Софийски университет „Св. Климент Охридски”

Факултет по математика и информатика

Курсов проект

Системи за паралелна обработка

Тема: *„Пресмятане на числото Pi-Chudnovsky”*

Владислава Венциславова Маркова, ф.н. 81271, спец. КН, поток II, група 6

Научен ръководител: проф. Васил Цунижев, ас. Христо Христов

Дата: 26.06.2020 г. ***Проверка: ……..***

СЪДЪРЖАНИЕ

1. Цел на проекта

2. Изисквания към програмата

3. Анализ на методите за решаване на задачата

4. Описание на алгоритъма

5. Реализация

6. Проведени тестове и измервания

1. Цел на проекта

Целта на проекта е реализация на паралелен алгоритъм, пресмятащ числото **Pi**(числото **Пи**) чрез сходящ ред с произволно висока точност(по формулата на братята Chudnovsky). Формулата, използвана за сходящ ред: 1/**pi**=12*∑(((1)^n*(6*n)!)/(3*n)!*((n)!)^3)*(13591409 + 545140134*n)/(6403203)^(n + ½)), където n = 0, … , ∞. Същата тази формула на 29.01.2020 г. е използвана за пресмятане на 50 трилиона цифри на числото Пи.

2. Изисквания към програмата

- Използване на паралелни процеси (нишки), за да се разпредели работата по изчислението на числото ***Pi***
- Наличие на команден параметър, който да задава точността на пресмятанията (брой цифри след десетичната запетая или брой членове на реда)
- Наличие на команден параметър, който да задава максималния брой нишки(задачи), на които да се разпределя работата по изчислението на числото ***Pi***

- Да се извеждат подходящи съобщения на различните етапи от работата на програмата, както и времето, отделено за изчисление, и резултата от изчислението (стойността на ***Pi***)
- Програмата да записва резултата от работата си (стойността на ***Pi***) в изходен файл, зададен с подходящ параметър. Ако тази параметър е изпуснат, се задава име по подразбиране
- Да се осигури възможност за *“quiet”* режим на работа на програмата, при който се извежда само времето отделено за изчисление на ***Pi***, отново чрез подходящо избран друг команден параметър

3.Анализ на методите за решаване на задачата

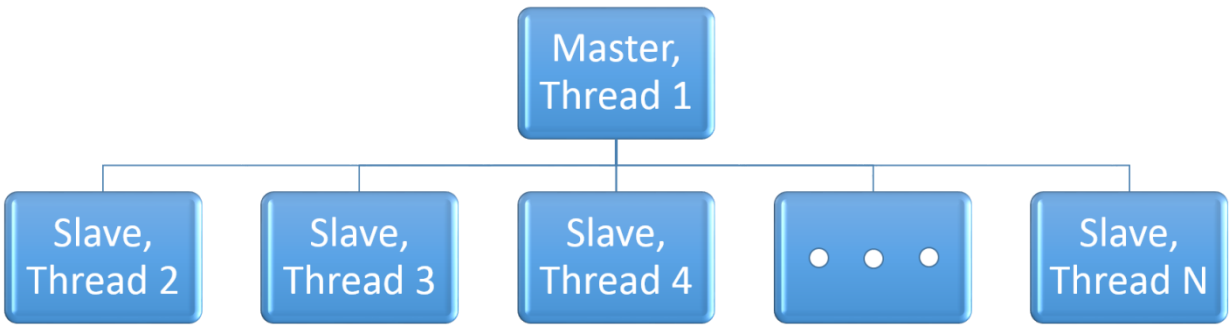
В алгоритъма, използван за пресмятане на сходящия ред, е включено разпределяне на членовете между отделни подпрограми (нишки) и съответно сумиране на резултатите. Нека обърнем внимание на две възможни архитектури и връзката им със синхронизацията между подпрограмите:

- ***Master-Slave*** е йерархична ***архитектура***, при която се избира една подпрограма, която влиза в ролята на Master и всички останали се определят като Slave. Подпрограмата в ролята на Master отговаря за разпределянето на работата, стартирането и завършването и на останалите подпрограми. В съответния случай в Master се извършва крайното сумиране на резултатите от всички подпрограми, в ролята на Slave, което е именно синхронно изчисление.
- ***Потоковата архитектура(Data Flow)*** с канали и филтри включва в характеристиката си поредица от подпрограми, където определена подпрограма получава резултата от изчислението от предишна подпрограма, извършва своята част от изчислението на сумата, сумира временния резултат и го предава на следващата подпрограма.

Един адекватен подход при съставянето на архитектурата би бил да заимстваме от „плюсовете” на тези два типа архитектура. В самото начало на изчислението ползваме единствена главна програма, отговаряща за създаването на отделните подпрограми и разпределянето на работата. За целта ни трябва начин, по който лесно да стартираме и конфигурираме всички подпрограми. Тук принципът на Master-Slave архитектурата е удачен и удобен. След стартирането се стремим към това да се освободим от Master-възела(„тясното място”), вече прилагайки принципа на потоковата архитектура за осъществяване на желаното предаване на резултата между съседните възли. Така постигаме намаляване на серийната работа на програмата и стигане до резултат, който е именно сумата, получена от последната подпрограма.

Ако трябва да обобщим сравнително характеристиките на двата типа архитектури, то можем да заключим следното:

- **Master-Slave** – удобно централизирано управление, серийно пресмятане
- **Потокова(Data Flow)** – задтруднен контрол, поради липса на централизация, паралелно извършване на междинните пресмятания



4.Описание на алгоритъма

Програмата, реализираща алгоритъма за изчисление на числото **Pi** чрез сходящ ред, открит от братята Chudnovsky, позволява на потребителя да изчисли стойността на ***Pi*** като определи броя цифри след десетичната запетая. Идеята на алгоритъма е всяка нишка да извършва изчисления с еднаква сложност, за приблизително еднакво време. Това се случва при итерацията на параметъра i при пресмятането на членовете, където се задава стъпка, която е броят нишки, които потребителят е задал при стартирането на програмата. Крайният резултат е сумата от работата на всяка нишка и това е стойността на числото **Pi**.

5.Реализация

Реализацията на алгоритъма и изчисляването на числото ***Pi*** е на езика Java. Използван е моделът за паралелизъм по данни, тоест Single Program-Multiple Data(SPMD). Използва се комуникация чрез споделена памет.

В началото заданието приема точността, която се търси, в брой цифри, след което се случва преобразуването към брой членове на реда, които да бъдат пресметнати.

Разпределянето на данните(декомпозицията) се извършва по зададен брой разполагаеми нишки threadStep, или по-точно – всички членове на реда, които трябва да бъдат пресметнати, се разделят на threadStep равни части от последователни членове.

Така се използва най-едрата възможна **грануларност**. **Грануларността** е делене на обработката. Тя може да бъде едра, средна и фина. Към едрата грануларност се отнасят

независимите задания и програми, ниво на ОС; подпрограми, ниво на мултипрограмирането.

Използвано е статично балансиране, по-точно при стартирането на всяка подпрограма бива зададен интервала от членове, чиято сума трябва да бъде пресметната. Изборът на едра грануларност влече това като естествено следствие.

При статичното балансиране разпределянето на заданията по възли и алоцирането на ресурси се извършва(и е известно) преди паралелната обработка – планиране, комплементиране(mapping, matchmaking, scheduling).

Важни свойства са и *линейност(linearity)* – стремеж ускорението да бъде плътно до линията HW, както и *мащабируемост(scalabilty).* /HW/SW – хардуерен/софтуерен паралелизъм/

С конфигурирането и стартирането на подпрограмите завършва първата част от алгоритъма, построена по Master-Slave архитектурата.

Втората част представлява паралелното изчисление на сумите в отделните нишки и събирането на отделните резултати. Използваме потокова архитектура с канали. Всяка подпрограма при стартирането си започва итериране през членовете на реда от зададения и интервал и смята междинната сума. Подпрограмата, която смята най-големия интервал, води до получаването на крайния резултат от пресмятането на числото Пи. За по-малките интервали пресмятанията на сумата от членовете се извършват по-бързо, но това се компенсира от времето, необходимо за събиране на резултата от предходния интервал.

Един начин за стартиране на програмата, след като тя е компилирана е например:

java SPO_proekt_81271/MainApp –p 100 –t 3.

Ще стартира програмата с три нишки за пресмятането на сто члена от формулата.

Главните функции, използвани за реализирането на алгоритъма са:

=== Записваме в променливи подадените параметри, съответно за точност на пресмятането на числото Pi и брой нишки, стига да са подадени коректно. След това ги подаваме на главния метод getRunner

Пример: Ако искаме да пресметнем стойността на числото Пи при зададени 2 на брой нишки и точност на пресмятането 5 000, тогава при итерациите на for-цикъла първата нишка ще пресметне резултата за k=(0, 2, 4, 6, 8, 10, …, 5 000), тъй като стъпката е зададена да е броя нишки, в случая 2. А пък втората нишка ще пресметне съответно резултата за k=(1, 3, 5, 7, 9, …, 4 999), отново със стъпка 2 и итерация, започваща от индекса на самата нишка.

=== Методът addResult викаме в метода getRunner, за да се добави текущият резултат от пресмятането на числото Пи. Този метод е синхронизиран, обновява стойността на глобалната променлива result, в която се записва финалния резултат. Тя е от тип BigDecimal, поради изискването за висока точност на пресмятането.

.=== В класа **Pi_number** използваме методът factorial за пресмятането на сходящия ред в метода getRunner, като factorial пресмята факториела на зададеното като аргумент на метода число

=== Извикваме метода **serializePiNumber** на класа **Pi_number** след приключването на работата на нишките по изчислението на числото ***Pi***, след което записваме стойността на

глобалната променлива **result** във файл с име **fileName**, което е зададено като втори аргумент на този метод

=== В метода **getRunner** се пресмята числото ***Pi***, самият метод връща обект от тип Runnable. За тази цел методът има 2 параметъра **from** и **to**, които са от тип **int** и чрез тях се задава една нишка от къде до къде ще се пресмята. Параметърът **threadStep** е от тип **int** и е стъпката на итерация на for-цикъла, като той е зададения от потребителя брой нишки. Това е най-важният параметър, който служи за равномерното разпределение на сложността на работа на всяка от нишките, за да се извършва тяхната работа за достатъчно близко време. Методът getRunner викаме в **main**-метода, когато създаваме и стартираме самите нишки.
Параметърът **quietMode** на метода **getRunner** е от тип **boolean** и чрез него се определя дали режимът на изпълнение на програмата ще е такъв, в зависимост от подадения команден параметър от потребителя. Ако го има, тоест е true, се извежда съобщение на конзолата, което включва само резултата и общото време, което е необходимо за завършването на програмата. В алтернативния режим, когато **quietMode** не е зададен, има стойност false, програмата изкарва същата тази информация, но и информация за работата на всяка нишка.

6. <u>Проведени тестове и измервания:</u>

Върху предоставената машина за изпълнение на тестове са проведени тестове за изчислението на числото ***Pi*** със сходящия ред(Chudnovsky), като зададената точност на изчислението е **n** = 5000. Събраните данни са представени в следната таблица:

Брой нишки	**Време в ms**	**Действително ускорение**	**Ефективност**	**Линейно ускорение**

1	1005728	1	1	1
2	598320	1,680919909	0,840459955	2
3	406997	2,471094382	0,823698127	3
4	319474	3,148074648	0,787018662	4
5	256004	3,928563616	0,785712723	5
6	211278	4,760211664	0,793368611	6
7	187046	5,376901939	0,768128848	7
8	172997	5,813557461	0,726694683	8
9	151633	6,632645928	0,736960659	9
10	141831	7,091030875	0,709103087	10
11	138791	7,246348827	0,658758984	11
12	132563	7,586792695	0,632232725	12
13	125303	8,026368084	0,61741293	13
14	120559	8,342205891	0,595871849	14
15	114566	8,778590507	0,585239367	15
16	113339	8,873626907	0,554601682	16

Метрика и анализ на производителността:

Използваме следните формули за пресмятане на ускорение и ефективност:

Tp = време за изпълнение на програмата с p на брой нишки

Ускорение: Sp = T1 / Tp = ускорение на програмата при използвани p на брой нишки

Ефективност: Ep = Sp / p е ефективност(нормирана стойност на ускорението) на програмата при p на брой нишки

Редно е Sp да е по-голямо от 1. Най-добрият случай е, включвайки n процеса, да намалим времето n пъти. В този случай линията HW е на ъгъл 45 градуса. При стойности над HW се наблюдават аномалии, каквито тук няма, под нея – наложени са ограничения.

Графиката на ускорението се подчинява на Закона на Amdahl(1967) – от даден момент нататък няма как да получаваме по-добро ускорение.

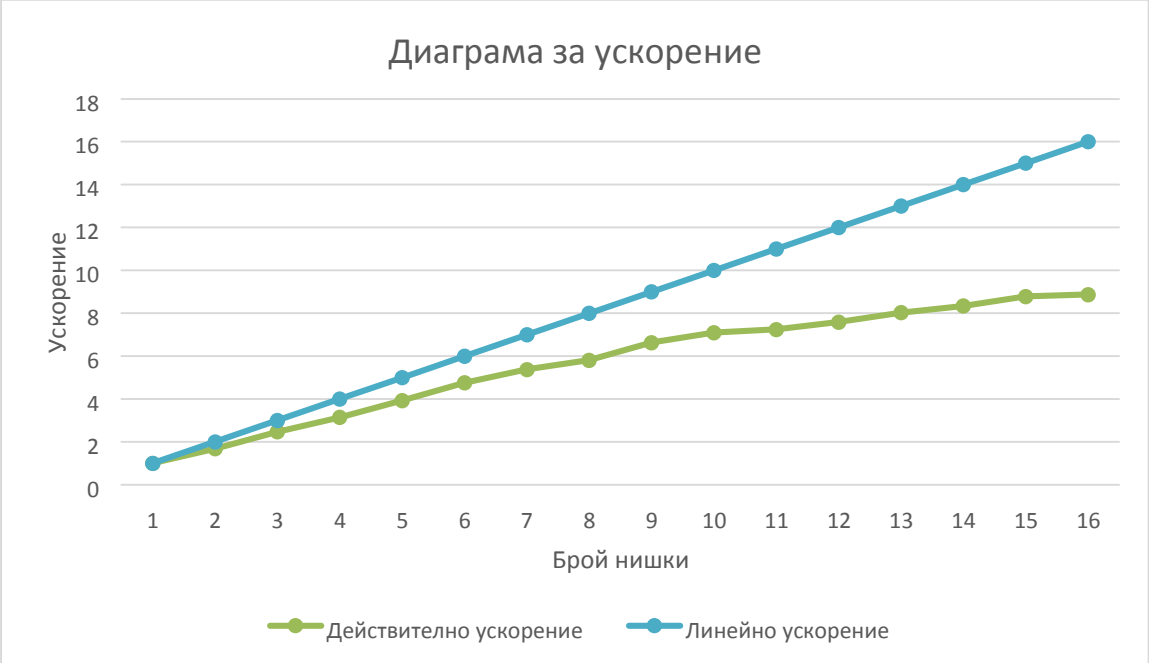
(При наличие на две интензивности в отношение f:(1-f) по брой на генерирани резултати/междинни и крайни/ общата инетнзивност на обработка е R(f) = (f/Rh+(1-f)/Rl) ,така следва, че при f->1 и при f->0, R(f)->и R(f)->Rl).

Като допълнение ще споменем още няколко параметъра при обработката на паралелен алгоритъм

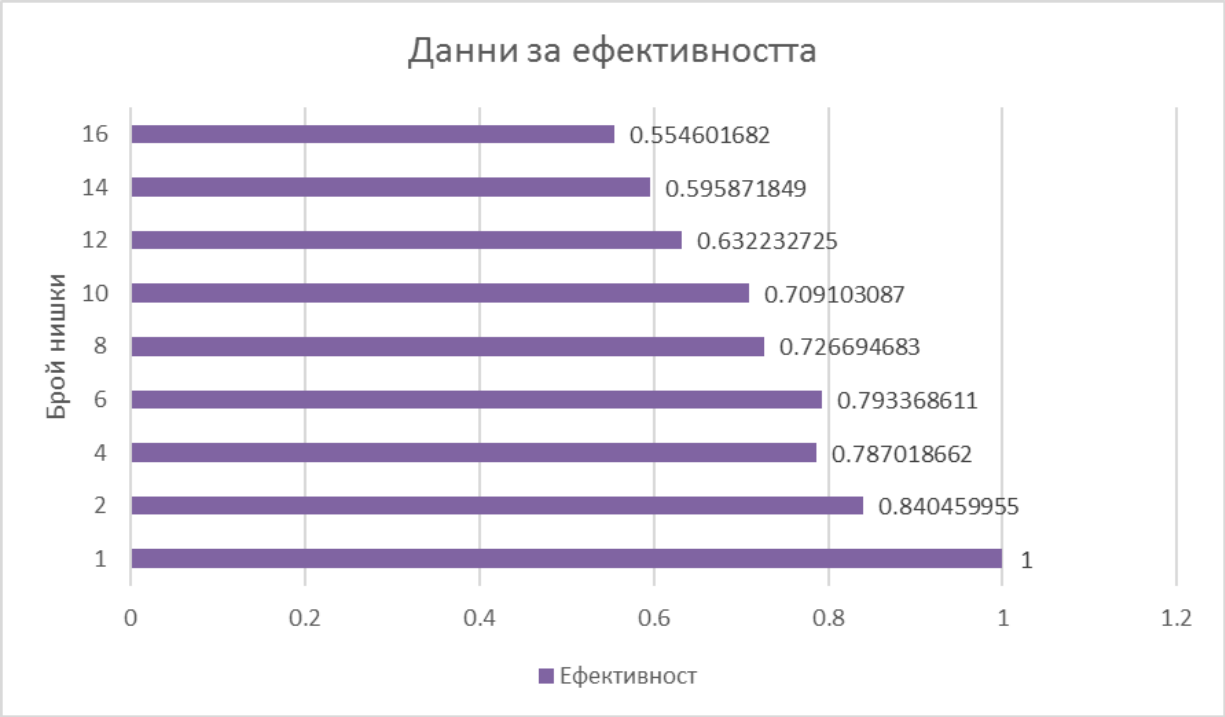
- цена(cost) – Cp=Tp*p, тоест максималният брой операции, които биха могли да се извършат за обработка на съответния паралелен алгоритъм
- коефициент на използване(utilization) – отношението на действителните към потенциалните операции, Up=Op/Cp(Op/(Tp*p))
- темп на обработка(execution rate) – представя се с няколко скали: MIPS, MFLOPS, MOPS и LIPS, архитектурно зависим параметър е.
- излишък(redundancy/(system)overload) – Rp=Op/O1 > 1, критерий е за свръхтовара, който се поражда при паралелната обработка на алгоритъма /Op-броят на операциите при обработка с p процесора, O1-броят операции при обработка на уникомпютър/
- Алгоритмична сложност – коректността на даден паралелен алгоритъм е архитектурно независима, но ефективността зависи и от изпълнителната платформа, поради това е целесъобразно сложността му да се

оценява и като функция на разпределянето(mapping). Алгоритмичната сложност O оценява времевите и пространствените характеристики на обработка – вр. сл.: задава се в брой елементарни операции и комуникации, простр. сл.: брой алоцирани регистри и клетки памет. Оценката обикновено се дава като горна и долна граница на тези величини или с приближение – асимптотична сложност. Една добра сложност, която се постига за пресмятане на числото Пи е O(n*((lg n)^3)).

- **Диаграма, показваща постигнатото ускорение спрямо брой нишки:**



- **Диаграма, показваща постигнатата ефективност спрямо броя нишки:**



Източници:

- http://rmi.yaht.net/
- https://learn.fmi.uni-sofia.bg/course/view.php?id=5928, лекции - Курс по СПО - летен семестър 2019/2020, проф. Васил Цунижев
- https://en.wikipedia.org/wiki/Chudnovsky_algorithm, Chudnovsky algorithm
- https://stackoverflow.com/questions/46166389/recursivechudnovsky-algorithm-in-java, Recursive Chudnovsky Algorithm in Java
- Schmidt, Bertil, Jorge Gonzales-Dominguez, Christian Hundt, Moritz Shlarb, Parallel Programming: Concepts and Practice, Kindle, 2018