Софийски университет „Св. Кл. Охридски”

Факултет по математика и информатика

Специалност
„Компютърни науки”

Дисциплина: Системи за паралелна обработка

Зимен семестър, 2020/2021 год.

Тема: Паралелизирана симулация на популационна динамика „Wa-Tor“

Курсов проект

Автор:

Александър Иванов Иванов, фак. номер 81841

Юли, 2021

София

Съдържание

Въведение

В настоящата работа ще разглеждаме паралелизирана имплементация на класическата симулация за популационна динамика „Wa-Tor“.

Описание на планетата Wa-Tor

Wa-Tor (от англ. Water-Torus) е хипотетична планета, която притежава форма, за разлика от познатите ни планети, на ротационната повърхнина „тор“ - Поничка с дупка. Цялата ѝ повърхност е покрита с океан, а населението се състои от точно два биологични вида – „риби“ и „акули“, със същите имена на представителите им. Океанът на планетата Wa-Tor е разбит, в математическия смисъл на разбиване на множество, на клетки, всяка от които е съседна с точно 4 други клетки. Времето тук се измерва чрез неделими времеви единици, наричани още хронони(Терминът е заимстван от хипотетичен квант за време, в смисъла на квантовата механика).

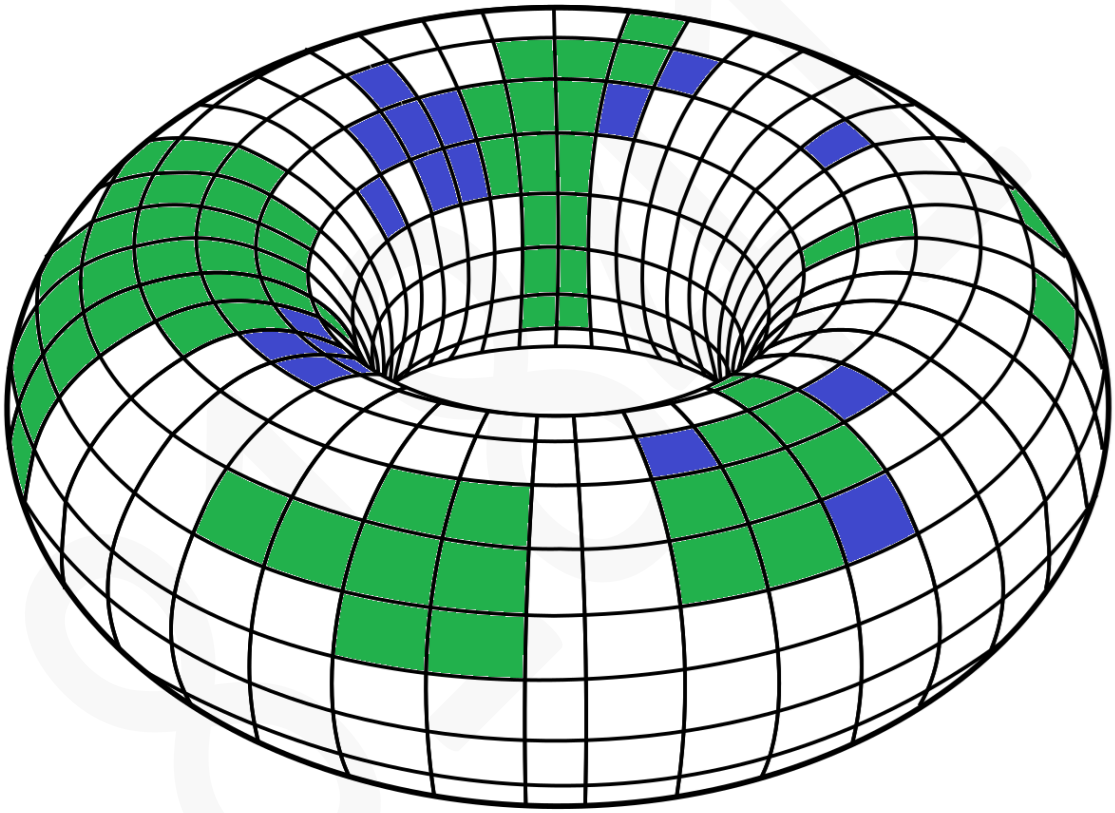
Динамиката на планетата се състои от това, че рибите и акулите могат да се преместват в съседни клетки и да се размножават. Рибите нямат нужда от храна, за да оцеляват, но животът на акулите зависи от редовната консумация на риби.

За рибите:

- За всяка времева единица, всеки представител от вида „риби“ се премества в съседна празна клетка, оставяйки на предишната си позиция празна клетка(освен при размножаване).
  - През определен брой времеви единици от своя живот, всяка риба се размножава, оставяйки на мястото на освободената след преместването ѝ клетка нова риба.
  - В случай, че за рибата няма съседни празни клетки, тя не се премества и следователно не се размножава.

- За акулите
  - За всяка времева единица, всеки представител от вида „акули“ се премества в съседна празна клетка, или такава, окупирана от риба, оставяйки на предишната си позиция празна клетка(освен при размножаване).
    - През определен брой времеви единици от своя живот, всяка акула се размножава, оставяйки на мястото на освободената след преместването ѝ клетка нова акула.
    - В случай, че за акулата няма съседни празни клетки или риби, тя не се премества и следователно не се размножава.
    - При преместване в клетка, окупирана от риба, акулата регенерира определена част от енергията си, а рибата бива изяждана т.е изчезва.

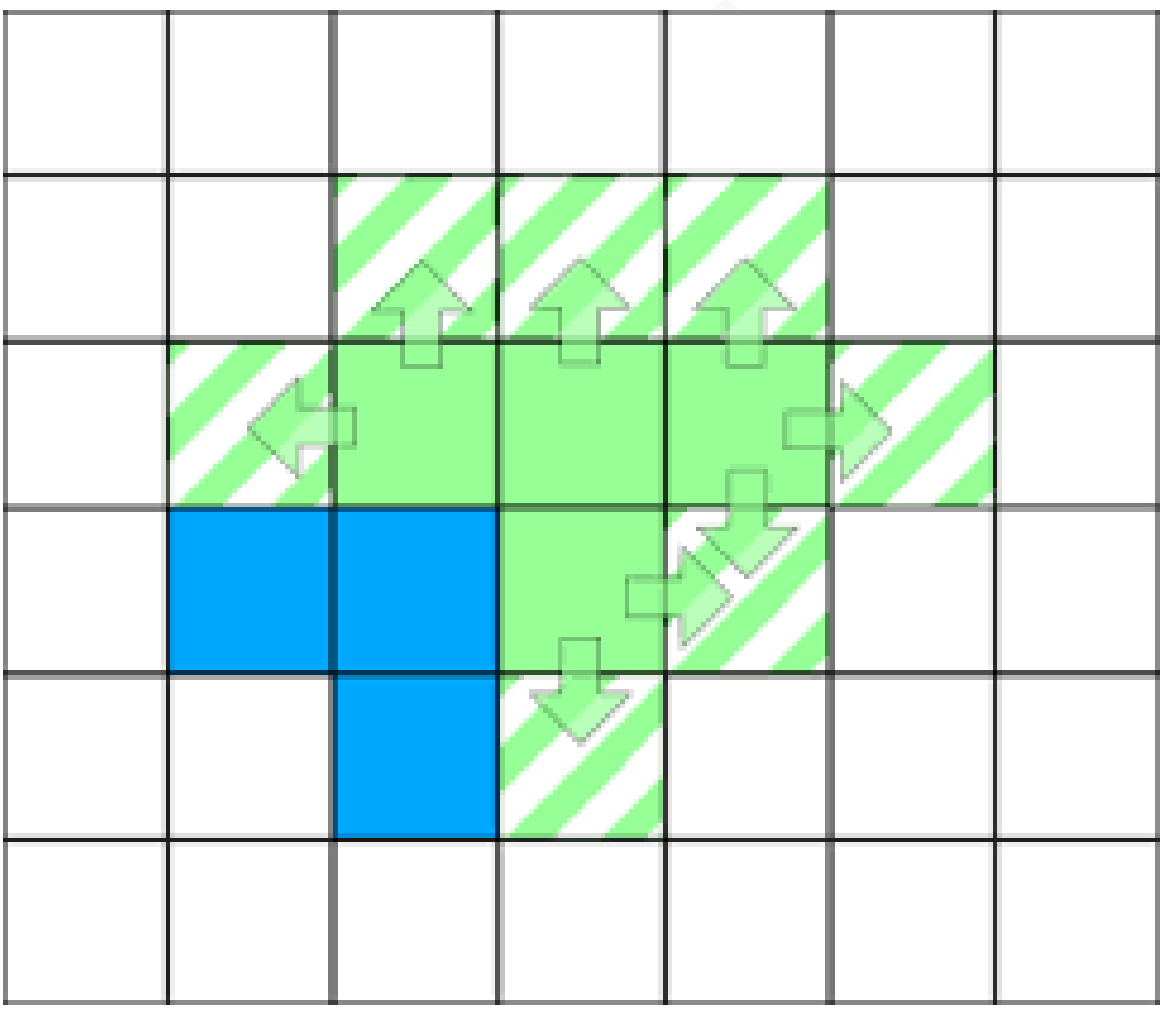
- За всяка времева единица от своя живот, акулата губи определено количество енергия, независимо дали се е преместила или не.
    - Ако в някоя времева единица енергията на акулата се изчерпа, тя умира, т.е изчезва и оставя на свое място празна клетка, вместо да се движи и размножава.



Фигура 1: Илюстрация на планетата Wa-Tor. Виждат се клетки, окупирани от риби(в зелен цвят), както и такива, окупирани от акули (в син цвят)

Матричен модел на Wa-Tor

За целите на повечето научни трудове, включително и настоящия, Wa-Tor се представя като променлива матрица с размер NxM, чиито клетки представляват тези от планетата и съответно във всеки момент всяка може да бъде празна или запълнени с риба или акула. За целта на илюстрацията, рибите отново са отбелязани в зелен цвят, а акулите – в син.



Фигура 2: Матричен модел на Wa-Tor

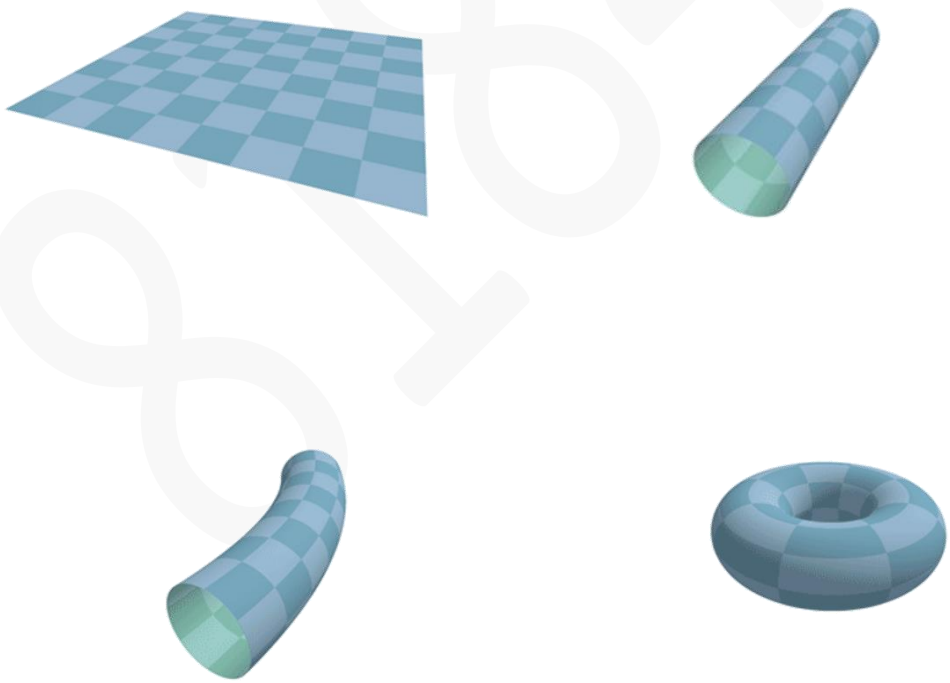
Клетките на матрицата ще номерираме с двойка цели числа:

- $i = 0 \ldots N$: индекс на реда на клетката
- $j = 0 \ldots M$: индекс на колоната на клетката

Четирите съседни на всяка клетка ще наричаме горна, долна, лява и дясна.

- За да постигнем коректност спрямо определението на Wa-Tor (т.е всяка клетка да има точно четири, съседни), ще въведем следните правила:
    - Клетка с индекс $i, 0$ има лява съседна клетка с индекс $i, M - 1$ и е нейна дясна съседна клетка.
    - Клетка с индекс $0, j$ има горна съседна клетка с индекс $N - 1, j$ и е нейна долна съседна клетка.

Така въведените правила за съседните клетки правят матричния модел напълно изоморфен на истинската хипотетична планета. Интуитивно можем да демонстрираме това, сгъвайки лист карирана хартия във формата на тръба и след това свързвайки двата ѝ края. По този начин, клетките на листа са съседни точно според правилата, а формата му е същата като на Wa-Tor. Получихме макет на планетата.



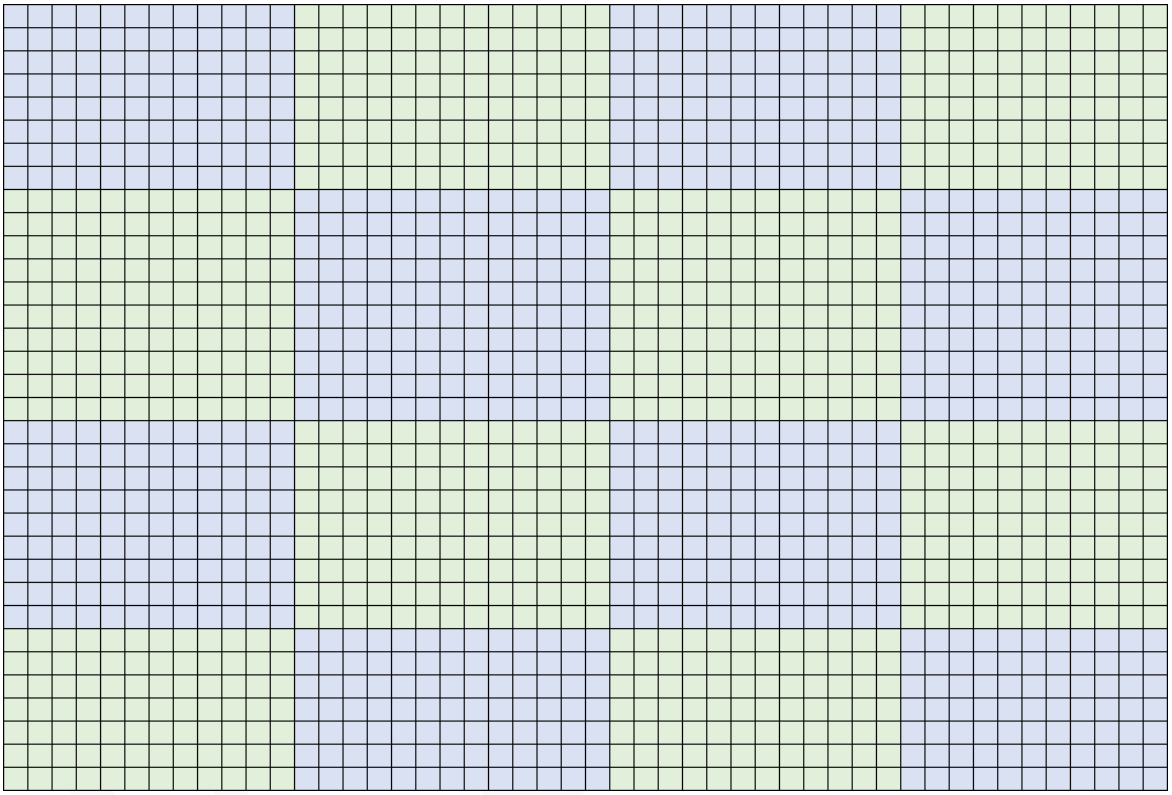
Фигура 3: Топологична илюстрация на Wa-Tor

Алгоритъм за симулация на популацията на Wa-Tor

В настоящата работа ще предложим паралелизиран алгоритъм за симулация на динамиката на планетата и ще разгледаме производителността на неговата имплементация.

Разделяне на подматрици

За целта на имплементацията, ще използваме логическо разбиване на матрицата на приблизително равни по брой редове и стълбове подматрици.



Фигура 4: Примерно разбиване на подматрици

Разбиването следва следните правила:

- Разбиването трябва да има вид на шахматна дъска
    - Всеки две хоризонтално съседни матрици имат равен брой редове
    - Всеки две вертикално съседни матрици имат равен брой стълбове
- Въвеждаме понятията „четна“ подматрица и „нечетна“ подматрица
    - Матрицата, съдържаща клетката с индекс 0,0 е четна
    - Две матрици наричаме съседни т.с.т.к една от тях съдържа клетка, съседна на клетка от другата
        - Всяка съседна на четна подматрица е нечетна
        - Всяка съседна нечетна подматрица е четна

За да запазим коректността спрямо правилата за съседни клетки е важно да отбележим, че броят редове от подматрици и броят стълбове от подматрици трябва да бъде четен (На фиг.4 това е спазено: имаме 4 реда и 4 стълба).

Същност на алгоритъма

С така въведените понятия за четни и нечетни подматрици, можем да забележим, че според правилата на симулацията, не е възможно една риба или акула да премине от четна в друга четна или от нечетна в друга нечетна подматрица в рамките на една времева единица. Тя ще остане в същата такава или ще се премести в съседната, която е от различна четност. Следователно, можем да твърдим, че не е нужна синхронизация в рамките на една времева единица при паралелна симулация на подматрици от една и съща четност. Това е в резонната основа на алгоритъм със следния псевдокод:

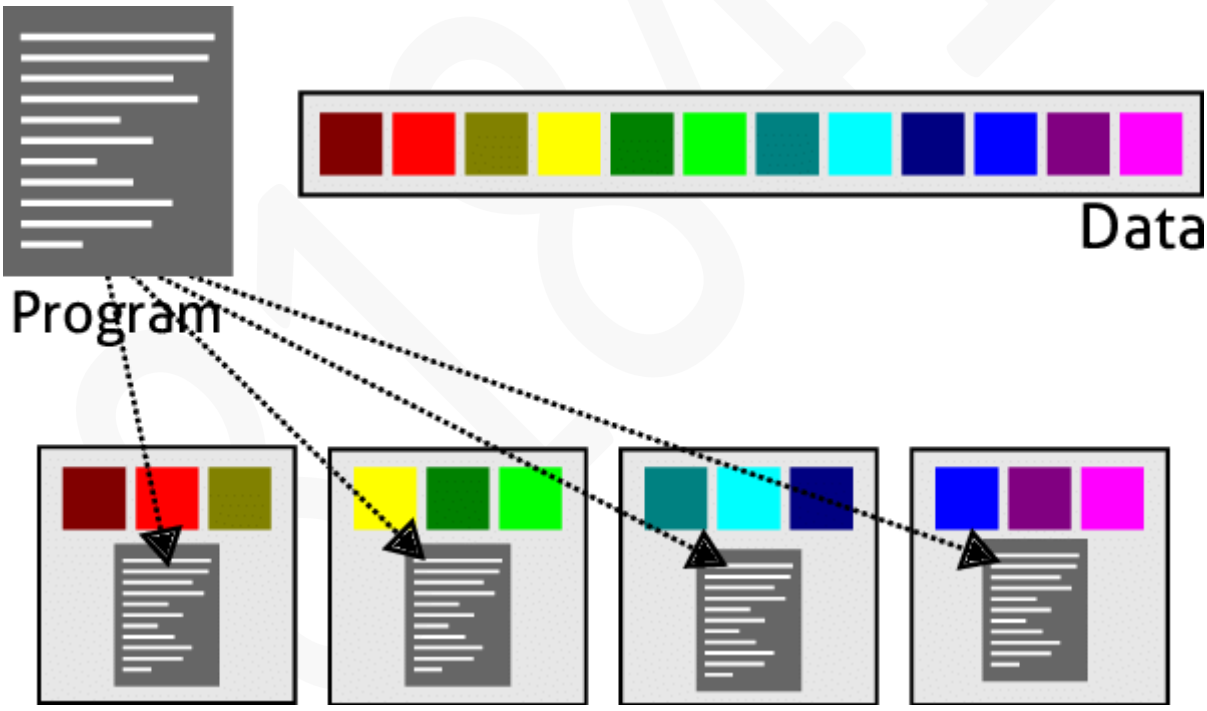
```
Океан <- [Матрица NxM с предварително зададен брой риби и акули]
ТекущоВреме <- 1
[предварително разбиваме Океан на четни и нечетни подматрици]

Повтаряме {
   [Симулираме действията на обитателите на всички четни подматрици]
   [Симулираме действията на обитателите на всички н.ч. подматрици]
   ТекущоВреме <- (ТекущоВреме + 1)
}
```

Както вече уточнихме, двете стъпки за симулация(съотв. на четни и нечетни подматрици, написани по-горе в получер шрифт) могат да бъдат изпълнени паралелно, без синхронизация и ограничение на броя нишки. Синхронизация е нужна само при „редуването“ на двете стъпки и увеличаването на текущото време с единица.

Класификация на алгоритъма

Базирайки се на горепосочените правила за разделяне на океана на подматрици и фактът, че държанието(в смисъл, behavior) на обитателите му се базира единствену върху съседните клетки, можем да разгледаме информацията във всички подматрици като отделни и независими късове данни, които обаче са от един и същ клас. Следователно, паралелната им обработка, тъй като следва еднакви за всички тях правила, спада към SPMD-класа на паралелните системи(Single Program, Multiple Data). На практика всички програми(в случая на имплементацията, която ще разгледаме, нишки) работят с обща споделена памет – целия океан, но те все пак са еднакви, но независими, програми, различаващи се единствено по зададените им параметри – позициите и размерите на подматриците, които обработват.



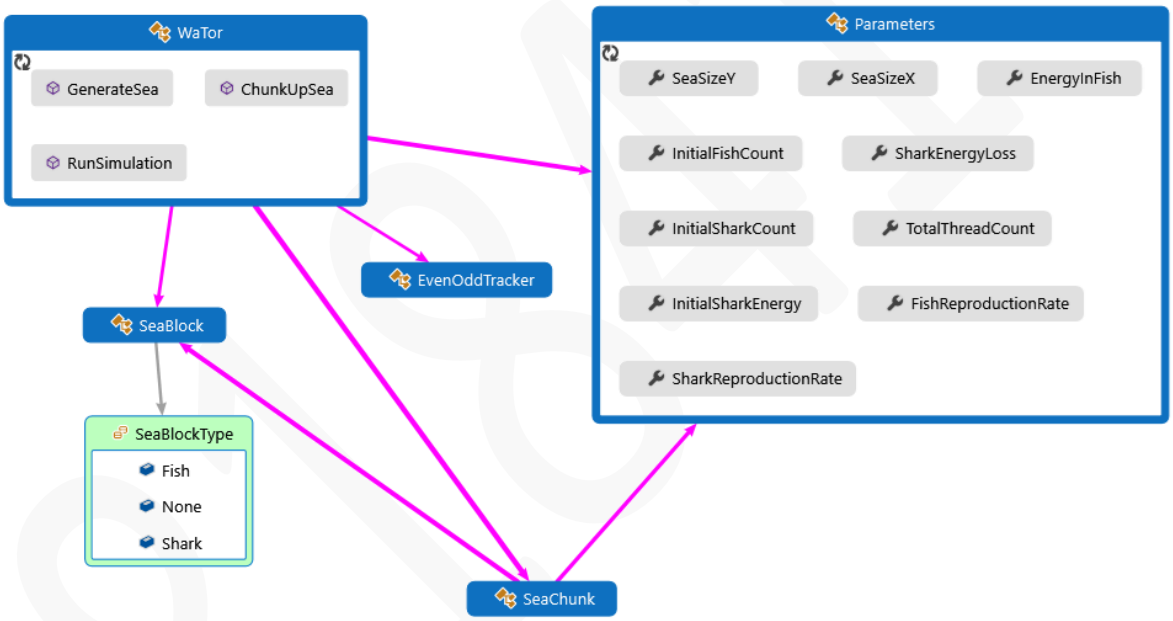
Фигура 5: Илюстрация на модела SPMD; Можем да си представим, че несподелените късове памет, с които всяка инстанция на програмата(нишка) работи са вътрешността на възложените им подматрици, а споделената памет (Data) принадлежи на оркестраторът(програмата, пуснала симулацията).

Имплементация на алгоритъма

Изходният код на приложението може да бъде открит на следния адрес:

https://github.com/startrunner/fmi-spo-wator

За да имплементираме библиотеката ще използваме езикът за програмиране C#, изпълняван в среда .NET Core 5.0. Следва диаграма и описание на имплементацията.

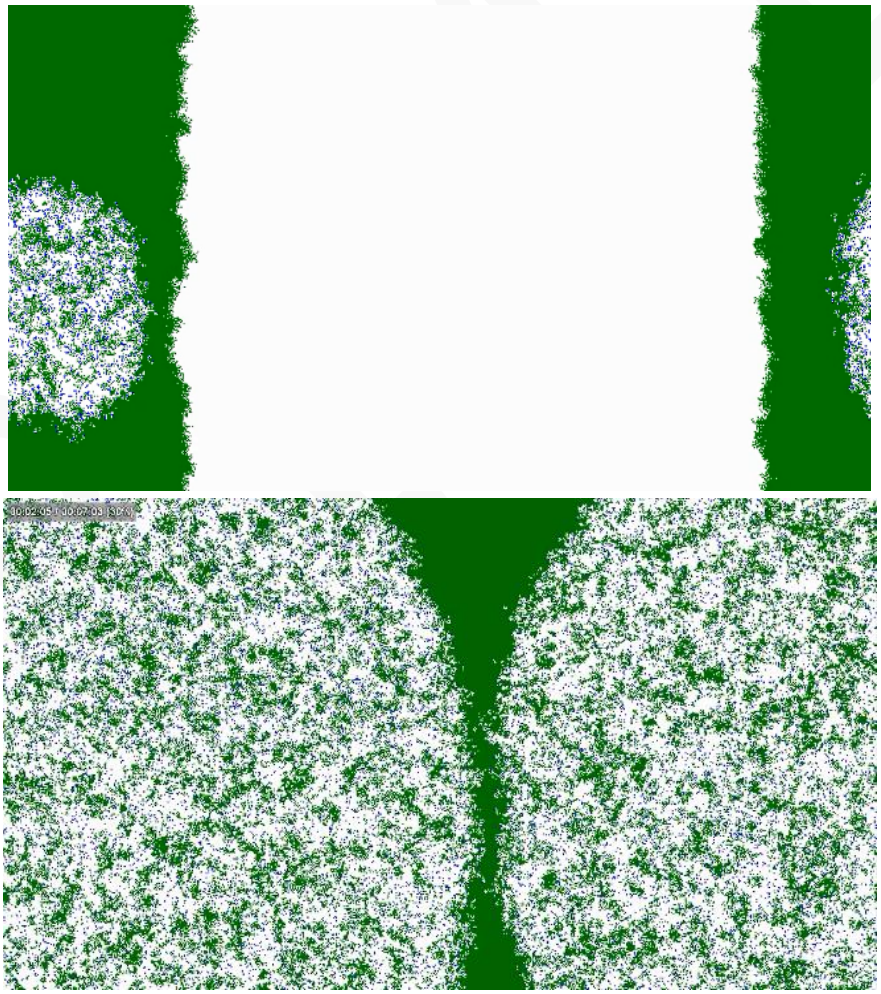


- Запис Parameters – съдържа всички параметри, нужни за симулацията
  - Полета `SeaSizeX`, `SeaSizeY` – съотв. броя редове и броя колони от клетки в океана
  - Полета `InitialFishCount`, `InitialSharkCount` – съотв. броя риби и броя акули **преди началото** на симулацията
  - Параметри за енергията на акулите
    1. Поле `InitialSharkEnergy` – броят единици енергия, с които всяка акула започва своето съществуване
    2. Поле `EnergyInFish` – колко единици енергия получава акула за всяка изядена риба

3. Поле `SharkEnergyLoss` – колко единици енергия губи всяка акула за единица време
   - Поле `TotalThreadCount` – брой нишки, които ще изпълняват симулацията
- Клас `SeaBlock` – представлява клетка от океана; Всяка инстанция има тип(акула, риба или празно); Също съдържа полета, носещи информация, нужна за проследяване на енергията(в случай, че е акула) и „рожденатa дата“ на своя окупант.
- Клас `EvenOddTracker` – техническо средство за синхронизация, осигуряващо правилното „редуване“ на нишките при изчислението на четни и нечетни подматрици.
- Клас `WaTor` – служи за имплементация на алгоритъма
   - Метод `GenerateSea`, който запълва двумерен масив от клетки `SeaBlock[,]` според зададените параметри за начало на симулацията. Този масив представлява матричния модел на планетата.
   - Метод `ChunkUpSea`, който по генерирания от горния метод масив и брой нишки за симулацията разделя морето на подматрици(Представени чрез клас `SeaChunk`, съдържащ указател към целия океан и индекси на интервала от клетки, които обхваща). Връща двумерен масив `SeaChunk[,]`. Спазват се условията за четен брой редове и колони от подматрици. Стреми се броят подматрици да е възможно най-близък до зададения брой нишки.
   - Метод `RunSimulation`
     1. Генерира океан и го разделя на подматрици чрез горните два метода, според подадените параметри
     2. Създава инстанция на `EvenOddTracker`
     3. Започва паралелно изпълнение на зададения брой нишки за симулация, като разпределя приблизително по равен брой подматрици на всяка. Всички нишки са една и съща подпрограма и използват същата инстанция на `EvenOddTracker` от стъпка 2. Техни единствени параметри биват множествата от разпределените им за изчисление подматрици

Графично представяне

За графично представяне на симулацията в реално време вграждаме кода за симулацията в Windows Forms приложение, което стартира симулацията и разполага с една допълнителна нишка, която показва състоянието на всяка клетка, оцветявайки един пиксел на екрана в зелен или син цвят, ако в клетката има съотв. риба или акула. Когато всички клетки са изобразени, тя започва отначало. Тя не е част от симулацията и не е синхронизирана с нишките от нея. Възможно е изобразената информация в части от екрана да е по-стара от тази в други, но това не пречи да добием представа за състоянието на симулацията.



Фигура 6: Откъси от графично изобразяване на симулация. Тук можем да видим миграцията на риби и акули от най-лявата колона в най-дясната т.к клетките в тях са съседни. В началото на тази симулация всички риби и акули се намират в най-лявата част на матрицата, за да се избегне чакане, докато се срещнат.

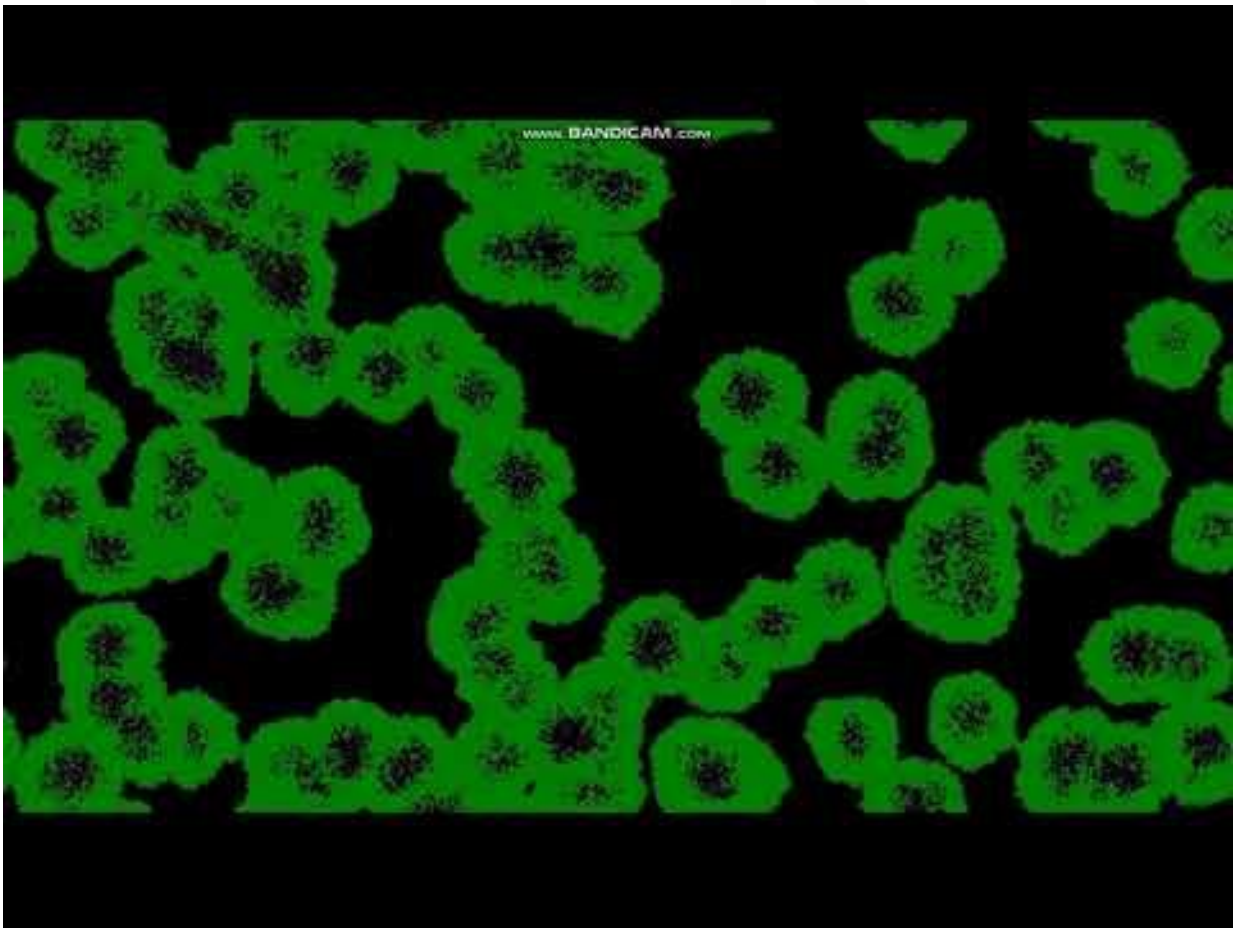
Запис на графичното представяне

Запис на графичното представяне може да бъде разгледан на следния адрес:

https://www.youtube.com/watch?v=bfd6Y05obV4

(Тук акулите са в лилав вместо син цвят, защото се различават по-добре.)

Разделителна способност на записа: 1920x1080



Симулация без графично представяне

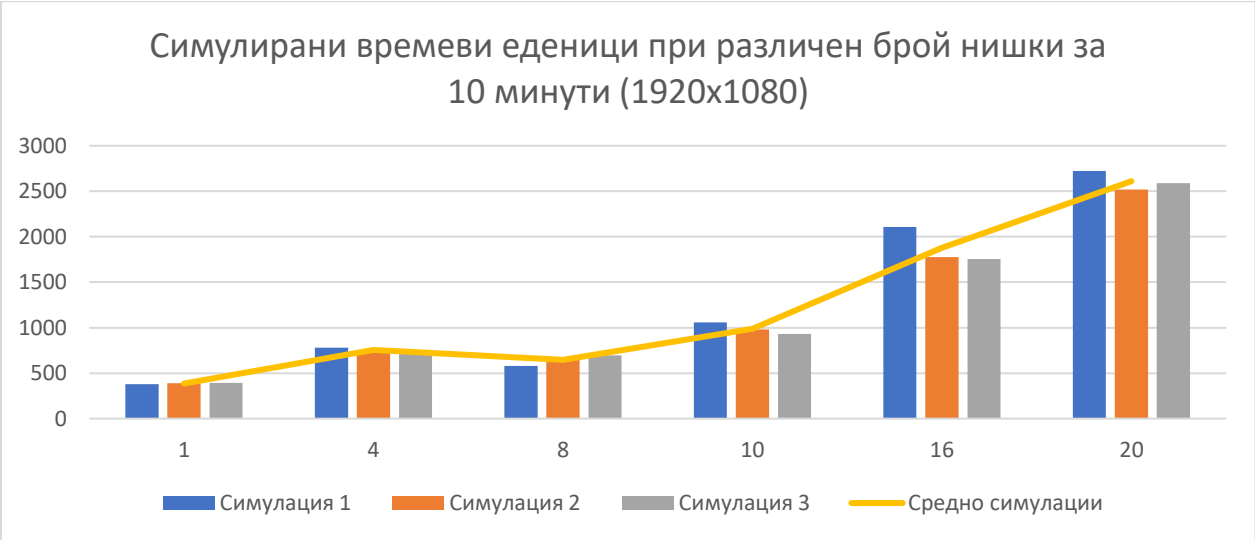
За да изследваме промяната в производителността на имплементацията, проведохме множество опити върху машина със следните параметри:

```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                32
On-line CPU(s) list:   0-31
Thread(s) per core:    2
Core(s) per socket:    8
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 45
Model name:            Intel(R) Xeon(R) CPU E5-2660 0 @ 2.20GHz
Stepping:              7
CPU MHz:               1394.470
CPU max MHz:           3000.0000
CPU min MHz:           1200.0000
BogoMIPS:              4389.52
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              20480K
NUMA node0 CPU(s):     0-7,16-23
NUMA node1 CPU(s):     8-15,24-31
```

Всеки опит бе изчислен по три пъти, за да елиминираме неточности, породени от външни фактори като натоварване на споделената машина от други потребители. Началният брой риби и акули е еднакъв при всички опити, а никой от двата вида не е изчезнал след края на някоя симулация.

Първи опит: океан с размер 1920x1080, 1-20 нишки

За този опит провеждаме симулация на океан с размер 1920x1080 клетки. Измерваме броя на симулираните времеви единици в рамките на 10 минути.



Фигура 7: Резултати от първи опит Хоризонтално: Брой нишки, Вертикално: Брой изчислени времеви единици

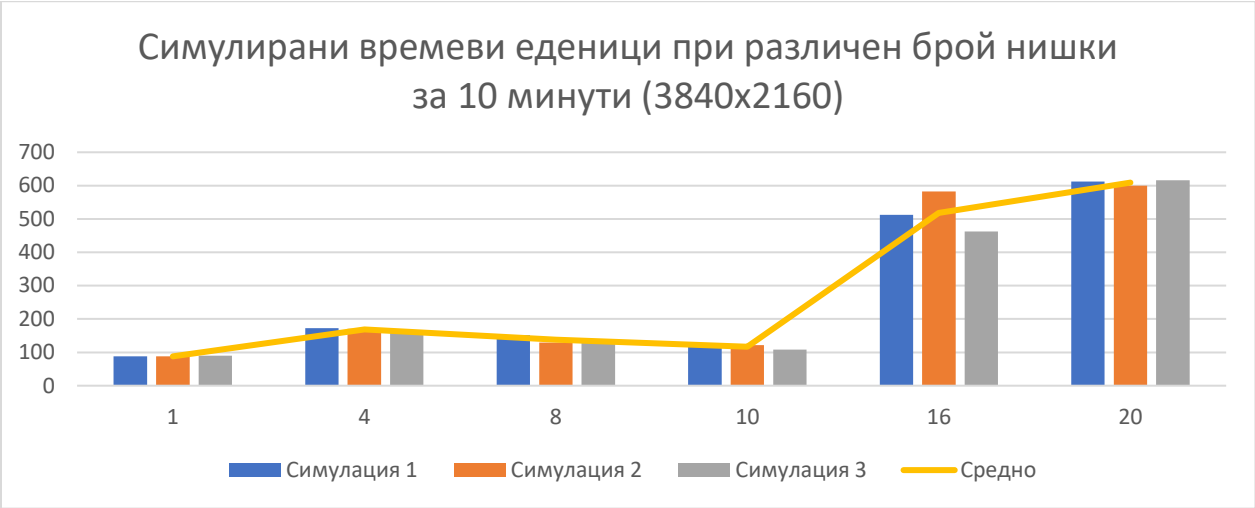
Брой нишки	Симулация 1	Симулация 2	Симулация 3	Средно
1	380	388	392	386
4	781	752	730	754
8	578	666	695	646
10	1059	980	930	989
16	2107	1775	1754	1878
20	2723	2519	2590	2610

Фигура 8: Таблични данни от първи опит

При този опит наблюдаваме около 40% увеличение на производителността при добавяне на три нишки.

Втори опит: океан с размер 3840x2160, 1-20 нишки

Този опит е аналогичен на първия, но този път океанът е с 4 пъти повече клетки – има размер 3840x2160



Фигура 9: Резултати от втори опит Хоризонтално: Брой нишки, Вертикално: Брой изчислени времеви единици

Брой нишки	Симулация 1	Симулация 2	Симулация 3	Средно
1	88	88	90	88
4	173	171	164	169
8	151	129	135	138
10	122	122	108	117
16	512	582	462	518
20	612	600	616	609

Тук резултатите за доста по-противоречиви. Има видимо подобрение при голям брой нишки, но то е незадоволително и непостоянно. По всяка вероятност това се дължи на сериозното натоварване на споделената машина по време на разработката на курсовите проекти.

Изводи от опитите

С помощтта на симулация върху реална машина, показахме главното предимство на системите за паралелна обработка: увеличена производителност, в случай, на правилна разработка на програмата, Както и възможността да извършваме множество различно операции едновременно върху една машина(Както показахме чрез Windows Forms приложението за визуализация **по време** на симулация). Именно заради тези предимства, в днешно време многопроцесорните машини са широко разпространени, а паралелното програмиране е в основите на съвременната информатика.

Разбира се, не е уместно да пропуснем да споменем и някои често срещани пречки при паралелно програмиране, които често(особено при лош подход за справяне със ситуацията) водят до увеличена цена за разработка на софтуер и дори загуби:

- Взаимно блокиране/изчакване (Deadlock)
  Срещу този класически проблем са се изправили мнозина системи и техните създатели, често без успех. В нашата разработка го избегнахме, но справянето с него е значително по-трудно в големи и сложни системи и изисква високо ниво на дисциплина при работа.
- Състояние на невалидни данни поради лоша синхронизация (Race Condition)
  Това отново е широко разпространен проблем, който избегнахме, именно защото чрез дизайна и имплементацията на алгоритъма сведохме нуждата от синхронизация до минимум, а за имплементацията ѝ отговаря единствено класът `EvenOddTracker`.
- Огладняване(Resource starvation)
  В тази ситуация, когато някоя от всички паралелно изпълнявани програми изостане, поради липса на нужен ѝ ресурс, причинявайки изоставане(замръзване) на други, зависещи от нейния резултат. Този проблем не успяхме да избегнем поради факта, че машината бе споделена с други студенти, също целящи да изследват производителността на своите паралелни алгоритми. т.к когато броят нишки надвиши броя процесорни ядра започва да се губи много процесорно време в т.нар. Task Scheduler на операционната система – паралелността вече се е привидна вместо реална, а производителността – страда. Поради тази причина, дори една нишка от нашата програма да изостане(т.е да бъде спряна от Task Scheduler-а на операционната система), това причинява и следващата „вълна“ от паралелно изпълнявани задачи – тези, обработващи подматрици с противоположна четност.

Източници

- Dewdney, Alexander Keewatin (December 1984). "Sharks and fish Wage an ecological War on the toroidal planet Wa-Tor"
- https://en.wikipedia.org/wiki/Wa-Tor
- https://en.wikipedia.org/wiki/Torus
- https://bg.wikipedia.org/wiki/Тор_(геометрия)

заб. Включените илюстрации