

**Софийски университет „Св. Климент Охридски”**

**Факултет по математика и информатика**

**Проект на тема**

Пресмятане на числото е

Изготвили:

Станислав Веселинов Станчев, 71277, Информационни системи

Веселина Венциславова Кръстева, 71283, Информационни системи

Ръководител:

ас. Христо Христов

**Разпределени ИТ архитектури – летен семестър, 2012 г**

Описание на задачата

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: 

Вашата задача е да напишете програма за изчисление на числото ***e*** изпозлвайки цитирания

ред, която използва паралелни процеси (нишки) и осигурява пресмятането на ***e*** със зададена от

потребителя точност. Изискванията към програмата са следните:

(o) Точността на пресмятанията, в брой цифри след десетичната точка, задаваме с

подходящо избран команден параметър – например **“-p 10000”;**

(о) Друг команден параметър задава максималния брой нишки (задачи) на които разделяме

работата по пресмятането на ***e*** – например **“–t 1”** или **“–tasks 3”;**

(о) Програмата извежда подходящи съобщения на различните етапи от работата си, както и

времето отделено за изчисление и резултата от изчислението (стойността на ***e***);

(o) 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;

(o) 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”;**

Уточнения:

**(о) В условието на задачата се говори за разделянето на работата на две или повече**

**нишки. Работата върху съответната задача на една нишка ще служи за еталон, по който да**

**измерваме евентуално ускорение (T1). Тоест в кода реализиращ решенията на задачите**

**трябва да се предвиди и тази възможност – задачата да бъде решавана от единствена нишка**

**(процес); Пускайки програмата да работи върху задачата с помощта на единствена нишка,**

**ще считаме че използваме серийното решение на задачата; Измервайки времето за работа на**

**програмата при работа с „p“ нишки - Tp, изчисляваме Sp. Представените на защитата данни**

**за работата на програмата, трябва да отразят и ефективността от работата и, тоест да се**

**изчисли и покаже Ep.**

**(о) Не се очаква от вас да реализирате библиотека, осигуряваща математически**

**операции със голяма точност. Подходяща за тази цел библиотека е например Apfloat**

**(http://www.apfloat.org).**

**(о) Командните аргументи (параметри) на терминална (конзолна) Java програма,**

**получаваме във масива String args[] на main() метода, на стартовия клас. За „разбирането“**

**им (анализирането им) може да ползвате и външни библиотеки писани специално за тази**

**цел . Един добър пример за това е: Apache Commons CLI (http://commons.apache.org/cli/).**

**(о) Интересен е въпросът, кога достигаме зададената точност на изчисленията? Тоест**

Зад. 8, 1/2 (3.0)

**кога сме сметнали “*e”* със зададените от потребителя брой цифри след десетичната точка.**

**Едно добро ограничение за серийната (последователната) програма е разликата между две**

**поредно изчислени стойности на “*e”* да е произволно малка.**

Описание на решението

Програмата е разработена на езика Java и използва разпалеляване на изчисленията за по-бързо пресмятане на *е*. За постигане на зададената прецизност на изчислението след десетичната запетая се пресмятат определен брой събираеми от редицата, посочена по-горе. За получаване на търсеното ускорение при използване на нишки, броя на нужните за пресмятането събираеми се разделя на интервали, които се задават на всяка нишка за пресмятане.

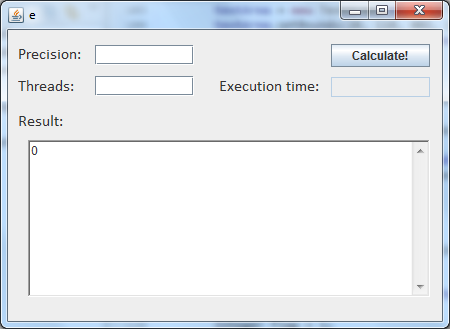
Програмата се състой от 3 класа – Runner, sumCalculation и GUI.

Runner – представлява главния клас от където се инициализират нишките и където се пресмята крайния резултат когато нишките извършат пресмятанията си.

sumCalculation – класът, в който се извършва самото пресмятане на събираемите.

GUI – изгражда графичния интерфейс, използвайки swing, и обработва потребителски вход.

Програмата разполага и с базов графичен интерфейс:

****

Тестване

За тестване на програмата бе ползван сървърът, предоставен ни от ас. Христов, който е с 12 процесора. Тествано бе времето за пресмятане на 5000 и 20000 цифри след десетичната запетая на числото *е*, изполвайки 1, 2, 3, 4, 6, 8, 10, 12, 16 и 20 нишки.

Първо ще тестваме скоростта на пресмятане с 5000 цифри след десетичната запетая.

По абсцисата на графиката са представени броят нишки за изчисление, а по ординатата – времето в милисекунди за изчисление.

Както се вижда серийното решение, т.е. на една нишка, изчислява до 20 000 цифри след запетаята за 11 861 милисекунди. При 2 нишки – за 8212 милисекунди и т.н. се вижда ускорение на пресмятането до 10 нишки, след това ускорението намаля драстично, а дори се забавя пресмятането при повече от 10 нишки.

На следващата графика е представено ускорението при използване на 2 и повече нишки. Както се вижда не е ефективно да се изполват повече от 16 нишки за пресмятане на малък брой цифри след десетичната запетая. Коефициента на ускорение получаваме като разделим времето за изпълнение от 1 нишка на времето за изпълнение на n нишки.

Тука е представена ефективността при изчисленията, която се пресмята като разделим ускорението, получено при различен брой нишки, разделено на броя нишки.

Следват резултатите, получени при пресмятане с точност до 20 000 цифри след запетаята.

Първата графика предстява времето, за изчисление.

Графика на ускорението:

Графика на ефективността.

На следващата графика ще сравним ускорението, което наблюдаваме при изчисляване на 5000 и 20 000 прецизност, изразени съответно чрез червената(долната) и синята(горната) линия.

Това са резултати от тестовете, като всеки ред е съответно: брой нишки, време за пресмятане, ускорение и ефективност. Горната таблица е за 5000, долната – 20 000.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 8 | 10 | 12 | 16 | 20 |
| 11681 | 8212 | 6415 | 5246 | 4369 | 3072 | 2457 | 2414 | 2377 | 2639 |
|  | 1.422431 | 1.820889 | 2.226649 | 2.67361 | 3.802409 | 4.754172 | 4.838857 | 4.914178 | 4.426298 |
|  | 0.711215 | 0.606963 | 0.556662 | 0.534722 | 0.475301 | 0.475417 | 0.403238 | 0.307136 | 0.221315 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 8 | 10 | 12 | 16 | 20 |
| 233303 | 150067 | 115409 | 91255 | 75781 | 47320 | 36451 | 32765 | 28824 | 27939 |
|  | 1.554659 | 2.021532 | 2.556605 | 3.078648 | 4.930325 | 6.400455 | 7.120494 | 8.094054 | 8.350442 |
|  | 0.777329 | 0.673844 | 0.639151 | 0.61573 | 0.616291 | 0.640046 | 0.593375 | 0.505878 | 0.417522 |