



Софийски университет „Св. Климент Охридски“

Факултет по математика и информатика

Курсов проект

ПО

Разпределени софтуерни архитектури

летен семестър, учебна година 2019/2020

Тема 13

Пресмятане на P_i – Chudonovsky

Ръководители: проф. Васил Цунижев, ас. Христо Христов

Изготвил: Ивайло Иванов, фн. 62133, Софтуерно инженерство, 3 курс

Дата: 15.07.2020г.

Съдържание

| | |
|--|----|
| Увод | 3 |
| Поставената задача | 3 |
| Цел на проекта | 3 |
| Анализ на различни решения на поставената задача | 3 |
| Източник [1] | 3 |
| Източник [3] | 4 |
| Източник [4] | 4 |
| Източник [5] | 5 |
| Източник [6] | 5 |
| Източник [7] | 5 |
| Проектиране | 5 |
| Описание на реализирания алгоритъм | 5 |
| Подобрение на реализирания алгоритъм | 5 |
| Разделяне на задачите | 6 |
| Диаграма на класовете | 6 |
| Диаграма на дейностите | 7 |
| Тестване | 8 |
| Архитектура на тестовата машина | 8 |
| Тестов план | 8 |
| Тестови резултати | 8 |
| Тест 1 | 8 |
| Тест 2 (подобрен алгоритъм) | 9 |
| Анализ на резултатите от тестовете | 12 |
| Източници | 13 |
| Таблица на фигурите | 13 |

Увод

Поставената задача

Числото (стойността на) π може да бъде изчислено по различни начини. Използвайки сходящи редове, можем да сметнем стойността на π с произволно висока точност. Един от бързо сходящите към π редове е този, открит от индийския математик Srinivasa Ramanujan през 1910-1914 година. През 1987 братята Чудоновски(Chudonovsky) откриват ред с още по голяма сходимост. Редът има вида:

$$\frac{1}{\pi} = 12 \sum_{k=0}^{\infty} \frac{(-1)^k (6k)! (13591409 + 545140134k)}{(3k)! (k!)^3 640320^{3k+3/2}}$$

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

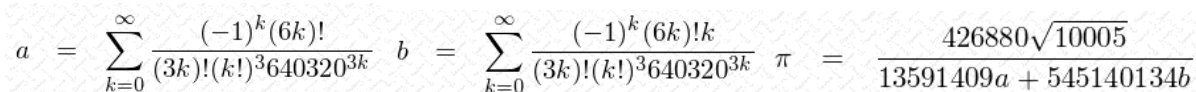
Цел на проекта

Настоящият курсов проект има за цел да разгледа и анализира вече съществуващи решения на поставената задача, също така и да се създаде програма, решаваща задачата.

Анализ на различни решения на поставената задача

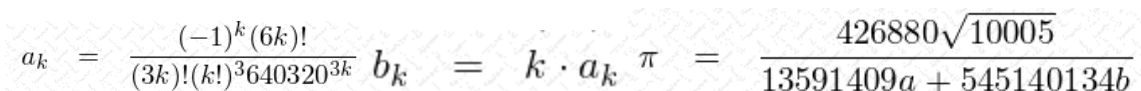
Източник [1]

В този източник е разгледана рекурентната редица на братята Чудоновски и последователен алгоритъм изчисляващ числото π написан на езика Python. Този източник се фокусира върху оптимизиране на алгоритъма. Едно от подобренията е редът да бъде разделен на няколко суми.


$$a = \sum_{k=0}^{\infty} \frac{(-1)^k (6k)!}{(3k)! (k!)^3 640320^{3k}} \quad b = \sum_{k=0}^{\infty} \frac{(-1)^k (6k)! k}{(3k)! (k!)^3 640320^{3k}} \quad \pi = \frac{426880\sqrt{10005}}{13591409a + 545140134b}$$

Фиг. 1 - Разделане на реда на два реда

Друго подобрение е пресмятането на реда като рекурентен ред.


$$a_k = \frac{(-1)^k (6k)!}{(3k)! (k!)^3 640320^{3k}} \quad b_k = k \cdot a_k \quad \pi = \frac{426880\sqrt{10005}}{13591409a + 545140134b}$$

Фиг. 2 - Рекурентна зависимост на членовете на редовете

Едно от подобренията на програмата е използването на Binary Splitting. Това е метод за забързване на изчисленията при пресмятането на редове като редът на братя Чудоновски. Този метод преобразува редът от сума на дроби в дроб от суми, тоест се извършва следното преобразуване:

$$S(a, b) = \sum_{n=a}^b \frac{p_n}{q_n} \longrightarrow S(a, b) = \frac{P(a, b)}{Q(a, b)}$$

Фиг. 3 - Преобразуване на ред от дроби в дроб от редове

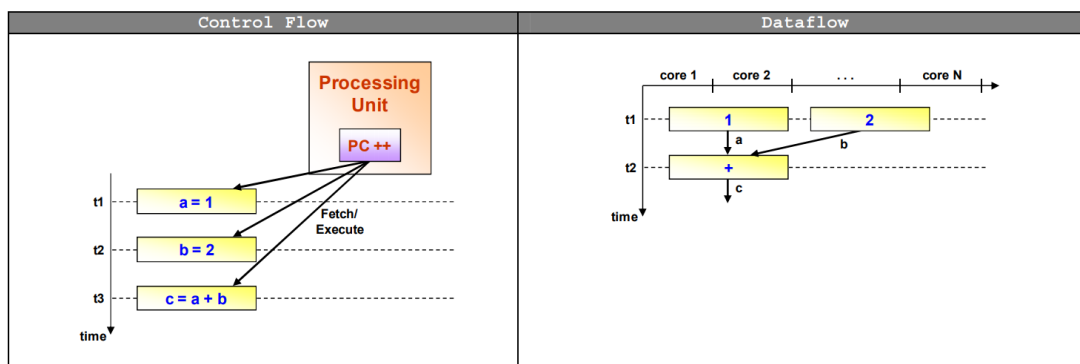
Този метод се разглежда по детайлно в Източник [2] и Източник [3].

Източник [3]

В този източник е разгледан математическият принцип на работа на метода като е предоставен и абстрактен алгоритъм, който може да се използва за пресмятането на множество стойности, пресметнати чрез редове, част от които са: факториел, Неперово число, тригонометрични функции синус, косинус и други. Също така е предоставено математическо доказателство за сложността на алгоритъма в общия случай. Разгледан е и случаят, в който алгоритъма се използва за пресмятането на реда на братята Чудоновски. Спомената е и възможността за паралелизиране на алгоритъма, но е дадена само идеята, че интервалът може да бъде разделен на под интервали и всеки да бъде изчислен паралелно.

Източник [4]

Представена е програма пресмятаща числото Pi, написана на езиците C и Lisp. В този документ се описва разликата между Dataflow и Control flow.



Фиг. 4 - Control flow vs Data flow

Описана е Dataflow семантика, по такъв начин, че да може да се паралелизира. Също е дадено обяснение за избора на езиците C и Lisp. В документа са публикувани и резултати от тестването на програмата. Получено е следното:

| Test Application | Single-threaded Control Flow | Multithreaded Dataflow |
|--|------------------------------|------------------------|
| Pi Number Calculation (GMP_pi.flp) | 167sec. | 7sec. |

Фиг. 5 - Резултати от източник [4]

Машината, на която е тествана програмата, разполага с 28 ядра работещи на 2.4GHz.

Източник [5]

В този източник се разглежда алгоритъмът за пресмятане на редицата на Чудоновски, като отново се извършва итеративно подобряване на алгоритъма. Подробно са описани и подобрения при операциите умножение, работата с дисковото пространство, също така и при работата с входни и изходни операции. Тези оптимизации са малки, но важни, имайки предвид броя извършвани операции.

В документа е засегнато и верифицирането на стойността на числото Pi , чрез използване на друг алгоритъм, конкретно Borwein-Bailey-Plouffe алгоритъма.

Описани са и алгоритмите използвани за забързване на умножението, разделяне на операндите(методът на Каратсуба), преминаване от една бройна система в друга. При умножение на големи числа се използва циклична конволюция на полиноми и други.

Източник [6]

В този източник е представена програма на C, която използва OpenMP. Направена е и оптимизация за разделяне на множителите. За изчислението на големи числа е използвана библиотеката GMP.

Източник [7]

В този източник е представена програма, написана на Java, която имплементира алгоритъмът за пресмятане на редицата на братя Чудоновски, с оптимизациите за разделяне на редът на два реда и представянето на редовете като рекурентни зависимости. Имплементирани са както паралелен, така и непаралелен алгоритъм.

Източниците [6], [7] и [8] не са разгледани подробно. Те представляват само програмен код на решения на проблема, разгледан в тази курсова работа.

Проектиране

Описание на реализирания алгоритъм

Имплементирани са два асинхронни паралелни алгоритъма, чийто софтуерен модел е **Master-Slave** с декомпозиция на данните **SPMD**. Използван е езикът **Java**.

Първият алгоритъм е базиран на първия алгоритъм, разгледан в Източник [1], а именно разделяне на редовете на два реда и използване на рекурентна зависимост.

Алгоритъмът преобразува редът на Чудоновски в рекурентен вид. След което данните на заданието, тоест членовете от реда, се разделят на последователни интервали. Тези интервали се разпределят по обработващите нишки. След приключването на работата на нишките получените стойности биват събрани по начин описан в Източник [1].

Подобрение на реализирания алгоритъм

Вторият алгоритъм е подобрение на първия като се използва вече разгледания метод Binary splitting. Алгоритъмът работи на следния принцип. Първо се пресмята колко члена на редицата трябва да бъдат намерени, като се използва прецизността на пресмятането като подаден аргумент. След това се създават равни по дължина интервали. Следващата стъпка е стартиране на определен брой нишки, като броя се подава като аргумент при стартиране на програмата. На всяка

от нишките се определя интервал, в който да изчисли нужните операнди. След приключването на работата на нишките получените операнди, съхранявани в един синхронизиран списък, биват събрани по алгоритъма на Binary splitting. Получената стойност за π се записва във файл, подаден отново чрез аргумент при стартиране на програмата.

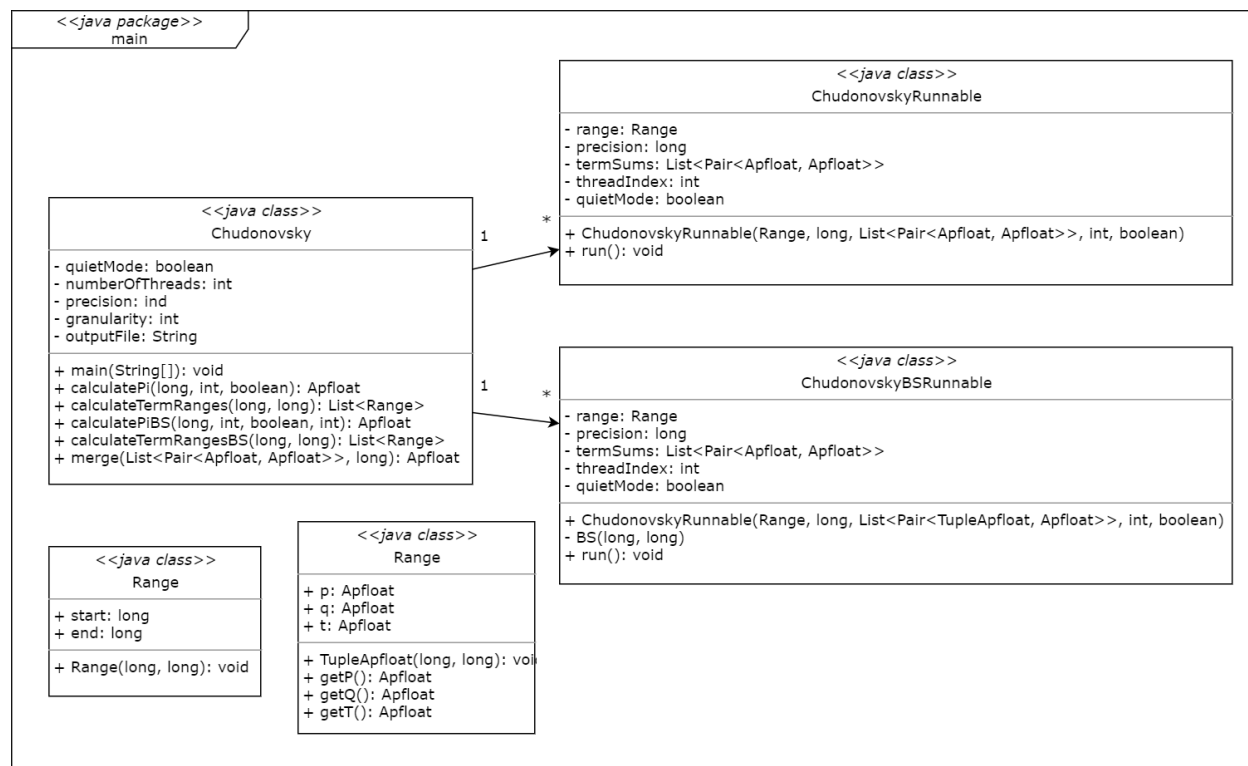
Разделяне на задачите

Разпределянето на задачите е статично. То става след като се изчисли броя на нужните итерации(членове на реда), а всъщност задачите са пресмятането на всеки от членовете на реда на Чудоновски. Задачите се разглеждат като интервали от членове на реда.

За избиране на оптимално разделяне е проведен тест за сравняване на едра и средна грануларност на алгоритъма. След анализ на резултатите е установено, че най-едрата грануларност е най-оптималният вариант(задачите се разделят на p -на брой интервала, а p е броят на използваните нишки).

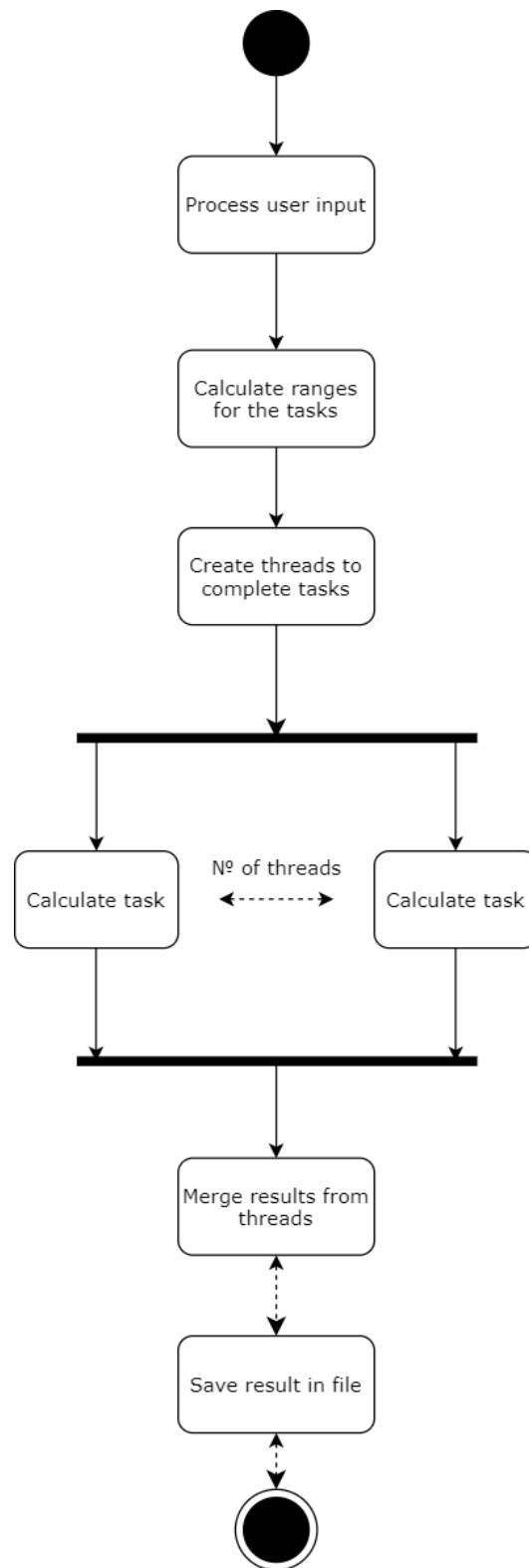
Балансиране на задачите, разпределени по нишките, може да се осъществи чрез вариране на големината на интервалите. Друг вариант за балансиране на задачите е при по-fino разделяне на задачите, нишките да се редуват при обработването на интервалите, тоест да се променя реда, в който нишките взимат интервал за изчисление. Това би се използвало при динамично разпределяне на задачите.

Диаграма на класовете



Фиг. 6 - Диаграма на класовете

Диаграма на дейностите



Фиг. 7 - Диаграма на дейностите

Тестване

Архитектура на тестовата машина

Решението, предложено в този курсов проект, е тествано на машината достъпна чрез SSH на адрес t5600.rmi.yaht.net. Характеристиките на машината са:

- 2x Intel Xeon E5-2660, 8 ядра, 16 нишки, 2.2 GHz, 32 kB L1 кеш
- 128 GB RAM
- Cent OS Linux release 7
- Версия на JDK: OpenJDK 1.8.0_252

Тестов план

- Тест 1) Сравняване на едра, средна грануларност на подобрения алгоритъм
 - Едра грануларност - редът се разделя на p на брой интервала, като всеки се ичислява от една нишка
 - Средна грануларност - интервалите за изчисление са разделят, така че всяка нишка да изчислява 4 интервала.
- Тест 2) Определяне на бързодействие и ефективност на подобрения алгоритъм
 - брой нишки $T = \{1, 2, 4, 6, 8, 10, 12, 14, 16\}$;
 - прецизност на изчислението $P = \{10000, 100000, 500000, 1000000\}$
 - всеки тест е повторен 5 пъти с цел извличане на оптимален резултат
 - сравнени са двата разгледани алгоритъма от

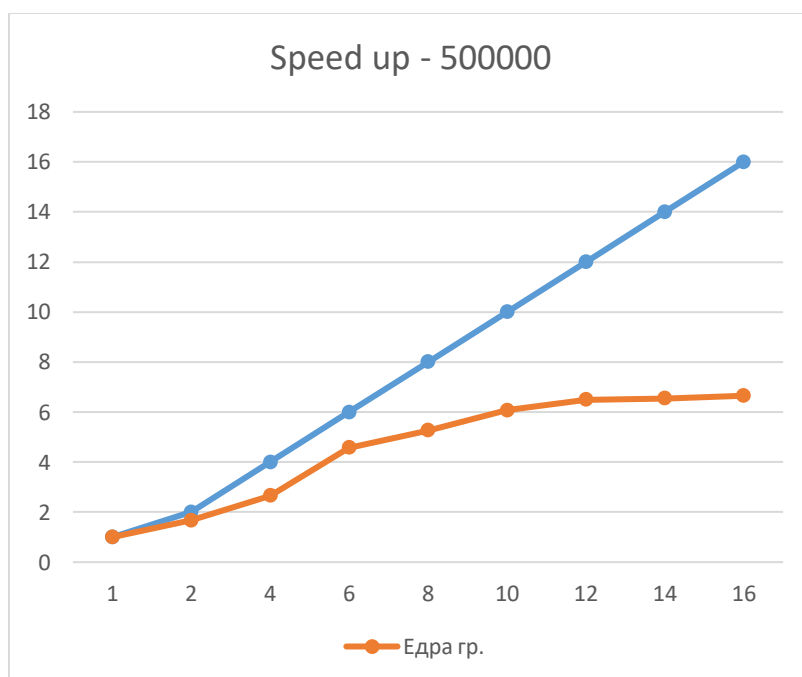
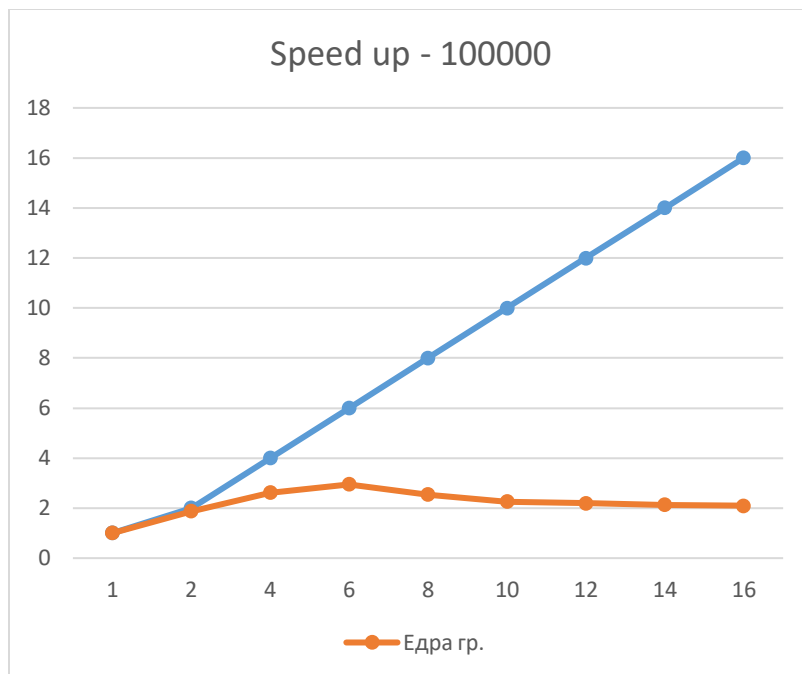
Тестови резултати

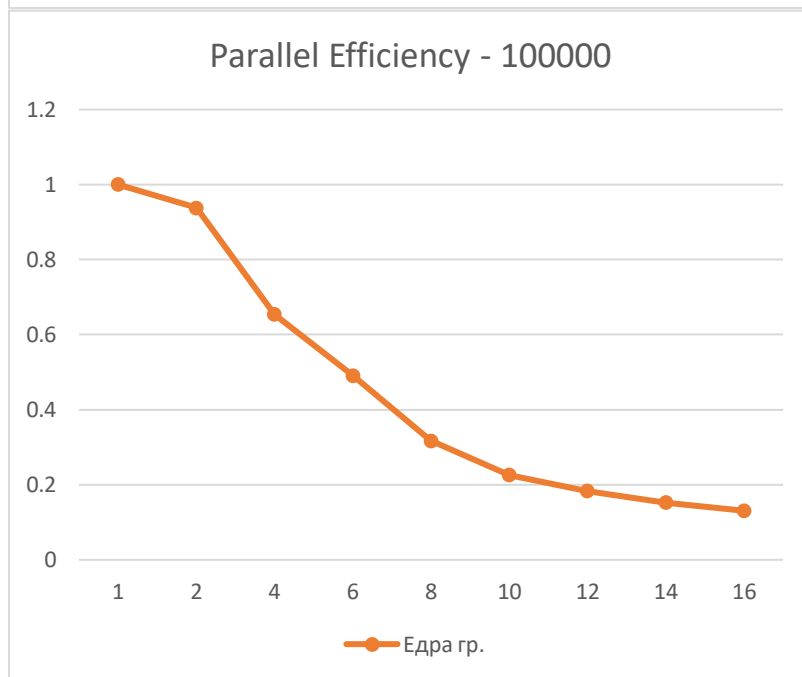
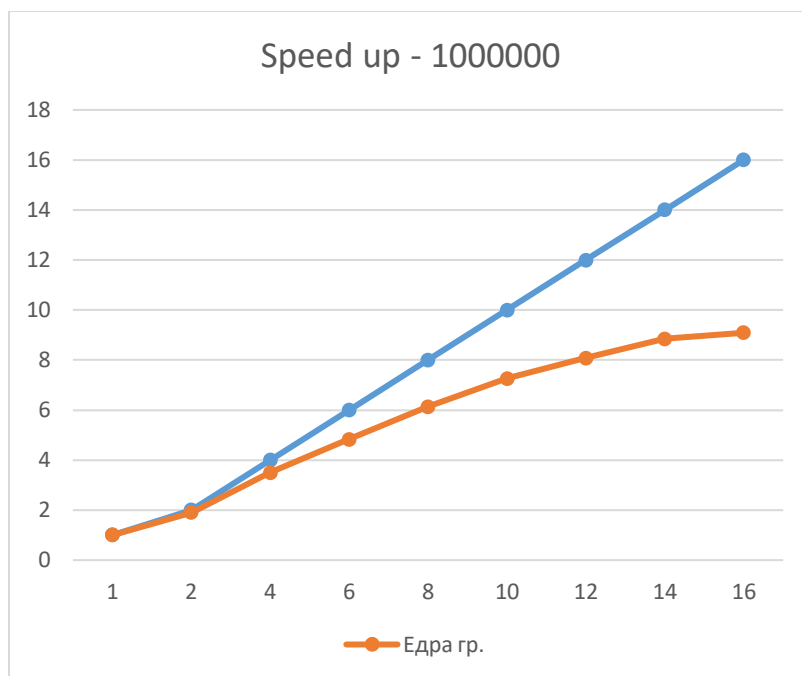
Тест 1

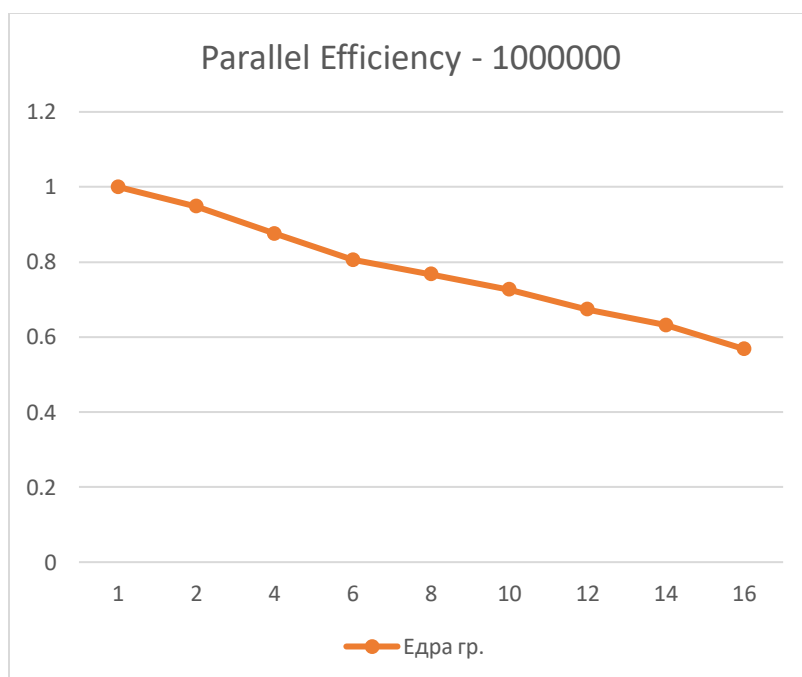
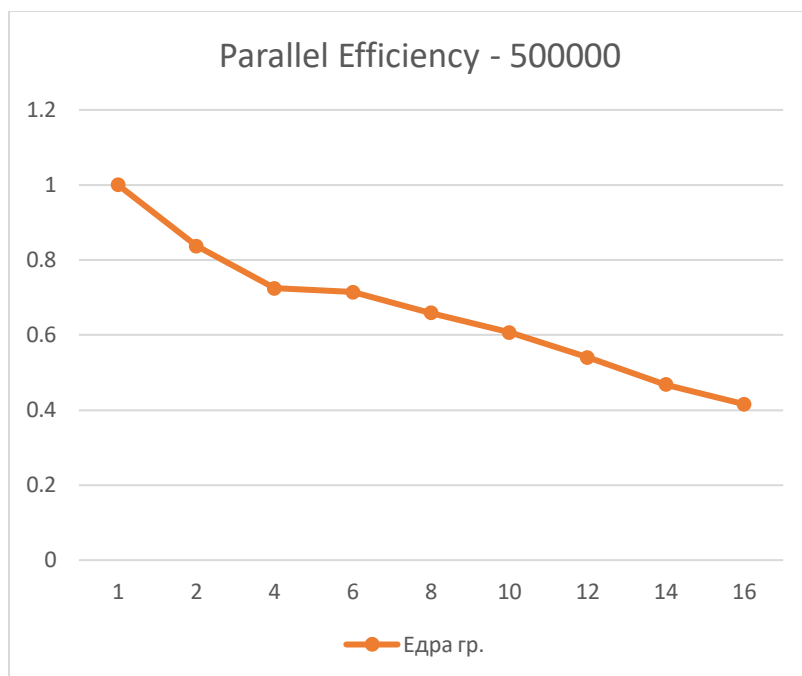
| Брой нишки | Прецизност | | | | | |
|---------------|-------------------|--------|---------|---------------------|--------|---------|
| | Едра грануларност | | | Средна грануларност | | |
| | 100000 | 500000 | 1000000 | 100000 | 500000 | 1000000 |
| 1 | 3590 | 48000 | 172804 | 4061 | 6735 | 12434 |
| 2 | 1913 | 28481 | 91175 | 3340 | 4349 | 7405 |
| 4 | 1372 | 16561 | 49365 | 3947 | 4428 | 6696 |
| 6 | 1219 | 11204 | 35760 | 6025 | 5763 | 8352 |
| 8 | 1415 | 9109 | 28155 | 5908 | 6900 | 9382 |
| 10 | 1591 | 7912 | 23790 | 5989 | 7722 | 11310 |
| 12 | 1631 | 7399 | 21390 | 6003 | 8666 | 13917 |
| 14 | 1683 | 7329 | 19535 | 3623 | 9414 | 16736 |
| 16 | 1716 | 7216 | 19012 | 3300 | 10379 | 17152 |

Тест 2 (подобрен алгоритъм)

| Бързодействие | | | | |
|----------------------------|-------------------|----------|----------|----------|
| Брой нишки | Прецизност | | | |
| | 10000 | 100000 | 500000 | 1000000 |
| 1 | 472 | 3590 | 48000 | 172804 |
| 2 | 370 | 1913 | 28481 | 91175 |
| 4 | 361 | 1372 | 16561 | 49365 |
| 6 | 349 | 1219 | 11204 | 35760 |
| 8 | 394 | 1415 | 9109 | 28155 |
| 10 | 378 | 1591 | 7912 | 23790 |
| 12 | 386 | 1631 | 7399 | 21390 |
| 14 | 412 | 1683 | 7329 | 19535 |
| 16 | 399 | 1716 | 7216 | 19012 |
| Speed up | | | | |
| 1 | 1 | 1 | 1 | 1 |
| 2 | 1.275676 | 1.876634 | 1.685334 | 1.8953 |
| 4 | 1.307479 | 2.616618 | 2.898376 | 3.500537 |
| 6 | 1.352436 | 2.945037 | 4.284184 | 4.832327 |
| 8 | 1.19797 | 2.537102 | 5.269514 | 6.137595 |
| 10 | 1.248677 | 2.256442 | 6.066734 | 7.263724 |
| 12 | 1.222798 | 2.201104 | 6.487363 | 8.078728 |
| 14 | 1.145631 | 2.133096 | 6.549325 | 8.845866 |
| 16 | 1.182957 | 2.092075 | 6.651885 | 9.089207 |
| Parallel Efficiency | | | | |
| 1 | 1 | 1 | 1 | 1 |
| 2 | 0.637838 | 0.938317 | 0.842667 | 0.94765 |
| 4 | 0.32687 | 0.654155 | 0.724594 | 0.875134 |
| 6 | 0.225406 | 0.490839 | 0.714031 | 0.805388 |
| 8 | 0.149746 | 0.317138 | 0.658689 | 0.767199 |
| 10 | 0.124868 | 0.225644 | 0.606673 | 0.726372 |
| 12 | 0.1019 | 0.183425 | 0.540614 | 0.673227 |
| 14 | 0.081831 | 0.152364 | 0.467809 | 0.631848 |
| 16 | 0.073935 | 0.130755 | 0.415743 | 0.568075 |







Анализ на резултатите от тестовете

От получените резултати може да заключим, че при едра грануларност алгоритъмът се справя по-добре при увеличаването на паралелизма(броя използвани нишки), а средната грануларност се справя по-добре при по-малък паралелизъм.

Теоретично тези факти са издържани, понеже слабото място на имплементираният алгоритъм е в събирането на резултатите от нишките. Понеже разделянето на задачите на по-голям брой води до повече операции за сумиране на резултатите от нишките, може да очаваме, че по-

фините грануларности ще се справят по-зле от по-едрите. Затова алгоритъмът е разгледан подробно при използване на едра грануларност.

От тестовите резултати на подобреният алгоритъм може да видим тенденцията, че при малки прецизности не се получава добро забързване при използване на множество нишки, това отново е породено от бавното сумиране на резултатите от нишките.

Източници

- [1] C. Wood, „Pi - Chudnovsky,“ [Онлайн]. Available: <https://www.craig-wood.com/nick/articles/pi-chudnovsky/>.
- [2] G. Xavier и S. Pascal, „Binary splitting method,“ [Онлайн]. Available: <http://numbers.computation.free.fr/Constants/Algorithms/splitting.html>.
- [3] B. Haible и T. Papanikolaou, „Fast multiprecision evaluation of series of rational numbers,“ [Онлайн]. Available: <https://www.ginac.de/CLN/binsplit.pdf>.
- [4] O. Pochayevets, „Dataflow in Practice: Calculating Pi Number with Chudnovsky Algorithm and GMP Library in Parallel Using Transparent Dataflow Programming Model for Multicore and Many-core,“ [Онлайн]. Available: http://bmdfm.com/pdf/Dataflow_Multicore_Manycore_PiGMP4BMDFM.pdf.
- [5] F. Bellard, „Computation of 2700 billion decimal digits of Pi using a,“ 11 Feb 2010. [Онлайн]. Available: <https://pdfs.semanticscholar.org/6cf7/1234c8662100277b1057467d5917c5954f40.pdf>.
- [6] D. Carver, „Parallel GMP-Chudnovsky using OpenMP with factorization,“ 07 11 2008. [Онлайн]. Available: <https://gmplib.org/list-archives/gmp-discuss/2008-November/003444.html>.
- [7] S. Wiedemann, „A parallel and non-parallel implementation of Chudnovsky algorithm to calculate pi,“ [Онлайн]. Available: <https://github.com/lemmingapex/ChudnovskyAlgorithm>.
- [8] quantum0813, „Experiments with calculating Pi using different parallel frameworks,“ [Онлайн]. Available: <https://github.com/quantum0813/FunWithPi>.
- [9] tutorialspoint, „Java - Multithreading,“ [Онлайн]. Available: https://www.tutorialspoint.com/java/java_multithreading.htm.

Таблица на фигурите

| | |
|--|---|
| Фиг. 1 - Разделане на реда на два реда | 3 |
| Фиг. 2 - Рекурентна зависимост на членовете на редовете | 3 |
| Фиг. 3 - Преобразуване на ред от дробни в дроб от редове | 4 |
| Фиг. 4 - Control flow vs Data flow | 4 |

| | |
|---|---|
| Фиг. 5 - Резултати от източник [4]..... | 4 |
| Фиг. 6 - Диаграма на класовете | 6 |
| Фиг. 7 - Диаграма на дейностите..... | 7 |