Multi-Tor

Многонишкова реализация на симулацията Wa-Tor

Автор: Александър Ангелков, фн. 71995

Преподавател: проф. Васил Цунижев

Съдържание:

1. Увод……2

1.1 Правила……2

1.2 Програмна реализация……3

2. Анализ……4

2.1 Технологични спецификации……4

3. Проектиране……5

3.1 Произволни стойности……7

3.2 Балансиране и фази……7

3.3 Обработка на елементите……9

4. Тестване……11

5. Източници……13

1. Увод

Wa-Tor е динамична симулация за популациите на два животински вида – хищник и плячка. Моделът е представен за първи път през декември 1984 година в списанието Scientific American от канадският математик Александър Дюдни. Статията е озаглавена „Компютърни развлечения: Акулите и рибите водят екологична война на тороидалната планета Wa-Tor“ [1] и е дълга пет страници. Описаният в нея свят се подчинява на следните прости

1.1 Правила:

- Акулите и рибите имат време за размножаване и време за умиране от глад.

- Акулите се хранят, изяждайки произволна риба около тях, а ако такава няма се движат в произволна незаета от други акули посока.

- Рибите се хранят движейки се в произволна, незаета от други риби или акули посока.

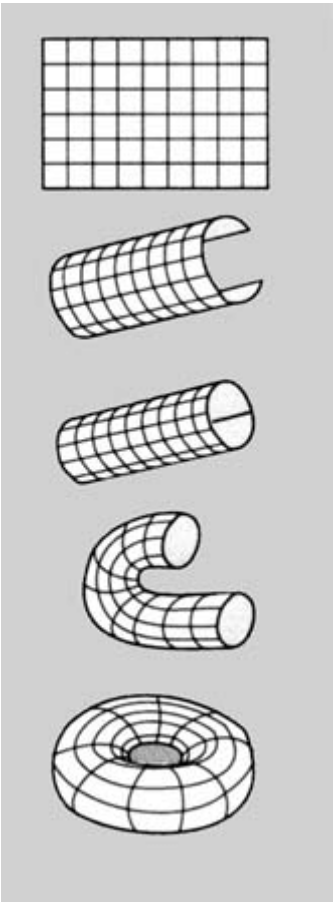
- След хранене, времето за умиране от глад се рестартира, а ако то изтече – съответният индивид умира и изчезва от картата

- След изтичане на времето за размножаване индивида оставя след себе си нов представител на вида при движение или хранене и времето му за размножаване се рестартира.

1.2 Програмна реализация:

За да реализираме програмно симулацията, тороидният океан представяме като матрица с n реда и m колони, като на всеки елемент от матрицата може да се намира акула, риба или празно пространство (вода). Животните се движат и хранят само нагоре, надолу, наляво и надясно, като при движение надясно от елемент с координати (i,m-1) за i=0..n-1 животното се премества на елемент (i,0). Обратното важи за движение наляво. При движение надолу от елемент с координати (n-1,j) за j=0..m-1 животното се премества на елемент (0,j). Обратното важи за движение нагоре (фиг. 1)

Матрицата се обработва паралелно ред по ред от няколко процеса и на три фази, като по този начин се избягват възможни конфликти с едновременната обработка на споделена памет.



Фиг. 1 Трансформация на правоъгълна матрица до тороид

2. Анализ

В програмната реализация наблюдаваме декомпозиция по данни по модела SPMD, тъй като матрицата се разделя по редове на подматрици, които се обработват паралено от даден брой нишки. Също така забелязваме декомпозиция по управление по модела Master-Slaves, защото главната нишка, стартираща останалите нишки има задачата да следи изпълнението на различните фази и да пуска нови – обработката е синхронна. Балансирането е статично и циклично с максимален паралелизъм (брой редове)/3. Има възможност за едра, средна и фина грануларност а обмена се осъществява чрез общи променливи. Софтуерът е написан на програмният език Java, в програмната среда Intellij IDEA 2021.3.2, компилиран с jdk-17.0.2 и използва библиотеката javafx-sdk-19.0.2.1 за визуализация.

2.1 Технологични спецификации на тестовата машина:

Тип	Настолен компютър
Процесор	Intel core i7-2600
Честота	3.4 GHz
Брой ядра	4
Cache L1	256 KB
Cache L2	1 MB

3. Проектиране

Работата на главната нишка е да инициализира масиви от произволни стойности, да стартира и следи работата на останалите нишки в три фази и да засича времето за изпълнение. Всички тези неща зависят единствено от няколко статични константи които потребителят може да променя преди стартирането на програмата. Те са:

ROW_COUNT, COLUMN_COUNT – брой редове и колони на матрицата

GRANULARITY – сумата на броя редове които се обработват във фази 1,2,3 от една нишка на една стъпка в една итерация

THREAD_COUNT – брой нишки които главната нишка стартира

STARTING_FISH_COUNT, STARTING_SHARK_COUNT – начален брой риби и акули

ITERATIONS_COUNT – брой итерации

EXPORT_TO_PNG – дали картата след всяка итерация да се записва във файл

TILE_SIZE – размер на един елемент от матрицата в пиксели

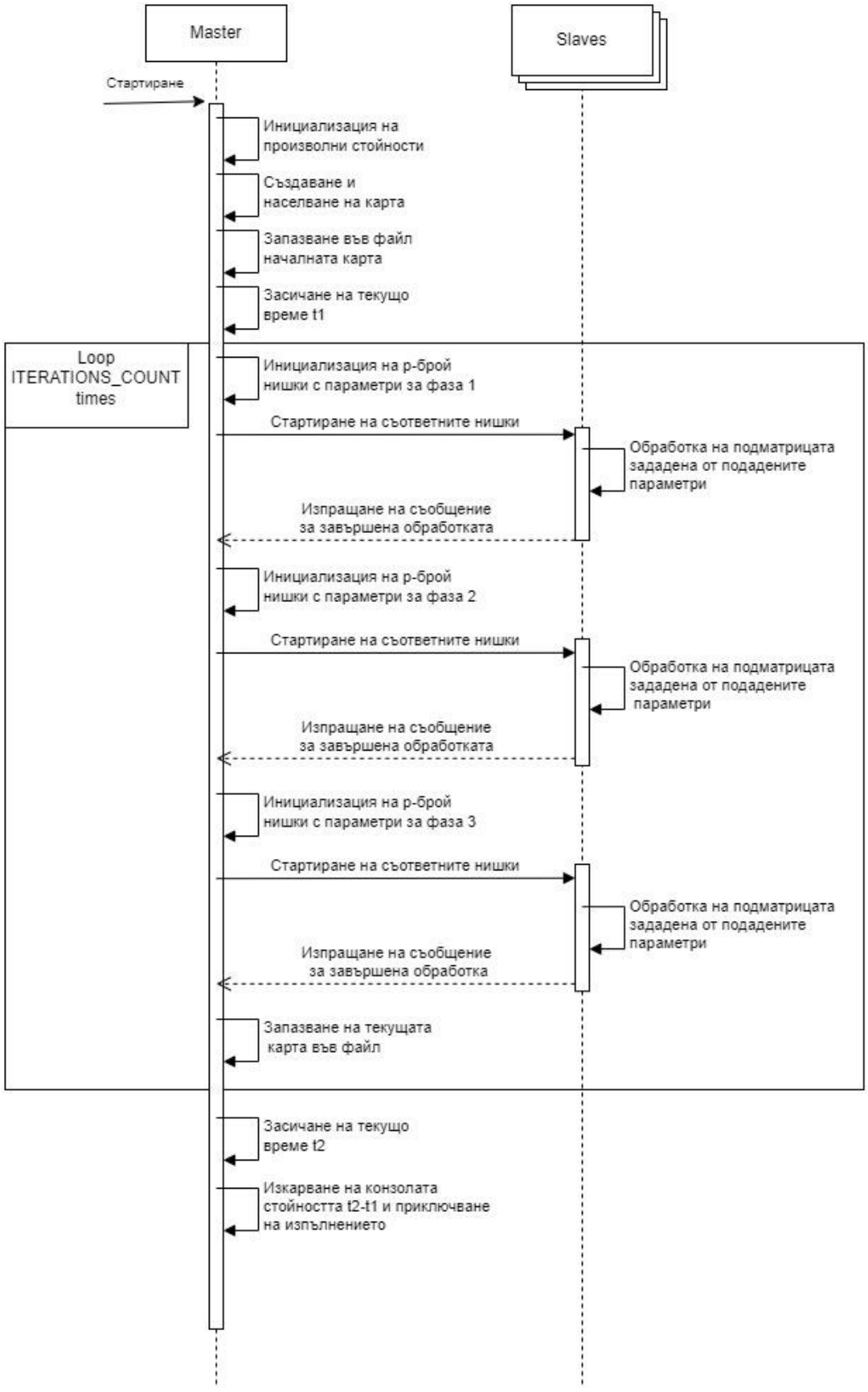
SHARK_TIME_TO_REPRODUCE, FISH_TIME_TO_REPRODUCE – време за размножаване

SHARK_TIME_TO_LIVE, FISH_TIME_TO_LIVE – време за умиране от глад

DEVIATION – най-голямо възможно отклонение от времето за размножаване и умиране от глад

SHARK_COLOR, FISH_COLOR, SEA_COLOR – цвят на акули, риби и океан

Диаграма на последователностите описваща работата на всяка нишка. Главната нишка стартира поредната фаза чак след като предходната е приключила.


Master
Slaves
Стартиране
Инициализация на произволни стойности
Създаване и населване на карта
Запазване във файл началната карта
Засичане на текущо време t1
Loop ITERATIONS_COUNT times
Инициализация на р-брой нишки с параметри за фаза 1
Стартиране на съответните нишки
Обработка на подматрицата зададена от подадените параметри
Изпращане на съобщение за завършена обработката
Инициализация на р-брой нишки с параметри за фаза 2
Стартиране на съответните нишки
Обработка на подматрицата зададена от подадените параметри
Изпращане на съобщение за завършена обработката
Инициализация на р-брой нишки с параметри за фаза 3
Стартиране на съответните нишки
Обработка на подматрицата зададена от подадените параметри
Изпращане на съобщение за завършена обработка
Запазване на текущата карта във файл
Засичане на текущо време t2
Изкарване на конзолата стойността t2-t1 и приключване на изпълнението

3.1 Произволни стойности

Първата работа на главната нишка е да инициализира седем масива с дължина (брой редове)*(брой колони) от произволни стойности, които нишките след това да използват. Това са масивите fishLive, fishReproduce, sharkLive, sharkReproduce които приемат стойности в съответните интервали: [FISH_TIME_TO_LIVE-DEVIATION, FISH_TIME_TO_LIVE+DEVIATION], [FISH_TIME_TO_REPRODUCE-DEVIATION, FISH_TIME_TO_REPRODUCE+DEVIATION], [SHARK_TIME_TO_LIVE-DEVIATION, SHARK_TIME_TO_LIVE+DEVIATION], [SHARK_TIME_TO_REPRODUCE-DEVIATION, SHARK_TIME_TO_REPRODUCE+DEVIATION]. Останалите три масива са sharkMove, sharkEat и fishMove които съдържат четирите посоки UP, DOWN, LEFT, RIGHT в разбъркан ред за всеки елемент. По този начин се постига истинска произволност без да се създава забавяне по време на паралелната обработка. Ако например риба се намира на координати (i,j) и иска да се движи, тя ще провери за свободни полета в реда в който са записани посоките във fishMove[i*COLUMN_COUNT+j]. Необходимо условие е DEVIATION да бъде по-малко от LIVE и REPRODUCE за рбите и акулите.

3.2 Балансиране и фази

Статичното циклично балансиране зависи от константите GRANULARITY=g, THREAD_COUNT=p и ROW_COUNT=r. В първа фаза i-тата нишка за i=0..p-1 обработва g-2 на брой редове започвайки от ред i*g+1, след което „прескача” g*p на брой редове и отново обработва g-2 реда. Прави r/(p*g) на брой „прескачания” или r/(p*g)+1, ако i<(r%(p*g))/g. Броят редове които се „прескачат” и броят „прескачания” при втора и трета фаза е същият като при първа. Броят редове които се обработват при тези фази е 1, а редът от който започва обработката е i*g за втора и (i+1)*g-1 за трета фаза. Разделяйки обработката на матрицата на фази се избягва евентуален конфликт с паралелизма върху споделена памет. Необходимо условие е r да се дели целочислено на g и r>=g*p

Пример с p=3, g=4, r=20

Нека имаме нишките A(i), B(i), C(i) които обработват матрица с 20 реда, i=1,2,3 за всяка фаза: първа е в зелено, втора в синьо, трета в розово

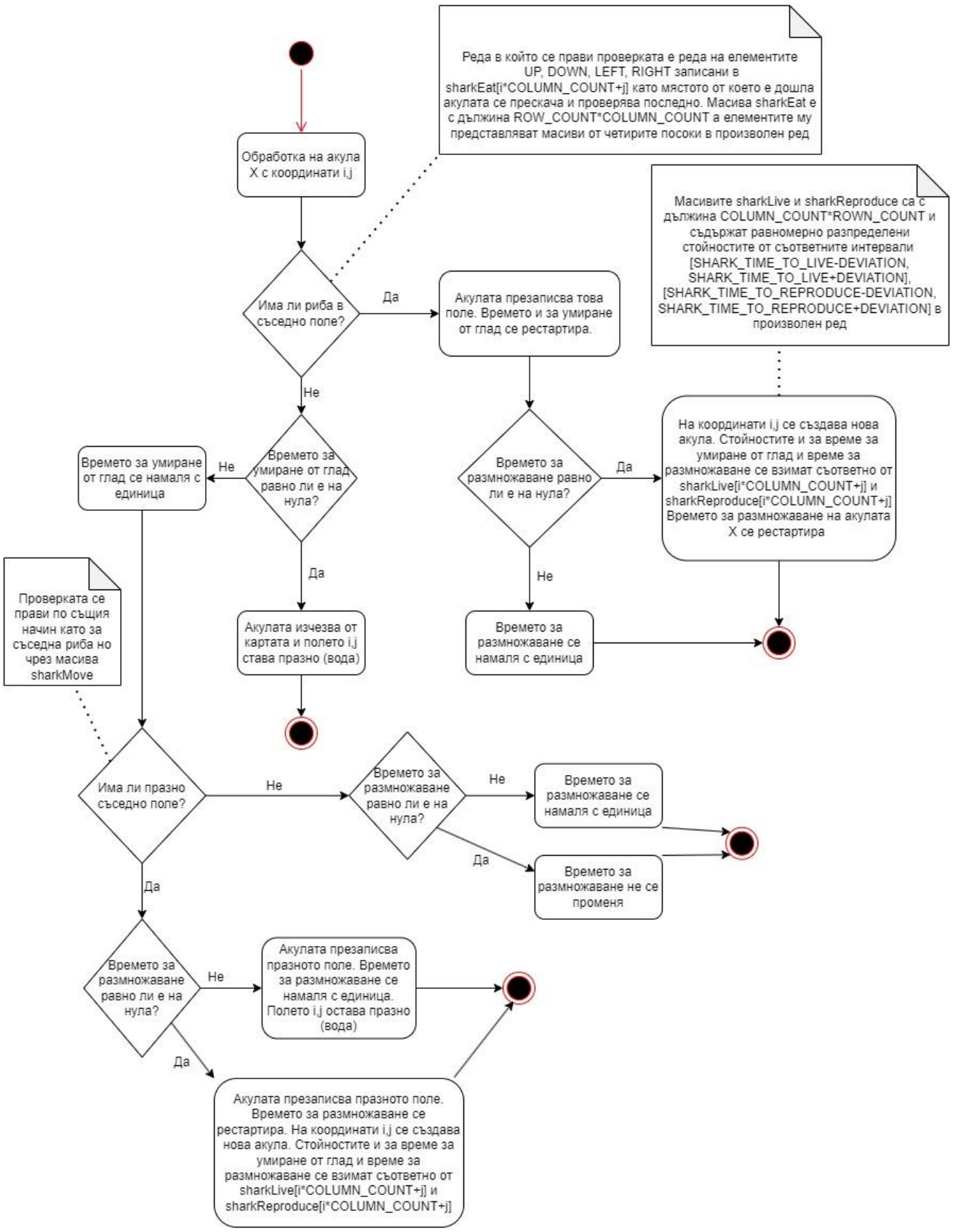
A2
A1
A1
A3
B2
B1
B1
B3
C2
C1
C1
C3
A2
A1
A1
A3
B2
B1
B1
B3

Първо трите нишки обработват зелената част. След приключване се пускат нови три нишки които да обработят синята част, а след това зелената. С това се изчерпва една итерация от обработването на матрицата и картата е готова да се запази във файл.

3.3 Обработка на елементите на матрицата

Нишките обхождат елементите на матрицата ред по ред и ги обработват само ако не са били обработени в текущата итерация. В противен случай на теория е възможно акула да изяде цял ред риби в една итерация. За тази цел при инициализирането на картата се разполагат на произволни места всички акули и риби и всеки елемент на матрицата се отбелязва като необработен. Ако текущият елемент е празно поле (вода), той просто се отбелязва като обработен и изпълнението продължава. Ако елемента е риба, тя се опитва да се придвижи на свободно поле, проверявайки полетата около нея в произволен ред като се връща там откъдето е дошла само ако няма други възможности. При успешно придвижване времето за умиране от глад се рестартира. Ако рибата не се придвижи времето за умиране от глад се намаля с единица или рибата изчезва от картата ако то стане равно на нула. Времето на размножаване се намаля с единица а ако стане равно на нула, рибата оставя след себе си нова риба при движение и времето за размножаване се рестартира. Ако времето за размножаване е нула и рибата не може да се придвижи то не се променя. Ако полето е акула тя първо се опитва да се придвижи на съседно поле на което има риба и чак тогава на празно поле, следвайки същият механизъм който използва и рибата. Ако успее да изяде риба времето за умиране от глад се рестартира, в противен случай се намаля с единица. Ако то стане нула акулата умира. Размножаването при акулите е същото като при рибите.

Диаграма на дейностите на обработка на поле с акула:


Обработка на акула X с координати i,j
Реда в който се прави проверката е реда на елементите UP, DOWN, LEFT, RIGHT записани в sharkEat[i*COLUMN_COUNT+j] като мястото от което е дошла акулата се прескача и проверява последно. Масива sharkEat е с дължина ROW_COUNT*COLUMN_COUNT а елементите му представляват масиви от четирите посоки в произволен ред
Има ли риба в съседно поле?
Да
Акулата презаписва това поле. Времето и за умиране от глад се рестартира.
Не
Масивите sharkLive и sharkReproduce са с дължина COLUMN_COUNT*ROWN_COUNT и съдържат равномерно разпределени стойностите от съответните интервали [SHARK_TIME_TO_LIVE-DEVIATION, SHARK_TIME_TO_LIVE+DEVIATION], [SHARK_TIME_TO_REPRODUCE-DEVIATION, SHARK_TIME_TO_REPRODUCE+DEVIATION] в произволен ред
Времето за умиране от глад равно ли е на нула?
Не
Времето за умиране от глад се намаля с единица
Времето за размножаване равно ли е на нула?
Да
На координати i,j се създава нова акула. Стойностите и за време за умиране от глад и време за размножаване се взимат съответно от sharkLive[i*COLUMN_COUNT+j] и sharkReproduce[i*COLUMN_COUNT+j] Времето за размножаване на акулата X се рестартира
Не
Времето за размножаване се намаля с единица
Да
Акулата изчезва от картата и полето i,j става празно (вода)
Проверката се прави по същия начин като за съседна риба но чрез масива sharkMove
Има ли празно съседно поле?
Не
Времето за размножаване равно ли е на нула?
Не
Времето за размножаване се намаля с единица
Да
Времето за размножаване не се променя
Да
Времето за размножаване равно ли е на нула?
Не
Акулата презаписва празното поле. Времето за размножаване се намаля с единица. Полето i,j остава празно (вода)
Да
Акулата презаписва празното поле. Времето за размножаване се рестартира. На координати i,j се създава нова акула. Стойностите и за време за умиране от глад и време за размножаване се взимат съответно от sharkLive[i*COLUMN_COUNT+j] и sharkReproduce[i*COLUMN_COUNT+j]

4. Тестване

Проведени бяха три вида теста. При първия се изследва влиянието на паралеността, тъй като тя се променя а грануларността остава същата. При втория тест се изследва влиянието на грануларността – паралелността в този тест остава непроменена. При третият тест се гледа как итерациите и размера на матрицата влияят на времето за обработка.

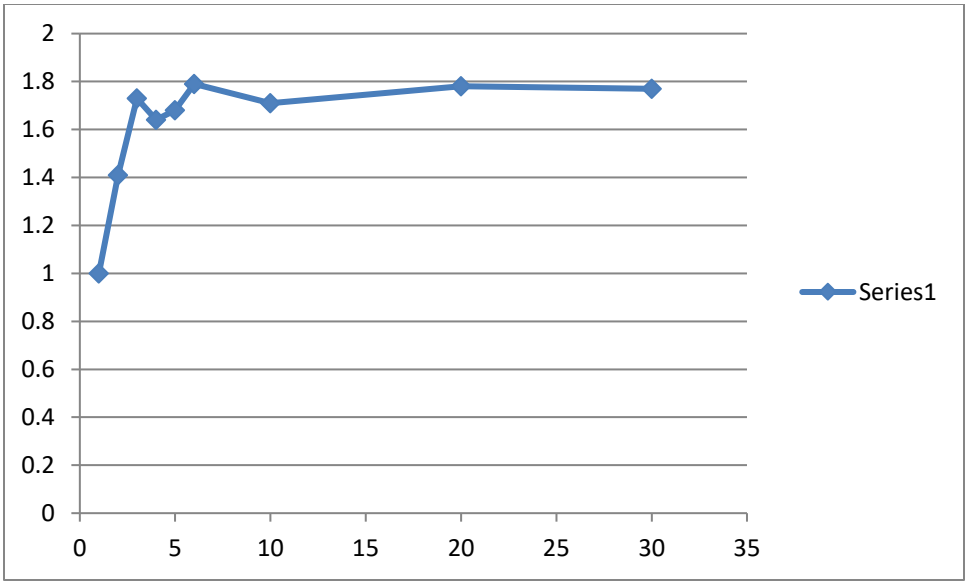
Проведени тестове върху матрица с размери 600x600 и 300 итерации

p	g	t(1) ms	t(2) ms	t(3) ms	t(min) ms	S=t1/t(min)	E=S/p
1	1	63885	62421	60601	60601	1	1.00
2	1	43158	44279	42838	42838	1.41	0.71
3	1	35656	38342	34861	34861	1.73	0.58
4	1	36898	37729	36788	36788	1.64	0.41
5	1	36471	35925	36416	35925	1.68	0.34
6	1	37121	34161	33827	33827	1.79	0.30
10	1	35413	35372	36287	35372	1.71	0.17
20	1	35016	33879	34911	33879	1.78	0.09
30	1	35184	34144	36686	34144	1.77	0.06

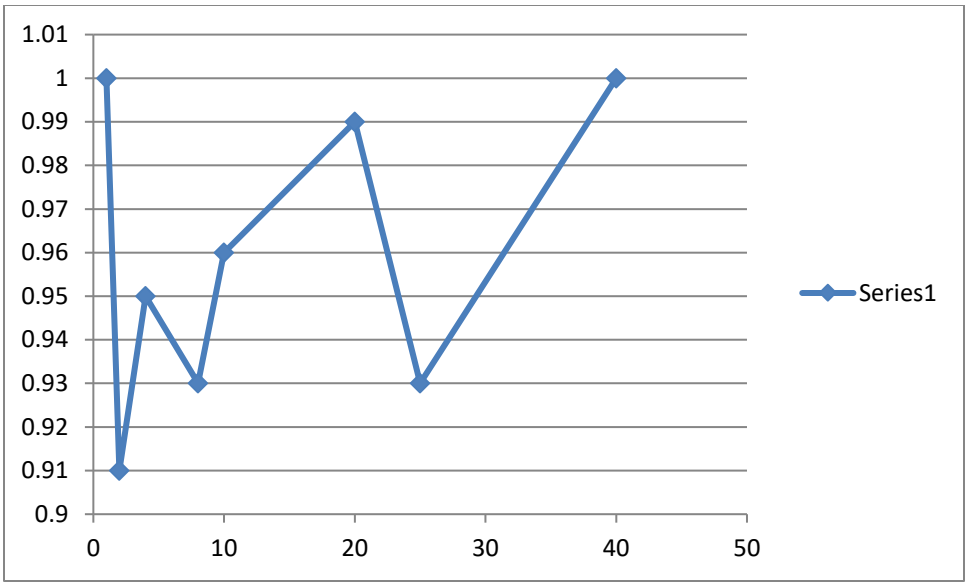
p	g	t(1) ms	t(2) ms	t(3) ms	t(min) ms	S=t1/t(min)	E=S/p
3	1	35656	38342	34861	34861	1	0.33
3	2	38366	38125	37933	37933	0.91	0.30
3	4	36746	36626	37241	36626	0.95	0.32
3	8	38865	37295	37259	37259	0.93	0.31
3	10	37343	36823	35985	35985	0.96	0.32
3	20	35213	37415	37284	35213	0.99	0.33
3	25	37228	38591	37665	37228	0.93	0.31
3	40	36907	34558	35515	34558	1	0.33

p	g	t(1) ms	t(2) ms	t(3) ms	t(min) ms	iterations	cells
1	1	63885	62421	60601	60601	300	360000
1	1	22192	24254	26560	22192	300	180000
1	1	27798	33140	31938	27798	150	360000

Интересно наблюдение от последния тест е че намаляйки итерациите наполовина или броя клетки наполовина ние намаляме общия брой клетки които трябва да се обработят наполовина. Виждаме, че и в двата случая необходимото време за изпълнение на програмата е сходно. Времето за обработка на първоначалният общ брой клетки обаче не е двойно повече, а три пъти повече.



Резултати от първият тест изследващ ускорението при различен паралелизъм. Оста X представлява броят нишки а оста Y – ускорението. Забелязваме, че ускорението при четири нишки спада въпреки че процесорът е четири ядрен



Резултати от втория тест, изследващ влиянието на грануларността. Оста X представя различната грануларност а оста Y – ускорението. Забелязваме че промяната на грануларността почти не оказва влияние върху времето за изпълнение на програмата. Достига се едва 9% промяна в ускорението. Още повече, всяка грануларност различна от единица влияе негативно на ускорението и скорост, по-бърза от тази при най-едрата възможна грануларност не се постига.

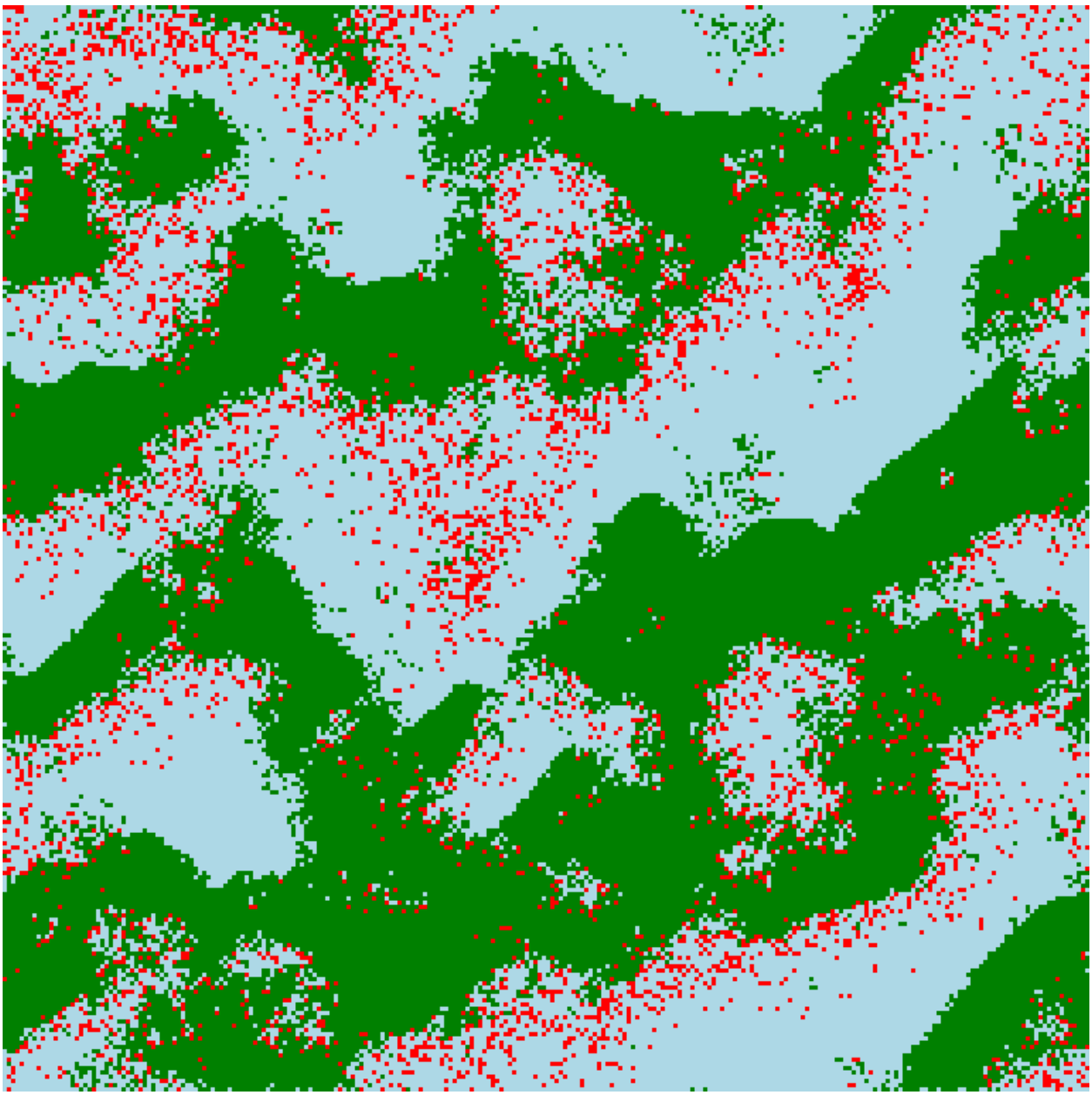
5. Източници:

[1] Computer Recreations: Sharks and fish wage an ecological war on the toroidal planet Wa-Tor

https://www.scientificamerican.com/article/computer-recreations-1984-12/

[2] Java concurrency tutorial

https://docs.oracle.com/javase/tutorial/essential/concurrency/



Картинка от дадена итерация на картата. Рибите са в зелено а акулите в червено