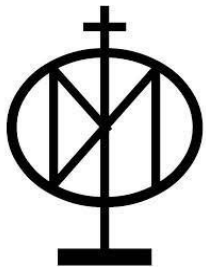
СУ “Св. Климент Охридски”
ФМИ – Софтуерно инженерство
Разпределени софтуерни архитектури



Курсов проект

Пресмятане на Pi – Monte Carlo

Изготвил:
Валентин Кирилов Кирилов
№ 61701, курс 3, група 3
Софтуерно инженерство

Проверил:

……
(ас. Христо Христов)

София, 12.6.2016

Съдържание

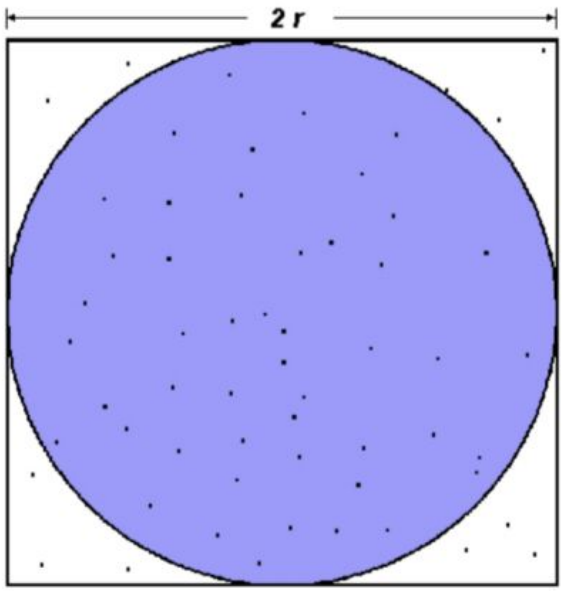
Условие на курсовата задача

Числото (стойността на) Pi може да бъде изчислено по различни начини. Разглеждаме следния метод за приближено пресмятане на Pi:

1) Разглеждаме окръжност, вписана в квадрат;
2) Генерираме по произволен начин точки в рамките на квадрата;
3) Определяме броя на точките, които се намират в окръжността;
4) Нека k е число равно на броя на точките в окръжността, разделен на броя на точките в квадрата;
5) Тогава: Pi ~ 4.0 * k;

Бележка: Колкото повече точки генерираме толкова по-добра ще бъде точността с която пресмятаме Pi, въпреки че трудно можем да я сравним с тази постигана при използването на сходящи редове;

Визуално, можем да представим метода със следната картинка:



$$A_S = (2r)^2 = 4r^2$$
$$A_C = \pi r^2$$
$$\pi = 4 \times \frac{A_C}{A_S}$$

Разглеждаме сериен (последователен) псевдокод реализиращ този метод:

```
=== cut ===
npoints = 10240
circle_count = 0

do j = 1, npoints
       generate 2 random numbers between 0 and 1
       xcoordinate = random1
       ycoordinate = random2
       if (xcoordinate, ycoordinate) inside circle
       then circle_count = circle_count + 1
end do

Pi = 4.0*circle_count/npoints
=== cut ===
```

Както се вижда от кода, по-голямата част от времето за изпълнение на програмата ще премине в изпълнение на операциите от цикъла.

Изисквания към програмата

Вашата задача е да напишете програма за изчисляване на числото Pi, по описания метод, която използва паралелни процеси (нишки). Изискванията към програмата са следните:

(o) Размерността на квадрата се задава в точки (пиксели) от подходящо избран команден параметър – например **“-s 10240”**;

(o) Точките в квадрата, генерираме произволно с помощта на **Math.random()**, (т.е. класа **java.util.Random**) или **java.util.concurrent.ThreadLocalRandom**;

Разликата между двата начина - **Math.random()** е достъпен във всички версии на Java и ужасно бавен. Не е проектиран за работа в много-нишкова (multi-threaded) среда; В Java 7 и 8, разполагаме с **ThreadLocalRandom**, който е специално проектиран за работа в рамките на отделен thread (респективно multi-threaded среда);

Ще бъде много интересно да реализирате програма, използваща и двата начина и съответно получим два вида резултати от работата на програмата;

(o) Друг команден параметър задава максималния брой нишки (задачи) на които разделяме работата по пресмятането на Pi – 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;

Примери за подходящи съобщения:
„Thread-<num> started.“,
„Thread-<num> stopped.“,
„Thread-<num> execution time was (millis): <num>“,
„Threads used in current run: <num>“,
„Total execution time for current run (millis): <num>“ и т.н.;

(о) Да се осигури възможност за **„quiet“** режим на работа на програмата, при който се извежда само времето отделено за изчисление на Pi (и самото число), отново чрез подходящо избран друг команден параметър – например **“-q”**;

Не е задължително „сляпо“ да следвате логиката на цитирания последователен код; Възможно е първо да решите задачата с генерирането на точки, след което задачата с определянето на това кои от тях принадлежат на окръжността. И въпреки че подобен подход не е много ефективен от гледна точка на памет, няма ограничения за количеството, което използва програмата Ви ;);

Забележка: (о) В условието на задачата се говори за разделянето на работата на две или повече нишки.

Работата върху съответната задача, в случаят в който е зададен **„-t 1“** (т.е. цялата задача се решава от една нишка) ще служи за еталон, по който да измерваме евентуално ускорение (т.е. това е **T1**). В кода реализиращ решението на задачата 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 използването на **„p“** нишки **–** намираме **Tp** и съответно можем да изчислим **Sp**. Представените на защитата данни за работата на програмата, трябва да отразят и ефективността от работата и, тоест да се изчисли и покаже **Ep**.

Като обобщение - данните събрани при тестването на програмата Ви, трябва да отразяват **Tp**, **Sp** и **Ep**. Желателно е освен табличен вид, да добавите и графичен вид на **Tp**, **Sp**, **Ep**, в три отделни графики.

Описание на курсовия проект

За да се реализира пресметянето на числото π последством алгоритъма на Монте Карло е реализирана програма на Java. Тя използва подход, който разпределя изчисленията на няколко нишки, чрез което се постига по-голяма ефективност и съответно - ускорение при извършването им. За тази цел са реализирани три основни класа - class TaskRunner, class PiThread и class Point:

Task Runner

Класът TaskRunner е основният за програмата и служи както за нейното стартиране, така и за организацията по цялото и изпълнение. Той е разделен на няколко метода, на които е отредено да извършат определени етапи от изпълнението на програмата.

Методът **readInput()** се грижи за прочитането на подадените при стартирането на програчата аргументи и съответно за тяхната обработка от страна на програмата. Работата на програмата може да се контролира посредством следните параметри:

- -q - ако този аргуметнт е подаден, то програмататрябва да работи в тих (quiet) режим. Това ознава, че след нейното изпълнение на стандартния изход ще бъде налична единствено информация за времето, което е отнело за изпълнението на програмата.
- *-s* - размер на страната на квадрата, в който е вписана окръжността, която служи за изчисленията на програмата. Въведената стойност трябва да бъде цяло число.
- -t - броят на нишките(процесите), които искаме да бъдат използвани от програмата за изчислението на числото π. Очакват се единствено целочислени стойности.

Методът **configure()** е изключително важен за програмата. Той валидира прочетените входни данни и в зависимост от това дали те са валидни продължава изпълнението на програмата или го прекратява с подходящо съобщение за грешка. Ако въведените данни са валидни, то се правят основните конфигурации, които са необходими за изпълнението на програмата. Задава се брой точки, които трябва да бъдат генерирни, създават се желаният брой нишки и се изчислява по колко точки трябва да генерира всяка една от тях. Ако не е избрано от потребителя изпълнението на програмата да протече в quite режим, то всяка една от нишките изписва на стандартния изход кога започва изпълнението си, кога завършва работа и за какво време е протекло изпълнението й.

Методът **run()** служи за синхронизиране на резултатите от отделните нишки(процеси) и на база

на получените данни изчислява приближение до числото π. След края на изпълнението, на стандартния изход се изписва времето, което е отнело за изпълнение на програмата.

PiThread

Класът PiThread е отговорен за създаването и управлението на нишките(процесите) в програмата. Той онаследява стандартния в Java клас Thread от пакета java.lang. Интересното в този клас е методът run(), в който е написано реално какво трябва да направи всяка нишка по време на изпълнението си.

Основно, изчислението се състои в това, че всяка нишка генерира определен брой точки с произволни координати в рамките на предварително зададен квадрат и проверява колко от тях се намират в неговите очертания и колко не.

За генерирането на произволни числа (при задаването на координати на точките) се използва един от стандартните за Java пакети - java.util.random. Работата на тази дейност е изнесена в отделен метод - getRandomNumber(), който очаква два аргумента, с които се спесифицира интервал, в който да се генерира произволното число.

За проверката на това дали дадена точка се намира в рамките на квадрата е предвиден специален метод isPointInCircle(), който приема два входни аргумента - радиус на кръга и самата точка, за която трябва да се извърши проверката.

В заключение, работата на нишката се състои в това да генерира голям брой точки с произволни координати и да провери дали те се намират в рамките на окръжност и ако да, да запомни техният брой.

Point

Класът Point служи за абстрактното представяне на точка от двумерната координатна система. Той е най-простият клас на програмата и се състои от две член променливи отговарящи за x и y, съответно отместването спрямо абцисната и ординатната оси.

Класът предоставя getter-и и setter-и за свойте член променливи, както и конструктор, който приема като аргументи x и y, с които установява техните стойности.

Диаграми на ускорението и ефективността

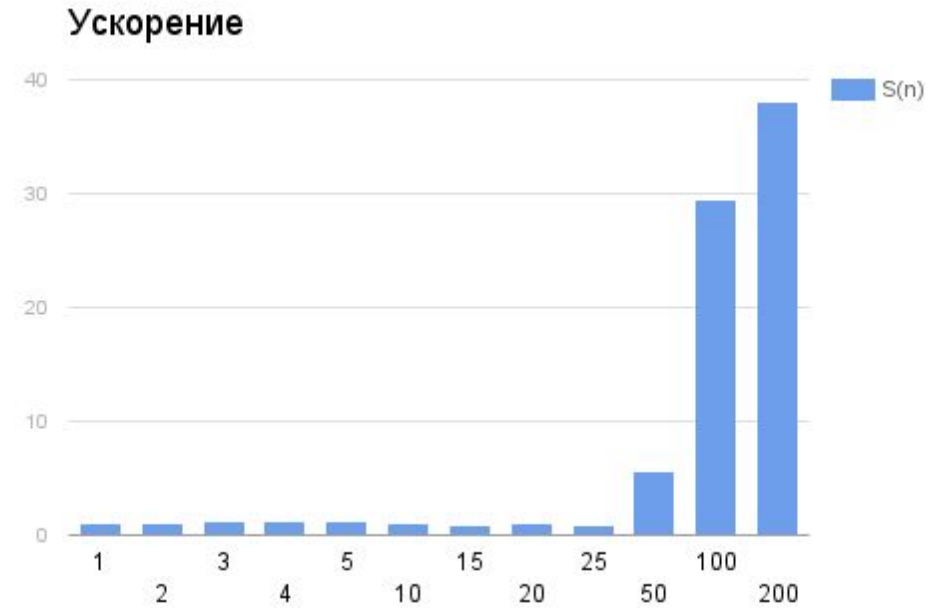
Тестването на проекта е извършено на 2 ядрен процесор като изследването е направено с цел да се оцени ускорението (S) и ефективността (E) на описания и реализирания алгоритъм. За изчислението им са използвани следните формули, където **n** е броят на използваните нишки, а **Tn** е времето необходиво за изпълнение на програмата използвайки **n** а брой нишки:

$$S(n) = \frac{T_1}{T}$$

$$E(n) = \frac{S(n)}{n}$$

Време за изпълнение

Ускорение



Ефективност