Пресмятане на e (Неперовото число)

Изготвил:

Дък Фан, Ф.Н. 81613, Компютърни науки, трети курс, втори поток, седма група

Съдържание:

1. Пресмятане на е (неперевото число) 3

1.1. Разглеждане на бърз алгоритъм за пресмятане на факториел и паралелни пресмятане на суми 3

1.2. Използвани технологии 5

2. Реализация 9

2.1. Имплементация на функции 10

2.2. Стартиране на програмата 11

3. Резултати 12

3.1. Обобщение 22

4.Източници 23

1. Пресмятане на e (неперевото число)

Целта на проекта е реализация на паралелен алгоритъм, намиращ стойността на e с произволна точност, посредством бързо сходящия ред: $e = \sum_{k=0}^{\infty} \frac{3 - 4*k^2}{(2*k\ +1)!}$

Точността се определя в брой членове на реда, който е предварително зададена константа.

1.1. Разглеждане на бърз алгоритъм за пресмятане на факториел и паралелни пресмятане на суми

[1] Peter Luschny, A new kind of factorial function, 2008, (https://oeis.org/A000142/a000142.pdf);

[2] Peter Luschny, Fast Factorial Functions, 2002, (http://www.luschny.de/math/factorial/FastFactorialFunctions.htm);

[3] Pete Luschny, Parallel Prime Swing, 2010, (https://github.com/PeterLuschny/Fast-Factorial-Functions/blob/master/JavaFactorial/src/de/luschny/math/factorial/FactorialParallelPrimeSwing.java);

[4] Peter Luschny, The most efficient factorial algorithms, (http://www.luschny.de/math/factorial/conclusions.html);

[5] Wikipedia contributors, “Master/slave (technology)”, Wikipedia, The Free Encyclopedia, last edited on 20 May 2020, https://en.wikipedia.org/wiki/Master/slave_(technology)

С помощта на [1] се дава пълен анализ на алгоритъма за PrimeSwing за намиране на факториел, като неговото сложност е равно на O($n(logn)^2 loglogn$)). Това показва, че алгоритъмът PrimeSwing е много по-добър заразлика от обинковения алгоритъм за намиране на факториел.

При [2] е обяснено комбинацията на паралалелното пресмятане и PrimeSwing. Като за тази цел се използва предимството на множество ядрените процесори. Получава се по бърз алгоритъм за пресмятане на факториел, който е ParallelPrimeSwing.

Реализацията за решаване на задачата за намирането на e (неперевото число) ще се пише на Java. Следователно се използва код на Java за ParallelPrimeSwing [3].

Въпросът е дали ParallelPrimeSwing е достатъчно бърз алгоритъм за намиране на факториел. На [4] е представено резултати от тестване на различни алгоритми при намиране на 10000000!. Като ParallelPrimeSwing го изчислява за 57,9 секунди с ranking = 0.91. С тези резултати то е на първо място и следователно е достатъчно бърз алгоритъм за целта на задачата.

Намирането на факториели е само половината част от задачата. Втората половина се състои в измисляне на алгоритъм, който е достатъчно бърз да се намери произволен ред на e. Алгоритъмът за пресмятането се базира на предимството пак на множество ядрените процесори [5], като ще се използва паралелни сметки, за да се допринесе за ускорение при сметките. Такъв алгоритъм ще бъде реализиран с статично и динамично балансиране, за да се покаже неговото ускорение.

Сравнителна таблица.

Образец	Функция	Коментар
[1]	Ускоряване на пресмятането на факториела чрез prime swing.	Представяне на алгоритъма за бързо пресмятане на факториел.
[2]	Разделяне на пресмятането на факториела чрез паралелизъм.	Обяснение на паралелизма за пресмятане на факториел.
[3]	Паралелно пресмятане на факториела.	Код на алгоритъма за паралелно пресмятане на факториел.
[4]	Избор на алгоритъм за бързо пресмятане на факториел.	Сравнение на различни алгоритми за пресмятане на факториел.
[5]	Пресмятане на неперовото число чрез паралелизъм.	Ускоряване пресмятането без загуба на информация.

1.2. Използвани технологии

- Решението е написано на програмния език Java.
- Използвани са външни библиотеки като: Apfloat и Apache Commons CLI.
- Реализиран е бърз паралелен алгоритъм за намиране на големи факториели.
- Паралелно пресмятане на неперовото число.
- Алгоритъмът е имплементиран на Java 13.

Програмата реализира алгоритъма ParallelPrimeSwing за бързо намиране на големи факториели, която позволява намирането на

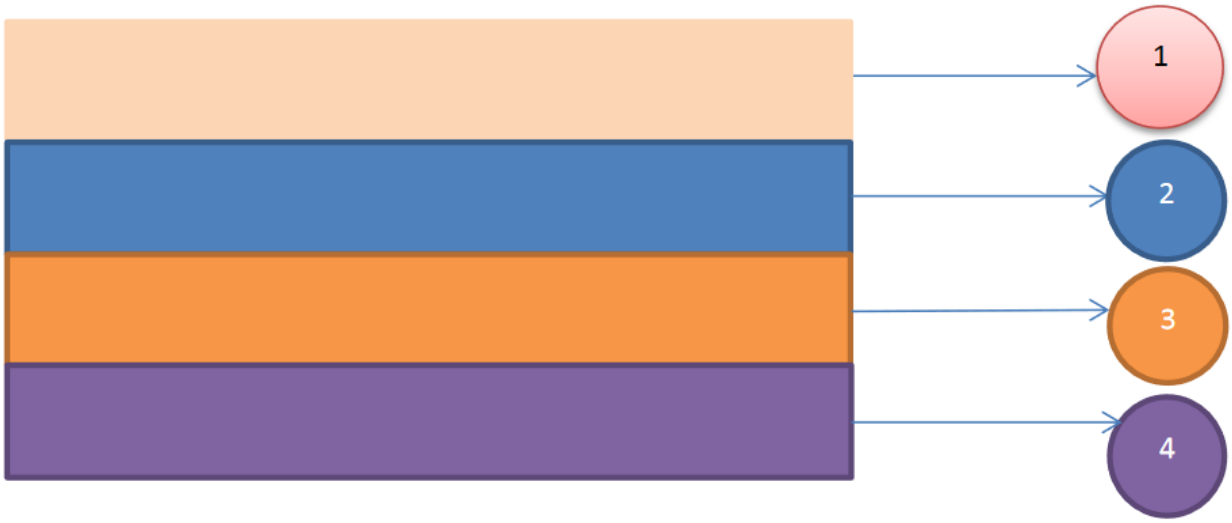
точността на e при много големи редове. Факториелът се изчислява по ефикасен начин чрез произведение на прости числа (ηl), което позволява да намерим факториела по формулата:

n! = $\lfloor n/2 \rfloor!^2 \times \eta\wr$. Например за 62! ще се реши по този начин:

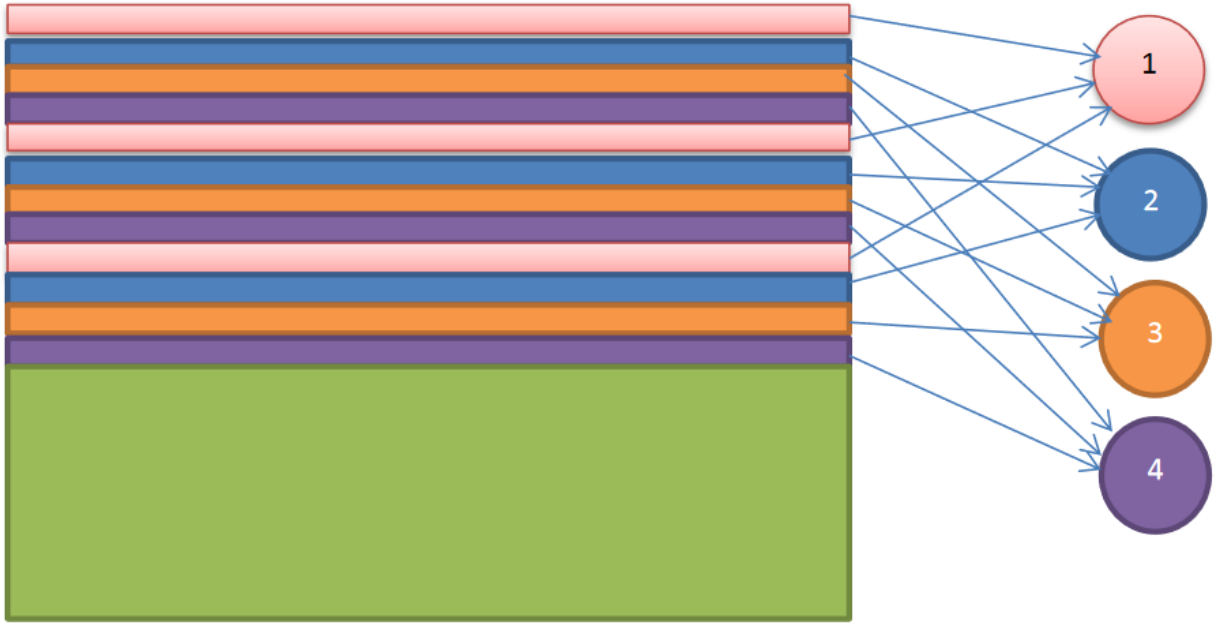
$$
\begin{aligned}
3\wr &= 2 \cdot 3 \\
7\wr &= 2^2 \cdot 5 \cdot 7 \\
15\wr &= 2^3 \cdot 3^2 \cdot 5 \cdot 11 \cdot 13 \\
31\wr &= 2^4 \cdot 3^2 \cdot 5 \cdot 17 \cdot 19 \cdot 23 \cdot 29 \cdot 31 \\
62\wr &= 2^5 \cdot 7 \cdot 11 \cdot 17 \cdot 19 \cdot 37 \cdot 41 \cdot 43 \cdot 47 \cdot 53 \cdot 59 \cdot 61 \\
\\
62! &= (((((1\wr)^2 3\wr)^2 7\wr)^2 15\wr)^2 31\wr)^2 62\wr
\end{aligned}
$$

Намирането на $\eta\wr$ е базирано на sieve of Eratosthenes. Като ще се използват odd swinging factorial. Моделът на паралелният алгоритъм е Master-Workers, като master – Factorial, а workers – Swing. Декомпозицията на проблемите са по данни (SPMD). Програмата след това разпределя работата в пресмятането на e според броя на заданените нишки и грануларността. Ясно е тогава, че при granular=1 имаме най-едрата възможна грануларност, толкова подзадачи, колкото са нишките и все по-фина с нарастващ коефициент на грануларност.

Примерна схема на разпределението (едра грануларност): thread



Примерна схема на разпределението (фина грануларност): thread



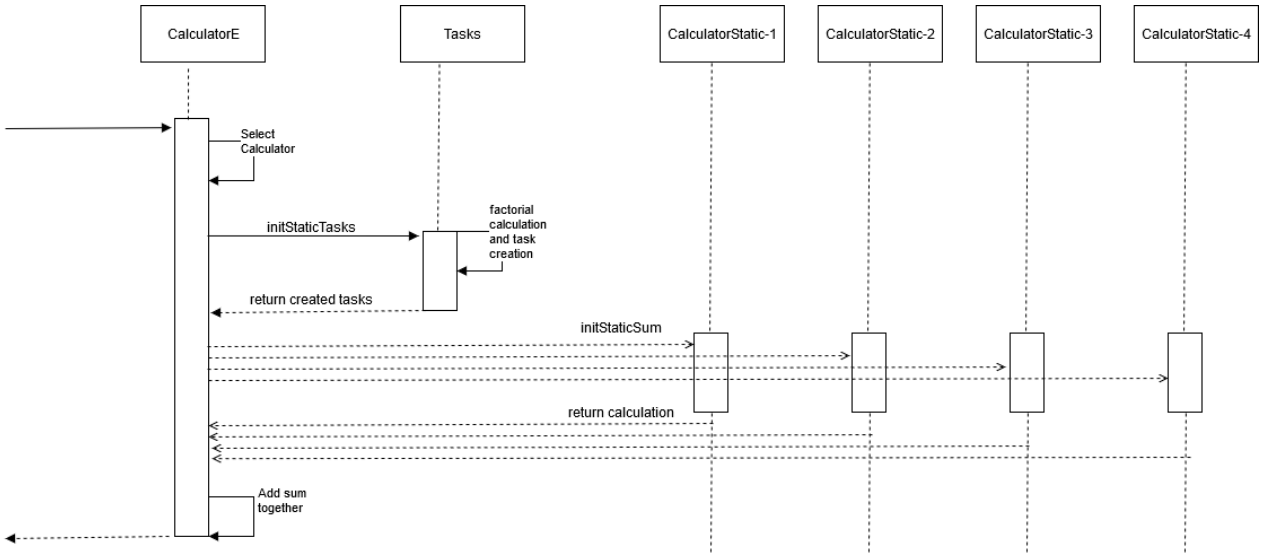
Определя се нужните факториели за работата на всяка задача, което позволява паралелни сметки, като нишките работят асинхронно. За да няма нужда всяка подзадача да изчаква друга подзадача да свърши намирането на факториела си, всяка подзадача има предварително факториела, който се нуждае. Като моделът на паралелния алгоритъм е също Master-Workers, като master – CalculatorE a workers – Calculator$ ($ - се определя дали

Calculator е със статично балансиране CalculatorStatic или с динамично балансиране CalculatorDynamic). Балансирането се определя спрямо команден параметър. Декомпозицията на проблемите са по данни (SPMD). При динамично балансиране PoolExe (ExecutorService) ще се грижи за нишките, като ще им раздава подзадачи, когато са свободни. В calculationDynamicTasks ще се намират всички подзадачите, които чакат да бъдат преработени. Докато при статично балансиране ще се използват нишки, които работят асинхронно (независима една от друга). Разпределянето на подзадачите се извършва циклично. С помощта на цикличното разпределение и грануларността може да се регулира баланса между натовареността на нишките (load balancing), а оттам и ускорението на паралелната програма. Предварително е избран балансирането от входен параметър дали да бъде статично (предварително фиксиране по равен брой N подзадачи между P нишки) или да бъде динамично.

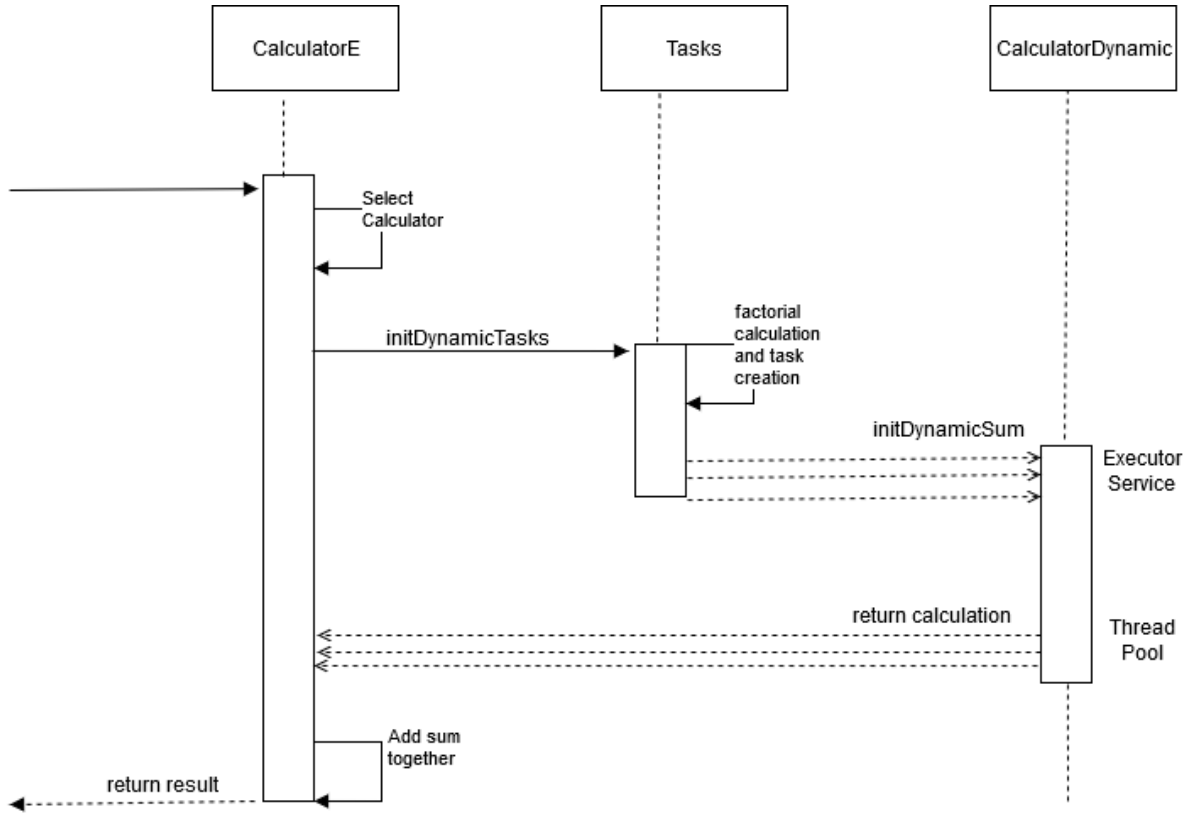
С използваните технологии ще се осъществи алгоритъм, който е достатъчно бърз за решаването на проблема за намиране на e (неперевото число) на точно определен с помощта на сходящата редица.

2. Реализация

1. Диаграма - Последователна диаграма при статично



2. Диаграма - Последователна диаграма при статично



2.1. Имплементация на функции

- public static void main(String[] args)

  В нея се обработват параметрите подадени от командния ред чрез commandExecutor, като в него се създава CalculatorE и се извиква функцията за изчисляване на е, която е startCalculation(). След извикването на фукнцията, започва работата на CalculatorE, като в него се извършва калкулацията на точността и засичането на времето за цялостното изпълнение на програмата.

- initCalculator(long row, int maxThreads, int granular, String filename, boolean quietMode, boolean isStatic)

  В нея предварително са зададени параметрите. Задачата за намирането на **e** се разделя в две части. Първата е да се пресметнат основните факториели, които са нужни в сметките, спрямо броя задачи, които ще се извършват. Това са подзадачите Task, които трябва да се изпълнят от предварително избран калкулатор. Определянето на Tasks се извършва чрез initStaticTasks и initDynamicTasks. Броят на Tasks се определя по следния начин:

  task = maxThreads ∗ granular

  Това разделяне става посредством броя нишки threads и коефициента на грануларност granular, подадени в командния ред. Разпределението на интервалите за пресмятането се определя по формулата:

  interval = row / (maxThreads * granular)

Втората част се определя какъв вид калкулатор – Calculator ще направи пресмятането. Избира се спрямо дали да е със статично

балансиране (CalculatorStatic) или е с динамично балансиране (CalculatorDynamic).

Резултатът от програмата се записва в файл, в който може да се види реда на е (неперевото число). Програмата има възможност да изпринти на конзолата отговора от изчислението на програмата.

2.2. Стартиране на програмата

Програмата се стартира от командния ред чрез подаване на следните параметри, като те самите могат да бъдат подадени в произволен ред:

- Първи команден параметър задава търсената точност в брой членове на реда – например: "**-p 1000**", ако не е зададен тогава по подразбиране е: "**-p 0**"
- Втори команден параметър задава максималния брой нишки на които разделяме работата по намирането на точността - например: "**-t 8**", ако не е зададен тогава по подразбиране е: "**-t 1**"
- Може да се зададе като команден параметър коефициент на грануларност - например: "**-g 2**", ако не е зададен тогава по подразбиране е: "**-g 1**"
- Може да се зададе като команден параметър вида на балансирането. Дали да бъде статично или динамично - например: "**-d**" ще направи балансирането динамично, ако не е зададен тогава по подразбиране е статично.
- Записва резултата от стойността на е в файл от подходящо избран команден параметър – например: **"-o Ecalculation.txt**". Ако този параметър е изпуснат по подразбиране е: "**-o result.txt**"

- Програмата извежда съобщения на различните етапи от работата си, както и времето отделено за изчисление и използваните нишки. Осигурен е възможност за “**quiet**” режим на работата на програмата, при който се извежда само времето за изчисляване на точността на e. Режимът се активира чрез подходящо избран команден паранетър – например: “**-q**”.

3. Резултати

Тестовете са правени за 10000 ред за пресмятане на e. Първите тестове за брой нишки 1,2,4 са правени върху лаптоп с показатели:

Architecture: 64 bit
CPU(s): 4
Model name: Intel® Core™ i5
Ram: 8GB

Следващите тестове за нишки от 5 до 32 са тествани на тестовия сървър t5600.rmi.yaht.net. С показатели:

Architecture: 64 bit
CPU(s): 32
Model name: Intel® Xeon™ CPU e5-2660 0 @ 2.20GHz
Ram: 62GB

Изследвано е поведението й при коефициенти на грануларност 1,2,4 с брой нишки от 1,.....,32. Първо е направено тестването с статично балансиране и след това с динамично балансиране.

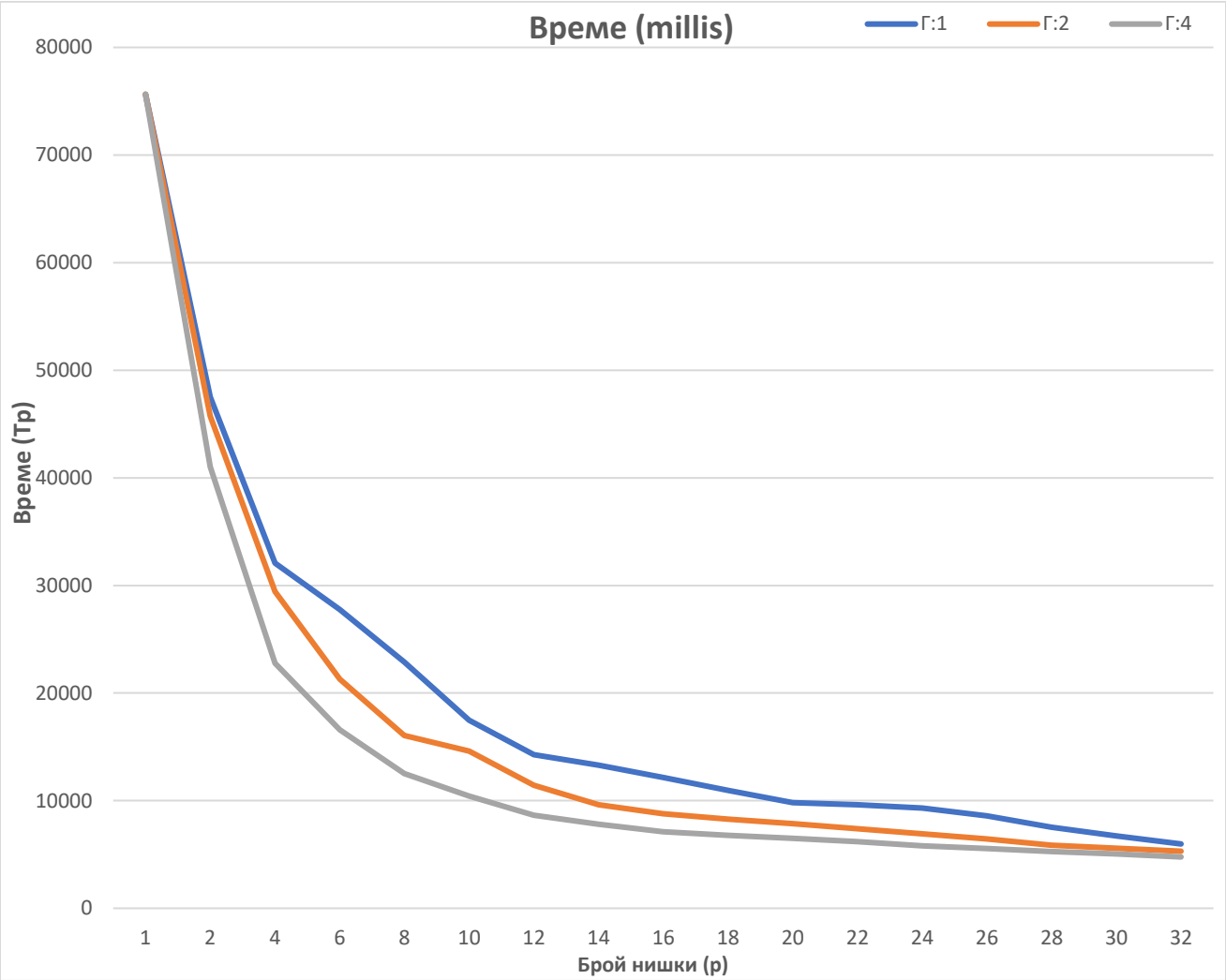
P - броя нишки, S_p – ускорение, E_p - ефективност, g-грануарност

- **Таблица 1.** Резултати при статично балансиране

#	p	g	$T_p^{(1)}$	$T_p^{(2)}$	$T_p^{(3)}$	T_p = min()	$S_p = T_1/T_p$	$E_p = S_p/p$
1	1	1	80059	75642	75865	75642	1	1
2	2	1	49034	52013	47505	47505	1.592295548	0.796147774
3	4	1	32478	32074	32874	32074	2.358358795	0.589589699
4	6	1	29093	27745	28520	27745	2.726329068	0.454388178
5	8	1	22886	23542	23439	22886	3.30516473	0.413145591
6	10	1	18523	18547	17487	17487	4.325613313	0.432561331
7	12	1	14268	14480	15027	14268	5.301513877	0.441792823
8	14	1	14229	13559	13286	13286	5.693361433	0.406668674
9	16	1	12700	12166	12704	12166	6.217491369	0.388593211
10	18	1	11196	10954	12181	10954	6.905422677	0.383634593
11	20	1	10398	10433	9806	9806	7.713848664	0.385692433
12	22	1	9607	9869	10854	9607	7.873633809	0.357892446
13	24	1	9324	9814	9878	9324	8.112612613	0.338025526
14	26	1	9546	9531	8599	8599	8.796604256	0.338330933
15	28	1	8176	7537	8271	7537	10.03608863	0.358431737
16	30	1	7385	6716	7057	6716	11.26295414	0.375431805
17	32	1	5977	6171	6298	5977	12.6555128	0.395484775
18	2	2	46438	45718	47465	45718	1.654534319	0.82726716
19	4	2	29524	29419	32162	29419	2.571195486	0.642798871
20	6	2	21848	21331	21281	21281	3.554438231	0.592406372
21	8	2	17334	16216	16067	16067	4.707910624	0.588488828
22	10	2	14617	14615	14772	14615	5.175641464	0.517564146
23	12	2	11593	11433	11489	11433	6.616111257	0.551342605
24	14	2	9793	9633	9789	9633	7.852382435	0.56088446
25	16	2	8793	8833	8889	8793	8.602524736	0.537657796
26	18	2	8352	8390	8285	8285	9.129993965	0.507221887
27	20	2	7854	7962	7871	7854	9.631016043	0.481550802
28	22	2	7431	7398	7457	7398	10.22465531	0.46475706
29	24	2	6963	6997	6907	6907	10.95149848	0.456312437
30	26	2	6591	6434	6440	6434	11.75660553	0.452177136
31	28	2	5933	5870	5953	5870	12.88620102	0.460221465
32	30	2	5593	5603	5591	5591	13.52924343	0.450974781

#	p	g	$T_p^{(1)}$	$T_p^{(2)}$	$T_p^{(3)}$	T_p = min()	$S_p = T_1/T_p$	$E_p = S_p/p$
33	32	2	5302	5358	5336	5302	14.26669181	0.445834119
34	2	4	41512	41023	43530	41023	1.843892451	0.921946225
35	4	4	24744	23455	22774	22774	3.321419162	0.830354791
36	6	4	17730	18410	16586	16586	4.560593271	0.760098879
37	8	4	14096	14940	12515	12515	6.044107072	0.755513384
38	10	4	10994	10417	10703	10417	7.261399635	0.726139964
39	12	4	8568	8539	8634	8539	8.858414334	0.738201195
40	14	4	7865	7850	7813	7813	9.68155638	0.691539741
41	16	4	7298	7175	7111	7111	10.63732246	0.664832654
42	18	4	6777	6869	6878	6777	11.16157592	0.620087551
43	20	4	6526	6494	6509	6494	11.64798275	0.582399138
44	22	4	6196	6187	6189	6187	12.22595765	0.555725348
45	24	4	5811	5951	5872	5811	13.01703665	0.542376527
46	26	4	5661	5565	5654	5565	13.59245283	0.522786647
47	28	4	5497	5264	5320	5264	14.36968085	0.513202888
48	30	4	5098	5062	5044	5044	14.9964314	0.499881047
49	32	4	4989	4771	4821	4771	15.85453783	0.495454307

- **Графика на времето спрямо броя нишки и грануарност.**

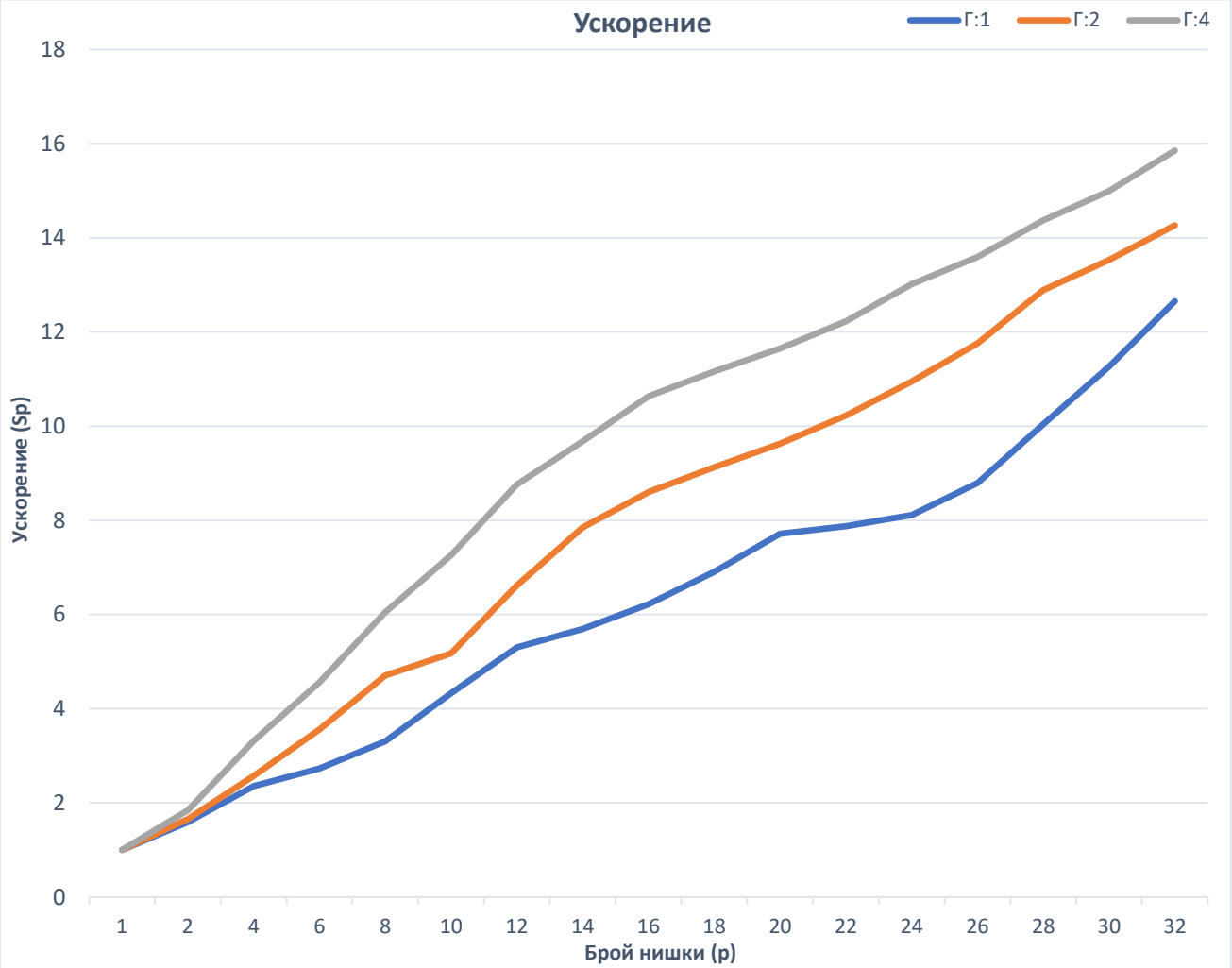


T_1 - времето за изпълнение на серийната програма (или програмата използваща една нишка/един компютър).

T_p - времето за изпълнение на паралелната програма, използваща p нишки.

Г:№ - грануларността.

- **Графика на ускорението спрямо броя нишки и грануарност.**

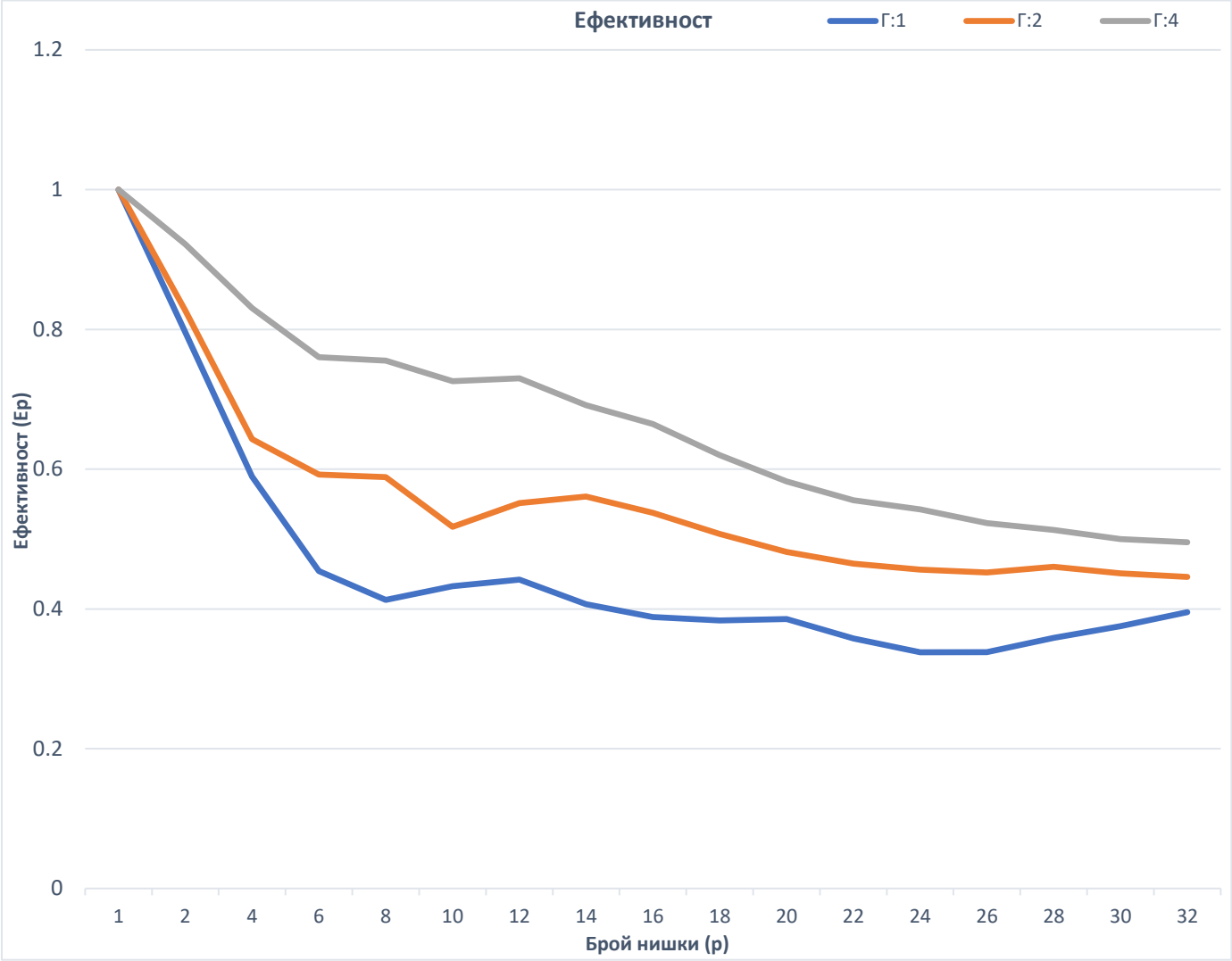


S_p - ускорението, което нашата програма има при използването на p нишки.

Ускорението се изчислява по формулата: $S_p = T_1/T_p$

Г:№ - грануларността.

- **Графика на ефективността спрямо броя нишки и грануарност.**



E_p - ефективността на нашата програма при използването на p нишки.

Ефективността се изчислява по формулата: $E_p = S_p/p$

Г:№ - грануларността.

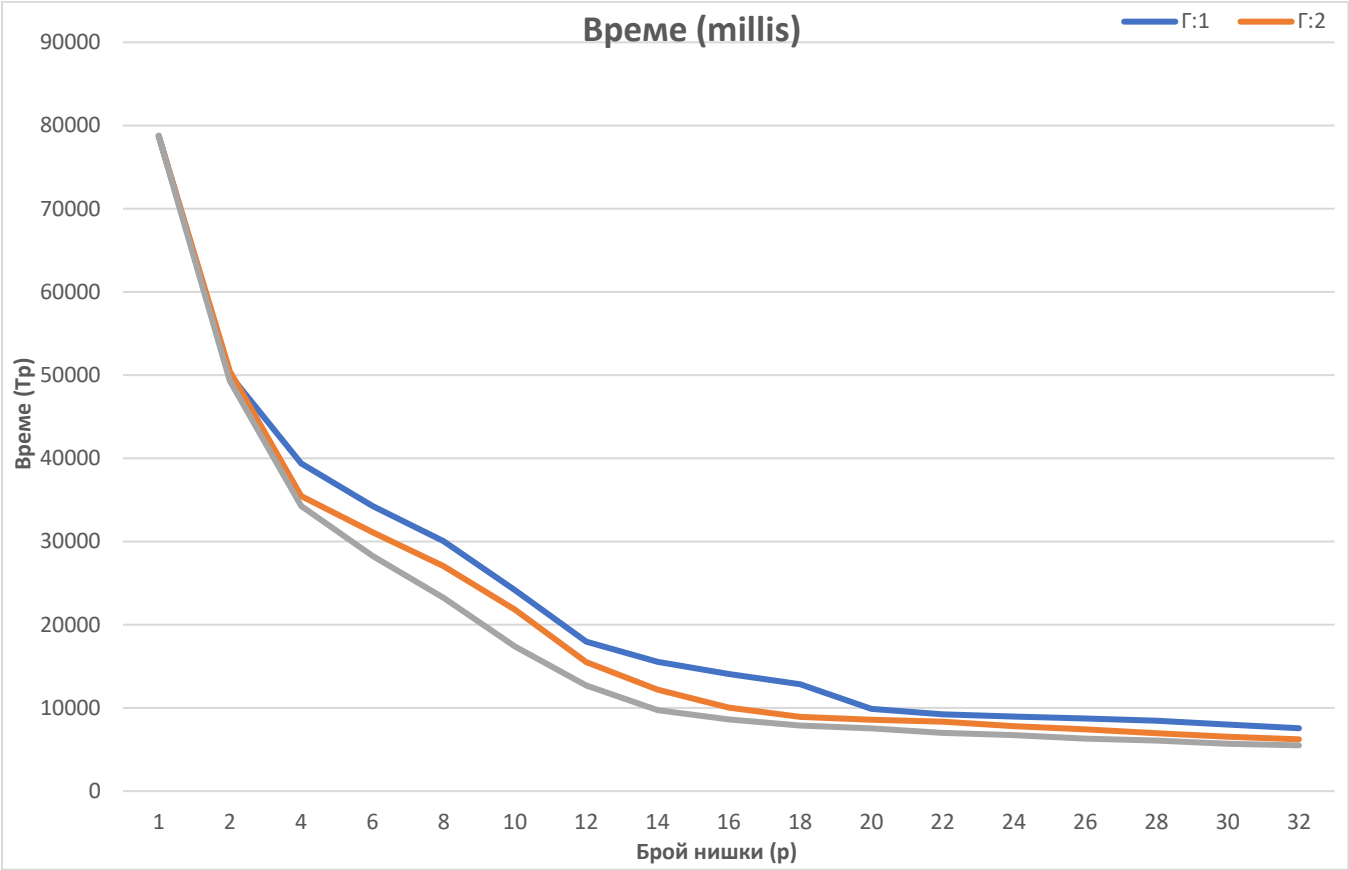
P - броя нишки, S_p – ускорение, E_p – ефективност, g-грануарност

- **Таблица 2.** Резултати при динамично балансиране

#	p	g	$T_p^{(1)}$	$T_p^{(2)}$	$T_p^{(3)}$	$T_p = \min()$	$S_p = T_1/T_p$	$E_p = S_p/p$
1	1	1	82574	78769	83474	78769	1	1
2	2	1	53447	52859	50043	50043	1.574026337	0.787013169
3	4	1	39371	39440	39476	39371	2.000685784	0.500171446
4	6	1	34356	34266	34471	34266	2.298750948	0.383125158
5	8	1	30096	30488	30039	30039	2.622224442	0.327778055
6	10	1	24353	25221	24169	24169	3.259092226	0.325909223
7	12	1	18014	18302	17981	17981	4.380679606	0.365056634
8	14	1	15573	15761	15529	15529	5.072380707	0.362312908
9	16	1	14646	14561	14094	14094	5.588832127	0.349302008
10	18	1	13106	13957	12845	12845	6.132269366	0.340681631
11	20	1	9974	9901	9912	9901	7.955661044	0.397783052
12	22	1	9243	9287	9324	9243	8.522016661	0.387364394
13	24	1	8949	9039	8950	8949	8.801989049	0.366749544
14	26	1	8864	8715	8814	8715	9.038324727	0.347627874
15	28	1	8466	8492	8514	8466	9.304157808	0.33229135
16	30	1	8045	8058	8013	8013	9.830151005	0.3276717
17	32	1	7694	7946	7562	7562	10.41642423	0.325513257
18	2	2	53744	50459	57290	50459	1.561049565	0.780524782
19	4	2	35699	35472	37850	35472	2.220596527	0.555149132
20	6	2	31112	31201	31851	31112	2.531788377	0.42196473
21	8	2	28963	27063	27030	27030	2.914132445	0.364266556
22	10	2	22601	23476	21792	21792	3.614583333	0.361458333
23	12	2	15763	15500	15911	15500	5.081870968	0.423489247
24	14	2	14227	13897	12197	12197	6.458063458	0.461290247
25	16	2	10963	10057	10992	10057	7.83225614	0.489516009
26	18	2	9624	9312	8926	8926	8.824669505	0.490259417
27	20	2	8797	8565	8817	8565	9.196614127	0.459830706
28	22	2	8363	8511	8412	8363	9.418749253	0.428124966
29	24	2	7826	7859	7994	7826	10.06503961	0.41937665
30	26	2	7425	7560	7485	7425	10.60861953	0.408023828

#	p	g	$T_p^{(1)}$	$T_p^{(2)}$	$T_p^{(3)}$	T_p = min()	$S_p = T_1/T_p$	$E_p = S_p/p$
31	28	2	7257	7001	6953	6953	11.32877894	0.404599248
32	30	2	6889	6755	6530	6530	12.062634	0.4020878
33	32	2	6534	6526	6229	6229	12.64552898	0.395172781
34	2	4	52802	52572	49294	49294	1.597942955	0.798971477
35	4	4	34570	34432	34278	34278	2.297946205	0.574486551
36	6	4	28893	28287	28445	28287	2.784636052	0.464106009
37	8	4	23236	24235	24868	23236	3.389955242	0.423744405
38	10	4	18409	17374	19350	17374	4.53372856	0.453372856
39	12	4	14878	13919	12695	12695	6.20472627	0.517060523
40	14	4	9908	9726	9765	9726	8.098807321	0.578486237
41	16	4	8602	8773	8844	8602	9.157056498	0.572316031
42	18	4	7907	8084	8009	7907	9.961932465	0.553440692
43	20	4	7610	7596	7530	7530	10.46069057	0.523034529
44	22	4	7031	7009	7142	7009	11.23826509	0.510830231
45	24	4	6816	6729	6850	6729	11.70589984	0.487745827
46	26	4	6666	6552	6293	6293	12.51692357	0.481420137
47	28	4	6937	6064	6181	6064	12.98961082	0.463914672
48	30	4	5915	5698	5860	5698	13.82397332	0.460799111
49	32	4	5577	5761	5504	5504	14.3112282	0.447225881

- **Графика на времето спрямо броя нишки и грануарност.**

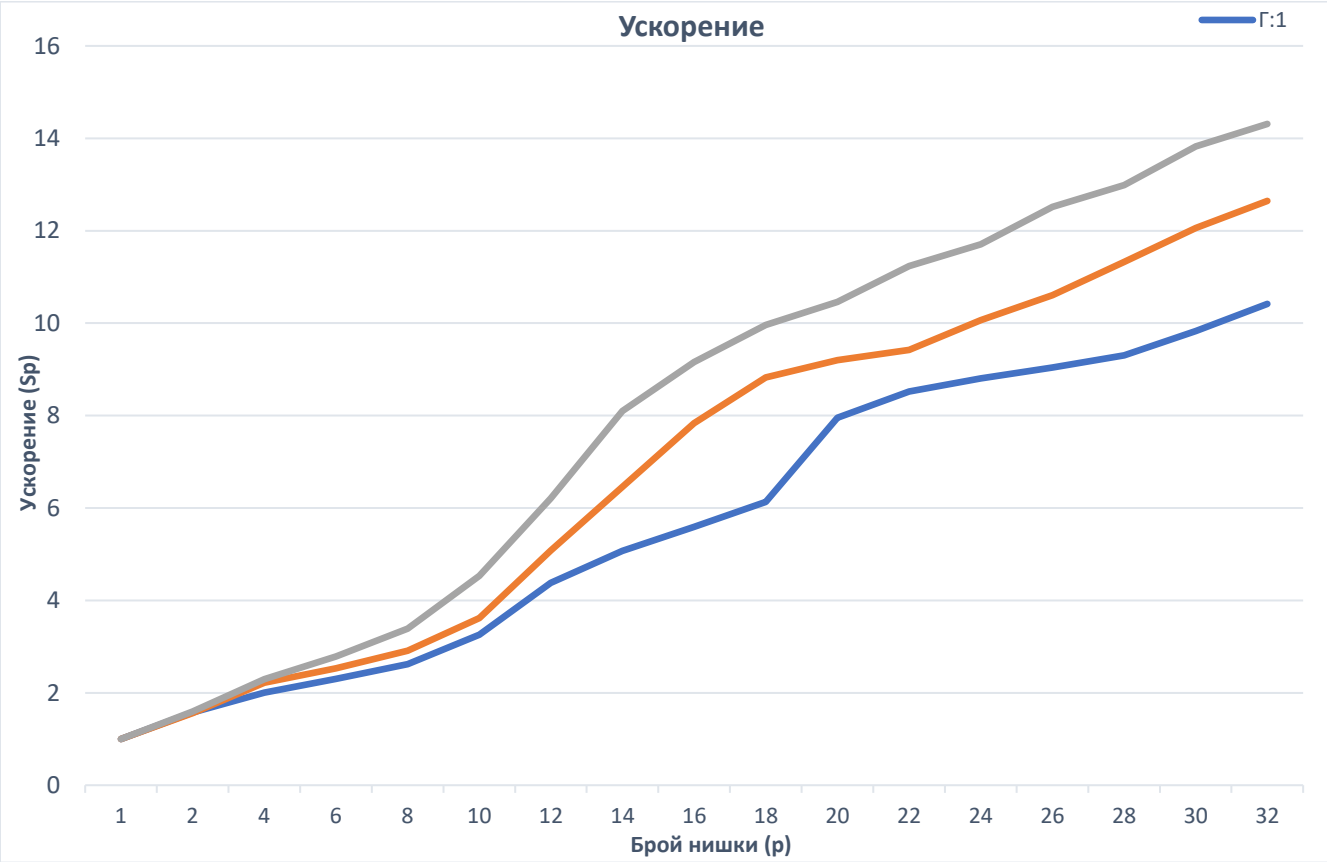


T_1 - времето за изпълнение на серийната програма (или програмата използваща една нишка/един компютър).

T_p - времето за изпълнение на паралелната програма, използваща p нишки.

Г:№ - грануларността.

- **Графика на ускорението спрямо броя нишки и грануарност.**

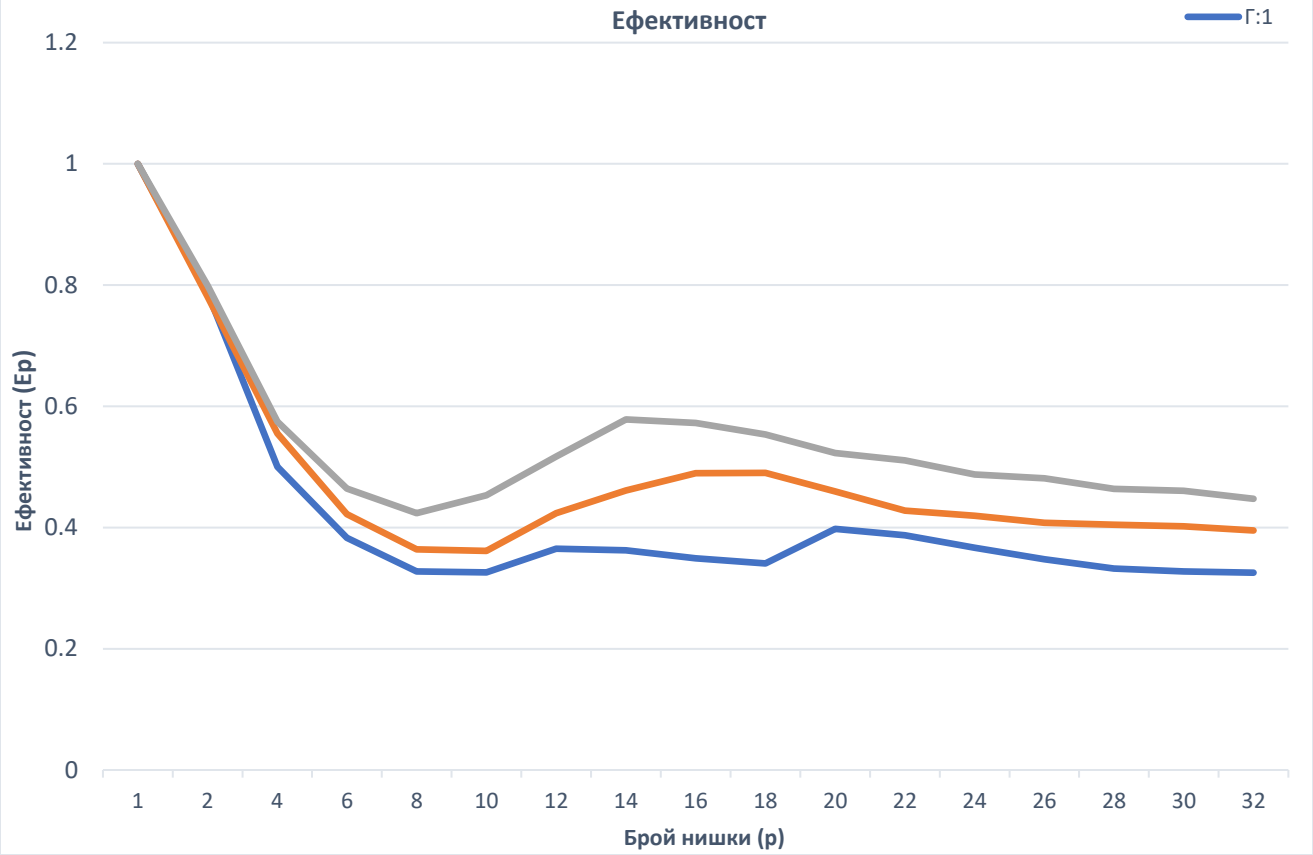


S_p - ускорението, което нашата програма има при използването на p нишки.

Ускорението се изчислява по формулата: $S_p = T_1/T_p$

Г:№ - гранулярността.

- **Графика на ефективността спрямо броя нишки и грануарност.**



E_p - ефективността (ефикасността) на нашата програма при използването на p нишки.

Ефективността се изчислява по формулата: $E_p = S_p/p$

Г:№ - грануларността.

3.1. Обобщение

На диаграмите се вижда, че при най-едрата грануларност (1) ускорението и ефективността са значително по-ниски в сравнение с грануларности (2, 4). С нарастване на броя нишки ефектът на грануларността се увеличава, като при 32 нишки най-финната грануларност отчита най-високо ускорение и най-добра ефективност. Най -добър резултат дава при статичното балансиране. Забелязва се ускорението, по-добро време и екефтиност при повече нишки и при по-голяма грануарност. Но при динамично балансиране

нишките извършват неравномерно брой подзадачи, това означава, че може да има нишка, която са решени в пъти повече подзадачи от друга нишка. Което довежда, че има нишки, които са дълго не активни в определен момент в програмата. При което не е оползотворен максимално дадените ресурси. Ускорението намалява заради вкарването по време на самата обратока допълнителни работа от самия PoolExe. Докато при статично балансиране нишките извършват еднакъв брой подзадачи или с минимална разлика сравнение при динамичното. Ресурсите се оползотворяват пълноценно. Със същите средства, алгоритъм, нишки и грануарност с статичното балансиране алгоритъма за намиране на e (неперевото число) е по-добър. В този проблем, който беше обяснен погоре, статичното балансиране е по-доброто тука за решаване на реда на e (неперевото число).

4.Източници

[1] Peter Luschny, A new kind of factorial function, 2008, (https://oeis.org/A000142/a000142.pdf);

[2] Peter Luschny, Fast Factorial Functions, 2002, (http://www.luschny.de/math/factorial/FastFactorialFunctions.htm);

[3] Pete Luschny, Parallel Prime Swing, 2010, (https://github.com/PeterLuschny/Fast-Factorial-Functions/blob/master/JavaFactorial/src/de/luschny/math/factorial/FactorialParallelPrimeSwing.java);

[4] Peter Luschny, The most efficient factorial algorithms, (http://www.luschny.de/math/factorial/conclusions.html);

[5] Wikipedia contributors, “Master/slave (technology)”, Wikipedia, The Free Encyclopedia, last edited on 20 May 2020, https://en.wikipedia.org/wiki/Master/slave_(technology)