Софийски университет "Св. Климент Охридски"
Факултет по математика и информатика

E

Тема №14

Изготвил:

Иван Божков - 61823

Съдържание

Идея

Едно важно за математиката число е Неперовото число (Ойлеровото число), тоест числото **e.** Използвайки сходящи редове, можем да сметнем стойността на е с произволно висока точност. Един от сравнително бързо сходящите към е редове е:

$$e = \sum \frac{(3k)^2 + 1}{(3k)!}, k = 0, \dots, n$$

Задача е да се напишете програма за изчисление на числото, използвайки цитирания ред,която използва паралелни процеси (нишки) и осигурява пресмятането на е със зададена от потребителя точност. Изискванията към програмата са следните:

- ✓ Команден параметър задава точността на пресмятанията. По Ваше желание, точността се изразява или в брой членове на реда. Командният параметър задаващ точността има вида - "-p 10240";
- ✓ Друг команден параметър задава максималния брой нишки (задачи) на които разделяме работата по пресмятането на е – "–t 1"
- ✓ Програмата извежда подходящи съобщения на различните етапи от работата си, както ивремето отделено за изчисление и резултата от изчислението (стойността на е);

  Примери за подходящи съобщения:
  „Thread <int> started calculation.",
  „Thread <int> is ready..",
  „Thread <int> has time (millis): <int>",
  „Time for whole calculation (millis): <int>" и т.н.;
- ✓ Записва резултата от работа си (стойността на е) във изходен файл, зададен с подходящ параметър, например "-o result.txt". Ако този параметър е изпуснат, се избира име по подразбиране;
- ✓ Да се осигури възможност за „quiet" режим на работа на програмата, при който се извежда само времето отделено за изчисление на е, отново чрез подходящо избран друг команден параметър –например "-q";

Анализ

Високопроизводителните изчисления най-общо се отнасят до практиката на агрегиране на изчислителната мощност по начин, който осигурява много по-висока производителност, отколкото може да се получи от типичен настолен компютър или работна станция, за да се решат големи проблеми в науката, инженерството или бизнеса. Има няколко неща, които бих искал да добавя, и няколко концепции, които бих искал да подчертая. Определението за паралелна ефективност обикновено се дефинира като ускорението, разделено на броя на изпълняващите единици (процесори, ядра, ...):

$$E = \frac{S}{p}$$

Но има и тънкост по отношение на ускорението. Бих го написал така:

$$S = \frac{Tseq}{Tp}$$

Където Tp е времето, необходимо за даден паралелен алгоритъм, преди да се получи отговор на p единици за изпълнение. Все пак, бих взел Tseq като време, използвано от най-добрия наличен последователен алгоритъм. Разбира се, "най-доброто" зависи от архитектурата, на която работи програмата. Забележете, че последователният алгоритъм може да бъде фундаментално различен от усъвършенстваната успоредна версия и да отбележим, че добре усъвършенстваният паралелен алгоритъм не е задължително да бъде ефективен изобщо, ако лесно надвишава успешния алгоритъм. Използвайки дефиницията, която често използва и Чарлз Лисерсън (MIT)

$$\frac{Tseq}{T\infty} = \lim_{p\to\infty} \frac{Tseq}{Tp}$$

Тази скорост е максималната възможна скорост, която даден алгоритъм може да постигне, и така показва нивото на паралелизъм, който алгоритъмът притежава. T∞ често се нарича спин на паралелния алгоритъм или критичния път на алгоритъма. Обикновено паралелната ефективност е паралелно ускоряване, разделено от паралелизма, т.е.

$$E_p = \frac{S_p}{p}$$

Където паралелното ускоряване е част от времето от време до еднопроцесорно решение за времето до паралелно решение за ниво p на паралелизъм,

$$S_p = \frac{T_1}{Tp}$$

Така "p" (нивото на паралелизъм) брои броя на възли, процесори, ядра, процеси или друго съобразено с контекста, трябва да се посочи за всеки отделен случай. Понякога има смисъл да се сравни с различна изходна линия, в сравнение и с един процесор, тъй като някои проблеми дори няма да се изпълняват на отделна единица, независимо от колко време го подаваме. Идеята е, че ускорението е сравнение колко пъти по-бързо проблемът може да бъде решен като функция от броя на паралелните единици, а ефективността е мярка за това колко голямо парче от това

подобрение получавате на една допринасяща единица. В идеалния случай ускорението е по-добро, колкото по-близко се получава, т.е. в идеалния случай p = 4 процесора решават проблем S4 = 4 пъти по-бърз от 1 процесор. Следователно ефикасността е по-добра, колкото по-близко е до 1. Тези ограничения обикновено не могат да се постигнат на практика, но те правят теоретични критерии за сравняване на решения, за да се разбере колко успешно е паралелизирано нещо.

Анализ на архитектурата

Важно е да се анализират и разграничат основните моделите за паралелни алгоритми. Моделът на паралелен алгоритъм се разработва чрез разглеждане на стратегия за разделяне на данните и метода за обработка и прилагане на подходяща стратегия за намаляване на взаимодействията. Най-масово използвани са:

Data parallel model - Паралелизмът на данните е следствие от единични операции, които се прилагат върху множество елементи от данни.

Task graph model - паралелизмът се изразява чрез граф на задачата. . Този модел се прилага за решаване на проблеми, при които количеството данни, свързани с задачите, е огромно в сравнение с броя изчисления, свързани с тях.

Work pool model - Задачите се разпределят динамично на процесите за балансиране на товара. Следователно, всеки процес може потенциално да изпълни всяка задача. Този модел се използва, когато количеството данни, свързани с задачите е сравнително по-малко от изчисленията, свързани с задачите.

Master slave model – това е моделът, който съм използвал за разработка на задачата, за това него ще го опишем в точка Софтуерен модел

*Producer consumer / pipeline mode*l - Задачите се разпределят динамично на процесите за балансиране на товара. Следователно, всеки процес може потенциално да изпълни всяка задача. Този модел се използва, когато количеството данни, свързани с задачите, е сравнително по-малко от изчисленията, свързани с задачите.

Hybrid model - състои се или от множество модели, приложени йерархично, или от множество модели, приложени последователно към различни фази на паралелен алгоритъм.

Анализ на дизайна

След избора на модел следва изборът на подходяща техника за проектиране на паралелния алгоритъм. Това е най-трудната и важна задача. Повечето от паралелните проблеми при програмирането могат да имат повече от едно решение. Ще обсъдим основните техники за проектиране на паралелни алгоритми, за да изберем най-подходящия за нас:

Divide and conquer - този подход проблемът е разделен на няколко малки подпроблеми. След това подпроблемите се решават рекурсивно и се комбинират, за да се намери решение на първоначалния проблем. Има няколко основни стъпки: Разделяне (на подпроблем), завладяване (решаване на подпроблема рекурсивно), комбиниране (на подпроблемите, за да се оформи крайно решение)

Greedy Method -при него най-доброто решение може да се избира във всеки един момент. Greedy работи рекурсивно за създаване на група от обекти от най-малките възможни компоненти. Наречен е така, защото когато се осигурява оптималното решение за по-малкия пример, алгоритъмът не разглежда цялата програма като цяло. След като се вземе решение, Greedy никога не разглежда същото решение отново.

Dynamic Programming – това е оптимизация, която разделя проблема на по-малки подпроблеми и след решаване на всеки подпроблем, динамичното програмиране съчетава всички решения, за да се постигне крайно решение. За разлика от метода за разделяне и завладяване, динамичното програмиране многократно използва повторно решението на подпроблемите.

Backtracking – това е оптимизационна техника за решаване на комбинирани проблеми. Прилага се както за програмните, така и за реалните проблеми. При него започваме с възможно решение, което отговаря на всички необходими условия. След това преминете към следващото ниво и ако това ниво не даде задоволително решение, върнете едно ниво назад и започнете с нова опция.

Branch & Bound – това е оптимизационна техника,която търси най-доброто решение за даден проблем в цялото пространство на решението. Ограниченията във функцията, която трябва да бъде оптимизирана, се обединяват със стойността на най-новото най-добро решение. Целта на алгоритъма е да се поддържа най-ниската цена на пътя към целта. След като бъде намерено решение, то може да продължи да подобрява решението. Основава се на търсене в дълбочина.

Linear Programming - Това е техника, за да получите най-добрия резултат като максимална печалба, най-кратък път или най-ниска цена. Тук имаме набор от променливи и ние трябва да им зададем абсолютни стойности, за да задоволим набор от линейни уравнения и да максимизираме или минимизираме дадена линейна обективна функция.

Използвани технологии

Езици за програмиране:

За разработване на проекта, избрах да ползвам езикът за програмиране Java, тъй като той предоставя много големи възможности. Езикът има вградена библиотека, която е идеална за целите на курса и предоставя всички функционалности, които бяха нужни за разработване на задачата

Платформи:

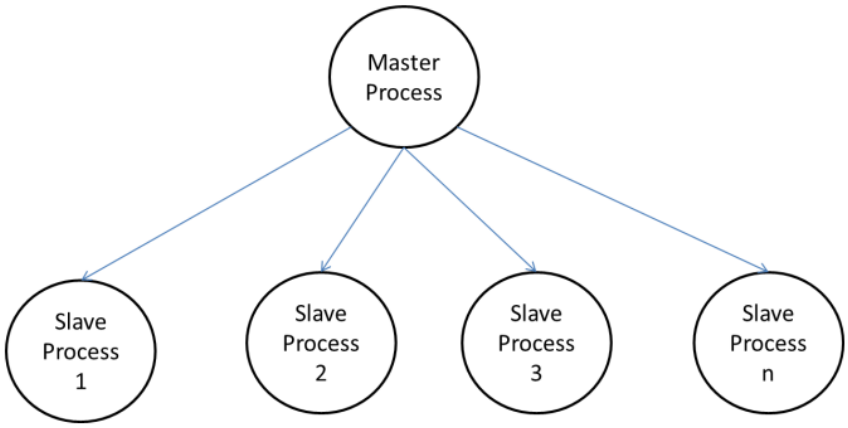
За разработването на задачата, реших да използвам eclipse, тъй като тази среда за разработка е създадена и разработена за ползване с езикът Java. Eclipse идеално се интегрира с езикът, предоставя възможност за допълване, следене за правописни грешки, следене за грешки в моделирането на класове и др. eclipse също така предоставя възможност и за конзола, в която се показва резултата от програмата. При грешка, проектът дори не се build-ва, което защитава от груби грешки

Софтуерен модел:

За разработването на проекта реших да използвам модела Master-Slave, където един или повече главни процеси генерират задача и я разпределят към подчинени процеси. Задачите могат да бъдат разпределени предварително, ако:

- Master може да оцени обема на задачите или
- произволното задание може да изпълни задоволително задачата за балансиране на натоварването или
- на slaves са назначени по-малки задачи в различно време

Този модел обикновено е еднакво подходящ за парадигмите с споделено адресно пространство или съобщения за предаване на съобщения, тъй като взаимодействието е естествено два начина. В някои случаи задачата може да се наложи да бъде завършена на етапи и задачата във всяка фаза трябва да бъде завършена преди следващата задачата да бъде генерирана в следващите фази. Master-Slave моделът може да бъде обобщен в йерархичен или модел с няколко нива, в който главният master подхранва голямата част от задачите на master-а на второ ниво, който допълнително подразделя задачите сред собствените си slave и може да изпълнява част от самата задача. Трябва да се вземе под внимание, че master-ът се превръща в точка на задръстване. Това може да се случи, ако задачите са твърде малки или работниците са сравнително бързи. Задачите трябва да бъдат избрани така, че разходите за изпълнение на дадена задача да зависят от цената на комуникацията и разходите за синхронизация. Асинхронното взаимодействие може да помогне за припокриване на взаимодействието и изчисленията, свързани с генерирането на работа от master-а.



Имплементирани функции

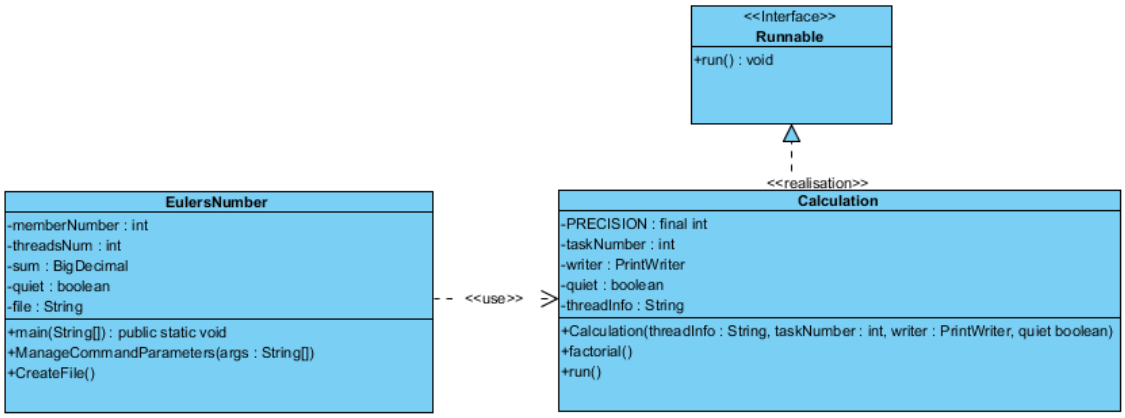
Реализацията на програмата е базирана на представената до тук информация, вземайки положителните страни на описани по горе модели и алгоритми и комбинирайки ги до създаване на функционална програма отговаряща на изискванията с минимални отрицателни ефекти.

Дефинирам променливи които да пазят информация за броя на събираемите, броя на нишките които да се използват за реализация на изчисленията, променлива за запазване на сумата, за името на фаила и булева променлива за тихата функционалност на програмата.

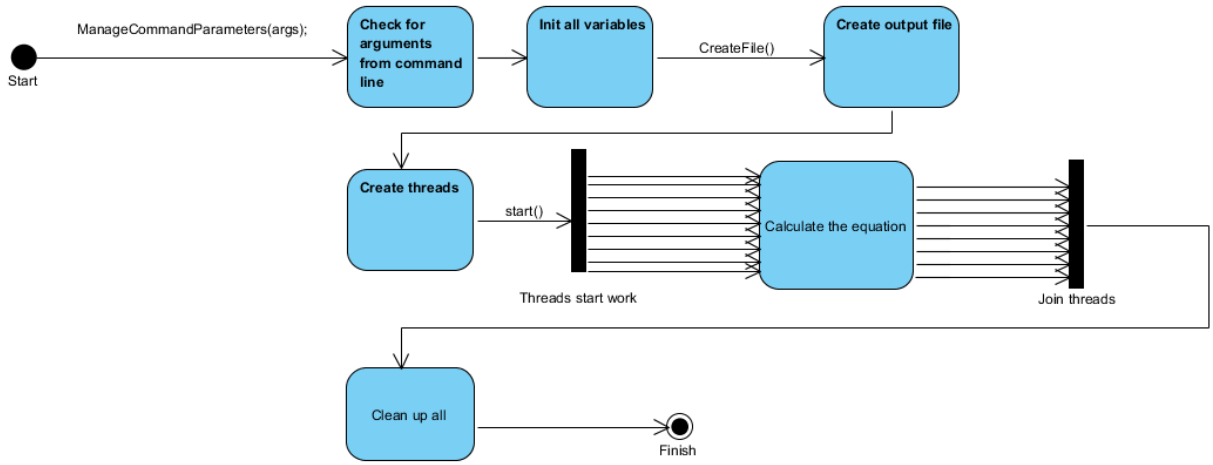
Започвам изпълнението на програмата, извиквайки функция управляваща командните параметри, подавани на програмата директно от командния ред. Последва извикване на функция суздаваща файл с име по подразбиране или въведено такова от потребителя. Създавам връзка през която да записвам данни в него. Създавам пул от нишки като подавам на всяка нишка да изпълнява изчисленията. За изчислението дефинирам точност на изчислението на е както и създавам конструктур с необходимите ми данни за нишката която извършва изчислението. Това са информация за името на нишката, нейния номер, информация за това къде да запише резултата и дали е зададен тих режим на работа. Имплементиеам функцията run наследена от интерфейса Runnable, за да използвам създадените нишки. Прилагам алгоритъм за пресмятане, чиято същност е да даде на всяка нишка работа с приблизително близка тежест. Това става като задавам на всяка нишка да изчислява събирамо равно на нейния номет а всяко следващо да бъде толкова по голямо колкото е броя на нишките. Така си създавам умерена гранулярност. Събирам всичко и изчаквам ако има незавършили нишки.

Диаграми

Class Diagram



Activity Diagram



Постигнати резултати

За тестване на коректността на алгоритъма, който реших да използвам, използвах наличните сървъри. Реших да направя няколко различни теста, на база различни параметри, за да преценя до колко използваният алгоритъм е коректен.

P=2000, сървър 1

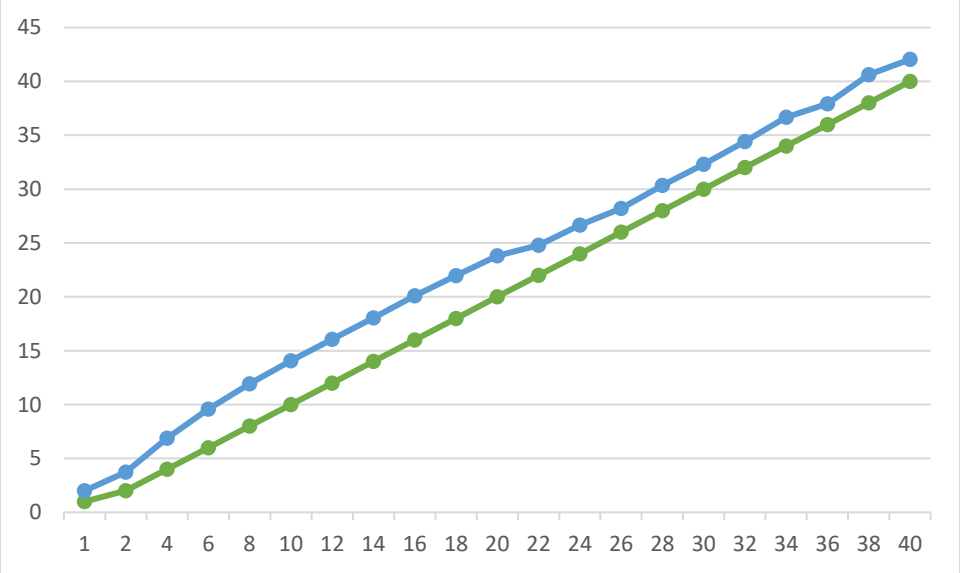
Таблица със стойности:

За създаване на таблицата и изготвяне на данните реших да направя по 3 изследвания за всеки брой ниши, след която да намеря средното им, за да имаме по-реалистично ускорение.

Брой нишки	Опит 1	Опит 2	Опит 3	Средно време за изпълнение	Ускорение	Ефективност
1	***9783***	***9270***	***9553***	9535.333333	1	1
2	***5563***	***5329***	***5506***	5466	1.744481034	0.872240517
4	***3456***	***3184***	***3277***	3305.666667	2.884541696	0.721135424
6	***2569***	***2685***	***2721***	2658.333333	3.586959248	0.597826541
8	***2671***	***2318***	***2303***	2430.666667	3.922929238	0.490366155
10	***2192***	***2481***	***2376***	2349.666667	4.058164279	0.405816428
12	***2146***	***2284***	***2622***	2350.666667	4.05643789	0.338036491
14	***2401***	***2489***	***2187***	2359	4.042108238	0.288722017
16	***2496***	***2320***	***2179***	2331.666667	4.089492495	0.255593281
18	***2419***	***2524***	***2266***	2403	3.968095436	0.220449746
20	***2495***	***2451***	***2550***	2498.666667	3.816168623	0.190808431
22	***3549***	***3220***	***3489***	3419.333333	2.788652759	0.126756944
24	***4094***	***3544***	***3067***	3568.333333	2.672209248	0.111342052
26	***4616***	***4410***	***3938***	4321.333333	2.206572046	0.084868156
28	***3350***	***4660***	***4151***	4053.666667	2.352273662	0.084009774
30	***4001***	***4150***	***4308***	4153	2.296010916	0.076533697
32	***4588***	***2612***	***4665***	3955	2.410956595	0.075342394
34	***2614***	***2922***	***5168***	3568	2.672458894	0.078601732
36	***2769***	***8619***	***3370***	4919.333333	1.938338528	0.053842737
38	***3340***	***3526***	***3997***	3621	2.633342539	0.069298488
40	***3188***	***5737***	***5070***	4665	2.04401572	0.051100393

Графики:


Ускорение
4,5
4
3,5
3
2,5
2
1,5
1
0,5
0
1 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40


45
40
35
30
25
20
15
10
5
0
1 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40

Анализ:

От графиките ясно се вижда, че на сървър 1, ускорението не надвишава 4.5. Можем да си обясним това с възможността сървърът да притежава по-малко ядра и така да бъде невъзможно за него да постигне по-големи ускорения. Наблюдаваме и някои странични отклонения. В края ускорението спада, което произлиза от по-слабата машина.

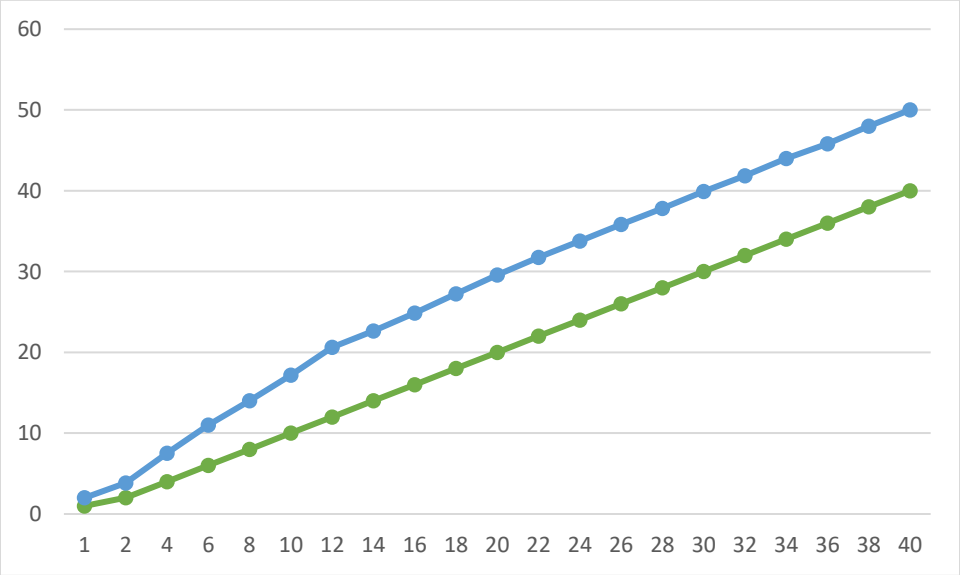
P=2000, сървър 2

Таблица със стойности:

Брой нишки	Опит 1	Опит 2	Опит 3	Средно време за изпълнение	Ускорение	Ефективност
1	***11657***	***11181***	***10576***	11138	1	1
2	***5585***	***5705***	***5857***	5715.666667	1.948679069	0.974339535
4	***3456***	***5657***	***5796***	4969.666667	2.241196593	0.560299148
6	***2988***	***3030***	***2999***	3005.666667	3.705667073	0.617611179
8	***1693***	***1729***	***1766***	1729.333333	6.440632228	0.805079029
10	***1824***	***1823***	***1886***	1844.333333	6.039038496	0.60390385
12	***1807***	***1837***	***1750***	1798	6.194660734	0.516221728
14	***1912***	***1811***	***1823***	1848.666667	6.024882798	0.430348771
16	***1827***	***1810***	***1900***	1845.666667	6.034675817	0.377167239
18	***1817***	***1793***	***1908***	1839.333333	6.055454875	0.33641416
20	***1796***	***1783***	***1971***	1850	6.020540541	0.301027027
22	***1998***	***1804***	***1788***	1863.333333	5.97745975	0.271702716
24	***1760***	***1955***	***1806***	1840.333333	6.052164463	0.252173519
26	***1959***	***2020***	***1819***	1932.666667	5.763021732	0.221654682
28	***1770***	***1779***	***1851***	1800	6.187777778	0.220992063
30	***1777***	***1771***	***1771***	1773	6.282007896	0.209400263
32	***2049***	***1798***	***2039***	1962	5.676860347	0.177401886
34	***1828***	***1768***	***1929***	1841.666667	6.047782805	0.177875965
36	***1817***	***1884***	***1838***	1846.333333	6.032496841	0.167569357
38	***1892***	***1859***	***1822***	1857.666667	5.995693522	0.157781408
40	***1813***	***1817***	***1800***	1810	6.15359116	0.153839779

Графики:


Ускорение
7
6
5
4
3
2
1
0
1 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40


60
50
40
30
20
10
0
1 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40

Анализ:

От графиките ясно се вижда, че на сървър 2, ускорението е по-добро от предишният опит. Вече се забелязва значителен ръст в ускорението и освен това можем да кажем, че почти няма спад.

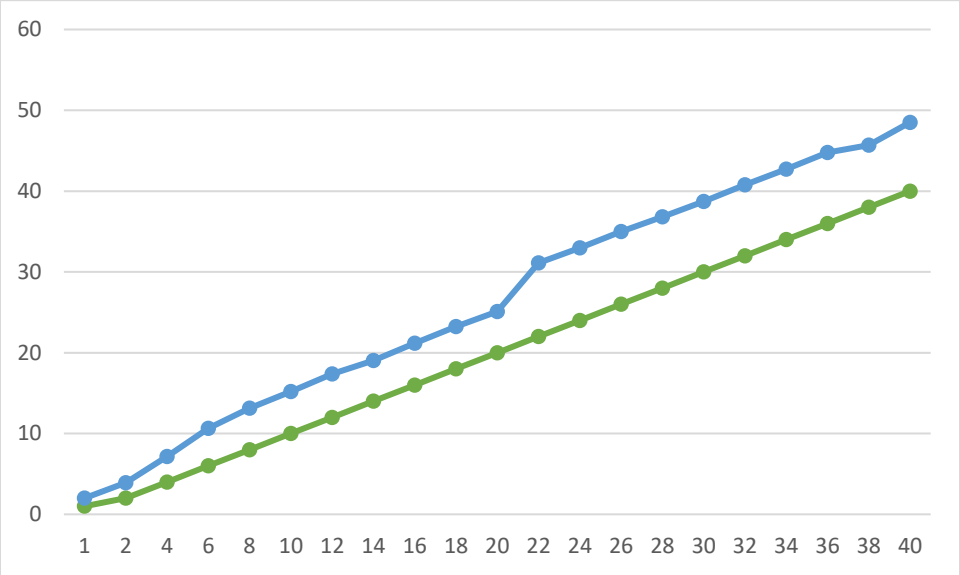
P=2000, сървър 3

Таблица със стойности:

Брой нишки	Опит 1	Опит 2	Опит 3	Средно време за изпълнение	Ускорение	Ефективност
1	***11657***	***11181***	***10576***	11138	1	1
2	***5585***	***5705***	***5857***	5715.666667	1.948679069	0.974339535
4	***3456***	***5657***	***5796***	4969.666667	2.241196593	0.560299148
6	***2988***	***3030***	***2999***	3005.666667	3.705667073	0.617611179
8	***1693***	***1729***	***1766***	1729.333333	6.440632228	0.805079029
10	***1824***	***1823***	***1886***	1844.333333	6.039038496	0.60390385
12	***1807***	***1837***	***1750***	1798	6.194660734	0.516221728
14	***1912***	***1811***	***1823***	1848.666667	6.024882798	0.430348771
16	***1827***	***1810***	***1900***	1845.666667	6.034675817	0.377167239
18	***1817***	***1793***	***1908***	1839.333333	6.055454875	0.33641416
20	***1796***	***1783***	***1971***	1850	6.020540541	0.301027027
22	***1998***	***1804***	***1788***	1863.333333	5.97745975	0.271702716
24	***1760***	***1955***	***1806***	1840.333333	6.052164463	0.252173519
26	***1959***	***2020***	***1819***	1932.666667	5.763021732	0.221654682
28	***1770***	***1779***	***1851***	1800	6.187777778	0.220992063
30	***1777***	***1771***	***1771***	1773	6.282007896	0.209400263
32	***2049***	***1798***	***2039***	1962	5.676860347	0.177401886
34	***1828***	***1768***	***1929***	1841.666667	6.047782805	0.177875965
36	***1817***	***1884***	***1838***	1846.333333	6.032496841	0.167569357
38	***1892***	***1859***	***1822***	1857.666667	5.995693522	0.157781408
40	***1813***	***1817***	***1800***	1810	6.15359116	0.153839779

Графики:


Ускорение
10
9
8
7
6
5
4
3
2
1
0
1 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40


60
50
40
30
20
10
0
1 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40

Анализ:

При изпълнение на програмата в сървър 3 забелязваме драстичен скок в ускорението. Очевидно машината има по-добри параметри.

Въпреки това за определен брой нишки има странни резултати. Това се случи с всички сървъри до сега. Това ме кара да опитам с по-голям брой събираеми.

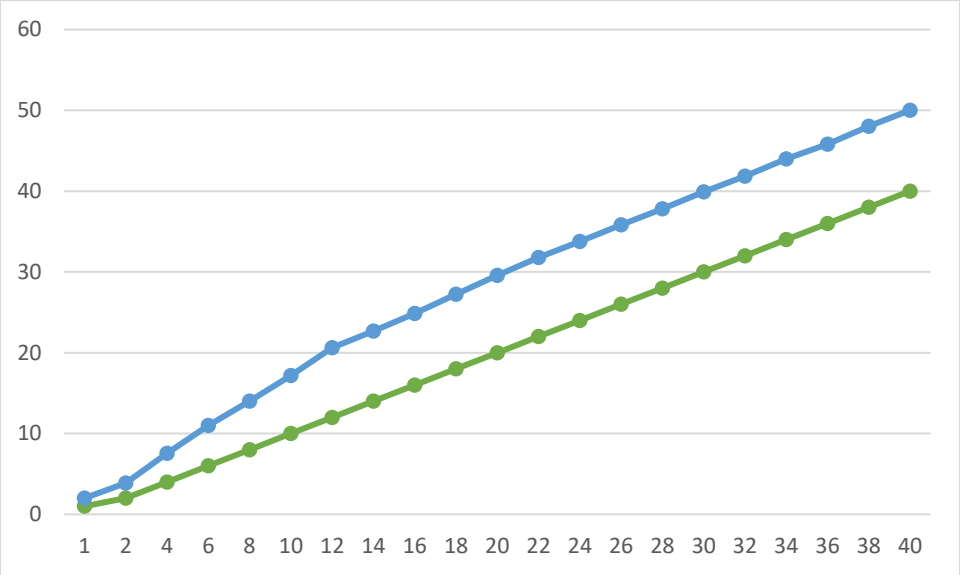
P=5000, сървър 3 - последен опит

Таблица със стойности:

Брой нишки	Опит 1	Опит 2	Опит 3	Средно време за изпълнение	Ускорение	Ефективност
1	143986	143502	145353	144280.3333	1	1
2	80156	77982	74892	77676.66667	1.857447539	0.928723769
4	40361	41121	41137	40873	3.529966808	0.882491702
6	28492	29282	28815	28863	4.998798924	0.833133154
8	25144	22916	24081	24047	5.999930691	0.749991336
10	20699	18378	21138	20071.66667	7.18825874	0.718825874
12	16547	16812	16788	16715.66667	8.631443556	0.719286963
14	16923	16809	16241	16657.66667	8.661497208	0.618678372
16	16457	16167	16186	16270	8.867875435	0.554242215
18	15660	15544	15606	15603.33333	9.246763512	0.513709084
20	14937	15086	15192	15071.66667	9.572951454	0.478647573
22	14643	14816	14852	14770.33333	9.768251676	0.44401144
24	14816	14641	14869	14775.33333	9.764946081	0.406872753
26	14797	14602	14641	14680	9.828360581	0.378013869
28	14640	14947	14465	14684	9.825683283	0.35091726
30	14571	14536	14563	14556.66667	9.9116327	0.330387757
32	14617	14424	14849	14630	9.86195033	0.308185948
34	14405	14400	14489	14431.33333	9.997713309	0.294050391
36	14454	14755	14867	14692	9.820333061	0.272787029
38	14346	14457	14412	14405	10.01598982	0.263578679
40	14321	14443	14488	14417.33333	10.00742162	0.250185541

Графики:


Ускорение
12
10
8
6
4
2
0
1 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40


60
50
40
30
20
10
0
1 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40

Анализ:

Забелязваме, че въпреки големият брой събираеми, които трябва да се изпълнят и по-голямото натоварване на машината, ускорението се запазва високо без да има спадове. Въпреки времето, което отнема на машината да извърши изчисленията, ускорението се доближава до идеалното като върви по успоредна линия с него.

Заключение

След обстойна проверка на алгоритъма и изпълнението на задачите можем да заключим, че алгоритъмът е ефективен и върши своята работа добре. Прави това, което се очаква от него. При по-добра машина достига до **ускорение** между **8** и **10**.

Източници

[1] http://www.cprogramming.com/parallelism.html

[2]: Ananth Grama, George Karypis, Vipin Kumar and Anshul Gupta, Introduction to Parallel Computing, 2nd edition, Addison-Wesley, 2003

[3] Tutorials Point, Parallel Algorithm, Tutorials Pont, 2015, https://www.tutorialspoint.com/parallel_algorithm/parallel_algorithm_tutorial.pdf

[4] What is high performance computing? insideHPC, https://insidehpc.com/hpc-basic-training/what-is-hpc/

[5] Java-Multithreading, Tutorials point, https://www.tutorialspoint.com/java/java_multithreading.html

[6] e (mathematical constant), Wikipedia, https://en.wikipedia.org/wiki/E_(mathematical_constant)

[7] Project Euler, https://projecteuler.net/

[8] Wolfram MathWorld, e Approximations, Wolfram, http://mathworld.wolfram.com/eApproximations.html

[9] Rod Pierce DipCE BEng, Math Is Fun, http://www.mathsisfun.com/numbers/e-eulers-number.html

[10] Robert Sedgewick and Kevin Wayne, Exp.java, 2000, http://introcs.cs.princeton.edu/java/13flow/Exp.java.Html

[11] https://www.sharcnet.ca/help/index.php/Measuring_Parallel_Scaling_Performance