**ТЕМА:**

**„ Пресмятане на числото PI (3,14159…) чрез**

**формулата на Рамануджан. “**

**Дисциплина:**

„Системи за паралелна обработка“

**Изготвил:**

Aнтония Тодорова

**Съдържание:**

Цел на проекта -> (3 стр.)

Описание на алгоритъма за решаване на проблема -> (3 стр.)

Архитектура на приложението и реализация на

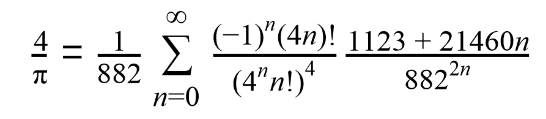
предложения алгоритъм -> (4 стр.)

Резултати и анализ -> (6 стр.)

**Цел на проекта.**

Пресмятане на числото PI с произволна висока точност, използвайки формулата за сходящ ред на Рамануджан.

Формулата има видя:



Да се напише програма за изчисление на числото PI, използвайки цитирания ред, която използва паралелни процеси (нишки) и осигурява пресмятането на числото PI със зададена от потребителя точност.

Функционалности:

* “-p” – задава броя на членовете, които да се включват в реда, като това определя точността на пресмятане на формулата
* “-t” or “-task” – задава броя на използваните нишки, на които е разделена работата на пресмятането на числото PI
* “-q” – задава тих режим на работа на програмата, при който се извежда само времето за работа отделено за изчисление на числото PI
* “-o” – задава изходен файл, в който се записва резултата от работата, а именно числото PI
* Програмата извежда подходящи съобщения на различни етапи от работата, както и времето отделено за изчисление на стойността на числото PI

**Описание на алгоритъма за решаване на проблема.**

Предложеното решение осъществява пълна имплементация на формулата за сходящ ред на Рамануджан, която служи за пресмятане на числото PI. В програмната логика са използвани няколко похвата за паралелна обработка, които да направят алгоритъма бърз и надежден.

За пресмятането на числото PI се използва тип BigDecimal, тъй като числата, които се получават са доста големи и не могат да се поберат в стандартните типове за променливи. В хода на програмата (при стартирането) се задават като параметри броя на използваните нишки и броя на членовете в сходящия ред. Възможни са още няколко параметъра, като име на изходен файл и възможност за тих режим, но те не отговарят толкова много за функционалността, а служат главно за извеждане на резултата. Алгоритъмът намира числото PI, спрямо зададените параметри, с точност десет хиляди цифри след десетичната запетая и връща този резултат, като окончателен отговор след прекратяване на работата си.

Формулата за сходящия ред се разделя на отделни части, за да се направи по-лесна и бърза за пресмятане, което ще рефлектира на цялостната функционалност на програмата. Първото нещо, върху което се фокусира програмата са отделни функции за пресмятане на различни компоненти от формулата. Една съставна част на сходящия ред е пресмятане на факториели. Тъй като не е желателно да се пресмята един и същи факториел във всяка нишка, то решението на този проблем е резултатът за отделните факториели да бъде записан в обща памет, която може да се достъпва от всички процеси. Възможно е нужната стойност да не бъде открита, като ако това се случи тя се изчислява моментално, а не се чакана за резултат от друга нишка. Функцията за пресмятане на 882^(2n) работи по аналогичен начин, като тази за факториелите. Това са част от нещата, които служат са пресмятане на всеки един елемент на сумата, в отделните процеси.

За улеснение на работата на отделните процеси, при стартирането на програмата се пуска нишка, която изчислява част от факториелите и ги записва в общата памет, предназначена за тази цел. Тази памет в последствие ще се използва от всички процеси. След това се стартират същинските нишки, върху които се прави join, за да се синхронизират, като така програмата ще работи значително по-бързо при по-голям брой членове на сумата. Всяка отделна нишка изпълнява пресмятане на съответния зададен член. Чрез параметрите се получава информация за броя нишки и броя членове, за които се реализира програмата. Изпълнява се цикъл, намиращ даден елемент на сходящия ред. Цикълът е със стъпка равна на броят на стартираните нишки, за да се избегне постоянното пресмятане на един и същи стойности от всяка нишка по отделно.

Накрая всички членове се пращат на функция, която има за задача да прибави дадения член към резултата, равен на намерената сума до момента. Поради синхронизацията на това добавяне няма да възникнат грешки в последователността на паметта. След като всички нишки приключат работата си, то имаме резултата от 0-вия до n-тия член на формулата. На този етап вече се извършва изчислението на PI и се записва във файла. Програмата завършва като се засече времето на работа.

**Архитектура на приложението и реализация на предложения алгоритъм.**

Реализацията на алгоритъма и паралелизирането на изчисляването е на езика за програмиране Java. Програмата се състои от един клас, наречен MainPI, като в него се изчислява числото PI, зададенo от сходящия ред на Рамануджан. Основата от която се започва в този алгоритъм е намиране на даден брой факториели и степенувания, които се записват в обща памет, така че тя да може да бъде достъпвана от всички нишки. Това става възможно, чрез функциите BigDecimal calculateFactoriel(int n) и BigDecimal calculateGradation(int n). Те са рекурсивни функции, които намирам n-тия факториел или съответно 882 на 2\*n-та степен и ги записват в съответните им глобални масиви, които са BigDecimal[] factorial и BigDecimal[] gradation. Това са основните части, чието разделение ще доведе до ефективно ускоряване на алгоритъма. След като бъде подготвена общата памет се стартират нишки за паралелна обработка на данните, чиито брой е подаден от командния ред чрез параметър. Всяка от нишките смята отделен елемент на сумата от формулата. Как да се случи това е оказанo във функцията Runnable runPI(int start, int end, int numberThreads, boolean x), която всяка нишка изпълнява при стартирането си. Тази функция връща обект от тип Runnable, като вътре в него има предефинирана функция void run(). Тя получава четири параметъра, които са int start (от кой член започва сумата), int end (брой членове, за които се стартира програмата), int numberThreads (брой на използваните нишки) и boolean x (дали функцията да работи в тих режим). Те са важни, за да определят какво точно трябва да направи всяка отделна нишка. Главното нещо е, че се изпълнява цикъл, намиращ даден елемент на реда, като той е със стъпка равна на броя на стартираните нишки. С този похват се избягва постоянно пресмятане на едни и същи стойности от всяка нишка поотделно. Така общата работа, която трябва да се извърши за пресмятане на сумата от формула, е разпределена горе долу по равно (обем и трудност) между всички стартирани нишки. След това, намерената стойност от функцията се изпраща до функцията synchronized void assignResult(BigDecimal pi), която от своя страна има за задача да прибави дадения член към общия резултата, който е равен на сумата до момента. Накрая се изчислява числото PI, на база вече получените резултати и се записва във файл със функцията void saveToFile(String filename, BigDecimal pi).

Всичко това се управлява от главната функция void main(String[] args) за стартиране на класа.

Помощни средства, които се използват от програмата за успешно реализиране на алгоритъма са BigDecimal за работа с много големи числа, MathContext за закръгляне и за прецизност при деление на много големи числа, Arrays за създаване на вектора от параметри и работата с тях и System.currentTimeMillis() за изчисляване на времето на работа.

**Резултати и анализ.**

На предоставената машина, за извършване на изпитване на програмата, са проведени тестове за намиране на стойността на числото PI, спрямо формулата на Рамануджан.

Броят на членовете на сходящия ред, който се използва в конкретния случай на тестване, е 5000. Разглеждат се брой стартирани нишки от 1 до 16.

Изчисленията, които се получават ни дават информация за времето на работа в ms, ускорението Sp и ефективността Ep.

|  |  |  |  |
| --- | --- | --- | --- |
| **Брой ядра (нишки)** | **Време в ms** | **Ускорение Sp** | **Ефективност Ep** |
| **1** | 113216 | 1 | 1 |
| **2** | 57226 | 1.9784014259 | 0.9892007129 |
| **3** | 39100 | 2.8955498721 | 0.9651832907 |
| **4** | 29141 | 3.8851103256 | 0.9712775814 |
| **5** | 24107 | 4.6963952378 | 0.9392790475 |
| **6** | 20339 | 5.5664486946 | 0.9277414491 |
| **7** | 17640 | 6.4181405895 | 0.9168772270 |
| **8** | 15797 | 7.1669304298 | 0.8958663037 |
| **9** | 14609 | 7.7497433089 | 0.8610825898 |
| **10** | 13480 | 8.3988130563 | 0.8398813056 |
| **11** | 12076 | 9.3752898310 | 0.8522990755 |
| **12** | 11o49 | 10.2467191601 | 0.8538932633 |
| **13** | 10434 | 10.8506804677 | 0.8346677282 |
| **14** | 9969 | 11.3568060989 | 0.8112004356 |
| **15** | 9712 | 11.6573311367 | 0.7771554091 |
| **16** | 9363 | 12.0918509024 | 0.8061233934 |

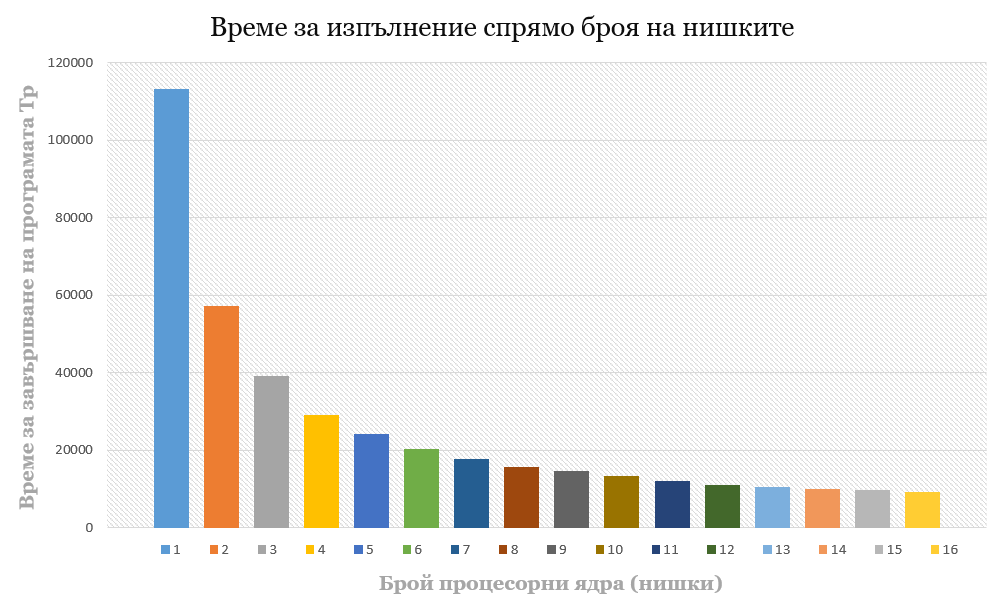
Забелязва се, че с увеличаване на броя процесорни ядра, които са използвани се намалява времето на изпълнение на задачата. Също така ускорението се увеличава, а ефективността намалява.

**Графика на времето спрямо броя нишки.**

Показва времето необходимо за пресмятане на числото PI по формулата на Рамануджан с точност до 5000 члена, участващи в сумата, при употреба на различен брой процесорни ядра, започвайки от едно ядро (серийната версия на програмата) до използване на шестнадесет ядра на предоставената машина.

Т1 – времето за изпълнение на програмата от една нишка

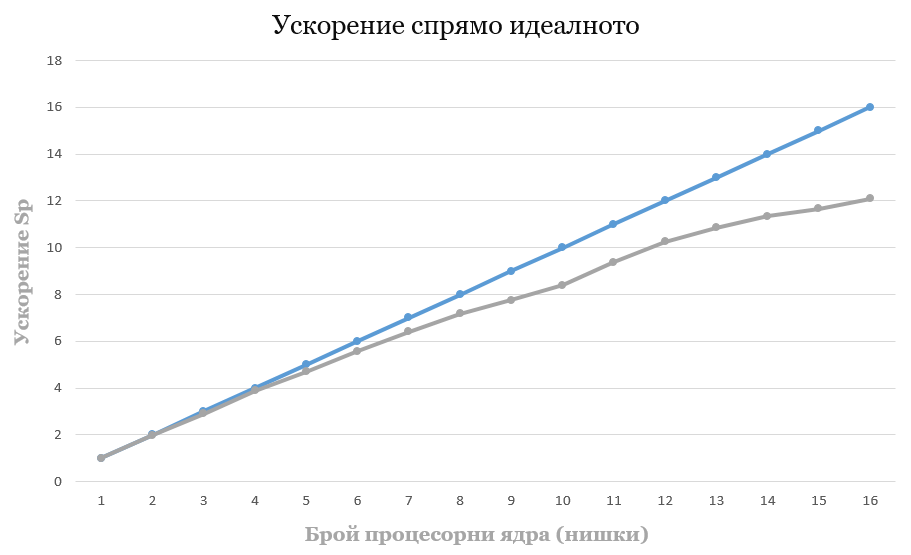
Тр – времето за изпълнение на програмата от р нишки



**Графика на ускореното спрямо броя нишки.**

Показва постигнатото ускорение при пресмятане на числото PI по формулата на Рамануджан с точност до 5000 члена, участващи в сумата, при употреба на различен брой процесорни ядра, започвайки от едно ядро (серийната версия на програмата) до използване на шестнадесет ядра на предоставената машина.

Sp = T1/Tp - ускорение



**Графика на ефективността спрямо броя нишки.**

Показва постигнатото ефективност при пресмятане на числото PI по формулата на Рамануджан с точност до 5000 члена, участващи в сумата, при употреба на различен брой процесорни ядра, започвайки от едно ядро (серийната версия на програмата) до използване на шестнадесет ядра на предоставената машина.

Ep = Sp/p = T1/Tp.p – ефективност

