Паралелна обработка на Манделброт. Изчислителна сложност, грануларност. Циклично статично срещу централизирано динамично балансиране.

Симеон Христов

Факултет по математика и информатика, Софийски университет

Август 2021

Целта на тази статия е да изследва параметри и начини за разпределяне на натоварването на една и повече нишки при паралелна обработка на матрица. Направени са сравнения в избора на подход с други статии.

Функционален анализ

Преглед на източници

За осъществяване на целта и демонстриране на резултатите е избрана задачата за генерирането на визуализация на множеството на Манделброт. Тя е била (и все още е) обект на множество изследвания от страна на математици и информатици [2, 4], защото може да се генерира от проста математическа формула и представлява фрактал: от едра страна, фигура с приложение в няколко естествени структури [4], и от друга - фигура, генерирането на която може да служи за илюстриране на нови идеи за паралелна обработка [2]. Генерирането се извършва чрез краен брой итерации

върху комплексно квадратно уравнение [1, 5, 6, 7]. Спрямо този брой на всеки пиксел се причислява цвят. Ако броят е равен на итерациите, то изследваната точка е част от множеството и никога няма да го напусне. В литературата тази проверка е известна като тестът на Манделброт.

За по-ефективна обработка съществуват няколко критерия за предопределяне на принадлежността на точка към множеството на Манделброт. Най-популярната метрика е Евклидово разстояние от точката до началото на координатната система [2, 4, 5, 6, 7], според която ако стойността на точката стане по-голяма от определена константа (напр. 2 [2, 5]), то точката не е от множеството. Съществуват обаче и методи, произлизащи от физиката и електростатиката, които могат се използват в случаите на генериране на множеството на Джулия [1], но те са извън обхвата на тази статия.

Въпреки контекста на матрици и реални числа, нерядко се използват две оси с реални числа и цикли по избраните размерности с транслирания на всяка точка в по-вътрешния цикъл [2, 3, 5, 6].

Ефективни начини за генериране на точки от множеството представя Pughineanu в [6]. Освен детайлно разглеждане на възможни точки, върху които да бъде итерирана квадратната функция, той дискутира и различни прозрачни за програмиста причини, поради които може да не се получи ускорение или пък да се получи свръхлинейно такова. Освен като мотивация за избрания начин на статично циклично разпределяне на задачите по редове, тази статия е и източник на примерен код за осъществяването на целите.

Обширни са дискусиите за вида декомпозиция, който може да се приложи върху матрицата. В [5] Gang разглежда и прави коментар на някои от тях. Поради хоризонталната симетричност на множеството на Манделброт най-интуитивното разпределение е по-редове. Както той описва, ако се пуснат само две нишки матрицата се разбие на две части, всяка с по ВИСОЧИНА/2 реда, то балансирането е абсолютно точно, ефективността ще бъде най-голяма и ускорението ще бъде линейно. Това обаче е един от малкото частни случаи, в които може да се постигне подобна ситуация. Много по-често ще е желателно да се постигне по-голямо ускорение от два пъти. Тук започват проблемите (Рис. 1).

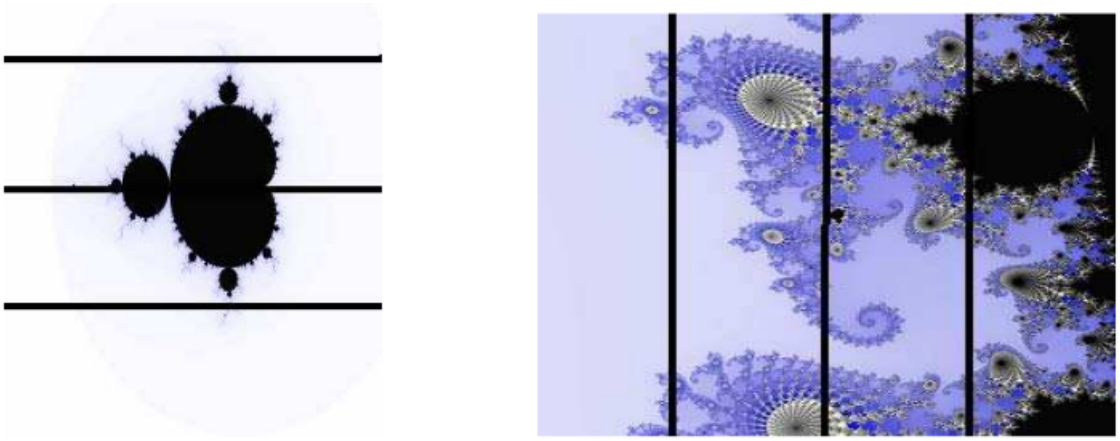

Рис. 1: Визуализация на разпределението на работата на 4 нишки, пуснати по редове (в ляво) и по колони (в дясно). Черната линия е границата между две нишки.

Черните части на картинките са зони, за които трябва да се направи най-много работа. В дясната картинка от общо четири нишки само две ще извършат почти цялата работа. Подобен е проблемът и в лявата картинка.

Друго интересно откритие е свързано със зависимостта по време за изпълнение. Оказва се, че, когато разпределението е по-колони, в случаите, когато изчислителната сложност е правоъгълник с по-голяма дължина отколкото широчина може да се

постигне по-бързо обработване, т.к. дължината на колоната е по-малка. Тези случаи обаче също са частни, т.к. в [2, 3, 5, 7] матрицата е квадратна и подобни резултати не са докладвани. Също, както и Gang докладва в [5], тогава нишките заемат повече на брой колони и в този случай едрата грануларност няма да доведе до по-голямо ускорение. Освен това значение има и избраната област - най-добре е тя да бъде фокусирана върху небалансирана част от фрактала.

С това е свързано и решението, предложено в [5] - стохастично статично балансиране с най-фина грануларност. Чрез случайно разпределяне на пикселите от два вектора Gang постига по-добри резултати. Тези резултати обаче могат да бъдат по-добрени още (Рис. 2).

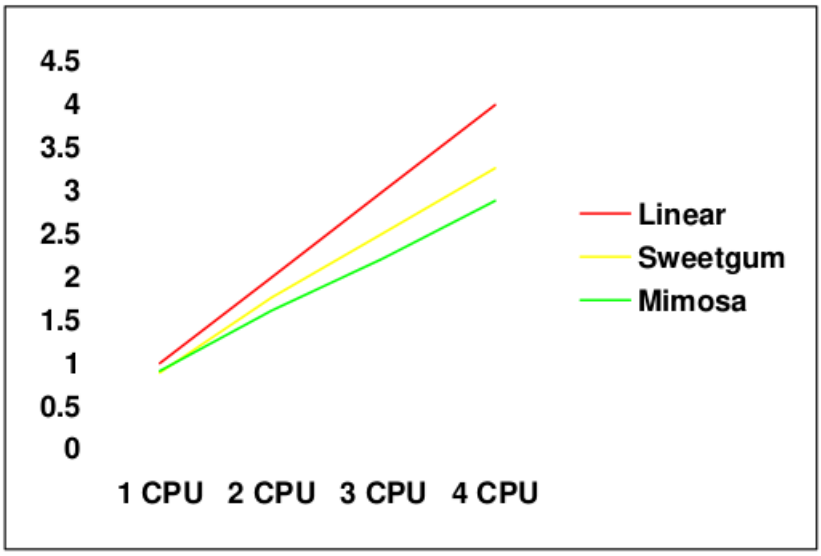

Рис. 2: Ускорението, което постига Gang в [5] чрез използване на два суперкомпютъра от Центърът на Мисисипи на проучвания със суперкомпютри.

Най-доброто ускорение е 3 при 4 процесора. При същите параметри - изчислителна сложност 10000x10000 и максимум 1000 итерации, ParMadel в развойна среда постига

ускорение 3.92 при циклично статично разпределение и средна грануларност.

Tracolli в [7] и Kuchen et al. в [3] демонстрират как за централизирано динамично балансиране може да се използват вече вградени инструменти за създаване и управление на множество нишки - ThreadPoolExecutor. Проведените им изследвания кореспондират с резултатите, постигнати от ParMandel - при (сравнително) малка изчислителна сложност и средна грануларност цикличното статично балансиране се справя по-добре от централизираното разпределено. Главната причина за това е свръхтоварът, които се поражда на логическо ниво, в нишката, която разпределя задачите (наричана по-долу "главна"). Свръхтоварът е резултат от характеристиките на този вид балансиране. Главната нишката следи състоянието на останалите нишките от една страна и състоянието на опашката със задачи от друга страна (информационна стратегия). На базата на тази информация, тя преценява на коя нишка да даде следващата задача (това действие е част от нейната стратегия за локализация). След като вземе решение тя прави трансфер на нужната информация. Очаквано, големината на паралелизма е правопропорционална на големината на свръхтовара. С други думи, всички тези действия не са проблем, когато паралелизмът е едноцифрено число, но с увеличаването му работата става неефективна.

Сравнителни таблици

Източник	Изчислителна сложност	Максимален брой нишки	Грануларност
1	595x842	8	едра
2	1024x1024	8	едра
3	8192x8192	64	едра
4	-	100	фина
5	10000x10000	4	фина
6	480x640	4	средна
7	100x100 - 12800x12800	16	фина до едра
ParMandel	4096x2160, 7680x4320	32	фина до едра

Таблица 1: Сравнение на първа част от параметрите, използвани в намерените източници и ParMandel.

Източник	Дълбочина	Декомпозиция	Вид балансиране
1	600	редове и колони	разпределено динамично
2	100000	редове	централизирано и разпределено динамично
3	-	редове	централизирано динамично и циклично статично
4	1000	пиксели	циклично статично и стохастично статично
5	1000	пиксели	стохастично статично
6	256	редове	циклично статично
7	10000	блокове	циклично статично и централизирано динамично
ParMandel	1024	редове	циклично статично и централизирано динамично

Таблица 2: Сравнение на втора част от параметрите, използвани в намерените източници и ParMandel.

Модел на паралелното приложение

Реализиран е модел на декомпозиция по данни (data decomposition) и съответно SPMD обработка чрез цикли по двете размерности и транслирания на точките. Точките формират две реални оси, вместо една реална и една комплексна. Използвани са две вариации на изчислителната сложност: 4096x2160 (4K) и 7680x4320 (8K). Броят на итерациите бе вариран измежду 256 и 1024, но понеже за първия вариант,

ускорението беше незначително, пълните тестове бяха проведени само с 1024 итерации. Като метрика за принадлежността на точка към множеството на Манделброт се използва Евклидовото разстояние между нея от началото на координатната ос с граница 2. Балансирането е статично циклично по (индекси на) редове като всяка нишка в зависимост от нейния индекс в масива обработва няколко реда накуп преди да се премести през определена стъпка до следващата група редове. Тази стъпка се определя от броя редове, които се обработват от всяка нишка и броя нишки, пуснати за изпълнението на цялата задача.

С илюстративна цел е реализирано и ръчно написано централизирано динамично разпределение на задачите. Реализиран е Master-Slave моделът [3, 7]. В началото главната нишка построява масив от индекси на групите редове, които трябва да бъдат обработени на база получените командни параметри, които са същите и при двата вида разпределяне. След построяване на масива, циклично се разпределят първите p, където p е броят на всички нишки-работници. В случай че грануларността е най-едра, дължината на масива от задачи съвпада с броя на пуснатите нишки и комуникация между главната нишка и работниците няма. Когато грануларността е по-фина, всяка нишка, приключила със своята задача, се обръща към публичен метод на Master-а, за да получи индекса на следващата група редове. По този начин работа с масива извършва само главната нишка, която споделя с останалите само стойността в масива със задачи на индекса на първата група необработени редове. Тази комуникация не оказва влияние спрямо статичното циклично разпределяне при най-едра грануларност, защото тогава комуникация няма, но при повече нишки и по-фина грануларност се наблюдава забавяне. То се поражда, когато повече от една нишка заяви желание да получи следваща задача.

Последователностна диаграма на процесите

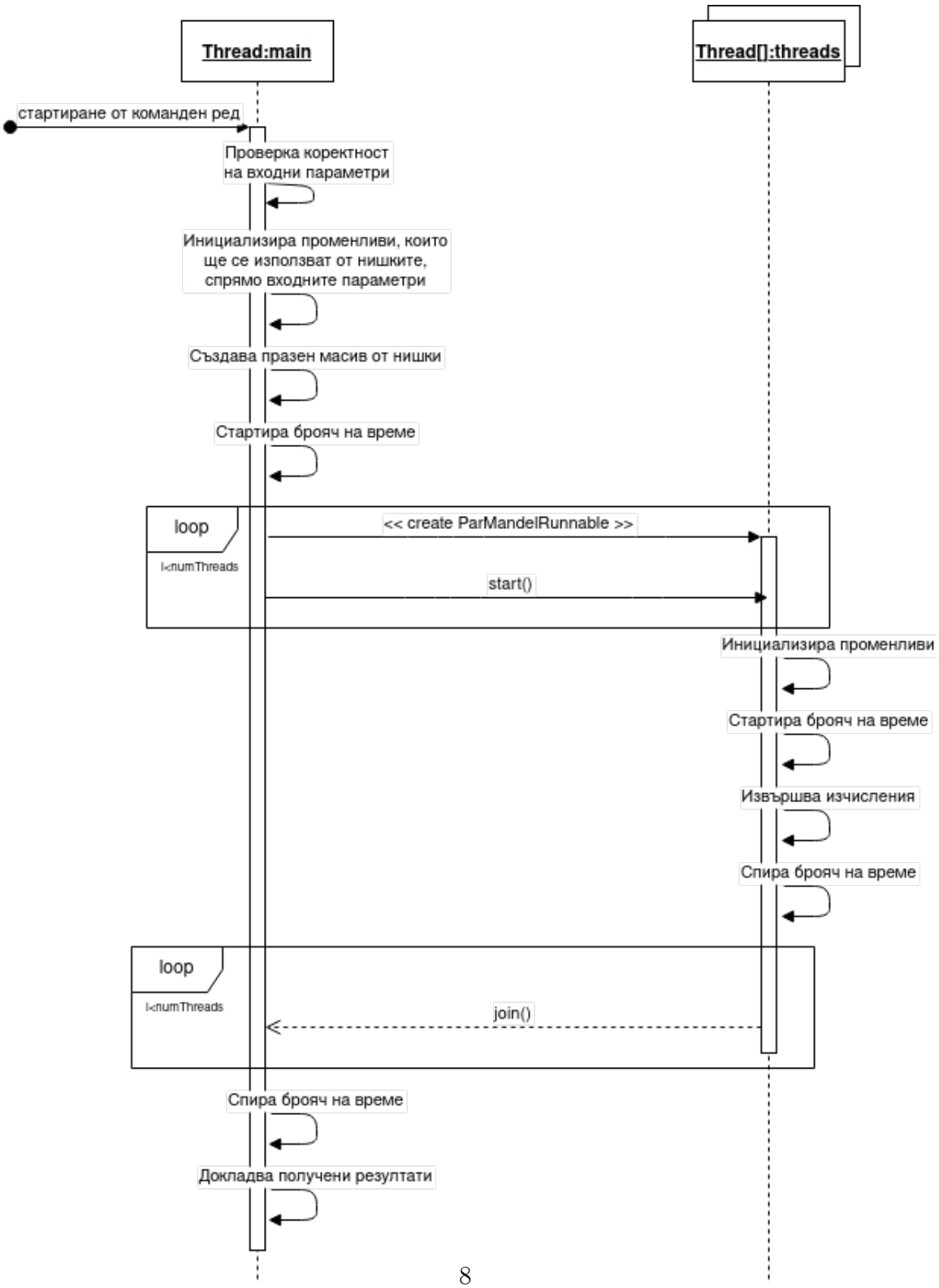

Рис. 3: Последователностна диаграма на процесите при статично циклично разпределяне на задачите.

Опис на командните параметри

ParMandel работи с три командни параметъра. Подаването на различен брой или подаването на стойности извън дефиниционното множество предизвиква показването на подсказка от страна на програмата, след което тя спира работа. Параметрите нямат имена, т.е. се характеризират единствено по реда, в който са подадени.

Индекс на команден аргумент	Семантика	Дефиниционно множество
0	брой нишки	**N**
1	грануларност	**N**
2	резолюция	4 (за 4096x2160) или (8 за 7680x4320)

Таблица 3: Таблица, представяща възможните командни параметри и тяхното значение за ParMandel.

ParMandel разполага с три скрипта, с помощта на които може да се компилира към изпълним код и стартира за едно или повече изпълнение на генерирания код. Те са наименувани makeMe.sh, runMe.sh, work.sh. Първият компилира програмата. Вторият стартира едно изпълнение. Третият стартира множество изпълнения и е много удобен за автоматично тестване на приложението. Подобни са скриптовете за работа с динамичното централизирано разпределяне. Единствената разлика е, че името има суфиксът "_mpmd".

Примерно пускане на програмата:

> cd ParMandel && ./makeMe.sh && ./runMe.sh 16 128 8

Горният код компилира и стартира ParMandel, задавайки софтуерен паралелизъм 16, грануларност 128 и резолюция 7680x4320.

```
#!/bin/bash

cs=(4 8)
gs=(1 5 10 16 32 64 128)
ps=(1 2 4 8 12 16 20 24 28 32)
for c in "${cs[@]}"; do
  for g in "${gs[@]}"; do
    for p in "${ps[@]}"; do
      ./runMe.sh $p $g $c
    done
  done
done
```

Рис. 4: Съдържание на файлът work.sh.

Технологично проектиране

Сравнителна таблица

Всички източници, които публично са обявили езикът, чрез който са провеждали проучването са избрали C++. ParMandel е написан на Java.

Източник	Ядра	Платформа	Комуникация	Процесор	Кеш данни
1	-	клъстер	MP	UltraSPARC-IIi	16KB
2	1x4	лаптоп	MP	AMD Ryzen	48KB
3	2x6	клъстер	MP и SV	Intel Xeon Westmere	32KB
4	-	сървър	SV	-	-
5	1x128	суперкомпютър	MP	SGI Origin 2800	-
6	-	суперкомпютър	-	-	-
7	13x4	клъстер	MP	Intel	-
Acer Aspire t5600	1x4 2x8	лаптоп сървър	SV SV	Intel Core i5 Intel Xeon E5-2660	32KB 32KB

Таблица 4: Сравнение на хардуерните характеристики в намерените източници и използвана развойна (Acer Aspire) и тестова среда.

Функция и код за стартиране на p нишки

```
for (int i = 0; i < numThreads; i++) {
    ParMandelRunnable runnable = new ParMandelRunnable(
            i, numThreads, rowsPerThread, HEIGHT, WIDTH
    );
    threads[i] = new Thread(runnable);
    threads[i].start();
}
```

Рис. 5: Кодът, използван за стартиране на зададения брой нишки, при статично циклично разпределяне на задачите.

```
tasksIdxs = new int[numTasks];
tasksIdxs[0] = 0;
for (int i = 1; i < numTasks; i++) {
    tasksIdxs[i] = tasksIdxs[i - 1] + rowsPerThread;
}
Thread[] threads = new Thread[numThreads];
long tik = System.nanoTime();
for (int i = 0; i < numThreads; i++) {
    if (currentTaskIdx + 1 == tasksIdxs.length) {break;}
    Slave runnable = new Slave(
            WIDTH, HEIGHT,
            tasksIdxs[++currentTaskIdx], rowsPerThread);
    threads[i] = new Thread(runnable);
    threads[i].start();
}
```

Рис. 6: Кодът, използван за постояването на масива от задачи и стартиране на зададения брой нишки, при динамично централизирано разпределяне на задачите.

Функция и код за обмен между нишките

Понеже генерирането на множеството на Манделброт се състои от асинхронни изчисления (такива задачи в литературата се реферират като embarrassingly parallel [4]) няма обмен между нишките.

Ако обаче разпределянето на задачите е динамично, то нишките-работници комуникират с главната нишка и създават скаларна антизависимост. С цел предотвратяване на аномалията, в която няколко нишки четат променлива, докато стойността ѝ се променя (известна като race condition), се използва възможността на Java за ограничаване (заключване) на изпълнението на даден метод до една нишка в кода на Master-а (Рис. 7).

```
public synchronized static int getNextJob() {
    if (currentTaskIdx + 1 == tasksIdxs.length) {
        return -1;
    }

    return tasksIdxs[++currentTaskIdx];
}
```

Рис. 7: Кодът, към който се обръщат нишките-работници, при нужда да получат нова задача.

Функция и код за системно време

Времето се засича с вградения инструмент в Java System.nanoTime(). Тъй като всяка нишка, която извършва работа, засича времето, което ѝ отнема, има възможност за преглед на натоварването чрез блокова диаграма.

За проверка дали изпълнението е коректно, времето се засича и от нишката, която създава масива. Причината, за двойното засичане, е, за да бъде сигурно, че скоростта на обработка съвпада със скоростта на най-бавната нишка.

Засичането започва точно преди да се започне извършването на работата. При нишката, която създава масивът, началото на засичането е последната операция преди разпределянето на задачите (Рис. 8).

Засичането спира точно след приключването на работата. При нишката, която създава масивът, краят на засичането е първата операция след получаване на сигнал от за приключването на най-бавната нишка (Рис. 8).

Засичането от стартираните нишки, започва при извикването на техния run метод и спира преди той да завърши (Рис. 9).

```
long tik = System.nanoTime();
for (int i = 0; i < numThreads; i++) {
    ParMandelRunnable runnable = new ParMandelRunnable(
            i, numThreads, rowsPerThread, HEIGHT, WIDTH
    );
    threads[i] = new Thread(runnable);
    threads[i].start();
}

for (int i = 0; i < numThreads; i++) {
    try {
        threads[i].join();
    } catch (InterruptedException e) {
        System.out.println("ERROR while joining");
        e.printStackTrace();
    }
}
long tok = System.nanoTime();

System.out.printf("Total=%f\n", (double) (tok - tik) / 1_000_000_000);
```

Рис. 8: Кодът (заедно с работата, която следи), използван за засичане на времето от нишката, която създава масива.

```java
@Override
public void run() {
    long tik = System.nanoTime();
    while (rowStart < HEIGHT) {
        int yEnd = rowStart + NUM_ROWS;

        for (int y = rowStart; y < yEnd && y < HEIGHT; y++) {
            for (int x = 0; x < WIDTH; x++) {
                double translatedX = minX + x * (maxX - minX) / WIDTH;
                double translatedY = minY + y * (maxY - minY) / HEIGHT;
                double cx = translatedX;
                double cy = translatedY;
                int iterations = 0;

                while (iterations < MAX_ITERATIONS) {
                    double trXX = translatedX * translatedX - translatedY * translatedY;
                    double trYY = 2 * translatedX * translatedY;

                    translatedX = trXX + cx;
                    translatedY = trYY + cy;

                    if (Math.abs(translatedX + translatedY) > INFINITY) {
                        break;
                    }

                    ++iterations;
                }
            }
        }

        rowStart += NUM_THREADS * NUM_ROWS;
    }
    long tok = System.nanoTime();

    System.out.printf("\tt=%f\n", (double) (tok - tik) / 1_000_000_000);
}
```

Рис. 9: Кодът (заедно с работата, която следи), използван за засичане на времето от нишките, които извършват работата.

По аналогичен начин се извършва засичането на времето за работа и когато разпределянето на задачите е динамично.

Тестване

Тестов план

Проведени са тестове на развойна - личен лаптоп, и на тестова среда - сървър, предоставен от Факултета по математика и информатика. За всяка стойност на грануларността са проведени по 4 теста, в които се регулира броя нишки. Решенията точно какви параметри да бъдат използвани (напр. броят на итерациите да бъде 1024), бяха взети преди преминаването на тестова среда. Попълнените тестове са налични в Appendix A.

По хоризонталната ос на следващите XY диаграми са нанесени стойностите на използвания софтуерен паралелизъм. По вертикалната ос са нанесени постигнатите ускорения. Сравненията се извършват на база постигнато ускорение за дадена грануларност.

За ускорение се приема пропорцията между времето за изпълнение на задачата при зададената грануларност от една нишка и времето за изпълнение на задачата при зададената грануларност от p нишки.

Изследваната област, в контекста на оригиналните размери на множеството на Манделброт, т.е. [-2, 2] по двете оси, е [-0.8, -0.3] по абсцисната ос и [0.3, 0.8] по ординатната (Рис. 10).

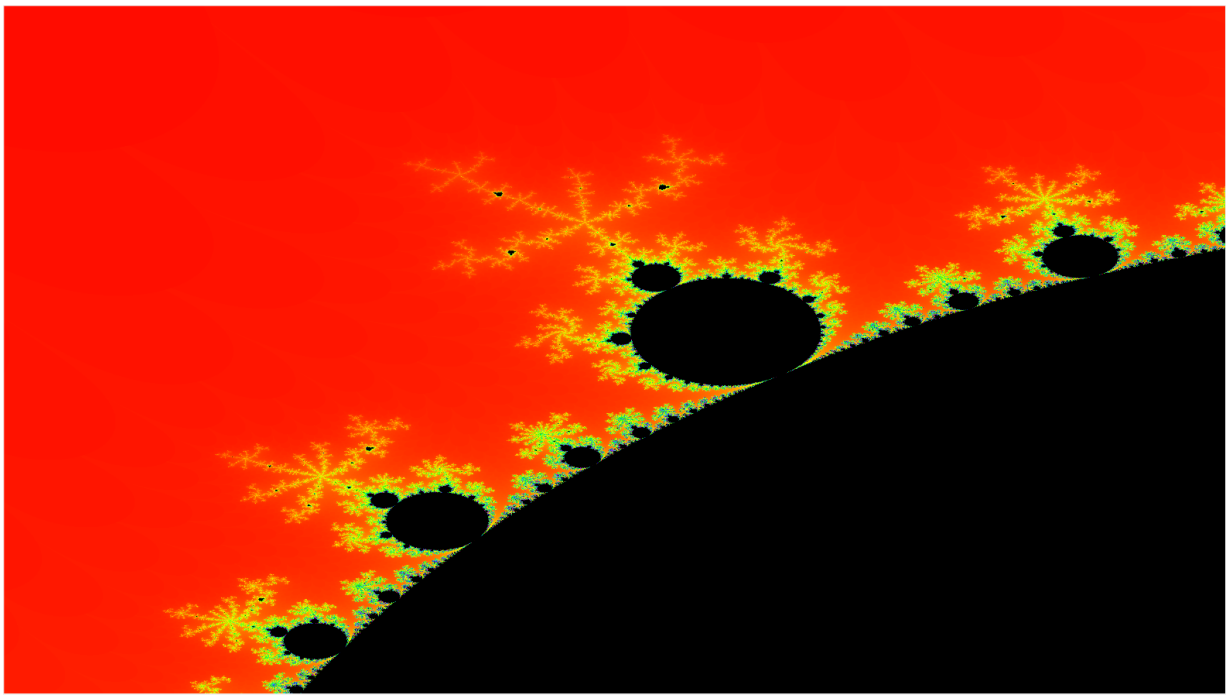

Рис. 10: Визуализация на изследваната област при 1024 итерации и резолюция 4K.

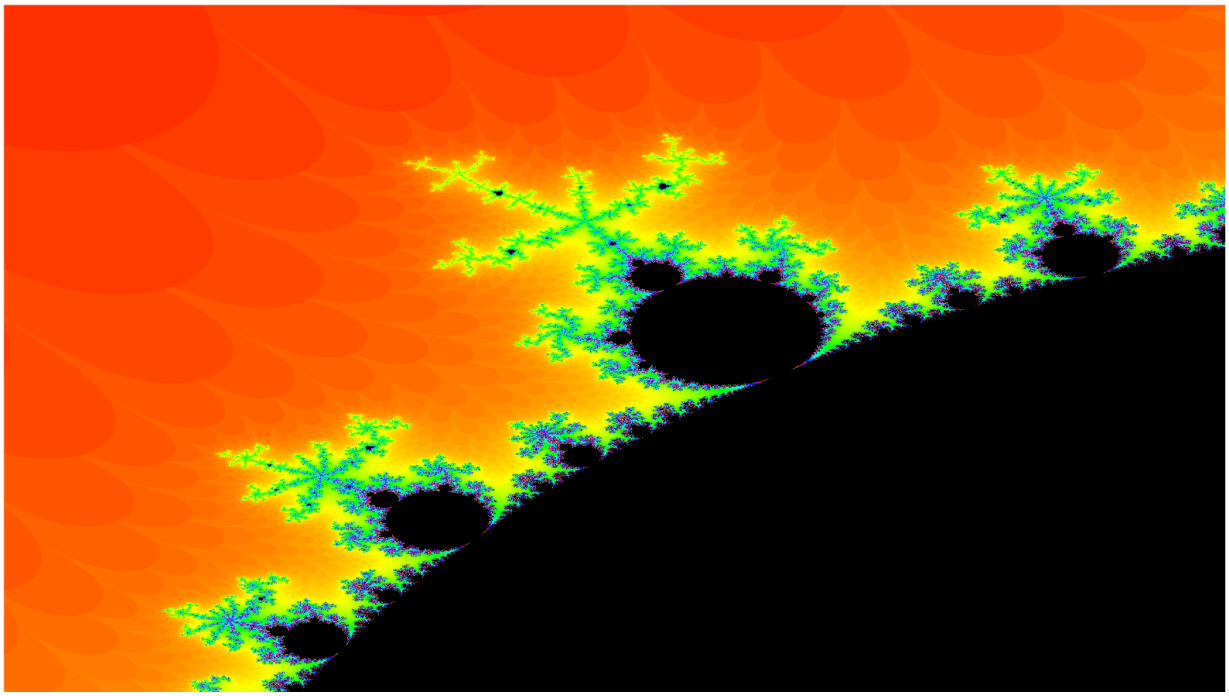

Рис. 11: Визуализация на изследваната област при 256 итерации и резолюция 4K.

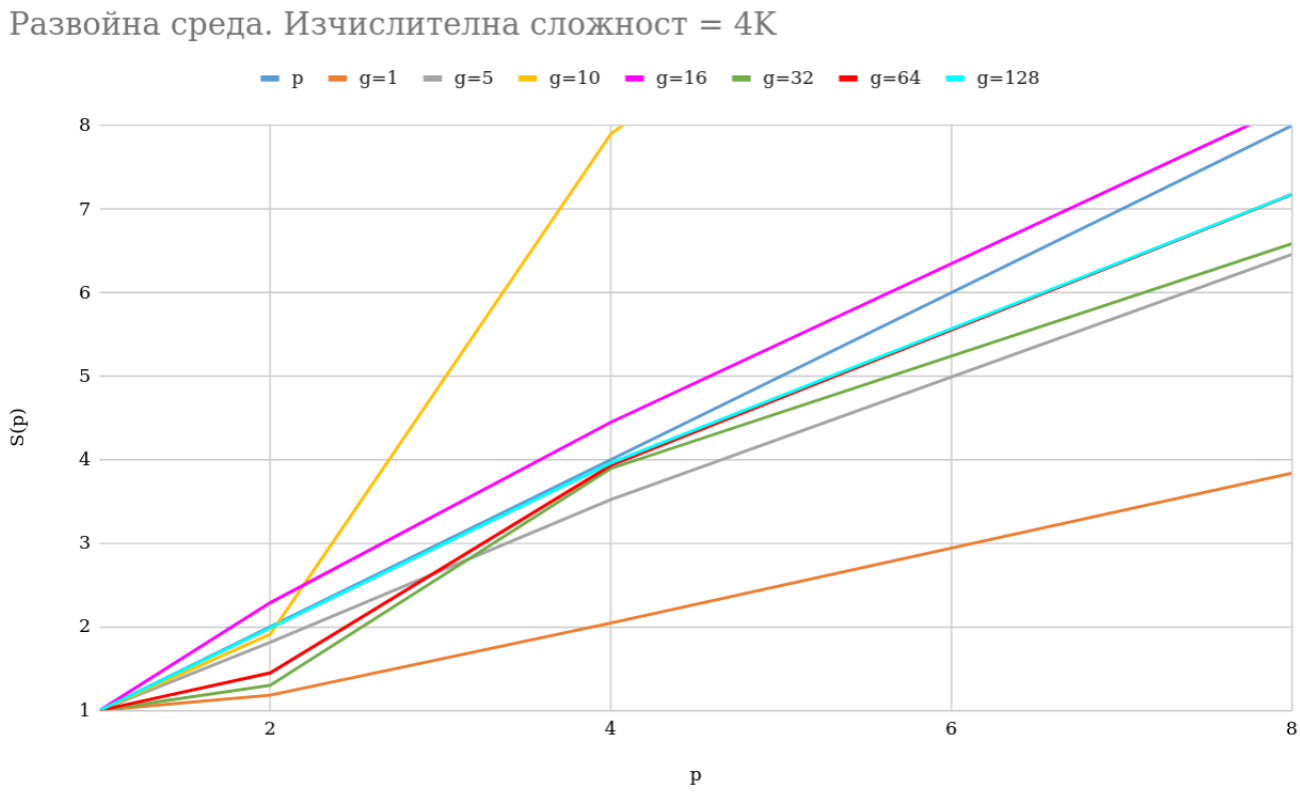

Рис. 12: Визуализация на постигнати ускорения на развойна среда с резолюция 4K. Попълнената таблица е налична в Appendix A.

Наблюдава се свръхлинейно ускорение при грануларност 10 и при грануларност 16. То се дължи на спестяването на скрити операции в процесора и, в следствие, постигната адаптация към кешът за данни, използван от инструкционния конвейер. Очаквано, най-малко е ускорението при грануларност 1, т.е. най-едра, което демонстрира как небалансирана област оказва благоприятно влияние на някои нишки, но задава най-трудните области на 1-2 нишки.

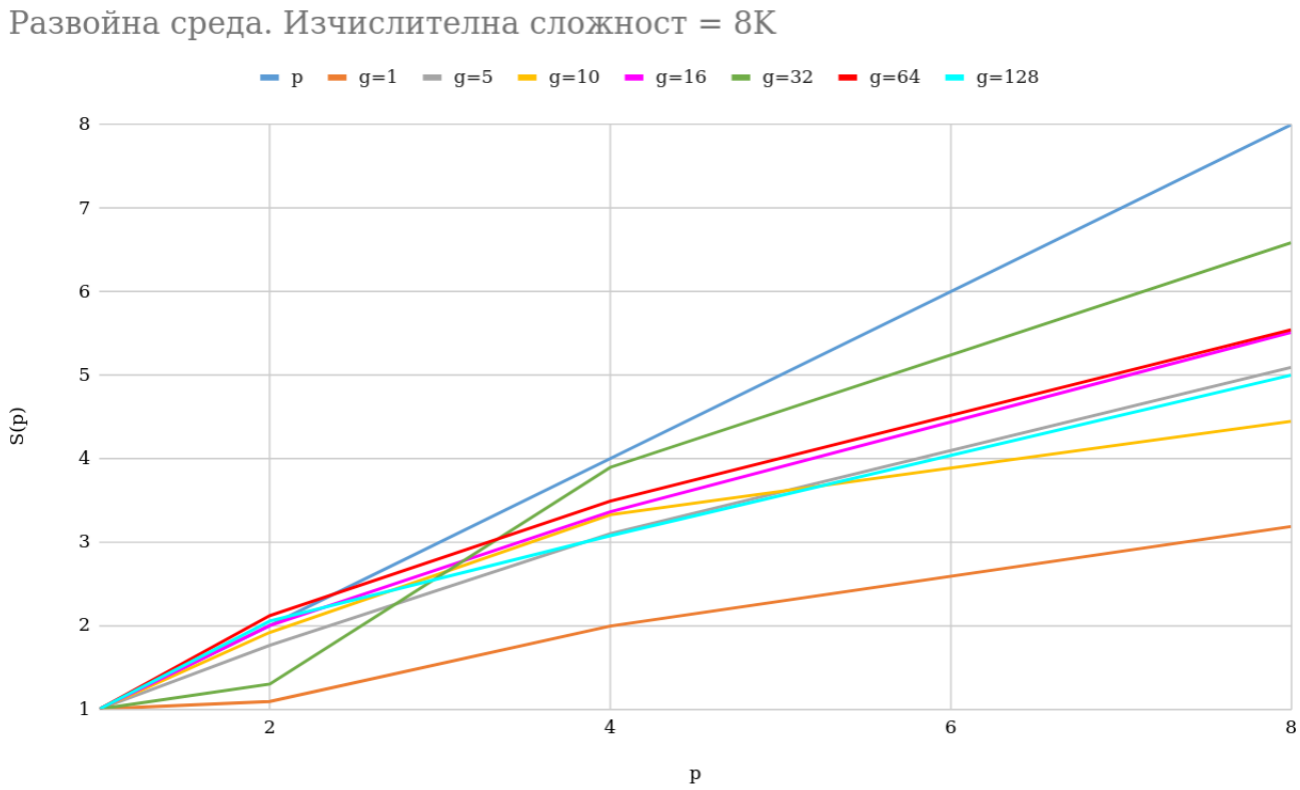

Рис. 13: Визуализация на постигнати ускорения на развойна среда с резолюция 8K. Попълнената таблица е налична в Appendix A.

При по-голяма изчислителна сложност нивото на най-доброто ускорение е по-ниско, но все пак при 4 ядра се запазва тенденцията често да се постига добро ускорение при пускане на 8 нишки.

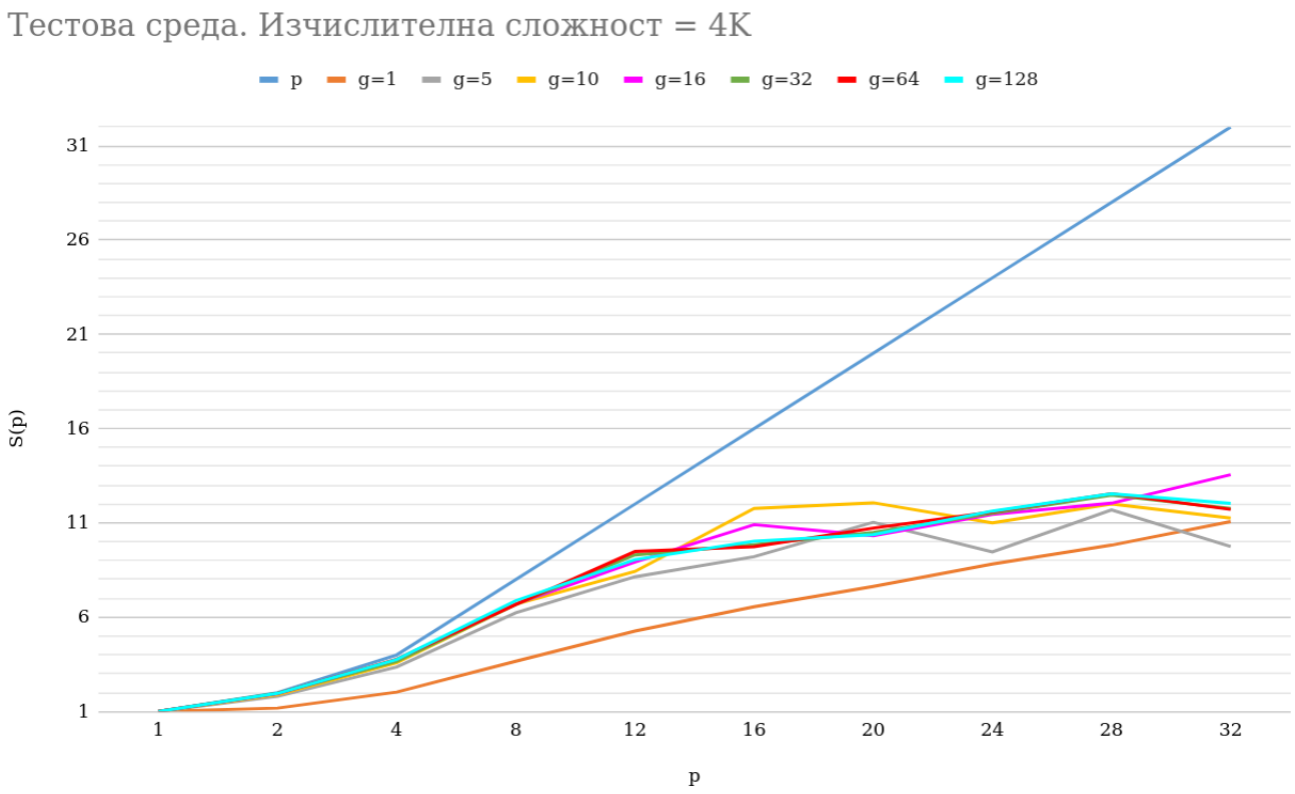

Рис. 14: Визуализация на постигнати ускорения на тестова среда с резолюция 4K. Попълнената таблица е налична в Appendix A.

Оказва се, че докато в развойна среда по-голямо ускорение се постига с по-малката резолюция, това не е така в тестовата среда. Ускорение има и се доближава до броя физически ядра. Отново най-добър резултат се постига със средна грануларност (при g=16).

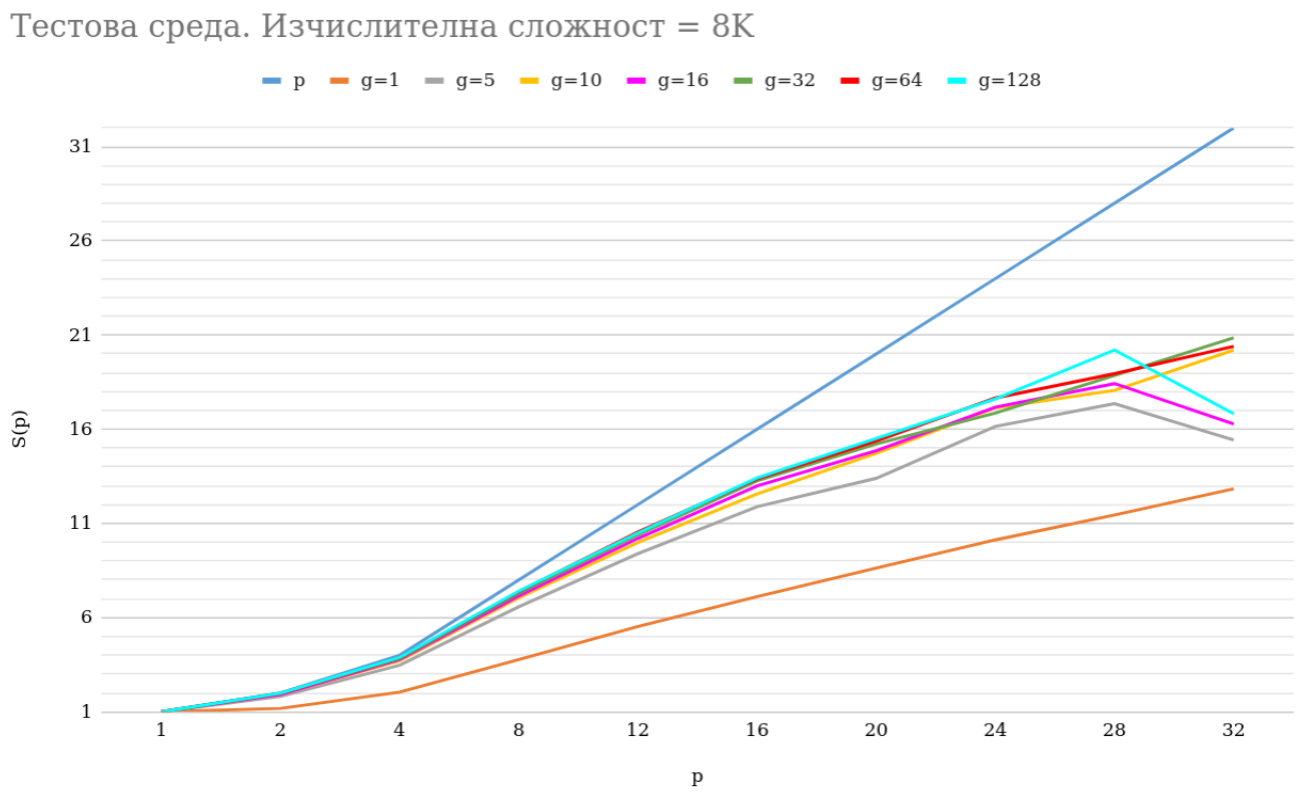

Рис. 15: Визуализация на постигнати ускорения на тестова среда с резолюция 8K. Попълнената таблица е налична в Appendix A.

При по-голяма изчислителна сложност се вижда, че ускорението е по-високо при всички стойности за грануларността.

Хистограма на разпределението при динамично балансиране

С илюстративна цел са направени сравнения на производителността на програмата при циклично статично балансиране и динамично централизирано балансиране.

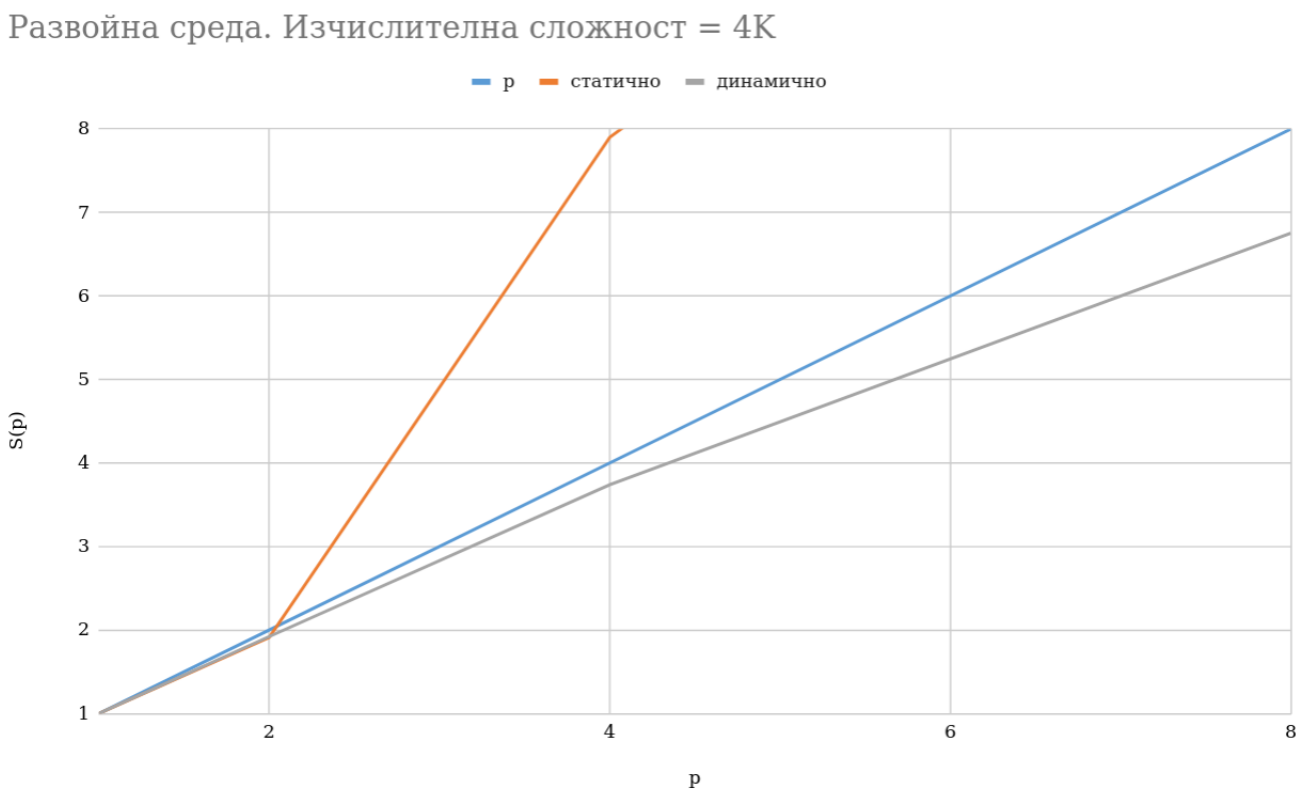

Рис. 16: Сравнение в ускорението, постигнато при двата вида балансиране.

От диаграмата следва, че динамичното разпределяне на задачите не помага за постигането на ускорение и въпреки, че такова се наблюдава, то няма потенциала на статичното.

Следват хистограми на общото време на работа на нишките при двата вида разпределения. Тестовете са проведени в тестова среда с изчислителна сложност 7680x4320 и грануларност 128. Попълнените таблици са налични в Appendix A.

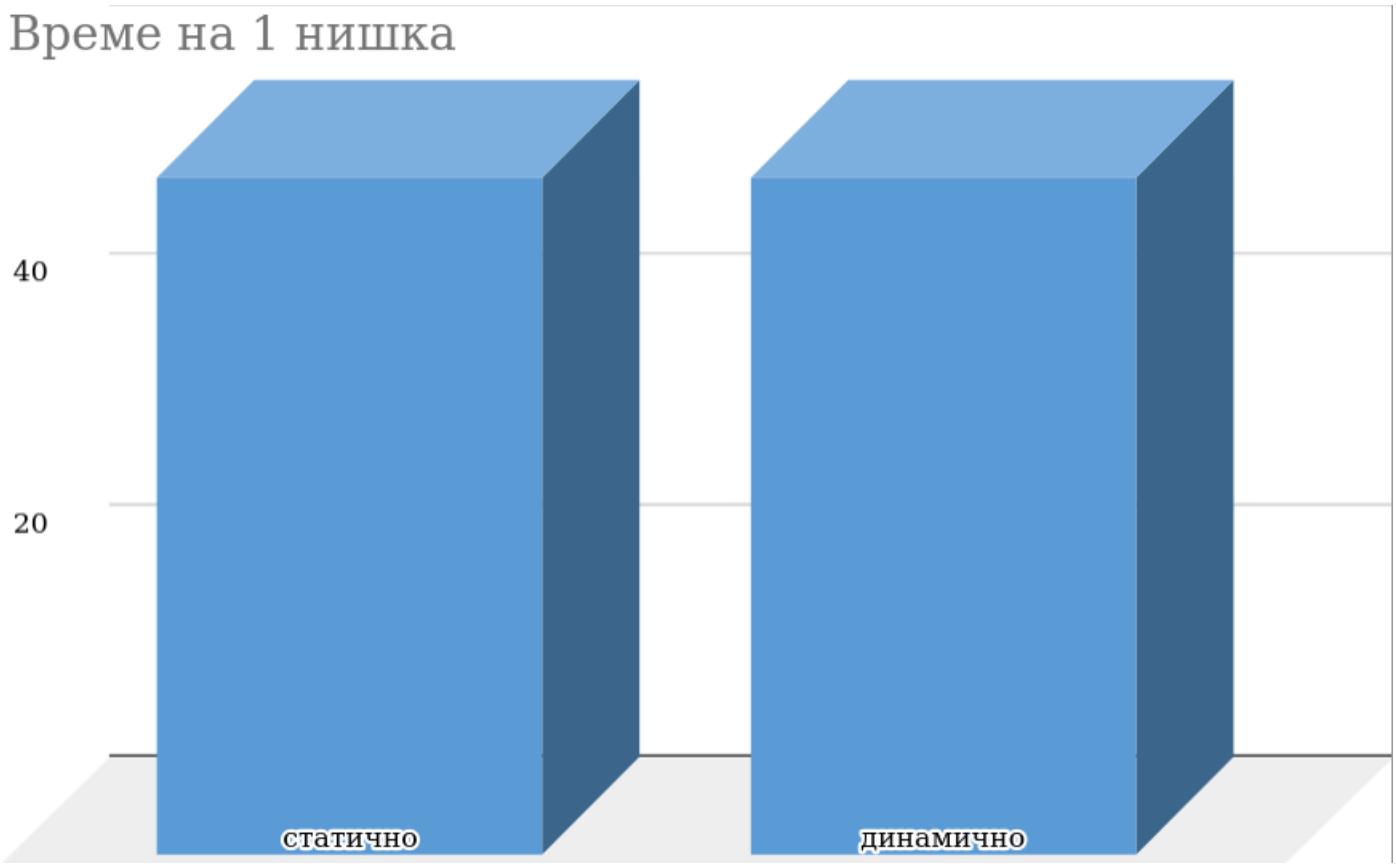

Рис. 17: Сравнение във времената на 1 нишка при двата вида балансиране. Попълнената таблица е налична в Appendix A.

При тези параметри не се наблюдава (съществена) разлика във времената.

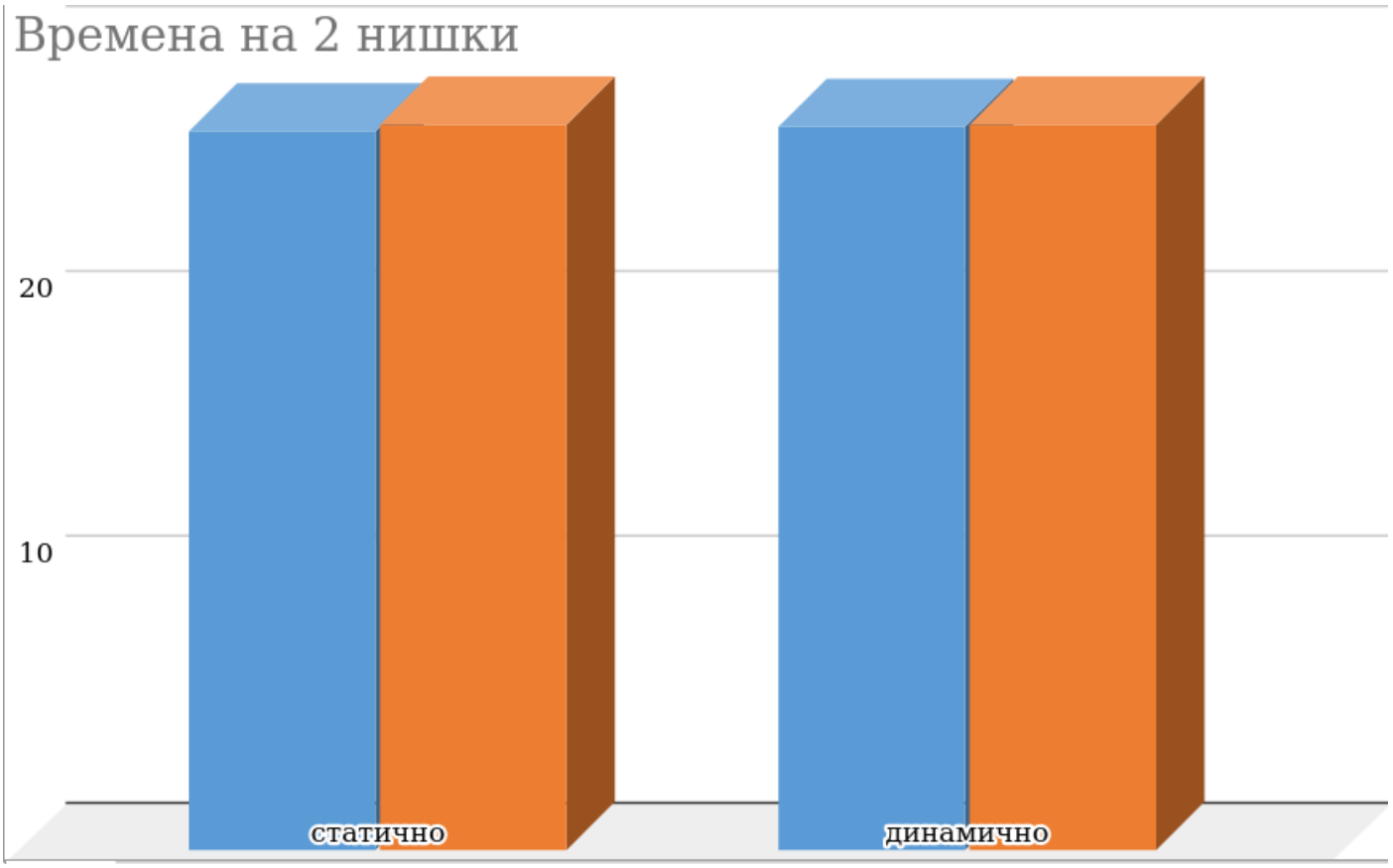

Рис. 18: Сравнение във времената на 2 нишки при двата вида балансиране. Попълнената таблица е налична в Appendix A.

Тази диаграма показва, в този случай динамичното балансиране се справя по-добре с разпределянето на задачите. Въпреки това съществено подобряване в ускорението не се наблюдава.

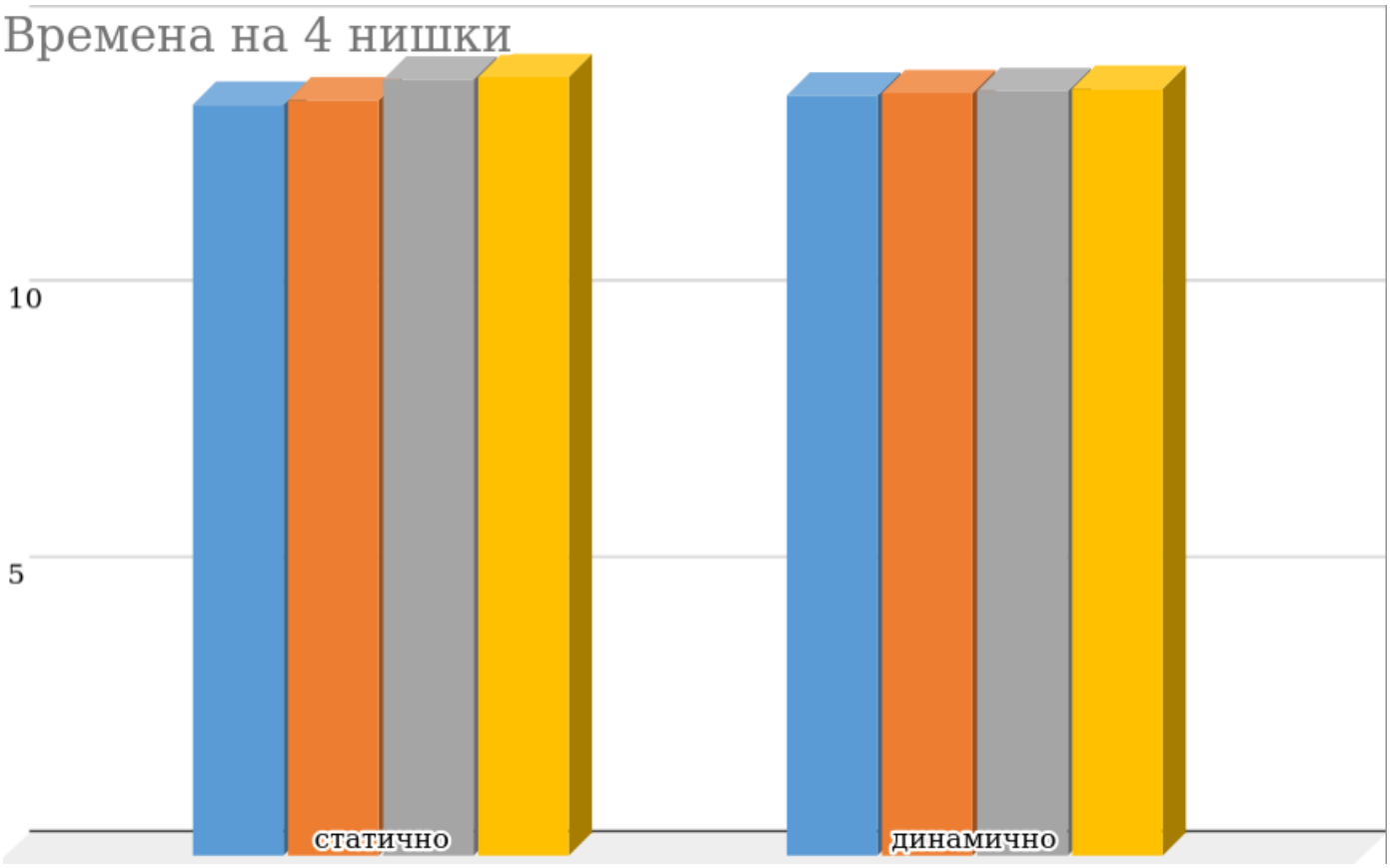

Рис. 19: Сравнение във времената на 4 нишки при двата вида балансиране. Попълнената таблица е налична в Appendix A.

Тенденцията, да се наблюдава по-равномерно разпределяне на работата между нишките при динамично централизирано балансиране, продължава да се наблюдава и тук, но отново - това не води до по-голямо ускорение.

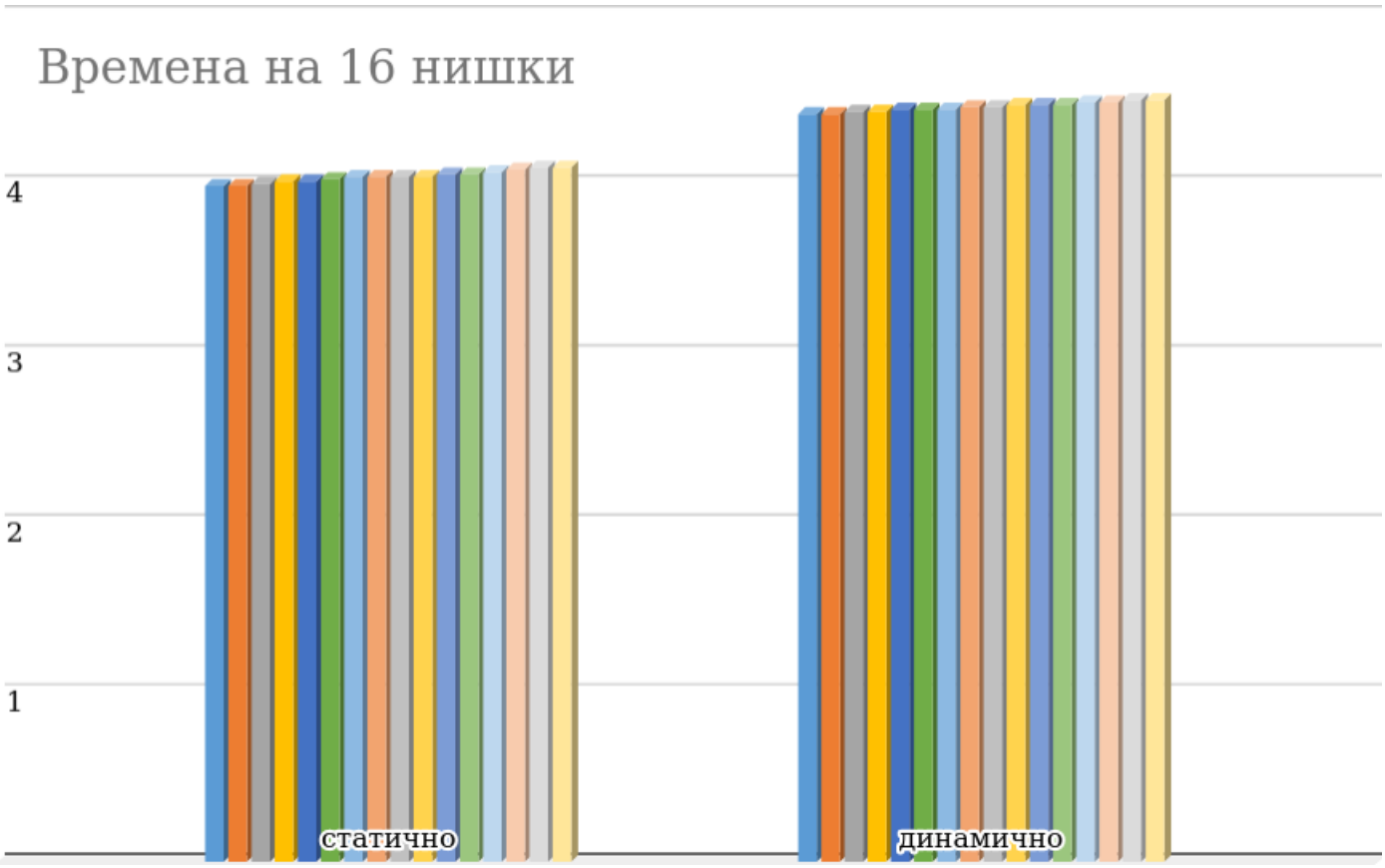

Рис. 20: Сравнение във времената на 16 нишки при двата вида балансиране. Попълнената таблица е налична в Appendix A.

Тази диаграма демонстрира най-добре каква е печалбата при прилагане на статично циклично разпределяне на работата между нишките. Динамичното разпределение отново поделя задачата по-равномерно, но с много малка разлика. По-важното нещо е, че ясно се вижда забавянето, предизвикано от комуникационния свръхтовар. Това е и причината за по-слабото ускорение.

От проведените тестове следва, че цикличното статично разпределяне на задачите дава по-добри резултати от динамичното централизирано. Макар и разлика да няма при по-ниско ниво на софтуерния паралелизъм, първият вид балансиране е

скалируемо, докато вторият не е.

Възможни подобрения и бъдеща работа

Възможно е да се изследва производителността при побитово кодиране на изследваните точки. Най-вероятно тя ще бъде по-голяма, поради по-оптималното използване на памет. Също е интересно сравнение между обработка по колони срещи обработка по редове.

Източници

1. V. Drakopoulos, N. Mimikou, T. Theoharis. An overview of parallel visualisation methods for Mandelbrot and Julia sets, Computers & Graphics (https://www.sciencedirect.com/science/article/pii/S0097849303001067)
2. Soto Gómez, Ernesto. (2020). MPI vs OpenMP: A case study on parallel generation of Mandelbrot set (https://www.researchgate.net/publication/344453347_MPI_vs_OpenMP_A_case_study_on_parallel_generation_of_Mandelbrot_set)
3. S. Ernsting, H. Kuchen. A Scalable Farm Skeleton for Hybrid Parallel and Distributed Programming (https://link.springer.com/article/10.1007/s10766-013-0269-2)
4. S. Tornboulian. Indirect Addressing and load balancing for faster solution to Mandelbrot set on SIMD architectures (https://ntrs.nasa.gov/citations/19890018019)
5. Isaac Gang. Parallel Implementation and Analysis of Mandelbrot Set Construction

(https://www.academia.edu/1399383/Parallel_Implemen
tation_and_Analysis_of_Mandelbrot_Set_Construction)

6. Seyed Mohamed Buhari. Parallel Processing and the Mandelbrot set
(https://ns.afahc.ro/ro/revista/Nr_2_2008/ART_CARMEN.pdf)

7. Mirco Tracolli. Parallel generation of a Mandelbrot set
(http://services.chm.unipg.it/ojs/index.php/virtlcomm/article/view/112)

Appendix A

РАЗВОЙНА СРЕДА								
ИЗЧИСЛИТЕЛНА СЛОЖНОСТ = 4K								
#	p	g	Tp (1)	Tp (2)	Tp (3)	Tp=min(Tp(i))	Sp = T1/Tp	Ep = Sp/p
1	1	1	**26.255233**	**26.251805**	**26.217473**	**26.217473**	1	1
2	2	1	22.197354	22.290007	22.280965	**22.197354**	1.181108028	0.5905540
3	4	1	12.835303	12.8182	12.848511	**12.8182**	2.045331872	0.5113329
4	8	1	7.003795	6.829559	6.838486	**6.829559**	3.838823707	0.4798529
5	1	5	**26.210371**	**26.238515**	**26.236747**	**26.210371**	1	1
6	2	5	14.45412	14.49166	14.497285	**14.45412**	1.813349481	0.9066747
7	4	5	7.466476	7.439455	7.458621	**7.439455**	3.523157409	0.8807893
8	8	5	4.115944	4.326625	4.058786	**4.058786**	6.457687348	0.8072109
9	1	10	**26.160981**	**26.223924**	**26.210176**	**26.160981**	1	1
10	2	10	13.724502	13.714902	13.690719	**13.690719**	1.910855157	0.9554275
11	4	10	3.311328	5.49787	7.056075	**3.311328**	7.90044991	1.9751124
12	8	10	4.004712	3.867126	1.879278	**1.879278**	13.92076159	1.7400951
13	1	16	**14.341351**	**15.531005**	**14.341351**	**14.341351**	1	1
14	2	16	6.267872	6.294911	8.093084	**6.267872**	2.288073368	1.1440366
15	4	16	3.224086	3.23625	5.769563	**3.224086**	4.448191208	1.1120478
16	8	16	3.803564	3.836905	1.739141	**1.739141**	8.246226729	1.0307783
17	1	32	**12.32635**	**12.32635**	**12.341735**	**12.32635**	1	1
18	2	32	9.728714	9.583941	9.485597	**9.485597**	1.299480676	0.6497403
19	4	32	3.164331	3.181488	6.480901	**3.164331**	3.895404747	0.9738511
20	8	32	4.264192	3.916548	1.871153	**1.871153**	6.587569269	0.8234461
21	1	64	**12.33051**	**15.565613**	**14.300731**	**12.33051**	1	1
22	2	64	8.514817	8.984165	9.213182	**8.514817**	1.4481239	0.7240619
23	4	64	3.142868	3.136313	6.278959	**3.136313**	3.931530431	0.9828826
24	8	64	1.78823	1.718209	2.359863	**1.718209**	7.176373771	0.8970467
25	1	128	**15.870022**	**14.925463**	**12.34729**	**12.34729**	1	1
26	2	128	6.217358	8.019404	8.760869	**6.217358**	1.9859384	0.9929692
27	4	128	3.123053	5.560621	3.117582	**3.117582**	3.960534158	0.9901335
28	8	128	1.720489	2.253696	1.837343	**1.720489**	7.176616648	0.8970770

Рис. 21: Тестов план на развойна среда с резолюция 4K.

РАЗВОЙНА СРЕДА								
ИЗЧИСЛИТЕЛНА СЛОЖНОСТ = 8K								
#	*p*	*g*	*Tp* (1)	*Tp* (2)	*Tp* (3)	*Tp*=min(*Tp* (*i*))	*Sp* = *T1*/*Tp*	*Ep* = *Sp*/*p*
1	1	1	***49.262487***	***51.725615***	***52.3461***	**49.262487**	1	1
2	2	1	*45.201702*	*45.368668*	*45.479047*	**45.201702**	1.089836993	0.54491849
3	4	1	*25.879059*	*27.739812*	*24.698983*	**24.698983**	1.994514794	0.49862869
4	8	1	*15.590183*	*16.333241*	*15.460871*	**15.460871**	3.186268549	0.39828356
5	1	5	***49.249588***	***52.864001***	***50.074742***	**49.249588**	1	1
6	2	5	*34.165492*	*27.930917*	*32.261669*	**27.930917**	1.763264271	0.88163213
7	4	5	*15.881365*	*17.310571*	*16.264918*	**15.881365**	3.101092885	0.77527322
8	8	5	*9.670051*	*10.22024*	*9.742834*	**9.670051**	5.093001888	0.63662523
9	1	10	***53.115313***	***49.248238***	***51.484606***	**49.248238**	1	1
10	2	10	*30.130765*	*28.683346*	*25.67215*	**25.67215**	1.91835269	0.95917634
11	4	10	*14.802417*	*15.352368*	*16.820427*	**14.802417**	3.327040307	0.83176007
12	8	10	*11.076009*	*12.267661*	*12.130636*	**11.076009**	4.446388406	0.55579855
13	1	16	***53.117739***	***51.104703***	***52.034827***	**51.104703**	1	1
14	2	16	*28.835496*	*25.563394*	*29.880835*	**25.563394**	1.999136069	0.99956803
15	4	16	*15.196366*	*17.144094*	*15.240741*	**15.196366**	3.362955525	0.84073888
16	8	16	*9.268594*	*9.860464*	*11.119065*	**9.268594**	5.513749227	0.68921865
17	1	32	***53.095936***	***53.100917***	***49.209062***	**49.209062**	1	1
18	2	32	*26.602013*	*25.492146*	*27.241579*	**25.492146**	1.930361689	0.96518084
19	4	32	*18.284406*	*16.750189*	*16.623494*	**16.623494**	2.960211734	0.74005293
20	8	32	*9.698545*	*12.294066*	*10.375861*	**9.698545**	5.073860254	0.63423253
21	1	64	***52.669623***	***52.8161***	***52.881246***	**52.669623**	1	1
22	2	64	*27.263448*	*25.068111*	*24.850812*	**24.850812**	2.119432677	1.05971633
23	4	64	*16.190841*	*15.089516*	*16.121497*	**15.089516**	3.490477958	0.87261948
24	8	64	*9.907999*	*9.49845*	*10.131853*	**9.49845**	5.545075565	0.69313444
25	1	128	***50.800255***	***51.885863***	***52.372545***	**50.800255**	1	1
26	2	128	*25.824063*	*24.727264*	*28.073336*	**24.727264**	2.054422802	1.02721140
27	4	128	*16.546984*	*17.832737*	*16.514097*	**16.514097**	3.076175161	0.76904379
28	8	128	*10.161666*	*10.333807*	*11.437914*	**10.161666**	4.999205347	0.62490066

Рис. 22: Тестов план на развойна среда с резолюция 8K.

ТЕСТОВА СРЕДА								
ИЗЧИСЛИТЕЛНА СЛОЖНОСТ = 4K								
#	*p*	*g*	*Tp* (1)	*Tp* (2)	*Tp* (3)	*Tp*=min(*Tp*(*i*))	*Sp* = *T1*/*Tp*	*Ep* = *Sp*/*p*
1	1	1	***13.338407***	***13.364929***	***13.3229***	**13.3229**	1	1
2	2	1	*11.307508*	*11.320749*	*11.293476*	**11.293476**	1.179698792	0.589849396
3	4	1	*6.634933*	*6.656703*	*6.561947*	**6.561947**	2.030327279	0.507581819
4	8	1	*3.63533*	*3.7132*	*3.735149*	**3.63533**	3.664839231	0.458104903
5	12	1	*2.561051*	*2.59337*	*2.530371*	**2.530371**	5.265196289	0.438766357
6	16	1	*2.032036*	*2.043477*	*2.059319*	**2.032036**	6.556429118	0.409776819
7	20	1	*1.764209*	*1.745502*	*1.756843*	**1.745502**	7.632703944	0.381635197
8	24	1	*1.538355*	*1.520514*	*1.510664*	**1.510664**	8.819234456	0.367468102
9	28	1	*1.356182*	*1.364438*	*1.371688*	**1.356182**	9.823828955	0.350851034
10	32	1	*1.248404*	*1.213308*	*1.203063*	**1.203063**	11.0741499	0.346067184
11	1	5	***13.326774***	***13.327034***	***13.327876***	**13.326774**	1	1
12	2	5	*7.438309*	*7.371524*	*7.418878*	**7.371524**	1.807872293	0.903936146
13	4	5	*3.969245*	*3.961189*	*4.023849*	**3.961189**	3.364336819	0.841084204
14	8	5	*2.203544*	*2.137952*	*2.204771*	**2.137952**	6.233429937	0.779178742
15	12	5	*1.681481*	*1.693748*	*1.636754*	**1.636754**	8.1421973	0.678516441
16	16	5	*1.499242*	*1.72727*	*1.446861*	**1.446861**	9.210818455	0.575676153
17	20	5	*1.605533*	*1.608249*	*1.207835*	**1.207835**	11.03360476	0.551680237
18	24	5	*1.440147*	*1.494215*	*1.407843*	**1.407843**	9.466093876	0.394420578
19	28	5	*1.139571*	*1.356214*	*1.341455*	**1.139571**	11.69455348	0.417662624
20	32	5	*1.380432*	*1.367523*	*1.457219*	**1.367523**	9.745191854	0.304537245

Рис. 23: Тестов план на тестова среда с резолюция 4K (част 1).

21	1	10	***13.321128***	***13.28499***	***13.291854***	**13.28499**	1	1
22	2	10	*7.118148*	*7.071457*	*7.043577*	**7.043577**	1.886114115	0.943057057
23	4	10	*3.724795*	*3.701684*	*3.733253*	**3.701684**	3.588904401	0.897226100
24	8	10	*2.079537*	*1.98528*	*2.124415*	**1.98528**	6.691746252	0.836468281
25	12	10	*1.653478*	*1.577083*	*1.61333*	**1.577083**	8.423773511	0.701981125
26	16	10	*1.129231*	*1.622846*	*1.580395*	**1.129231**	11.76463452	0.735289657
27	20	10	*1.100828*	*1.307779*	*1.661326*	**1.100828**	12.06817959	0.603408979
28	24	10	*1.382622*	*1.207144*	*1.369753*	**1.207144**	11.00530674	0.458554447
29	28	10	*1.106166*	*1.123939*	*1.109718*	**1.106166**	12.00994245	0.428926516
30	32	10	*1.179911*	*1.343034*	*1.475505*	**1.179911**	11.25931532	0.351853603
31	1	16	***13.28922***	***13.287338***	***13.311798***	**13.287338**	1	1
32	2	16	*6.843641*	*6.944877*	*6.861961*	**6.843641**	1.941559763	0.970779881
33	4	16	*3.713807*	*3.58407*	*3.576274*	**3.576274**	3.715413864	0.928853465
34	8	16	*2.042157*	*1.983751*	*2.034623*	**1.983751**	6.698087613	0.837260951
35	12	16	*1.756082*	*1.488504*	*1.580776*	**1.488504**	8.926639095	0.743886591
36	16	16	*1.21818*	*1.568006*	*1.719145*	**1.21818**	10.90753255	0.681720784
37	20	16	*1.302554*	*1.287286*	*1.288761*	**1.287286**	10.32197818	0.516098089
38	24	16	*1.160315*	*1.432814*	*1.379674*	**1.160315**	11.45149205	0.477145502
39	28	16	*1.102863*	*1.111537*	*1.305723*	**1.102863**	12.04804042	0.430287157
40	32	16	*1.336398*	*1.369481*	*0.980264*	**0.980264**	13.55485665	0.423589270
41	1	32	***13.295008***	***13.290405***	***13.335162***	**13.290405**	1	1
42	2	32	*6.753919*	*6.777277*	*6.772487*	**6.753919**	1.967806395	0.983903197
43	4	32	*3.667225*	*3.652024*	*3.635716*	**3.635716**	3.655512422	0.913878105
44	8	32	*2.03773*	*2.054764*	*1.990046*	**1.990046**	6.678441101	0.834805137
45	12	32	*1.569685*	*1.428321*	*1.467349*	**1.428321**	9.304914652	0.775409554
46	16	32	*1.364894*	*1.34933*	*1.392336*	**1.34933**	9.849632781	0.615602048
47	20	32	*1.266569*	*1.275665*	*1.288641*	**1.266569**	10.49323408	0.524661704
48	24	32	*1.155104*	*1.262511*	*1.177517*	**1.155104**	11.50580814	0.479408672
49	28	32	*1.078703*	*1.099927*	*1.066021*	**1.066021**	12.4673013	0.445260760
50	32	32	*1.138396*	*1.146009*	*1.129711*	**1.129711**	11.76442913	0.367638410

Рис. 24: Тестов план на тестова среда с резолюция 4K (част 2).

51	1	64	***13.446752***	***13.284825***	***13.327064***	**13.284825**	1	1
52	2	64	*6.911741*	*6.990955*	*6.788476*	**6.788476**	1.956967219	0.978483609
53	4	64	*3.578451*	*3.562673*	*3.522196*	**3.522196**	3.771744957	0.942936239
54	8	64	*1.98955*	*2.002941*	*1.996762*	**1.98955**	6.6773014	0.834662675
55	12	64	*1.541657*	*1.504147*	*1.400731*	**1.400731**	9.484208603	0.790350716
56	16	64	*1.363776*	*1.472969*	*1.377199*	**1.363776**	9.7412075	0.608825468
57	20	64	*1.238519*	*1.282925*	*1.317421*	**1.238519**	10.72637965	0.536318982
58	24	64	*1.148522*	*1.146831*	*1.151498*	**1.146831**	11.58394306	0.482664294
59	28	64	*1.058074*	*1.079857*	*1.078009*	**1.058074**	12.55566718	0.448416685
60	32	64	*1.144922*	*1.134034*	*1.146402*	**1.134034**	11.71466199	0.366083187
61	1	128	***13.341409***	***13.323069***	***13.289707***	**13.289707**	1	1
62	2	128	*6.82382*	*6.783972*	*6.771955*	**6.771955**	1.962462391	0.981231195
63	4	128	*3.538974*	*3.543348*	*3.555255*	**3.538974**	3.755242904	0.938810725
64	8	128	*1.940534*	*1.957635*	*1.931208*	**1.931208**	6.88155134	0.860193917
65	12	128	*1.595198*	*1.467805*	*1.562064*	**1.467805**	9.054136619	0.754511384
66	16	128	*1.35462*	*1.324687*	*1.545989*	**1.324687**	10.03233745	0.627021090
67	20	128	*1.280043*	*1.300697*	*1.352213*	**1.280043**	10.38223482	0.519111740
68	24	128	*1.142836*	*1.17738*	*1.17738*	**1.142836**	11.62870876	0.484529531
69	28	128	*1.064709*	*1.068132*	*1.05899*	**1.05899**	12.5494169	0.448193460
70	32	128	*1.144279*	*1.133039*	*1.103733*	**1.103733**	12.04069009	0.376271565

Рис. 25: Тестов план на тестова среда с резолюция 4K (част 3).

ТЕСТОВА СРЕДА								
ИЗЧИСЛИТЕЛНА СЛОЖНОСТ = 8K								
#	*p*	*g*	*Tp* (1)	*Tp* (2)	*Tp* (3)	*Tp*=min(*Tp* (*i*))	*Sp* = *T1*/*Tp*	*Ep* = *Sp*/*p*
1	1	1	***52.802102***	***52.790214***	***52.797578***	**52.790214**	1	1
2	2	1	*44.746385*	*44.769166*	*44.699467*	**44.699467**	1.18100321	0.5905016
3	4	1	*26.57966*	*25.874524*	*26.143212*	**25.874524**	2.04023904	0.5100597
4	8	1	*13.999328*	*14.220636*	*13.978223*	**13.978223**	3.776604079	0.4720755
5	12	1	*9.540239*	*9.699838*	*9.614072*	**9.540239**	5.533426783	0.4611188
6	16	1	*7.54228*	*7.459391*	*7.417777*	**7.417777**	7.116716235	0.4447947
7	20	1	*6.118229*	*6.185929*	*6.141693*	**6.118229**	8.628348824	0.4314174
8	24	1	*5.260508*	*5.258551*	*5.212949*	**5.212949**	10.12674668	0.4219477
9	28	1	*4.636593*	*4.621063*	*4.610662*	**4.610662**	11.44959531	0.4089141
10	32	1	*4.213171*	*4.11352*	*4.169305*	**4.11352**	12.83334322	0.4010419
11	1	5	***52.775456***	***52.775979***	***53.335427***	**52.775456**	1	1
12	2	5	*29.195517*	*29.281372*	*29.099788*	**29.099788**	1.813602766	0.9068013
13	4	5	*15.275044*	*15.238569*	*15.192809*	**15.192809**	3.47371286	0.8684282
14	8	5	*8.096069*	*8.027137*	*8.203921*	**8.027137**	6.574630033	0.8218287
15	12	5	*5.617786*	*5.6388*	*5.620975*	**5.617786**	9.39435144	0.7828626
16	16	5	*4.4375*	*4.63335*	*4.513068*	**4.4375**	11.89306051	0.7433162
17	20	5	*3.939558*	*3.963858*	*4.039724*	**3.939558**	13.39628862	0.6698144
18	24	5	*3.497277*	*3.268296*	*3.45918*	**3.268296**	16.14769776	0.6728207
19	28	5	*3.040634*	*3.068955*	*3.042267*	**3.040634**	17.35672758	0.6198831
20	32	5	*3.419376*	*3.459683*	*3.475418*	**3.419376**	15.43423595	0.4823198

Рис. 26: Тестов план на тестова среда с резолюция 8K (част 1).

21	1	10	***52.77999***	***52.848873***	***52.791168***	**52.77999**	1	1
22	2	10	*27.552468*	*27.617221*	*27.634294*	**27.552468**	1.915617505	0.9578087
23	4	10	*14.406215*	*14.473698*	*14.19738*	**14.19738**	3.717586625	0.9293966
24	8	10	*7.4857*	*7.654236*	*7.536796*	**7.4857**	7.050775479	0.8813469
25	12	10	*5.336694*	*5.403618*	*5.281944*	**5.281944**	9.992531159	0.8327109
26	16	10	*4.323393*	*4.341449*	*4.199327*	**4.199327**	12.56867827	0.7855423
27	20	10	*3.73317*	*3.586776*	*3.800085*	**3.586776**	14.71516203	0.7357581
28	24	10	*3.280588*	*3.31475*	*3.080201*	**3.080201**	17.13524215	0.7139684
29	28	10	*2.922514*	*2.993395*	*3.179843*	**2.922514**	18.0597903	0.6449925
30	32	10	*2.611873*	*2.669462*	*2.67786*	**2.611873**	20.20771684	0.6314911
31	1	16	***52.817256***	***52.827167***	***52.776591***	**52.776591**	1	1
32	2	16	*27.692889*	*27.507512*	*27.429298*	**27.429298**	1.924095578	0.9620477
33	4	16	*14.023162*	*14.001467*	*14.058325*	**14.001467**	3.769361525	0.9423403
34	8	16	*7.377922*	*7.411621*	*7.421084*	**7.377922**	7.153313765	1.7883284
35	12	16	*5.341743*	*5.32669*	*5.17123*	**5.17123**	10.20581003	1.2757262
36	16	16	*4.081433*	*4.063376*	*4.126418*	**4.063376**	12.98836017	1.0823633
37	20	16	*3.552956*	*3.551058*	*3.598566*	**3.551058**	14.86221599	0.9288884
38	24	16	*3.308223*	*3.073111*	*3.072947*	**3.072947**	17.1745855	0.8587292
39	28	16	*2.89112*	*2.89694*	*2.864037*	**2.864037**	18.42734259	0.7678059
40	32	16	*3.282233*	*3.241414*	*3.273336*	**3.241414**	16.28196552	0.5814987
41	1	32	***52.772989***	***52.806754***	***52.808916***	**52.772989**	1	1
42	2	32	*26.671582*	*26.535315*	*26.527519*	**26.527519**	1.989367683	0.9946838
43	4	32	*13.890533*	*13.86052*	*13.842745*	**13.842745**	3.812321111	0.9530802
44	8	32	*7.318371*	*7.239657*	*7.328191*	**7.239657**	7.28943222	0.9111790
45	12	32	*5.075224*	*5.204144*	*5.1758*	**5.075224**	10.39815957	0.8665132
46	16	32	*3.980293*	*4.016774*	*4.192371*	**3.980293**	13.2585689	0.8286605
47	20	32	*3.511554*	*3.596419*	*3.468933*	**3.468933**	15.21303208	0.7606516
48	24	32	*3.209739*	*3.130146*	*3.161182*	**3.130146**	16.85959345	0.7024830
49	28	32	*2.807203*	*2.799002*	*3.081872*	**2.799002**	18.85421625	0.6733648
50	32	32	*3.196472*	*2.530046*	*3.196118*	**2.530046**	20.85850969	0.6518284

Рис. 27: Тестов план на тестова среда с резолюция 8K (част 2).

51	1	64	***52.83528***	***52.838378***	***52.795484***	**52.795484**	1	1
52	2	64	*26.770218*	*26.713235*	*26.923746*	**26.713235**	1.976379274	0.9881896
53	4	64	*13.850986*	*13.706229*	*13.718467*	**13.706229**	3.851933599	0.9629833
54	8	64	*7.248433*	*7.157033*	*7.263078*	**7.157033**	7.376727759	0.9220909
55	12	64	*5.007939*	*5.445718*	*5.091957*	**5.007939**	10.54235764	0.8785298
56	16	64	*3.952081*	*4.030486*	*4.193754*	**3.952081**	13.35890737	0.8349317
57	20	64	*3.621876*	*3.601401*	*3.435396*	**3.435396**	15.36809264	0.7684046
58	24	64	*3.366665*	*2.98953*	*3.153214*	**2.98953**	17.66012852	0.7358386
59	28	64	*2.785976*	*2.793827*	*2.826747*	**2.785976**	18.95044466	0.6768015
60	32	64	*3.196396*	*3.170235*	*2.587607*	**2.587607**	20.40320806	0.6376002
61	1	128	***52.817614***	***52.803734***	***52.836412***	**52.803734**	1	1
62	2	128	*26.936336*	*26.595096*	*26.582646*	**26.582646**	1.98639872	0.9931993
63	4	128	*13.749842*	*13.658171*	*13.599753*	**13.599753**	3.882698017	0.9706745
64	8	128	*7.126325*	*7.158693*	*7.272218*	**7.126325**	7.409672447	0.9262090
65	12	128	*5.043288*	*5.057657*	*5.031748*	**5.031748**	10.49411338	0.8745094
66	16	128	*3.962183*	*3.936363*	*3.952401*	**3.936363**	13.41434568	0.8383966
67	20	128	*3.404063*	*3.465032*	*3.427337*	**3.404063**	15.51197319	0.7755986
68	24	128	*2.997856*	*3.151826*	*3.121247*	**2.997856**	17.61383269	0.7339096
69	28	128	*2.824311*	*2.612981*	*2.842232*	**2.612981**	20.20823496	0.7217226
70	32	128	*3.161304*	*3.168471*	*3.138732*	**3.138732**	16.8232694	0.5257271

Рис. 28: Тестов план на тестова среда с резолюция 8K (част 3).

p	*g*	*min Tp ЦСБ*	*min Tp ДЦБ*	*Sp ЦСБ*	*Sp ДЦБ*
1	10	**26.160981**	**12.393183**	**1**	**1**
2	10	**13.690719**	**6.448693**	**1.910855157**	**1.921813149**
4	10	**3.311328**	**3.312945**	**7.90044991**	**3.740835722**
8	10	**1.879278**	**1.835308**	**13.92076159**	**6.752644788**

Рис. 29: Тестов план на развойна среда с резолюция 4K, който сравнява двата вида балансиране.

статично	динамично
53.944317	53.84033

Рис. 30: Тестов план на разпределението на времената на 1 нишка при двата вида балансиране.

статично	динамично
27.065041	27.250004
27.325905	27.275795

Рис. 31: Тестов план на разпределението на времената на 2 нишки при двата вида балансиране.

статично	динамично
13.588619	13.757229
13.707624	13.811591
14.073118	13.851448
14.13132	13.895618

Рис. 32: Тестов план на разпределението на времената на 4 нишки при двата вида балансиране.

статично	динамично
3.971485	4.401983
3.977081	4.405572
3.992625	4.417886
3.997071	4.41495
4.001422	4.421378
4.015927	4.420946
4.023532	4.430224
4.030083	4.437005
4.035003	4.441619
4.030837	4.450523
4.040755	4.453887
4.041404	4.458948
4.053338	4.464576
4.069089	4.471565
4.084469	4.477625
4.087859	4.481979

Рис. 33: Тестов план на разпределението на времената на 16 нишки при двата вида балансиране.