

Софийски Университет „Св. Климент Охридски“

Проект по „Системи за Паралелна Обработка“,

специалност „Компютърни науки“,

на тема

**Паралелно пресмятане на *e***

**(с графичен интерфейс)**

(Задача 12)

Изготвили:

Павел Тодоров Хаджиев, Ф.Н: 81297

Симеон Пламен Георгиев, Ф.Н. 81139

Ръководител:

ас. Христо Христов

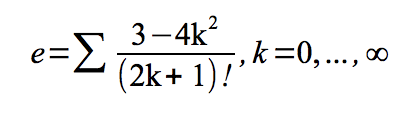
Проверил: .........

(ас.Христо Христов)

София, Юни 2018 г.

1. Тема на проекта

Едно важно за математиката число е Неперовото число (Ойлеровото число), тоест числото ***e***. Използвайки сходящи редове, можем да сметнем стойността на ***e*** с произволно висока точност. Един от сравнително бързо сходящите към ***e*** редове е:



Задачата е да напишем програма за изчисление на числото ***e*** използвайки цитирания ред, която използва паралелни процеси (нишки) и осигурява пресмятането на ***e*** със зададена от потребителя точност.

Видове командни параметри, които се подават при стартиране на програмата:

* „-p=<integer*>*”, където като аргумент сезадава прецизността на изчислението, или по – конкретно колко знака след десетичната запетая да се извърши калкулацията и броя членове на реда.
* „-t=<integer*>*”, където като аргумент се задават броя нишки, с които да се изпълни изчислението.
* „-o=<string*>*”, където като аргумент се подава име на файл, в който да се запише резултата от изчислението.
* „-q”, където ‘q’ е съкратено от *quiet* и означава режим на работа на програмата, при който се извежда само времето отделено за изчисление на ***e,*** без ориенировъчни съобщения в хода на изпълнението на програмата.
* Програмата извежда подходящи съобщения на различните етапи от работата си, както и времето отделено за изчисление и резултата от изчислението (стойността на e); Примери за подходящи съобщения: „Thread- started.“, „Thread- stopped.“, „Thread- execution time was (millis): “, „Threads used in current run: “, „Total execution time for current run (millis): “

*Забележка:* Тъй като нашата програма реализира и графичен интерфейс, параметърът –q означава режим на работа, при който **не се пуска** графичния интерфейс.

1. Декомпозиция на задачата

Алгоритъмът изчислява Неперовото число с точност след десетичната запетая, зададена от потребителя и същата дължина на реда, използвайки подхода на многонишковото програмиране. Всяка нишка пресмята отделна част от членовете на реда, даден в условието и накрая се натрупва резултата в променлива.

Тъй като членовете в началото на реда се смятат значително по-лесно от по-следващите, за да може ефективността на програмата да не пада при използването на нишковия подход, сме избрали алгоритъм, в който дадена нишка пресмята членовете на реда през дадена стъпка, а не няколко последователни члена. Така изчисленията се разпределят равномерно и всички нишки работят приблизително еднакво време. Тази стъпка е равна на броя на нишките.

След завършване на работата на всички нишки, в променлива имаме окончателния вид на Неперовото число. Главната нишка изчаква приключването на нишките, за да може да вземе резултата от изчисленията им. При този подход колкото повече нишки се пуснат (на теория), толкова по – бързо ще се извърши изчислението, тъй като не се изисква от нишките да се изчакват и на практика калкулацията е паралелна.

Имплементацията на предложеният алгоритъм е на езика Java и се състои от 5 класа: **Main, CalculateE, CalculateThread, StartThreads и GUI**.

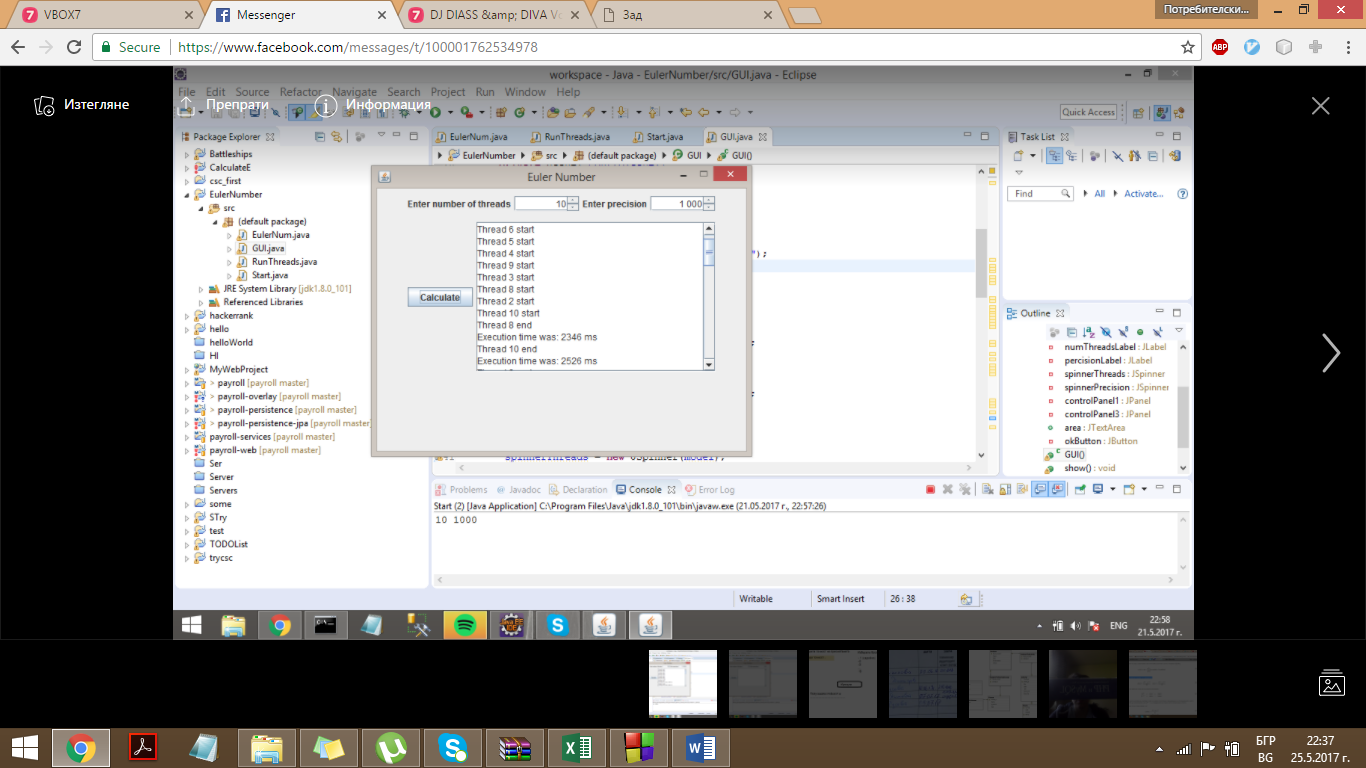
В класът **Main** е реализиран командния интерфейс на програмата, чрез библиотеката на[**Apache Commons CLI**](http://commons.apache.org/proper/commons-cli/). Той има main метод, чиито аргументи се обработват и спрямо тях се задават параметрите на теста, а именно прецизността, броя нишки, файла за писане на резултати и подробния или тихия режим на работа. След това се стартира програмата. Ако е подадена опцията -q се зарежда и графичния интерфейс.

В класът **CalculateE** се намират статичните методи **calculateSumMember** за намиране на даден пореден член на реда от условието.

В класът **CalculateThread** се имплементира Java интерфейса **Runnable**. Това е класа, който ползваме за нишка. По зададени първи член и стъпка, нишката намира търсените членовете на редицата спрямо алгоритъма ни. Също така, ако не сме в тих режим, изобразява в интерфейса и пише във изходния файл съобщения за стартирането на нишка, приключването на нейната работа, и времето и за работа.

В класът **StartThreads** се намира статичен метод **start**, който приема като аргумент брой нишки и стартира съответния брой нишки, изпълнявайки предложения алгоритъм. Изчаква се завършването на всички нишки, след което се пресмята резултата (събират се резултатите на всички нишки) и се извежда във изходния файл и на графичния интерфейс. Извежда се и времето за изпълнение на програмата.

В класът **GUI** се задава графичния интерфейс на програмата. Той изглежда по следния начин:



Интерфейсът се състои от два тесктови етикета: за брой на използваните нишки **numThreadsLabel** и за точност **percisionLabel,** както и 2 spinner бутона: **spinnerThreads и spinnerPrecision.** Чрез бутона **calculate** програмата започва своята работа, а изходът от програмата излиза в текстовото поле **area.**

1. Проведени тестове и измервания

Разработено е приложение на Java и са проведени тестови изчисления на Intel Core i7 процесор с 4 ядра със скорост 2.9 GHz. Целта на теста, е да се оценят ускорението Sp (speed-up) , ефективността Е (efficiency) и Tp (времето необходимо за завършване на работата с p на брой нишки) на използвания алгоритъм. Sp и Ep се представят по следния начин:

**Ускорение: S(p) = T(1)/T(p)**

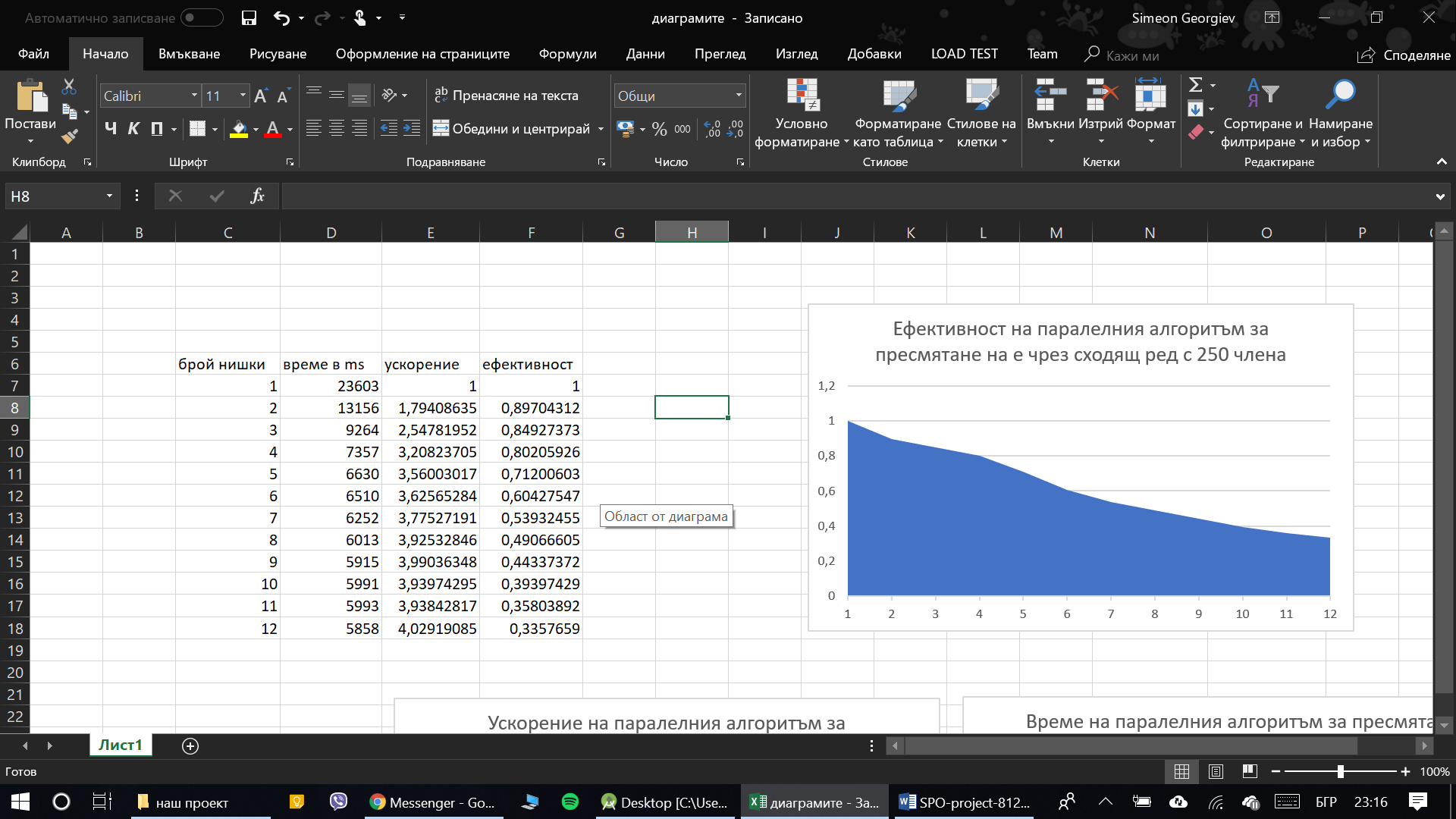
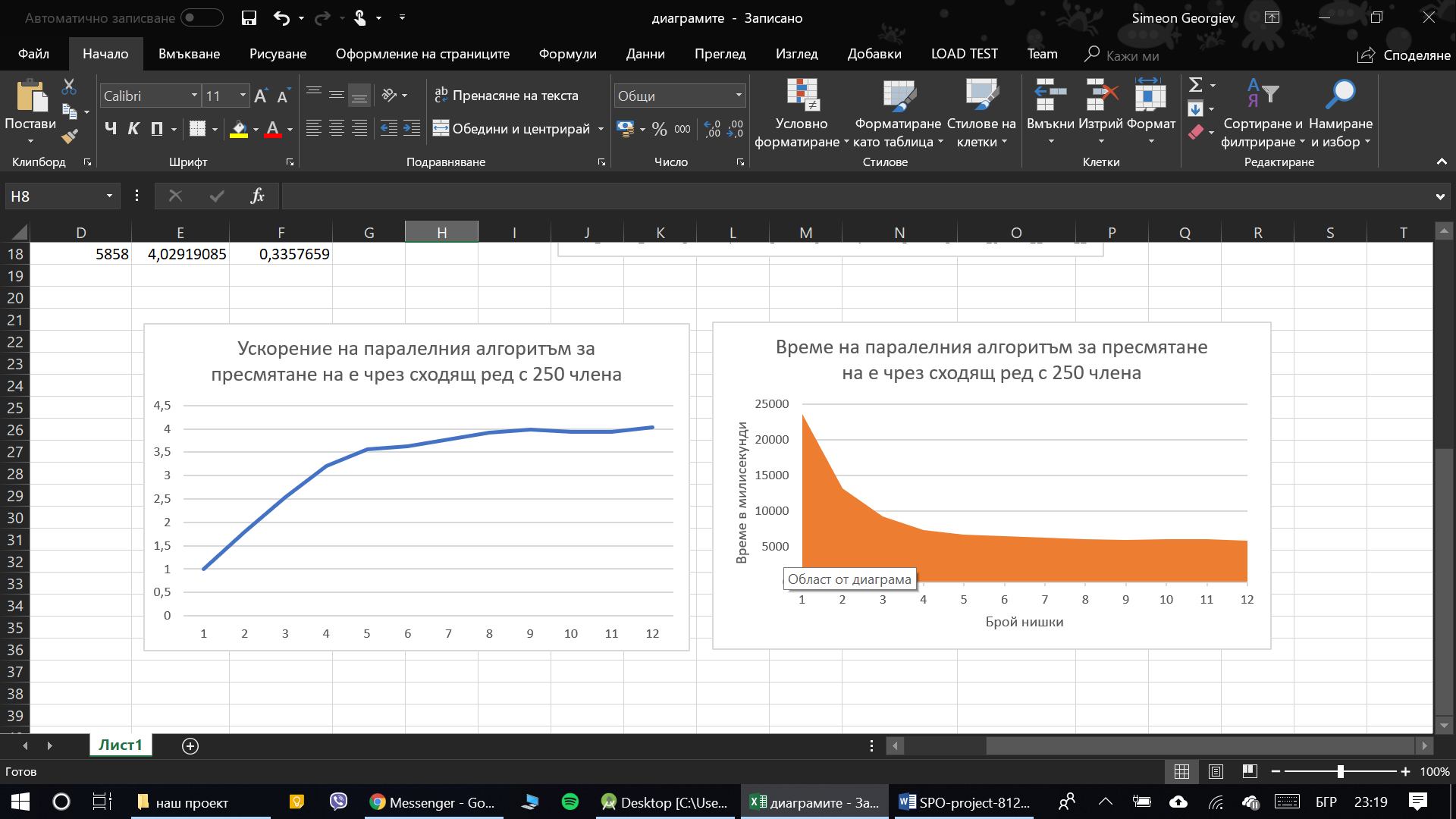
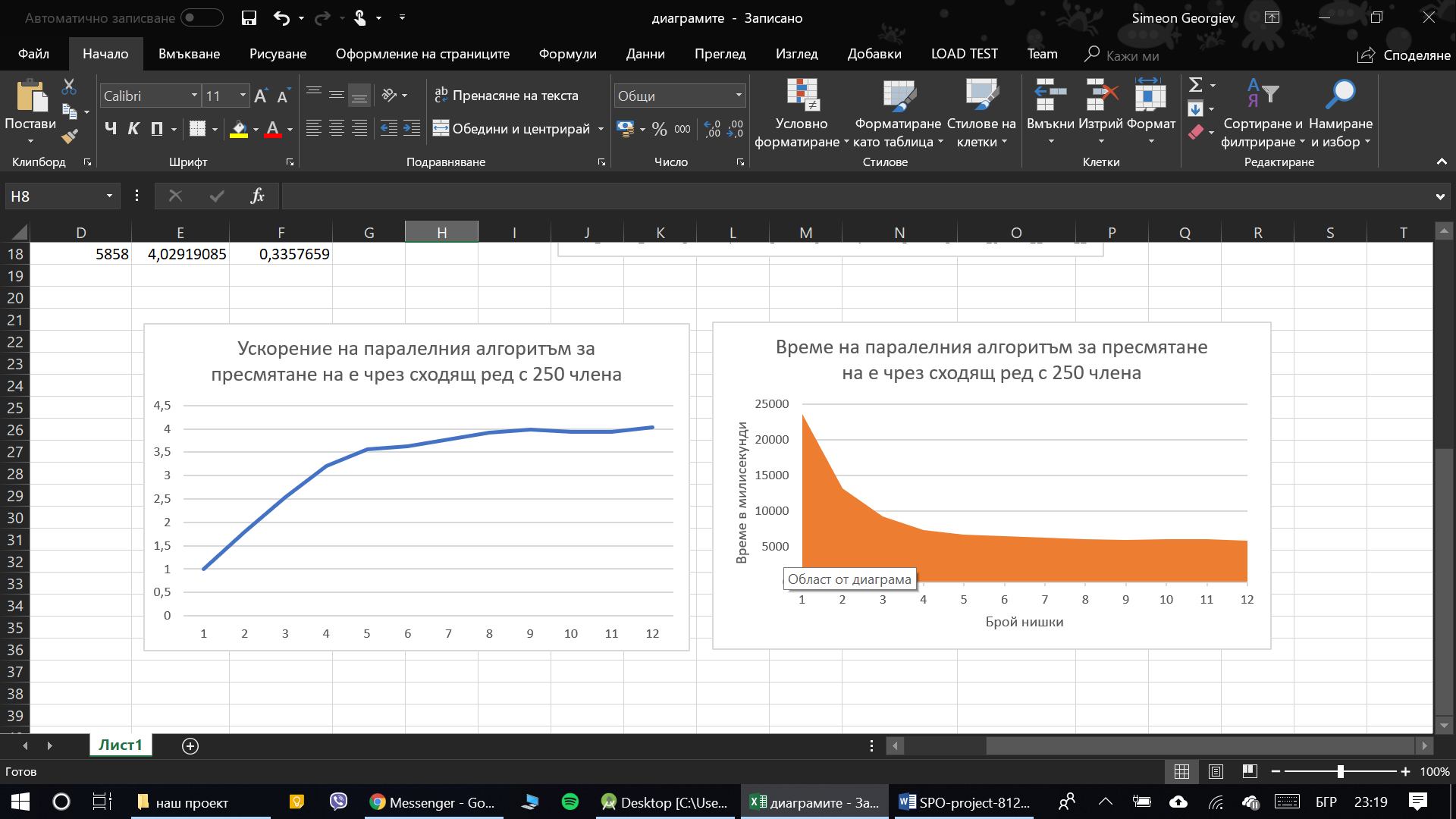
**Ефективност: E(p) = S(p)/p**

Замерванията са направени със прецизност на изчислението 250 единици, т.е 250 знака след десетичната запетая ще бъдат калкулирани и дължината на реда ще е 250. Алгоритъма, по който пресмятаме резултата работи по следния начин:

Ако **thread** е номера на дадена нишка, а **threadsCount** - общия брой нишки, всяка нишка пресмята първо члена на редицата с последователен номер **thread** (при ***k=thread***) и след това всеки следващ член през **threadsCounts** разстояние, т.е. итерира по членовете на реда със стъпка **threadsCount** и начало thread.

1. Резултати от проведените тестови изчисления:

|  |  |  |  |
| --- | --- | --- | --- |
| Брой нишки | Време в ms | Ускорение | Ефективност |
| 1 | 23603 | 1 | 1 |
| 2 | 13156 | 1,79408635 | 0,89704312 |
| 3 | 9264 | 2,54781952 | 0,84927373 |
| 4 | 7357 | 3,20823705 | 0,80205926 |
| 5 | 6630 | 3,56003017 | 0,71200603 |
| 6 | 6510 | 3,62565284 | 0,60427547 |
| 7 | 6252 | 3,77527191 | 0,53932455 |
| 8 | 6013 | 3,92532846 | 0,49066605 |
| 9 | 5915 | 3,99036348 | 0,44337372 |
| 10 | 5991 | 3,93974295 | 0,39397429 |
| 11 | 5993 | 3,93842817 | 0,35803892 |
| 12 | 5858 | 4,02919085 | 0,33576590 |



Забележка: при използвания хардуер и алгоритъм, изпълняване на тестовете с повече от 12 нишки (дори с повече 8) се оказва безсмислено. При повече от 12, започва да се наблюдава забавяне на работата, поради все по-тежките операции, които трябва да изпълняват последните нишки.