  
import java.awt.event.WindowEvent;  
import static nbodygui.Frames.\*;  
  
public class Frame extends JFrame {  
 public Frame(int width, int height, Panel panel) {  
 super(DEFAULT\_TITLE);  
 if (width < MIN\_WIDTH || width > MAX\_WIDTH) {  
 throw new WidthOutOfBoundsException();  
 }  
 if (height < MIN\_HEIGHT || height > MAX\_HEIGHT) {  
 throw new HeightOutOfBoundsException();  
 }  
 add(panel);  
 addWindowListener(new WindowAdapter() {  
 @Override  
 public void windowClosing(WindowEvent e) {  
 Timer timer = panel.timer();  
 timer.stop();  
 }  
 });  
 setSize(width, height);  
 setResizable(false);  
 setLocationRelativeTo(null);  
 setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  
 }  
}

А.5.3 Код класса Frames

package nbodygui;  
  
public class Frames {  
 public static final String DEFAULT\_TITLE = "Движение тел";  
 public static final int MIN\_WIDTH = 200;  
 public static final int MAX\_WIDTH = 2000;  
 public static final int DEFAULT\_WIDTH = 500;  
 public static final int MIN\_HEIGHT = 200;  
 public static final int MAX\_HEIGHT = 2000;  
 public static final int DEFAULT\_HEIGHT = 500;  
}

А.5.4 Код класса Panel

package nbodygui;  
  
import nbody.NbodySolver;  
import nbodygui.exceptions.DurationMillisOutOfBoundsException;  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import static nbodygui.Panels.\*;  
public class Panel extends JPanel implements ActionListener {  
  
 private final NbodySolver solver;  
 private final Timer timer;  
 private final int durationMillis;  
 private int consumedMillis;  
  
 public Panel(NbodySolver solver, int durationMillis) {  
 if (durationMillis < MIN\_DURATION\_MILLIS || durationMillis > MAX\_DURATION\_MILLIS) {  
 throw new DurationMillisOutOfBoundsException();  
 }  
 this.solver = solver;  
 timer = new Timer(solver.dt(), this);  
 timer.start();  
 consumedMillis = 0;  
 this.durationMillis = durationMillis;  
 }  
 public Timer timer() {  
 return timer;  
 }  
 private void drawRandomPoints(Graphics gr) {  
 Graphics2D graphics = (Graphics2D) gr;  
 graphics.setPaint(DEFAULT\_POINTS\_COLOR);  
 for (int i = 0; i < solver.n(); i++) {  
 int x = solver.bodyX(i);  
 int y = solver.bodyY(i);  
 graphics.fillOval(x, y, DEFAULT\_POINTS\_SIZE, DEFAULT\_POINTS\_SIZE);  
 }  
 }  
 @Override  
 public void paintComponent(Graphics gr) {  
 super.paintComponent(gr);  
 solver.recalcBodiesCoords();  
 drawRandomPoints(gr);  
 }  
 @Override  
 public void actionPerformed(ActionEvent actionEvent) {  
 if (consumedMillis >= durationMillis) {  
 timer.stop();  
 System.out.println(consumedMillis);  
 return;  
 }  
 consumedMillis += timer.getDelay();  
 repaint();  
 }  
}

А.5.5 Код класса Panels

package nbodygui;  
  
import java.awt.\*;  
public class Panels {  
 public static final Color DEFAULT\_POINTS\_COLOR = Color.GREEN;  
 public static final int DEFAULT\_POINTS\_SIZE = 15;  
 public static final int MIN\_DURATION\_MILLIS = 10\_000;  
 public static final int MAX\_DURATION\_MILLIS = 3\_600\_000;  
 public static final int DEFAULT\_DURATION\_MILLIS = MIN\_DURATION\_MILLIS;  
}

ПРИЛОЖЕНИЕ Б

Параллельный алгоритм

Б.1 Код класса App

import nbody.Coords;  
import nbody.NbodySettings;  
import nbody.NbodySolver;  
import nbodygui.Frame;  
import nbodygui.Panel;  
import java.awt.\*;  
import java.io.IOException;  
import java.util.Random;  
  
public class App {  
 public static void main(String[] args) throws IOException {  
 final AppSettingsParser parser = new AppSettingsParser();  
 final AppSettings settings = parser.parseSettings();  
 final Dimension coordsBounds = new Dimension(settings.width - 100, settings.height - 100);  
 final Coords[] randomCoordsArr = randomCoordsArr(settings.bodiesNum, coordsBounds);  
 final NbodySettings nbodySettings = new NbodySettings(settings.bodyMass, settings.deltaTime, settings.errorDistance, settings.threadsNum);  
 final NbodySolver solver = new NbodySolver(randomCoordsArr, nbodySettings);  
 final Panel panel = new Panel(solver, settings.durationMillis);  
 final Frame frame = new Frame(settings.width, settings.height, panel);  
 frame.setVisible(true);  
 long startMillis = System.currentTimeMillis();  
 panel.start();  
 long finishMillis = System.currentTimeMillis();  
 long consumedMillis = finishMillis - startMillis;  
 System.out.println("Consumed millis: " + consumedMillis);  
 }  
 public static Coords[] randomCoordsArr(int num, Dimension coordsBounds) {  
  
 final Coords[] randomCoordsArr = new Coords[num];  
 int x, y;  
 final Random random = new Random();  
 for (int i = 0; i < randomCoordsArr.length; i++) {  
 x = Math.abs(random.nextInt()) % coordsBounds.width;  
 y = Math.abs(random.nextInt()) % coordsBounds.height;  
 randomCoordsArr[i] = new Coords(x, y);  
 }  
 return randomCoordsArr;  
 }  
}

Б.2 Код класса AppSettings

public class AppSettings {  
 public final int width;  
 public final int height;  
 public final int bodiesNum;  
 public final double bodyMass;  
 public final int deltaTime;  
 public final double errorDistance;  
 public final int durationMillis;  
 public final int threadsNum;  
 public AppSettings(  
 int width,  
 int height,  
 int bodiesNum,  
 double bodyMass,  
 int deltaTime,  
 double errorDistance,  
 int durationMillis,  
 int threadsNum  
 ) {  
 this.width = width;  
 this.height = height;  
 this.bodiesNum = bodiesNum;  
 this.bodyMass = bodyMass;  
 this.deltaTime = deltaTime;  
 this.errorDistance = errorDistance;  
 this.durationMillis = durationMillis;  
 this.threadsNum = threadsNum;  
 }  
}

Б.3 Код класса AppSettingsParser

import nbody.Bodies;  
import nbody.NbodySolvers;  
import nbodygui.Frames;  
import nbodygui.Panels;  
  
import java.io.IOException;  
import java.nio.file.Files;  
import java.nio.file.Path;  
  
public class AppSettingsParser {  
  
 private static final String WIDTH\_TAG\_NAME = "width";  
 private static final String HEIGHT\_TAG\_NAME = "height";  
 private static final String BODIES\_NUM\_TAG\_NAME = "bodiesNum";  
 private static final String BODY\_MASS\_TAG\_NAME = "bodyMass";  
 private static final String DELTA\_TIME\_TAG\_NAME = "deltaTime";  
 private static final String ERROR\_DISTANCE\_TAG\_NAME = "errorDistance";  
 private static final String DURATION\_SECONDS\_TAG\_NAME = "durationSeconds";  
 private static final String THREADS\_NUM\_TAG\_NAME = "threadsNum";  
  
 private final String settingsXml;  
  
 public AppSettingsParser() throws IOException {  
 settingsXml = new String(Files.readAllBytes(Path.of("settings.xml")));  
 }  
  
 public Integer width() {  
 Integer width;  
 try {  
 width = Integer.parseInt(value(WIDTH\_TAG\_NAME));  
 } catch (NumberFormatException e) {  
 width = null;  
 }  
 return width;  
 }  
  
 public Integer height() {  
 Integer height;  
 try {  
 height = Integer.parseInt(value(HEIGHT\_TAG\_NAME));  
 } catch (NumberFormatException e) {  
 height = null;  
 }  
 return height;  
 }  
  
 public Integer bodiesNum() {  
 Integer bodiesNum;  
 try {  
 bodiesNum = Integer.parseInt(value(BODIES\_NUM\_TAG\_NAME));  
 } catch (NumberFormatException e) {  
 bodiesNum = null;  
 }  
 return bodiesNum;  
 }  
  
 public Double bodyMass() {  
 Double bodyMass;  
 try {  
 bodyMass = Double.parseDouble(value(BODY\_MASS\_TAG\_NAME));  
 } catch (NumberFormatException | NullPointerException e) {  
 bodyMass = null;  
 }  
 return bodyMass;  
 }  
  
 public Integer deltaTime() {  
 Integer deltaTime;  
 try {  
 deltaTime = Integer.parseInt(value(DELTA\_TIME\_TAG\_NAME));  
 } catch (NumberFormatException e) {  
 deltaTime = null;  
 }  
 return deltaTime;  
 }  
  
 public Double errorDistance() {  
 Double errorDistance;  
 try {  
 errorDistance = Double.parseDouble(value(ERROR\_DISTANCE\_TAG\_NAME));  
 } catch (NumberFormatException | NullPointerException e) {  
 errorDistance = null;  
 }  
 return errorDistance;  
 }  
  
 public Integer durationMillis() {  
 Integer durationMillis;  
 try {  
 durationMillis = Integer.parseInt(value(DURATION\_SECONDS\_TAG\_NAME)) \* 1000;  
 } catch (NumberFormatException e) {  
 durationMillis = null;  
 }  
 return durationMillis;  
 }  
  
 public Integer threadsNum() {  
 Integer threadsNum;  
 try {  
 threadsNum = Integer.parseInt(value(THREADS\_NUM\_TAG\_NAME));  
 } catch (NumberFormatException e) {  
 threadsNum = null;  
 }  
 return threadsNum;  
 }  
  
 private String value(String tagName) {  
 String[] left = settingsXml.split("<" + tagName + ">");  
 if (left.length < 2) {  
 return null;  
 }  
  
 String[] leftAndRight = left[1].split("</" + tagName + ">");  
 if (leftAndRight.length < 1) {  
 return null;  
 }  
  
 return leftAndRight[0];  
 }  
  
 public AppSettings parseSettings() {  
 final int parsedWidth = (width() == null) ? Frames.DEFAULT\_WIDTH : width();  
 final int parsedHeight = (height() == null) ? Frames.DEFAULT\_HEIGHT : height();  
 final int parsedBodiesNum = (bodiesNum() == null) ? NbodySolvers.DEFAULT\_BODIES\_NUM : bodiesNum();  
 final double parsedBodyMass = (bodyMass() == null) ? Bodies.DEFAULT\_BODY\_MASS : bodyMass();  
 final int parsedDeltaTime = (deltaTime() == null) ? NbodySolvers.DEFAULT\_DELTA\_TIME : deltaTime();  
 final double parsedErrorDistance = (errorDistance() == null) ? NbodySolvers.DEFAULT\_ERROR\_DISTANCE : errorDistance();  
 final int parsedDurationMillis = (durationMillis() == null) ? Panels.DEFAULT\_DURATION\_MILLIS : durationMillis();  
 final int parsedThreadsNum = (threadsNum() == null) ? NbodySolvers.DEFAULT\_THREADS\_NUM : threadsNum();  
  
 return new AppSettings(  
 parsedWidth,  
 parsedHeight,  
 parsedBodiesNum,  
 parsedBodyMass,  
 parsedDeltaTime,  
 parsedErrorDistance,  
 parsedDurationMillis,  
 parsedThreadsNum  
 );  
 }  
}

Б.4 package nbody

Б.4.1 package nbody.exceptions

Б.4.1.1 Код класса BodiesNumOutOfBoundsException

package nbody.exceptions;  
public class BodiesNumOutOfBoundsException extends IndexOutOfBoundsException {  
}

Б.4.1.2 Код класса BodyMassOutOfBoundsException

package nbody.exceptions;  
public class BodyMassOutOfBoundsException extends IndexOutOfBoundsException {  
}

Б.4.1.3 Код класса DeltaTimeOutOfBoundsException

package nbody.exceptions;  
public class DeltaTimeOutOfBoundsException extends IndexOutOfBoundsException {  
}

Б.4.1.4 Код класса ErrorDistanceOutOfBoundsException

package nbody.exceptions;  
public class ErrorDistanceOutOfBoundsException extends IndexOutOfBoundsException {  
}

Б.4.1.5 Код класса ThreadsNumOutOfBoundsException

package nbody.exceptions;  
public class ThreadsNumOutOfBoundsException extends IndexOutOfBoundsException {  
}

Б.4.2 Bodies

package nbody;  
  
public class Bodies {  
  
 public static final double G = 6.67e-11;  
  
 public static final double MIN\_BODY\_MASS = 1e4;  
 public static final double MAX\_BODY\_MASS = 9e15;  
 public static final double DEFAULT\_BODY\_MASS = 1e10;  
}

Б.4.3 Body

package nbody;  
  
import nbody.exceptions.BodyMassOutOfBoundsException;  
import static nbody.Bodies.MAX\_BODY\_MASS;  
import static nbody.Bodies.MIN\_BODY\_MASS;  
  
public class Body {  
 private final double m;  
 private final Coords p;  
 private final Coords v;  
 private final Coords f;  
 public Body(Coords xyCoords, double mass) {  
 if (mass < MIN\_BODY\_MASS || mass > MAX\_BODY\_MASS) {  
 throw new BodyMassOutOfBoundsException();  
 }  
  
 m = mass;  
 p = xyCoords.clone();  
 v = new Coords(0.0, 0.0);  
 f = new Coords(0.0, 0.0);  
 }  
  
 public Body(Coords xyCoords, double mass, Coords v) {  
 if (mass < MIN\_BODY\_MASS || mass > MAX\_BODY\_MASS) {  
 throw new BodyMassOutOfBoundsException();  
 }  
 m = mass;  
 p = xyCoords.clone();  
 this.v = v.clone();  
 f = new Coords(0.0, 0.0);  
 }  
 public Coords p() {  
 return p;  
 }  
 public Coords v() {  
 return v;  
 }  
 public Coords f() {  
 return f;  
 }  
 public void setP(double x, double y) {  
 p.set(x, y);  
 }  
 public void setV(double x, double y) {  
 v.set(x, y);  
 }  
 public void setF(double x, double y) {  
 f.set(x, y);  
 }  
 public double m() {  
 return m;  
 }  
}

Б.4.4 Coords

package nbody;  
  
public class Coords {  
  
 private double x;  
 private double y;  
 public Coords(double x, double y) {  
 this.x = x;  
 this.y = y;  
 }  
 public double x() {  
 return x;  
 }  
 public double y() {  
 return y;  
 }  
 public void set(double x, double y) {  
 this.x = x;  
 this.y = y;  
 }  
 public Coords clone() {  
 return new Coords(x, y);  
 }  
}

Б.4.5 Helpers

package nbody;  
  
public class Helpers {  
  
 public static int[][] ranges(int startNum, int endNum, int rangesNum) {  
  
 float lowLimit = startNum;  
 float partition = (endNum - lowLimit) / rangesNum; // n is the number partitions of the range [a,b]  
 if (partition < 1.0) {  
 throw new IllegalArgumentException();  
 }  
  
 int[][] ranges = new int[rangesNum][2];  
  
 ranges[0][0] = Math.round(lowLimit);  
 for (int rowIndex = 0; rowIndex < rangesNum - 1; rowIndex++) {  
 lowLimit += partition;  
 ranges[rowIndex][1] = Math.round(lowLimit);  
 ranges[rowIndex + 1][0] = ranges[rowIndex][1] + 1;  
 }  
 ranges[rangesNum - 1][1] = endNum;  
  
 return ranges;  
 }  
}

Б.4.6 NbodySettings

package nbody;  
  
import nbody.exceptions.BodyMassOutOfBoundsException;  
import nbody.exceptions.DeltaTimeOutOfBoundsException;  
import nbody.exceptions.ErrorDistanceOutOfBoundsException;  
import nbody.exceptions.ThreadsNumOutOfBoundsException;  
  
import static nbody.NbodySolvers.\*;  
  
public class NbodySettings {  
  
 public final double bodyMass;  
 public final int deltaTime;  
 public final double errorDistance;  
 public final int threadsNum;  
  
 public NbodySettings(  
 double bodyMass,  
 int deltaTime,  
 double errorDistance,  
 int threadsNum  
 ) {  
  
 if (bodyMass < Bodies.MIN\_BODY\_MASS || bodyMass > Bodies.MAX\_BODY\_MASS) {  
 throw new BodyMassOutOfBoundsException();  
 }  
  
 if (deltaTime < MIN\_DELTA\_TIME || deltaTime > MAX\_DELTA\_TIME) {  
 throw new DeltaTimeOutOfBoundsException();  
 }  
  
 if (errorDistance < MIN\_ERROR\_DISTANCE || errorDistance > MAX\_ERROR\_DISTANCE) {  
 throw new ErrorDistanceOutOfBoundsException();  
 }  
  
 if (threadsNum < MIN\_THREADS\_NUM || threadsNum > MAX\_THREADS\_NUM) {  
 throw new ThreadsNumOutOfBoundsException();  
 }  
  
 this.bodyMass = bodyMass;  
 this.deltaTime = deltaTime;  
 this.errorDistance = errorDistance;  
 this.threadsNum = threadsNum;  
 }  
  
 public NbodySettings(  
 int deltaTime,  
 double errorDistance,  
 int threadsNum  
 ) {  
  
 bodyMass = Bodies.DEFAULT\_BODY\_MASS;  
  
 if (deltaTime < MIN\_DELTA\_TIME || deltaTime > MAX\_DELTA\_TIME) {  
 throw new DeltaTimeOutOfBoundsException();  
 }  
  
 if (errorDistance < MIN\_ERROR\_DISTANCE || errorDistance > MAX\_ERROR\_DISTANCE) {  
 throw new ErrorDistanceOutOfBoundsException();  
 }  
  
 if (threadsNum < MIN\_THREADS\_NUM || threadsNum > MAX\_THREADS\_NUM) {  
 throw new ThreadsNumOutOfBoundsException();  
 }  
  
 this.deltaTime = deltaTime;  
 this.errorDistance = errorDistance;  
 this.threadsNum = threadsNum;  
 }  
}

Б.4.7 NbodySolver

package nbody;  
  
import nbody.exceptions.BodiesNumOutOfBoundsException;  
  
import java.util.concurrent.\*;  
  
import static nbody.NbodySolvers.\*;  
  
public class NbodySolver {  
  
 private final Body[] b;  
 private final int dt;  
 private final double errorDistance;  
  
 private final RecalcingCallable[] recalcingCallables;  
 private final MovingCallable[] movingCallables;  
  
 private final ExecutorService executor;  
 private final Future[] recalcingFutures;  
 private final Future[] movingFutures;  
  
 public NbodySolver(Coords[] bodiesCoords, NbodySettings settings) {  
  
 if (bodiesCoords.length < MIN\_BODIES\_NUM || bodiesCoords.length > MAX\_BODIES\_NUM) {  
 throw new BodiesNumOutOfBoundsException();  
 }  
  
 b = new Body[bodiesCoords.length];  
 for (int i = 0; i < b.length; i++) {  
 b[i] = new Body(bodiesCoords[i], settings.bodyMass);  
 }  
  
 dt = settings.deltaTime;  
 this.errorDistance = settings.errorDistance;  
  
 final int[][] recalcingRanges = Helpers.ranges(0, b.length - 2, settings.threadsNum);  
 final int[][] movingRanges = Helpers.ranges(1, b.length, settings.threadsNum);  
  
 recalcingCallables = new RecalcingCallable[recalcingRanges.length];  
 movingCallables = new MovingCallable[movingRanges.length];  
 for (int i = 0; i < recalcingRanges.length; i++) {  
 recalcingCallables[i] = new RecalcingCallable(recalcingRanges[i][0], recalcingRanges[i][1]);  
 movingCallables[i] = new MovingCallable(movingRanges[i][0], movingRanges[i][1]);  
 }  
  
 executor = Executors.newFixedThreadPool(settings.threadsNum);  
 recalcingFutures = new Future[settings.threadsNum];  
 movingFutures = new Future[settings.threadsNum];  
 }  
  
 public NbodySolver(Body[] b, NbodySettings settings) {  
  
 if (b.length < MIN\_BODIES\_NUM || b.length > MAX\_BODIES\_NUM) {  
 throw new BodiesNumOutOfBoundsException();  
 }  
  
 this.b = b.clone();  
  
 dt = settings.deltaTime;  
 this.errorDistance = settings.errorDistance;  
  
 final int[][] recalcingRanges = Helpers.ranges(0, b.length - 2, settings.threadsNum);  
 final int[][] movingRanges = Helpers.ranges(1, b.length, settings.threadsNum);  
  
 recalcingCallables = new RecalcingCallable[recalcingRanges.length];  
 movingCallables = new MovingCallable[movingRanges.length];  
 for (int i = 0; i < recalcingRanges.length; i++) {  
 recalcingCallables[i] = new RecalcingCallable(recalcingRanges[i][0], recalcingRanges[i][1]);  
 movingCallables[i] = new MovingCallable(movingRanges[i][0], movingRanges[i][1]);  
 }  
  
 executor = Executors.newFixedThreadPool(settings.threadsNum);  
 recalcingFutures = new Future[settings.threadsNum];  
 movingFutures = new Future[settings.threadsNum];  
 }  
  
 public int n() {  
 return b.length;  
 }  
  
 public int dt() {  
 return dt;  
 }  
  
 public int bodyX(int index) {  
 return (int) b[index].p().x();  
 }  
  
 public int bodyY(int index) {  
 return (int) b[index].p().y();  
 }  
  
 public void recalcBodiesCoords() {  
 recalcBodiesForces();  
 moveNBodies();  
 }  
  
 private void recalcBodiesForces() {  
 for (int i = 0; i < recalcingCallables.length; i++) {  
 recalcingFutures[i] = executor.submit(recalcingCallables[i]);  
 }  
  
 for (Future f : recalcingFutures) {  
 try {  
 f.get();  
 } catch (ExecutionException | InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 private void moveNBodies() {  
 for (int i = 0; i < movingCallables.length; i++) {  
 movingFutures[i] = executor.submit(movingCallables[i]);  
 }  
  
 for (Future f : movingFutures) {  
 try {  
 f.get();  
 } catch (ExecutionException | InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 public void stop() {  
 executor.shutdown();  
 try {  
 executor.awaitTermination(1, TimeUnit.MINUTES);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
  
 private class RecalcingCallable implements Callable<Void> {  
  
 private final int leftIndex;  
 private final int rightIndex;  
  
 public RecalcingCallable(int leftIndex, int rightIndex) {  
 this.leftIndex = leftIndex;  
 this.rightIndex = rightIndex;  
 }  
  
 @Override  
 public Void call() {  
 double distance;  
 double magnitude;  
 Coords direction;  
  
 for (int k = leftIndex; k <= rightIndex; k++) {  
 for (int l = k + 1; l < b.length; l++) {  
 distance = distance(b[k], b[l]);  
 magnitude = (distance < errorDistance) ? 0.0 : magnitude(b[k], b[l], distance);  
 direction = direction(b[k], b[l]);  
  
 b[k].setF(  
 b[k].f().x() + magnitude \* direction.x() / distance,  
 b[k].f().y() + magnitude \* direction.y() / distance  
 );  
  
 synchronized (NbodySolver.this) {  
 b[l].setF(  
 b[l].f().x() - magnitude \* direction.x() / distance,  
 b[l].f().y() - magnitude \* direction.y() / distance  
 );  
 }  
 }  
 }  
  
 return null;  
 }  
 }  
  
 private class MovingCallable implements Callable<Void> {  
  
 private final int leftIndex;  
 private final int rightIndex;  
  
 public MovingCallable(int rangeStart, int rangeEnd) {  
 this.leftIndex = rangeStart - 1;  
 this.rightIndex = rangeEnd - 1;  
 }  
  
 @Override  
 public Void call() {  
  
 Coords dv; // dv = f/m \* dt  
 Coords dp; // dp = (v + dv/2) \* dt  
  
 for (int i = leftIndex; i <= rightIndex; i++) {  
 dv = dv(b[i], dt);  
 dp = dp(b[i], dt, dv);  
  
 b[i].setV(  
 b[i].v().x() + dv.x(),  
 b[i].v().y() + dv.y()  
 );  
  
 b[i].setP(  
 b[i].p().x() + dp.x(),  
 b[i].p().y() + dp.y()  
 );  
  
 b[i].setF(0.0, 0.0);  
 }  
  
 return null;  
 }  
 }  
}

Б.4.8 NbodySolvers

package nbody;  
  
import static nbody.Bodies.G;  
  
public class NbodySolvers {  
  
 public static final int MIN\_BODIES\_NUM = 1;  
 public static final int MAX\_BODIES\_NUM = 1024;  
 public static final int DEFAULT\_BODIES\_NUM = 4;  
  
 public static final int MIN\_DELTA\_TIME = 16;  
 public static final int MAX\_DELTA\_TIME = 128;  
 public static final int DEFAULT\_DELTA\_TIME = MIN\_DELTA\_TIME;  
  
 public static final double MIN\_ERROR\_DISTANCE = 1e2;  
 public static final double MAX\_ERROR\_DISTANCE = 1e3;  
 public static final double DEFAULT\_ERROR\_DISTANCE = MIN\_ERROR\_DISTANCE;  
  
 public static final int MIN\_THREADS\_NUM = 1;  
 public static final int MAX\_THREADS\_NUM = 64;  
 public static final int DEFAULT\_THREADS\_NUM = MIN\_THREADS\_NUM;  
  
 public static double distance(Body b1, Body b2) {  
 return Math.sqrt(  
 Math.pow(b1.p().x() - b2.p().x(), 2) + Math.pow(b1.p().y() - b2.p().y(), 2)  
 );  
 }  
  
 public static double magnitude(Body b1, Body b2, double b1b2distance) {  
 return G \* b1.m() \* b2.m() / Math.pow(b1b2distance, 2);  
 }  
  
 public static Coords direction(Body b1, Body b2) {  
 return new Coords(  
 b2.p().x() - b1.p().x(),  
 b2.p().y() - b1.p().y());  
 }  
  
 public static Coords dv(Body b, long dt) {  
 return new Coords(  
 b.f().x() / b.m() \* dt,  
 b.f().y() / b.m() \* dt);  
 }  
  
 public static Coords dp(Body b, long dt, Coords dv) {  
 return new Coords((b.v().x() + dv.x() / 2) \* dt,  
 (b.v().y() + dv.y() / 2) \* dt);  
 }  
}

Б.5 package nbodygui

Б.5.1 package nbodygui.exceptions

Б.5.1.1 Код класса DurationMillisOutOfBoundsException

package nbodygui.exceptions;  
  
public class DurationMillisOutOfBoundsException extends IndexOutOfBoundsException {  
}

Б.5.1.2 Код класса HeightOutOfBoundsException

package nbodygui.exceptions;  
  
public class HeightOutOfBoundsException extends IndexOutOfBoundsException {  
}

Б.5.1.3 Код класса WidthOutOfBoundsException

package nbodygui.exceptions;  
  
public class WidthOutOfBoundsException extends IndexOutOfBoundsException {  
}

Б.5.2 Код класса Frame

package nbodygui;  
  
import nbodygui.exceptions.HeightOutOfBoundsException;  
import nbodygui.exceptions.WidthOutOfBoundsException;  
  
import javax.swing.\*;  
  
import static nbodygui.Frames.\*;  
  
public class Frame extends JFrame {  
  
 public Frame(int width, int height, Panel panel) {  
 super(DEFAULT\_TITLE);  
  
 if (width < MIN\_WIDTH || width > MAX\_WIDTH) {  
 throw new WidthOutOfBoundsException();  
 }  
  
 if (height < MIN\_HEIGHT || height > MAX\_HEIGHT) {  
 throw new HeightOutOfBoundsException();  
 }  
  
 add(panel);  
  
 setSize(width, height);  
 setResizable(false);  
 setLocationRelativeTo(null);  
 setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  
 }  
}

Б.5.3 Код класса Frames

package nbodygui;  
  
public class Frames {  
  
 public static final String DEFAULT\_TITLE = "Движение тел";  
  
 public static final int MIN\_WIDTH = 200;  
 public static final int MAX\_WIDTH = 2000;  
 public static final int DEFAULT\_WIDTH = 500;  
  
 public static final int MIN\_HEIGHT = 200;  
 public static final int MAX\_HEIGHT = 2000;  
 public static final int DEFAULT\_HEIGHT = 500;  
}

Б.5.4 Код класса Panel

package nbodygui;  
  
import nbody.NbodySolver;  
import nbodygui.exceptions.DurationMillisOutOfBoundsException;  
  
import javax.swing.\*;  
import java.awt.\*;  
  
import static nbodygui.Panels.\*;  
  
public class Panel extends JPanel {  
  
 private final NbodySolver solver;  
 private final int durationMillis;  
  
 public Panel(NbodySolver solver, int durationMillis) {  
 if (durationMillis < MIN\_DURATION\_MILLIS || durationMillis > MAX\_DURATION\_MILLIS) {  
 throw new DurationMillisOutOfBoundsException();  
 }  
  
 this.solver = solver;  
  
 this.durationMillis = durationMillis;  
 }  
  
 public void start() {  
 for (int t = 0; t <= durationMillis; t = t + solver.dt()) {  
 solver.recalcBodiesCoords();  
 repaint();  
 }  
 solver.stop();  
 }  
  
 private void drawRandomPoints(Graphics gr) {  
 Graphics2D graphics = (Graphics2D) gr;  
 graphics.setPaint(DEFAULT\_POINTS\_COLOR);  
  
 for (int i = 0; i < solver.n(); i++) {  
 int x = solver.bodyX(i);  
 int y = solver.bodyY(i);  
 graphics.fillOval(x, y, DEFAULT\_POINTS\_SIZE, DEFAULT\_POINTS\_SIZE);  
 }  
 }  
  
 @Override  
 public void paintComponent(Graphics gr) {  
 super.paintComponent(gr);  
 drawRandomPoints(gr);  
 }  
}

Б.5.5 Код класса Panels

package nbodygui;  
  
import java.awt.\*;  
  
public class Panels {  
  
 public static final Color DEFAULT\_POINTS\_COLOR = Color.GREEN;  
 public static final int DEFAULT\_POINTS\_SIZE = 15;  
  
 public static final int MIN\_DURATION\_MILLIS = 10\_000;  
 public static final int MAX\_DURATION\_MILLIS = 3\_600\_000;  
 public static final int DEFAULT\_DURATION\_MILLIS = MIN\_DURATION\_MILLIS;  
}