Алогитъмът Wa-Tor. Изследване на скалируемостта му при симулация на статично балансиране

Биляна Инджева

3 юни 2024 г.

Въведение

Wa-Tor е симулационен модел, предназначен за изследване на динамиката на популациите. Той е създаден от Александър Дюдний и публикуван през 1984 година в научното списание „Scientific American“. Светът на Wa-Tor е покрит с вода и обитават го два вида същества - риби и акули. Симулацията моделира еволюцията на тези популации през времето.

Симулацията е позиционирана върху тороид, а не например сфера, поради възможността тороидът да се разгъне в двумерна плоскост с форма на правоъгълник. Това улеснява симулацията и обработката на данните.Цяла та площ на тороида е разделена на равни квадратни клетки, като всяка клетка може да съдържа най-много едно същество (риба или акула) в даден момент на времето.

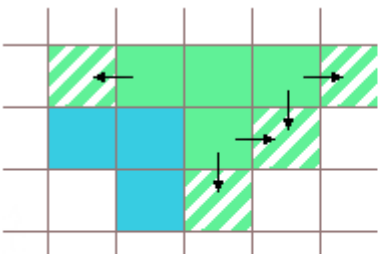
Времето в симулацията тече дискретно и равномерно за всички същества. Всеки следващ цикъл или итерация на симулацията се случва след изтичането на един времеви интервал за всички същества.

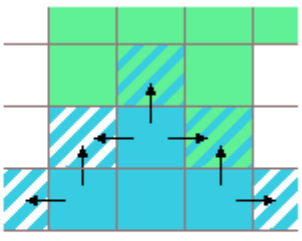
Правила за рибите:

- Движение: През всеки времеви интервал, всяка риба има възможност да се премести на произволна празна позиция до текущата си клетка (нагоре, надолу, наляво или надясно).
- Смърт: Рибите могат да умрат в следствие на изяждане от акула, достигане на максимална възраст, или ако не могат да се преместят в нова позиция.
- Размножаване: Всяка риба притежава брояч, който се нулира при достигане на определена възраст. При това събитие, нова риба се появява на мястото, заемано от първоначалната риба.

Правила за акулите:

- Движение: Във всеки времеви интервал, всеки акула се движи към произволна близка позиция, в която се намира риба и я изяжда. Ако няма съседни риби, акулата се премества на произволна позиция.
  Смърт: Всяка акула има счетоводител за живот, който намалява с едно след всеки изминал интервал и се увеличава с едно, когато изяде риба. Ако броячът падне на нула, акулата умира. Акула така също умира, ако не може да се премести на ново място.
- Размножаване: Самото размножаване се случва по същия начин както при рибите.





В наш интерес е да направим предположение, че всички същества се движат с еднаква скорост и че няма същества с по-висока скорост от другите. Освен това, ще предположим, че съществата не подлежат на болести и че болестите не могат да причинят смъртта им.

За симулиране на такъв тип задачи е особено подходящо да се използват инструментите на паралелното програмиране.

Декомпозиция

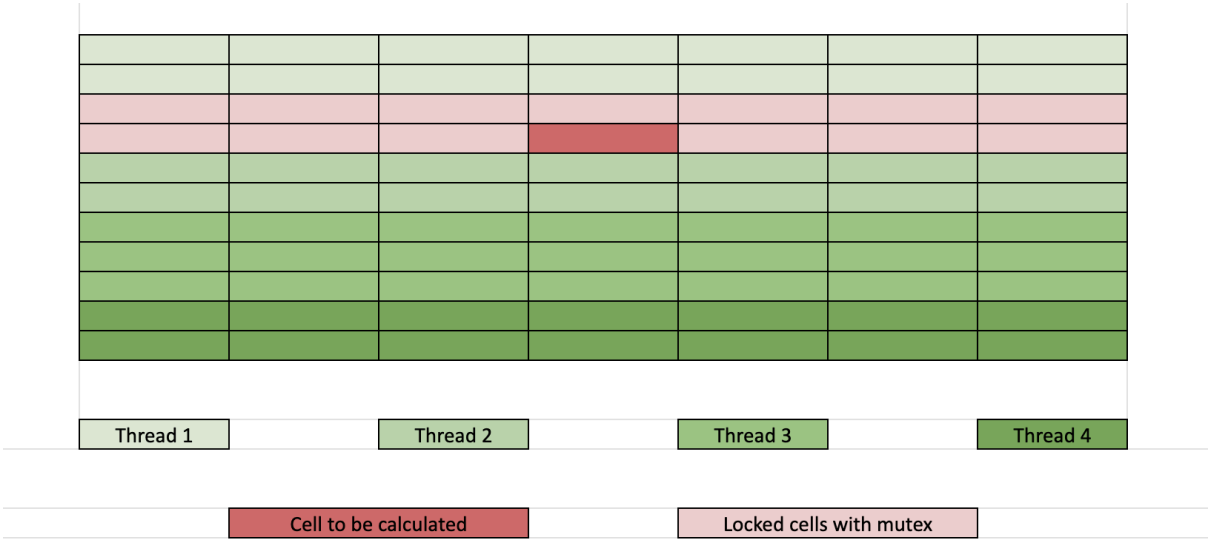
Декомпозицията позволява ефективна паралелизация като разбива задачата на по-малки подзадачи или поддомейни. В случая с Wa-Tor, пространството (равнината) бива разделено на по-дребни поддомейни, като всяка нишка получава отговорност за конкретен поддомейн. Животните движещи се между поддомейните са основният фактор за нуждата от синхронизация.

Методът за декомпозиция може да бъде статичен или динамичен. Статичната възлага неразменяеми поддомейни на всяка нишка от самото начало, докато динамичната включва възможността за преразпределяне на поддомейните през времето, за да бъде получено равномерно разпределение на работата.

Избираме решението за статична декомпозиция, с поддомейните представляващи фиксирани ленти от редове на равнината, заради голямата динамика в симулацията.

Синхронизация

Синхронизацията, основана на мютекси, е ключова за избягване на конфликти при достъп до общите ресурси, особено когато животни движещи се измежду на различни поддомейни.



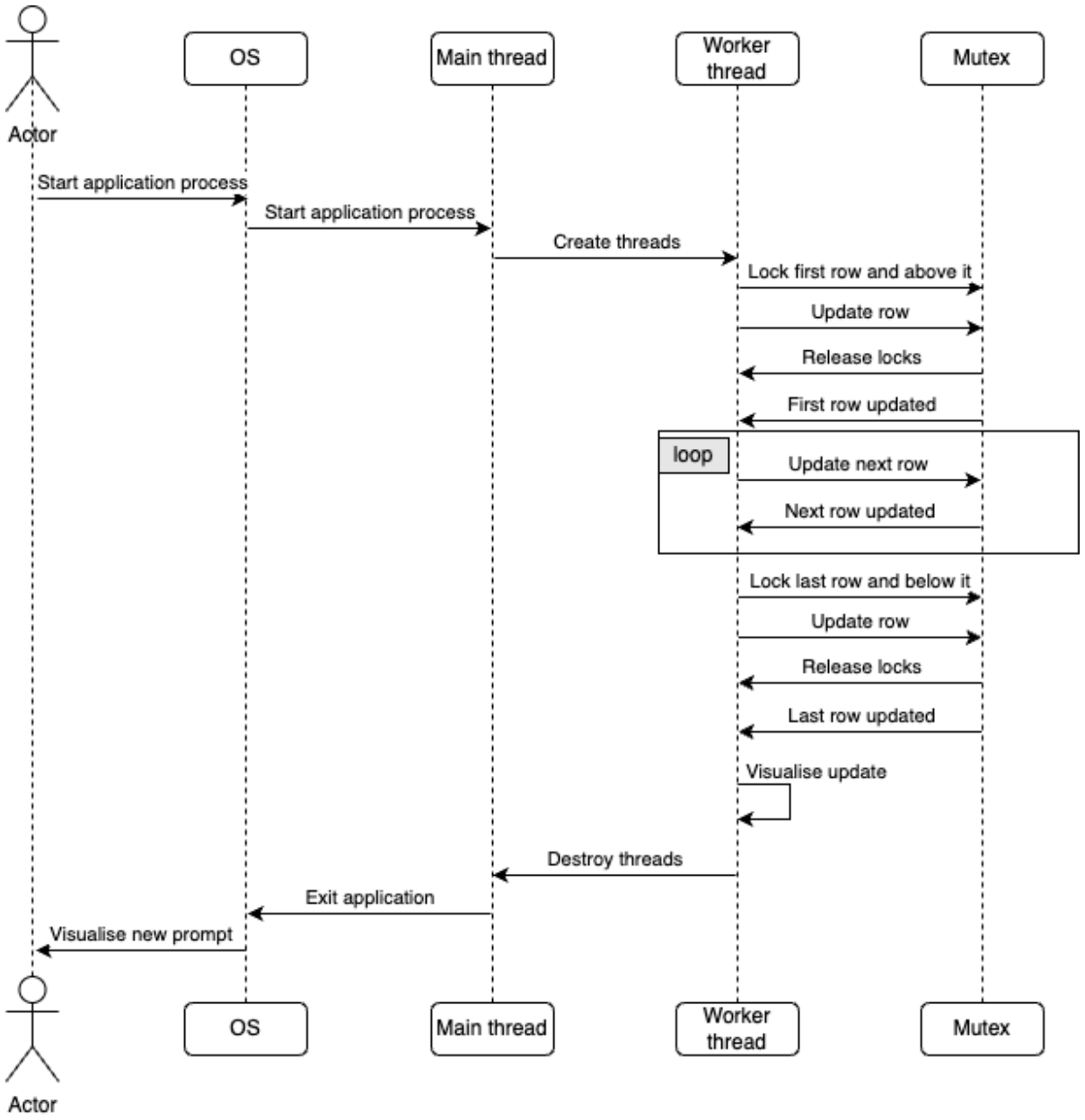
Паралелен алгоритъм

Паралелният алгоритъм, основаващ се на статичната декомпозиция, ще използва подходящо разделение на работата между нишките чрез разпространяване на отговорността за обработване на поддомейни между различни нишки. Последователното обработване на редове вътре в поддомейна и синхронизацията на граничните редове между съседни поддомейни ще бъде осигурено с мютекс.

В цялост, избраният подход осигурява баланс между ефективността на производителността при паралелната обработка и нуждата от синхронизация при динамични среди като Wa-Tor.

Архитектура

UML пакетната диаграма е мощен инструмент за визуализация и организация на елементите в твоята система. Тя може да помогне при планирането и анализа на структурата на системата, и може да поясни отношенията и зависимостите между различни модули или компоненти. Сега ще разгледаме диаграма на нашето приложение.



Диаграмите на последователностите показват в какъв ред ще се изпълняват действията, извършвани, когато настъпи дадено събитие. При този тип диаграма времето нараства в посока от горе надолу, а със стрелки между отделните вертикални линии се показва комуникацията между актьорите и линиите на живота.

В нашия случай потребителят се обръща към операционната система, за да стартира приложението. Тя от своя страна стартира приложението в нов процес и приложението започва обработка според поставените му параметри.

Всяка нишка започва като обработва първия ред, заключва този над него и самия ред. След това променя данните на реда и освобождава заключените

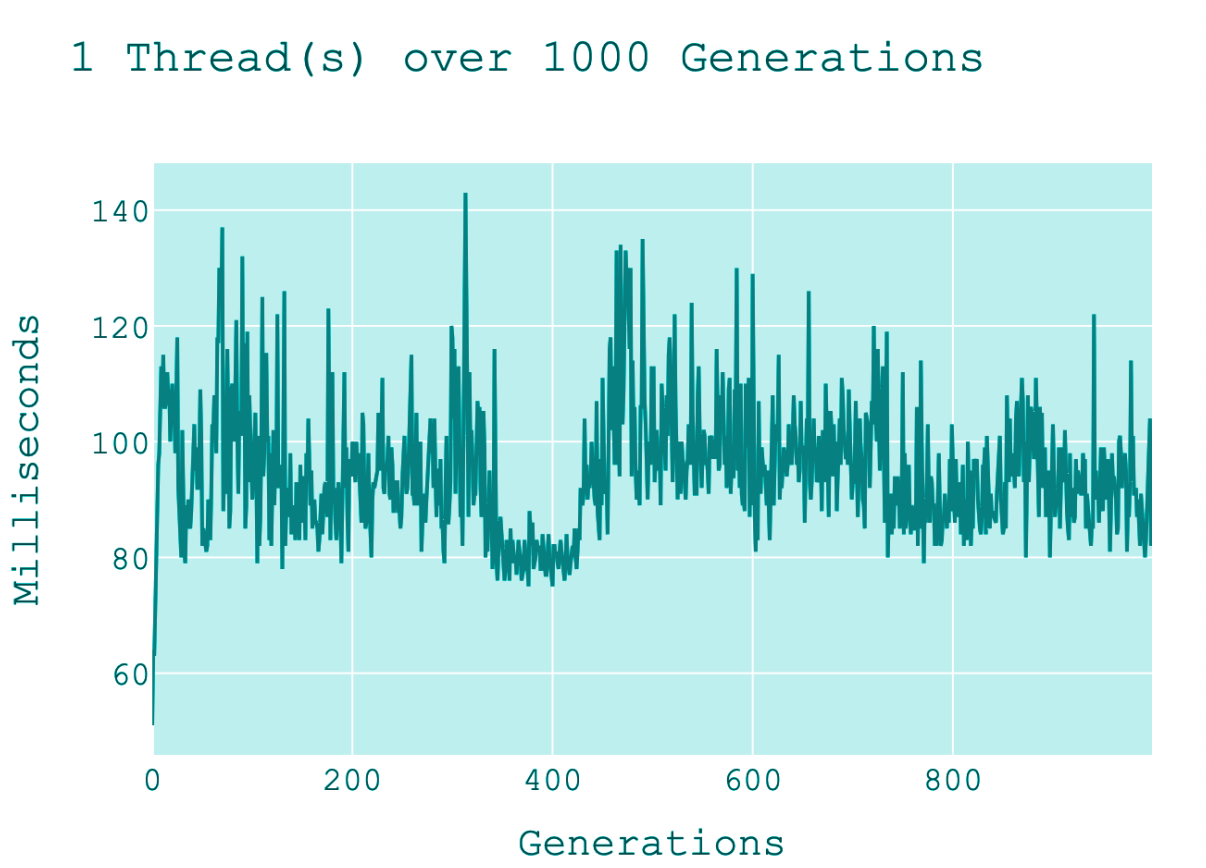
клетки. Последователно се смятат всички редове до последния. На последния ред първо заключваме реда и този под него, след това се актуализира и ги отключваме. Също така се визуализират промените.

Система

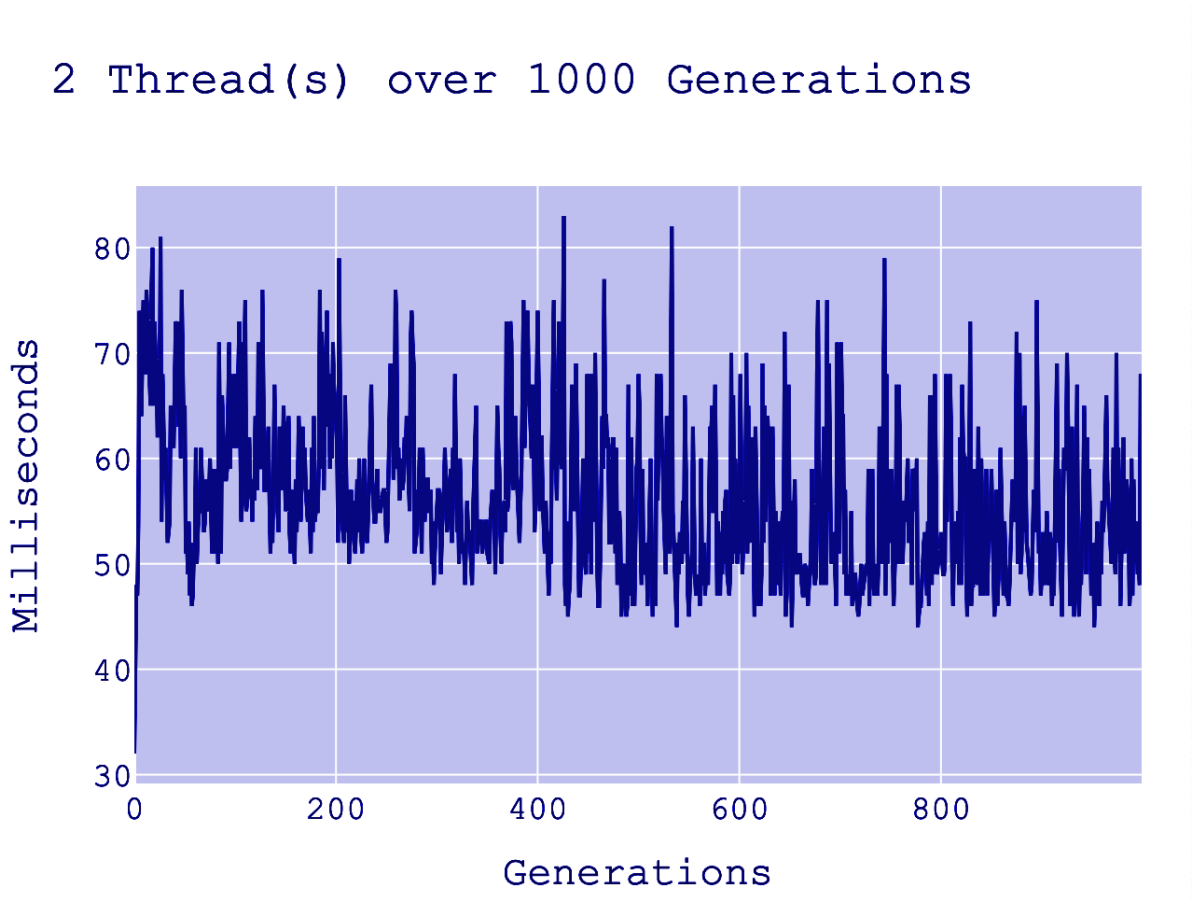
```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Address sizes:         39 bits physical, 48 bits virtual
Byte Order:            Little Endian
CPU(s):                8
On-line CPU(s) list:   0-7
Vendor ID:             GenuineIntel
Model name:            11th Gen Intel(R) Core(TM) i5-1155G7 @ 2.50GHz
  CPU family:          6
  Model:               140
  Thread(s) per core:  2
  Core(s) per socket:  4
  Socket(s):           1
  Stepping:            2
  CPU max MHz:         4500.0000
  CPU min MHz:         400.0000
  BogoMIPS:            4992.00
Caches (sum of all):
L1d:                   192 KiB (4 instances)
L1i:                   128 KiB (4 instances)
L2:                    5 MiB (4 instances)
L3:                    8 MiB (1 instance)
```

Тестове

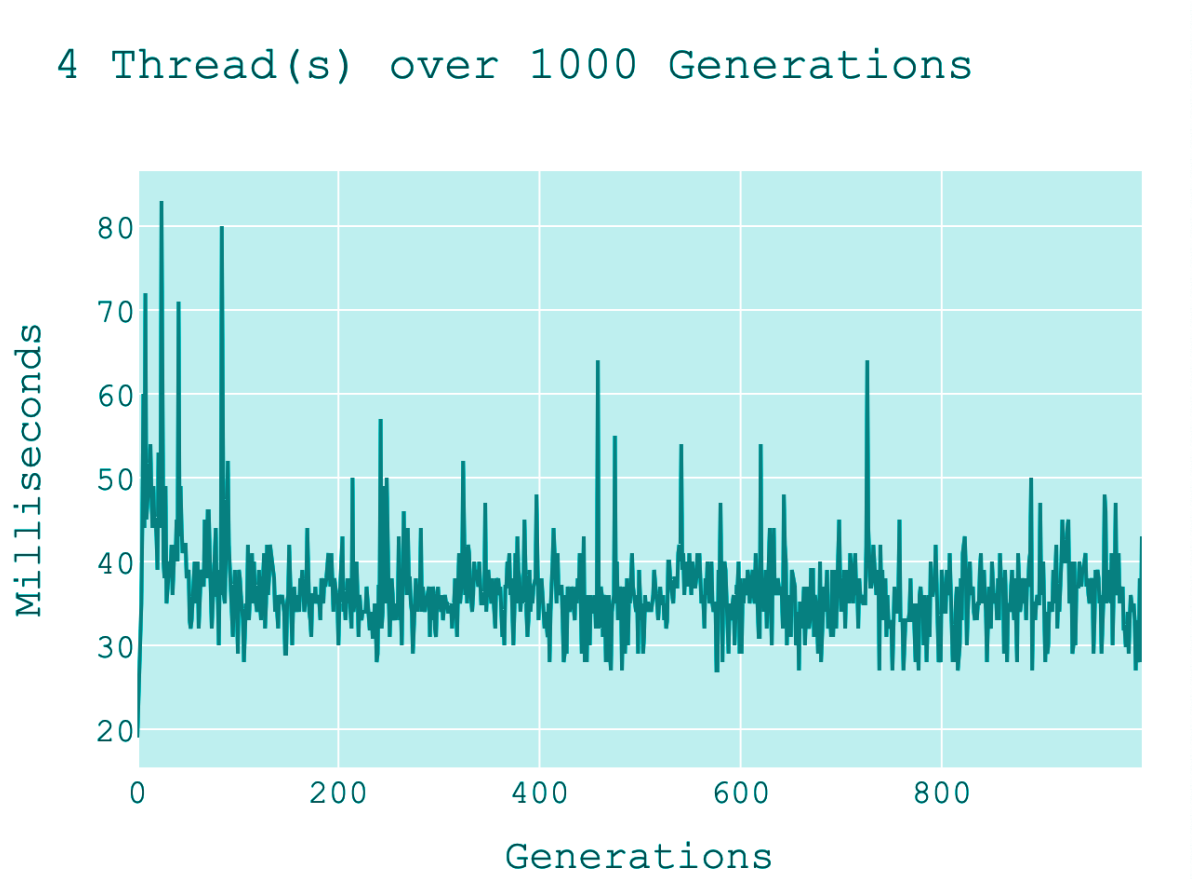
Всяка от следващите диаграмите записват времетраенето в милисекунди (по ординатата) за една генерация на играта (по абсицасата) за различен брой нишки.



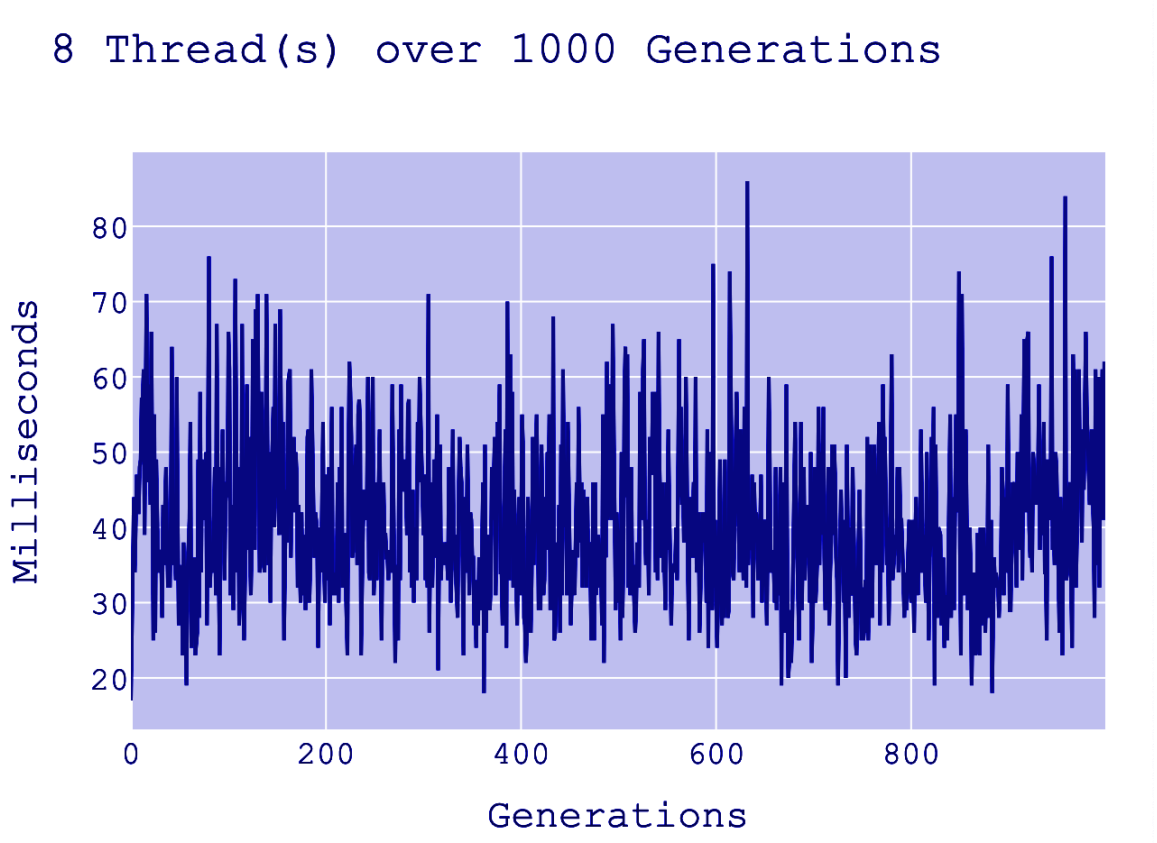
Можем да забележим, че между 300 и 500 генерация има значително забързване - вероятно, защото рибите са били почти изчезнали. Като най-високата точка е към 140.


2 Thread(s) over 1000 Generations
Milliseconds
80
70
60
50
40
30
0
200
400
600
800
Generations

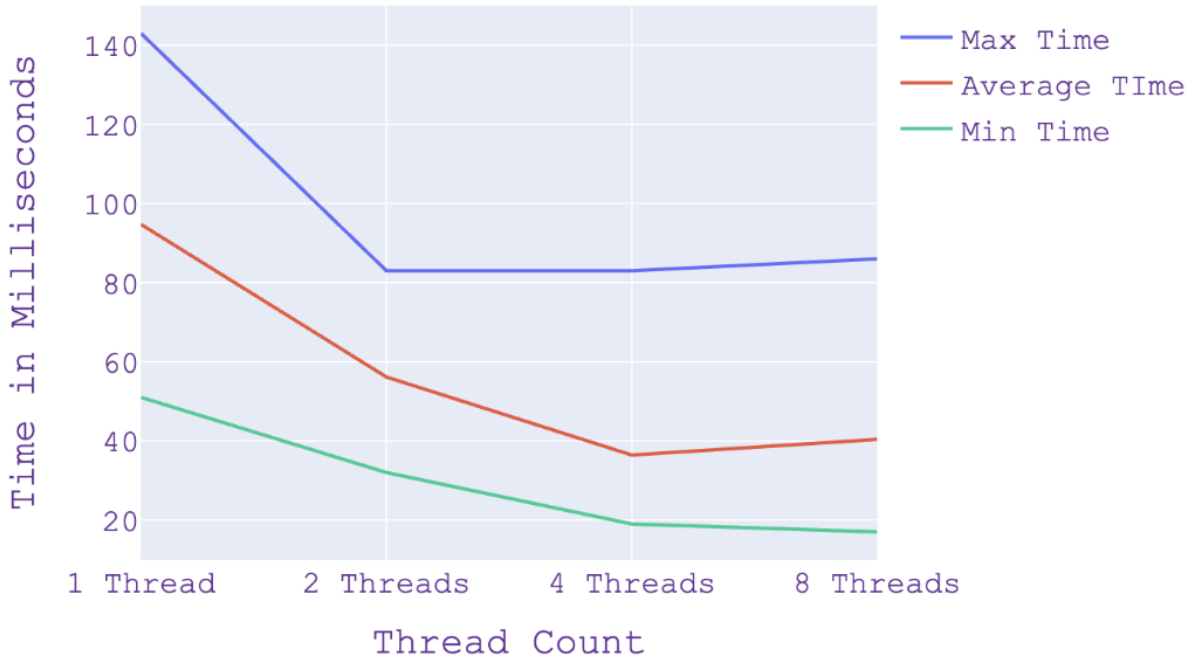
Най-високата точка е леко над 80, което е с около 50 милисекунди по-бързо от с 1 нишка.



Лесно се забелязва, че има изключително забързване в средното време сравнение с миналите 2 диаграми. Само че понякога има резки подскоци до 80, което вероятно се дължи на увеличение на заключването на мютекси около границата на полето.



Забелязваме, че 8 нишки вече са твърде много за нашата система и има забане спрямо с 4 нишки. Въпреки това се справя по-добре от еднонишкото.



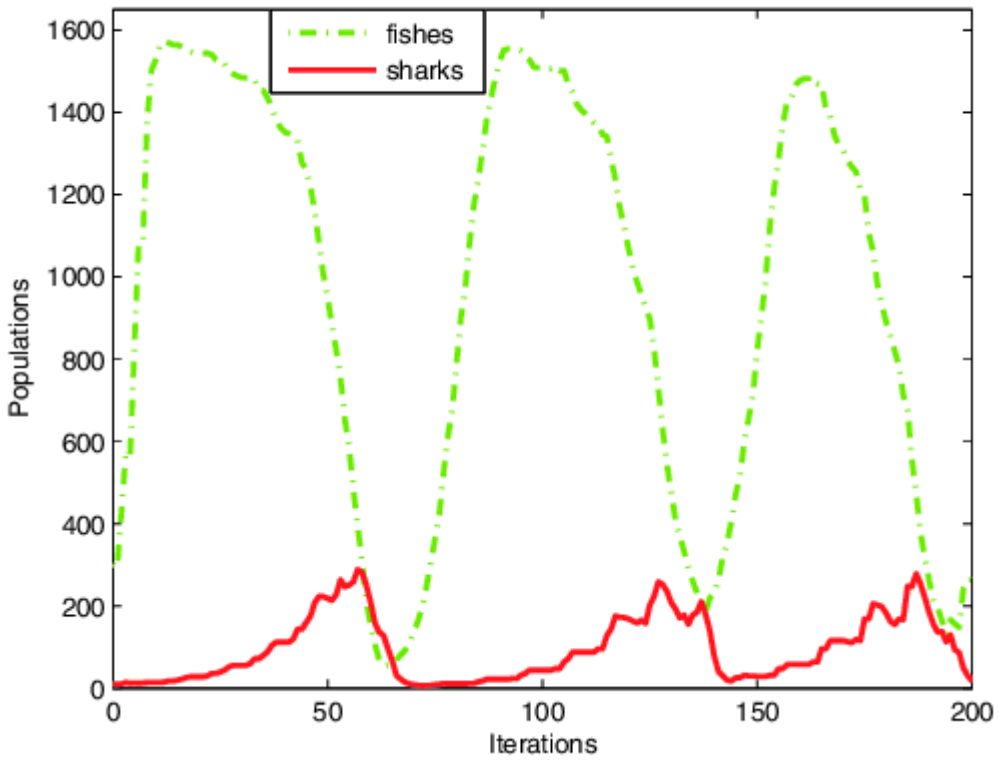
Максималното време спада значително при появата на над една нишка, но не се виждат повече подобрения за повече нишки.

Средното време спада постепенно при повече нишки, но при достигането на 8 се качва леко.

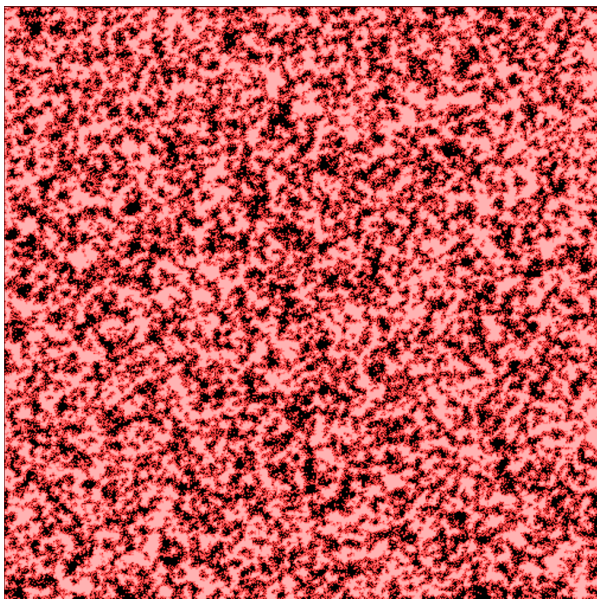
Интересно е, че минималното време се смъква допълнително при 8 нишки, тоест следва, че в някой случаи е по-оптимално да имаме 8 нишки. Въпреки това, средното време и консистентността е от най-голямо значение и затова бихме избрали да ползваме 4 нишки.

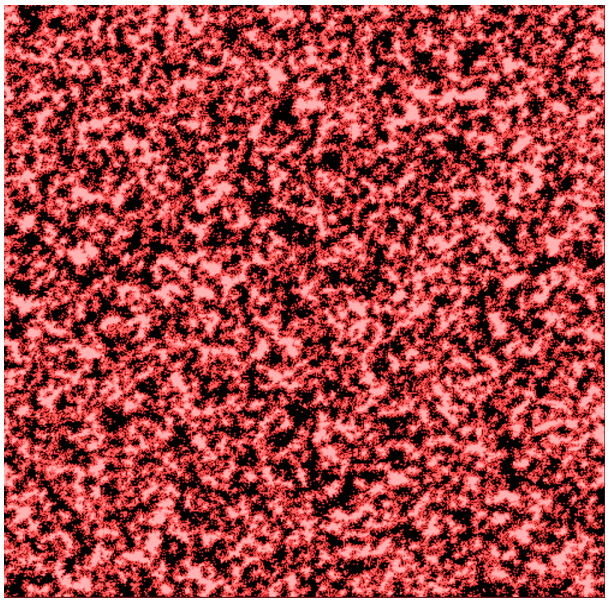
Допълнително, използването на rand() използва мютекси, които забавят значително смятането на генерация. Сменяйки го с rand48(), тъй като не използва мютекси, ускори значително многонишковото времетраене. Преди това еднонишкото беше бързо, колкото многонишковото.

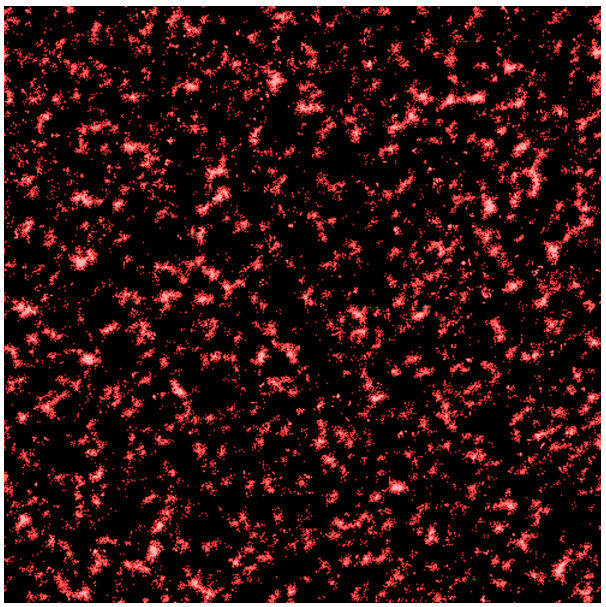
Още нещо интересно (от интернет):

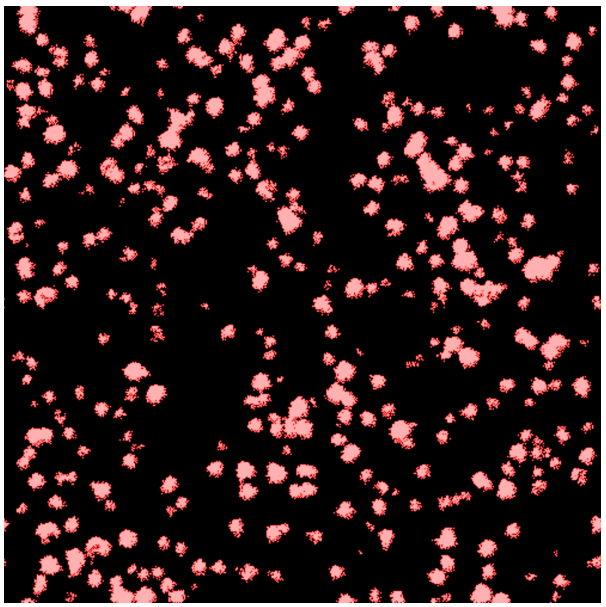


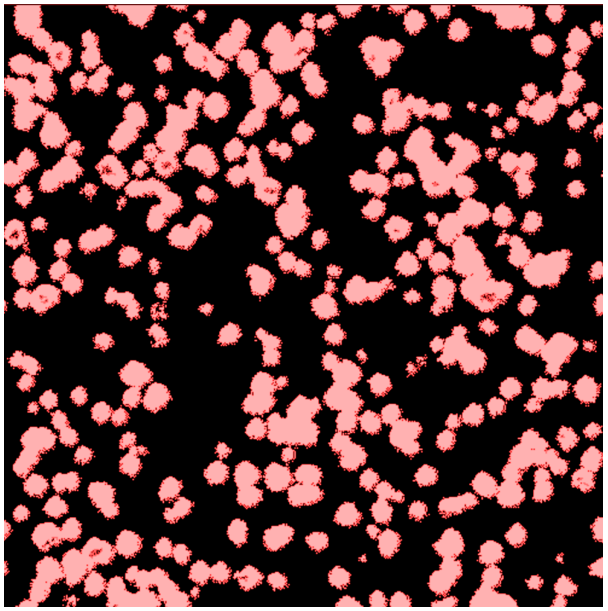
Визуализация

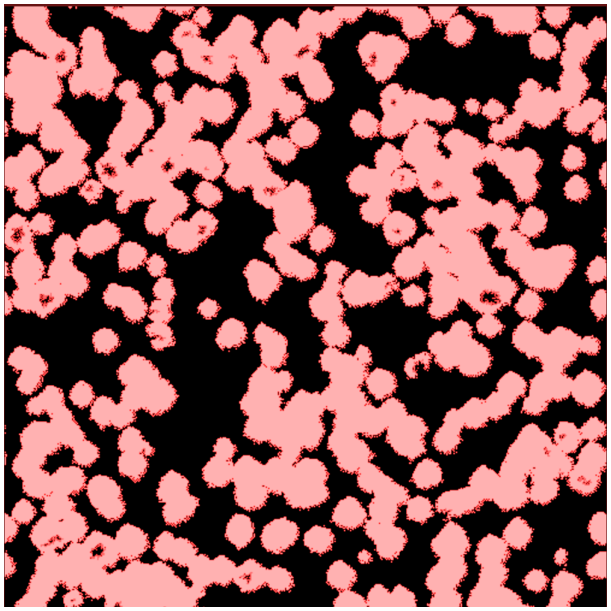












Литература

- https://en.wikipedia.org/wiki/Wa-Tor
- https://en.cppreference.com/w/cpp/thread/mutex
- https://www.sfml-dev.org/
- https://www.gnu.org/software/make/