

**СОФИЙСКИ УНИВЕРСИТЕТ “СВ. КЛИМЕНТ ОХРИДСКИ”**

Факултет по математика и информатикa

Проект

по Разпределени Софтуерни Системи

Софтуерно Инженерство

2016/2017

Изчисляване на Неперовото число - Тема 14

**Изготвили**:

Ружа Боботанова, 61835

Красимира Бадова, 61866

**Ръководител:**

**…………………………………...**

ас. Христо Христов

гр. София

2017г

# 

Съдържание

[**Задание**](#_joff60vnunhv) **2**

[**Имплементация**](#_3keu7bsfphhm) **3**

[**Стартиране на програмата**](#_ls6yhof4ps5h) **4**

[**Използвани технологии**](#_d3qphabhdxyt) **6**

[**Тестове и измервания**](#_5e6ljl32k468) **6**

# Задание

**Условие:**

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:

, к = 0, ….. ∞

Вашата задача е да напишете програма за изчисление на числото e използвайки цитирания ред, която използва паралелни процеси (нишки) и осигурява пресмятането на e със зададена от потребителя точност. Изискванията към програмата са следните:

* Команден параметър задава точността на пресмятанията. По Ваше желание, точността се изразява или в брой цифри след десетичната запетая или в брой членове на реда. Командният параметър задаващ точността има вида - **“-p 10240”;**
* Друг команден параметър задава максималния брой нишки (задачи) на които разделяме работата по пресмятането на e – например **“–t 1”** или **“–tasks 3”;**
* Програмата извежда подходящи съобщения на различните етапи от работата си, както и времето отделено за изчисление и резултата от изчислението (стойността на e);
* Записва резултата от работа си (стойността на e) във изходен файл, зададен с подходящ параметър, например **“-o result.txt”.** Ако този параметър е изпуснат, се избира име по подразбиране;
* Да се осигури възможност за **„quiet“** режим на работа на програмата, при който се извежда само времето отделено за изчисление на e, отново чрез подходящо избран друг команден параметър – например **“-q”;**

ЗАБЕЛЕЖКА:

* При желание за направата на подходящ графичен потребителски интерфейс (GUI) с помощта на класовете от пакета javax.swing задачата може да се изпълни от двама души; Разработването на графичен интерфейс не отменя изискването Вашата програма да поддържа изредените командни параметри. В този случай към функцията на параметъра параметъра „-q“ се добавя изискването да не пуска графичният интерфейс. Причината за това е, че Вашата програма трябва да позволява отдалечено тестване, а то ще се извършва в terminal.

# Имплементация

За да извършим пресмятането на Неперовото число по най - бърз и ефективен начин, нашата програма се базира на принципите на паралелното програмиране. За да разпределим цялата работа по сумирането на отделните членове поравно на дадения брой нишки, имлементираме алгоритъма за фината грануларност. Т.е. нашата програма има за цел да извлече потребителски данни и да върне резултат за възможно най-малко време.

Потребителските данни се задават през графичен интерфейс или ръчно чрез изпълнение на даден скрипт. (в.ж “Стартиране на програма”). И в двата случая се изпълява еднотипна имплементация на алгоритъма.

Файлът, който отговаря за пресмятането на сумата е под името “neper\_number.py”.

Главната фунцкия “main” се грижи за събирането на подходящите съобщения и записването им във желания от потребитля файл, както и за извикването на сумирането на реда (“calculate\_row\_sum”).

Разпределянето на работата на всички нишки поравно се извтешва във фунцията “calculate\_row\_sum”. Генерираме речник с ключ - съответната нишка и стойност - списък от k-тите членове, които тази нишка трява да сметне(“generate\_work\_per\_thread”).

Пример:

При 100 члена и 6 нишки, речника, който генерираме има формата:

{

0: [0, 6, 12, 18, 24, 30, 36, 42, 48, 54, 60, 66, 72, 78, 84, 90, 96],

1: [1, 7, 13, 19, 25, 31, 37, 43, 49, 55, 61, 67, 73, 79, 85, 91, 97],

2: [2, 8, 14, 20, 26, 32, 38, 44, 50, 56, 62, 68, 74, 80, 86, 92, 98],

3: [3, 9, 15, 21, 27, 33, 39, 45, 51, 57, 63, 69, 75, 81, 87, 93, 99],

4: [4, 10, 16, 22, 28, 34, 40, 46, 52, 58, 64, 70, 76, 82, 88, 94],

5: [5, 11, 17, 23, 29, 35, 41, 47, 53, 59, 65, 71, 77, 83, 89, 95]

}

След като знаем коя нишка кой точно k-ти член трябва да пресметне, за всяка нишка пускаме пресмятането на генерирания k-ти член, като използваме класа “Process” от python пакета “multiprocessing”. Пресметнатия резултат за всеки член се натрува в предварително инстанцирана опашка (класа “Queue” от “multiprocessing”). Накрая сумираме всички елементи на опашката.

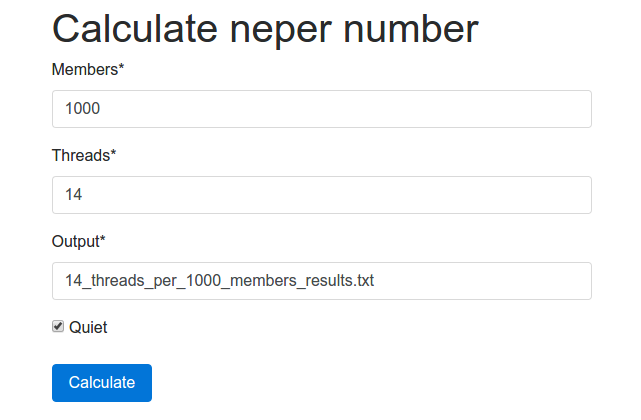
# Стартиране на програмата

* през графичен интерфейс:

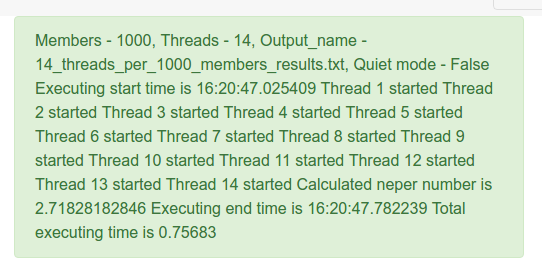
Попълване на форма със следните полета:

* **members** - брой k члена на сумата, която изчисляваме (задължителен атрибут)
* **threads** - брой нишки (стойността по подразбиране е 1)
* **output** - име на файл, в който ще се записва резултата (името по подразбиране на файла е “results.txt”)
* **quiet** - възможност за “тих” режим (извежда само времето за изпълнение на програмата, без допълнителни пояснения)

Примерен вход:



Примерен изход:



* за отдалечено тестване

Примерен вход:

$ python neper\_number.py -p 1000 -t 14 -q=True -o 14\_threads\_per\_1000\_members\_results.txt

Примерен изход:

Members - 1000, Threads - 14, Output\_name - 14\_threads\_per\_1000\_members\_results.txt, Quiet mode - False  
 Executing start time is 16:20:47.025409  
 Thread 1 started  
 Thread 2 started  
 Thread 3 started  
 Thread 4 started  
 Thread 5 started  
 Thread 6 started  
 Thread 7 started  
 Thread 8 started  
 Thread 9 started  
 Thread 10 started  
 Thread 11 started  
 Thread 12 started  
 Thread 13 started  
 Thread 14 started  
 Calculated neper number is 2.71828182846  
 Executing end time is 16:20:47.782239  
 Total executing time is 0.75683

Тестовете на програмата бяха изпълнени на Linux операционна система на личен компютър, както и на сървър на ФМИ (u61866@ats24). На база от тестовите резултати, са направени заключенията за ефективността на използвания алгоритъм.

# Използвани технологии

За да можем да изпълним програмата на тестовите машини, скрипта, който пресмята изчисленията е съобразен с Python(v 2.7.5).

За реализиране на проекта под графична среда изпозлваме Web framework-a Django(v1.10.7) и Python(v3.5.4).

За визуализация на графичните данни използваме<https://plot.ly/>.

# 

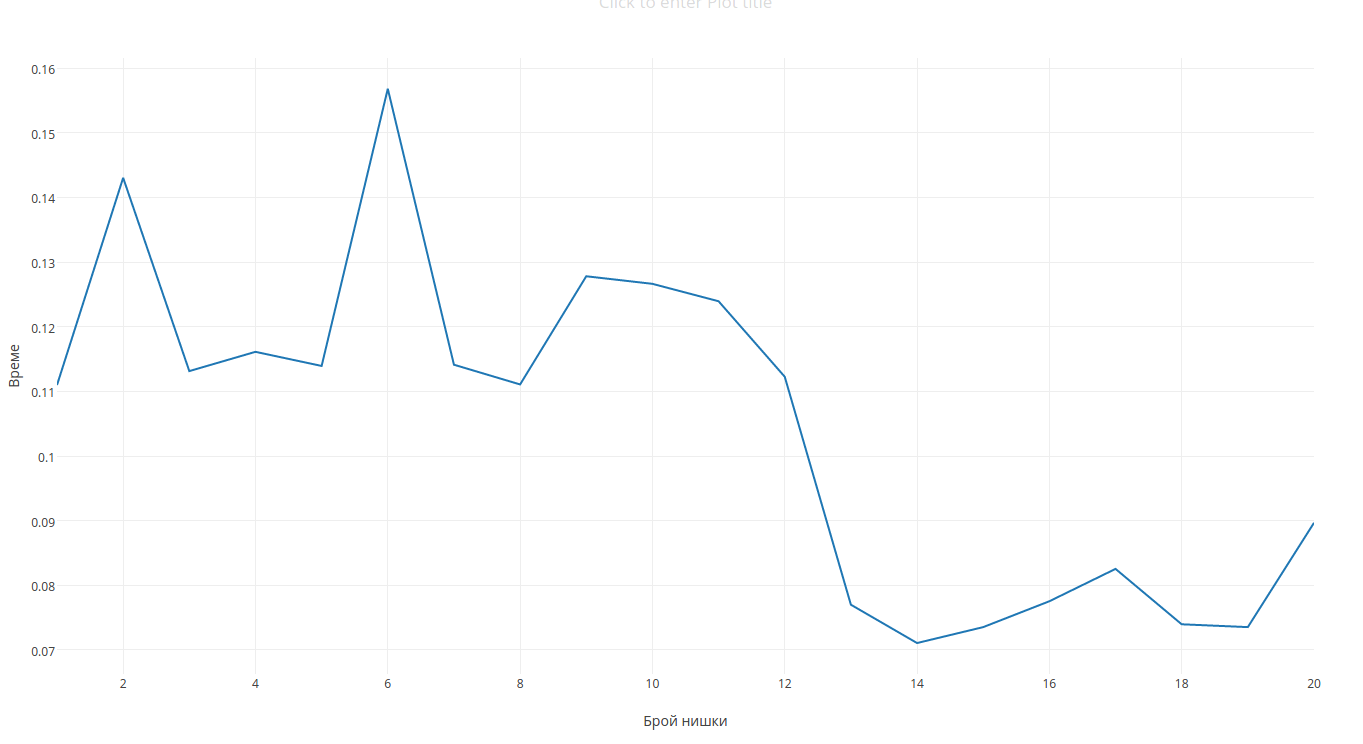
# Тестове и измервания

На таблиците отдолу е показано:

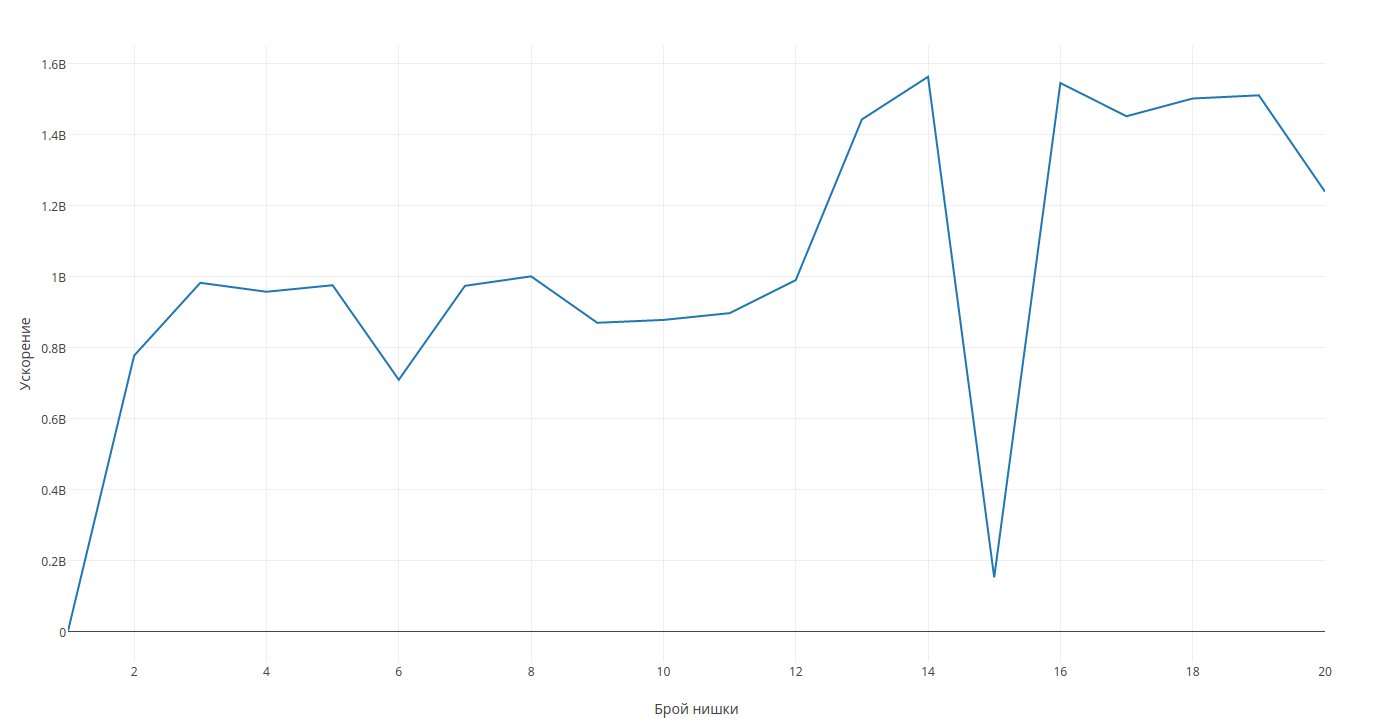
* **ускорение** (speed-up) на програмата при p на брой нишки -
* **ефективност** на програмата при p на брой нишки -
* **време** за изпълнение на програмата с p на брой нишки -
* Тест таблица 1: членове на редицата - 100

|  |  |  |  |
| --- | --- | --- | --- |
| **брой процеси** | **време за изпълнение (секунди)** | **успокорение** | **ефективност** |
| 1 | 0.110969 | 1 | 1 |
| 2 | 0.143023 | 0,775882201 | 0,387941101 |
| 3 | 0.113132 | 0,980880741 | 0,326960247 |
| 4 | 0.116112 | 0,955706559 | 0,23892664 |
| 5 | 0.113928 | 0,974027456 | 0,194805491 |
| 6 | 0.156794 | 0,707737541 | 0,117956257 |
| 7 | 0.114119 | 0,972397234 | 0,138913891 |
| 8 | 0.111066 | 0,999126645 | 0,124890831 |
| 9 | 0.127798 | 0,868315623 | 0,096479514 |
| 10 | 0.126627 | 0,876345487 | 0,087634549 |
| 11 | 0.123933 | 0,895395093 | 0,081399554 |
| 12 | 0.112260 | 0,988499911 | 0,082374993 |
| 13 | 0.077003 | 1,441099697 | 0,110853823 |
| 14 | 0.071075 | 1,561294407 | 0,111521029 |
| 15 | 0.073536 | 1,50904319 | 0,100602879 |
| 16 | 0.077537 | 1,543650128 | 0,096478133 |
| 17 | 0.082541 | 1,450067239 | 0,085298073 |
| 18 | 0.073976 | 1,500067589 | 0,083337088 |
| 19 | 0.073538 | 1,509002149 | 0,079421166 |
| 20 | 0.089669 | 1,237540287 | 0,061877014 |

* Графика за време на изпълнение, членове на редицата - 100, процеси - 20



* Графика за ускорение, членове на редицата - 100, процеси - 20

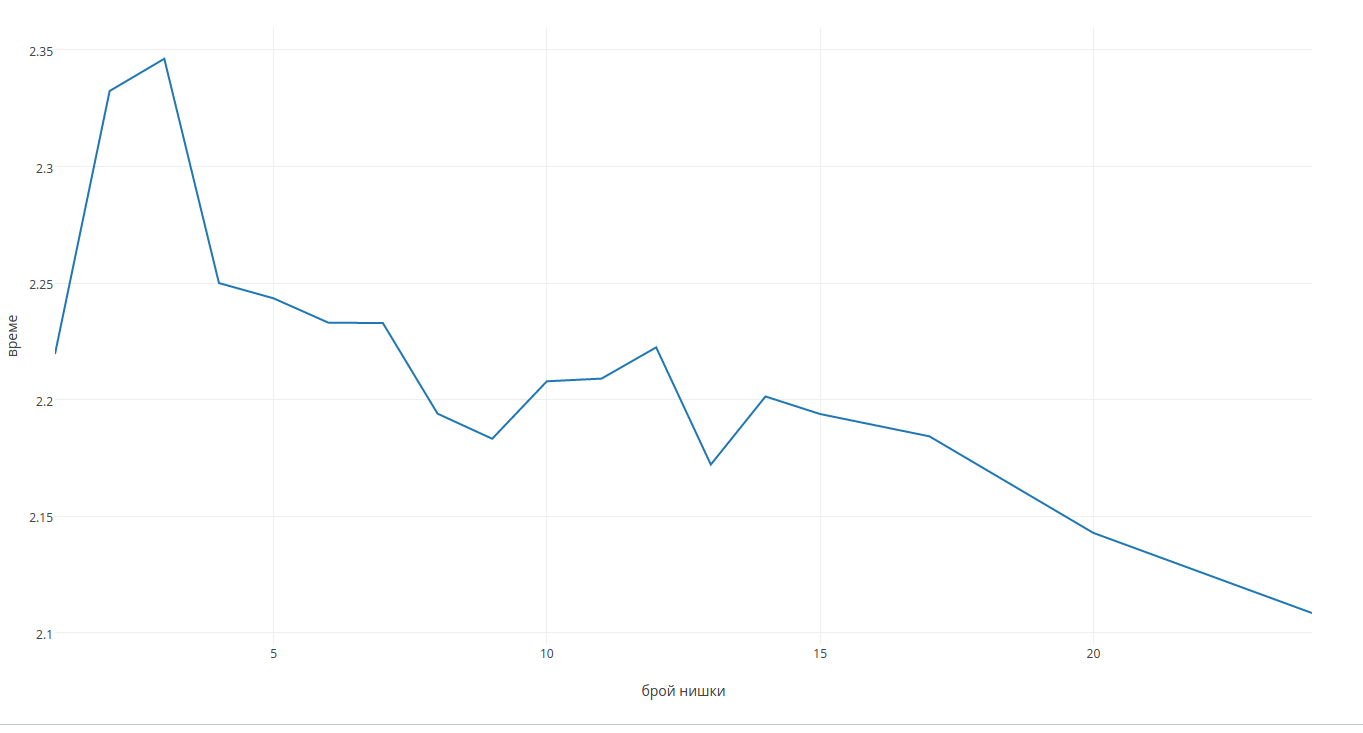


Наблюдаваме

* Тест таблица 2: членове на редицата - 2000, брой процеси - 24

|  |  |  |  |
| --- | --- | --- | --- |
| **брой процеси** | **време за изпълнение (секунди)** | **успокорение** | **ефективност** |
| 1 | 2.219608 | 1 | 1 |
| 2 | 2.3324 | 0,951641228 | 0,475820614 |
| 3 | 2.346219 | 0,946036154 | 0,315345385 |
| 4 | 2.250033 | 0,986477976 | 0,246619494 |
| 5 | 2.243484 | 0,989357624 | 0,197871525 |
| 6 | 2.233066 | 0,993973308 | 0,165662218 |
| 7 | 2.232868 | 0,994061449 | 0,142008778 |
| 8 | 2.194015 | 1,011664916 | 0,126458114 |
| 9 | 2.183272 | 1,01664291 | 0,112960323 |
| 10 | 2.207912 | 1,005297313 | 0,100529731 |
| 11 | 2.209059 | 1,004775336 | 0,091343212 |
| 12 | 2.222458 | 0,998717636 | 0,08322647 |
| 13 | 2.172196 | 1,02182676 | 0,078602058 |
| 14 | 2.201402 | 1,008270184 | 0,072019299 |
| 15 | 2.193869 | 1,011732241 | 0,067448816 |
| 17 | 2.184304 | 1,016162585 | 0,05977427 |
| 20 | 2.142888 | 1,035802151 | 0,051790108 |
| 24 | 2.10853 | 1,052680303 | 0,043861679 |

* Графика за време на изпълнение, членове на редицата - 2000, процеси - 24



От двата теста наблюдаваме зависимостта между времето за изпълнение на програмата и броя нишки, които я изпълняват, съответно и как се ускорява времето за изпълнение, в зависимост от броя нишки. Виждаме, че

колкото повече нишки се използват, толкова по-малко време отнема изпълнението на програмата. Съответно при голям брой членове (2000), времето за сумирането им намалява значително и се ускорението за работа се покачва.

Линк към кода за изпълнение, заедно с всички графики:

<https://github.com/kbadova/Neper-Number-FMI-Project>