Софийски университет „Св. Климент Охридски”
Факултет по математика и информатика

Предмет: Системи за паралелна обработка
Летен семестър, 2022/2023 год.

Курсов проект

Тема: Wa-Tor

Проверка на ефективността на CPU pinning и NUMA aware memory allocation върху NUMA базирани процесорни архитектури.

Изготвил:
Пламен Венелинов Динев, 82155
Компютърни науки

Ръководители:
проф. д-р Васил Цунижев
ас. Христо Христов

1. Анотация

Целта на настоящия проект е да анализира поведението на NUMA архитектурите и техниките за оптимизация на програмите, работещи върху тях с цел максимизиране на производителността. За целта ще разгледаме поведението на две техники за оптимизация върху имплементация на симулационната популационна динамика „Wa-Tor“. Избраните техники са CPU pinning и NUMA aware алокация на памет, които позволяват разпределението на памет близо до обработващата я нишка.

CPU pinning (или processor affinity) е опция на ядрото, по-точно scheduler - а, която оказва множество от ядра (хардуерни нишки), само на които е позволено дадена (софтуерна) нишка да работи.

NUMA aware memory allocation също е опция на ядрото, което позволява по време на алокация на памет, тя да бъде алокирана точно на определен NUMA node.

Двете техники вървят ръка за ръка, защото те позволят паметта и инструкциите за работа с нея да се намират на точно определен NUMA node.

В примерните океани представени в този документ, белите квадратчета означават, че квадратът е празен (има само вода), сините - риба, а червените - акула.

2. Описание на играта

Wa-Tor е симулация, която симулира популациите на риби и акули на планетата Wa-Tor, която има формата на тор (поничка с дупка). Планетата е съставена само от океан и е разделена на квадрати, като за улеснение можем да я представим разгъната като правоъгълник чийто ляв и десен, както и горен и долен край са свързани.

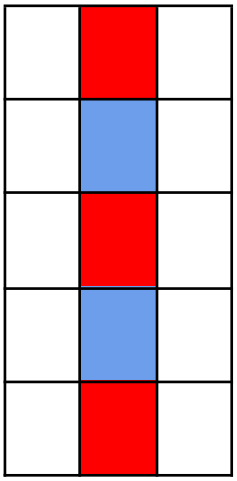
Океанът съдържа неограничено количество планктон, с който рибата се храни, а акулите - с риба. В даден момент, едно квадратче може само да е: свободно, заето от риба или заето от акули. Времето на планетата, също, тече на дискретни интервали: хронони, понякога наричани в този документ итерации.

На всеки хронон рибата се премества на случайно свободно съседно квадратче, ако има такова, като ако оцелее достатъчно дълго се размножава.

Поведението на акулите е подобно, с малката разлика че те са длъжни, ако има риба на съседно квадратче да се преместят там и да я изядат, и умират ако не са яли храна прекалено дълго.

3. Анализ

Като за начало, една важна забележка е, че редът в който съществата (акули и риби) биват придвижвани (симулирани) е от съществено значение за това какво ще ни е разпределението на съществата върху картата след един или няколко хронона. Това се вижда в следния пример:



Ако преместим нещо различно от акулата в най-горната клетка, то подреждането което ще получим в края на хронона ще е различно.

Ако приемем симулация за коректна, тогава когато съществата биват придвижвани в реда от горе надолу, от ляво надясно, то това прави задачата на практика непаралелизуема, защото това би изисквало паралелното построяване на дърво на зависимости и впоследствие паралелната обработка върху това дърво, което е доста неефективно имайки предвид, че придвижването на едно същество е особено проста задача от изчислителна гледна точка.

Затова ще приемем симулацията за коректна, ако за един хронон всички същества са се придвижили точно веднъж (стига да е имало свободна клетка), без значение от реда в който са били придвижени.

Wa-Tor спада към групата на асинхронни клетъчни автомати (asynchronous cellular automata). Такъв тип задачи може да бъде ефективно изчисляван в паралел, чрез разделяне на поддомейни (Domain Decomposition) [1]. Възможни са два типа разделяне на поддомейни:

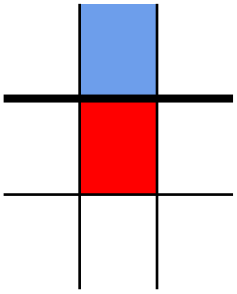
- **Динамично разделяне на поддомейни**

  Този подход преизчислява разделянето на домейни по време на симулацията с цел да изравни сложността на изчислението на всеки поддомейн.

- **Статично разделяне на поддомейни**

При този подход разделянето на поддомейни се извършва само веднъж.

Друг фактор при разделянето на поддомейни е броят преки съседи на всеки домейн. Това разделение изисква комуникация/синхронизация между отделните поддомейни за четене от или писане върху чужда за домейна част от океана. Четене се налага, за да се провери състоянието на клетка, съседна до клетката на същество, което се намира между два домейна.



Писането се налага, когато същество сменя домейна в който се намира. Намаляйки броят на съседите на всеки поддомейн се намалява и свръхтоварът породен от нуждата за синхронизация/комуникация между поддомейните.

Имайки предвид и, че разпределението на рибите и акулите е приблизително равномерно, избраната имплементация използва статично разделяне на поддомейни и по-точно хоризонтални линии (правоъгълници). Първо, този подход минимизира броят на съседни поддомейни на всеки поддомейн, до точно два. После, така разположени клетките на линиите са последователни в паметта, което съчетано с прочита напред (read ahead) на процесорите подобрява производителността. Броят поддомейни е равен на броят нишки умножен по грануларността.

Различните имплементации се различават и по това как се извършва синхронизацията между отделните поддомейни:

- **Комуникация чрез предаване на съобщения (message passing)**

  Отделните нишки си предават информация за състоянието на клетките, които се намират на границата с други домейни. [1] Заедно с това, всяко съобщение носи и информация за итерацията на която се намира изпращача на съобщението. Ако нишка получи съобщение за писание с номер на итерация по-малък от итерацията на която се намира в момента, то тя е длъжна да се върне назад (да направи rollback) до итерацията която е в съобщението.[2] Предимството на този подход е, че той е приложим в дистрибутирани системи (системи, при които имаме много компютри, които си комуникират в мрежа) и може да бъде комбиниран с някой от другите методи.

- **Синхронизация със семафори и споделена памет**

  Всеки поддомейн има два семафора (или reader writer mutex) съответно за горната и долната граница. По време на симулация, една нишка може да заключи семафора на своя домейн или на съседен домейн. Важна забележка е, че трябва да се осигури синхронизация по броя итерации извършени от всяка нишка. Имайки предвид, че комуникацията със споделена памет е по-ефективна от предаване на съобщения (message passing) то е очаквано този метод да е по-ефективен от предишния.

- **Избягване на синхронизацията между отделните нишки**

  Умножаваме поддомейните на две и на всяка нишка получава два пъти повече поддомейни. Разделяме домейните на четни и нечетни, както и всяка итерация я разделяме на две „полуитерации“. На всяка полуитерация симулираме океана само за нечетните или четните линии, като редуваме четни и нечетни. След като бъдат пуснати нишки за симулация на едната четност ленти от океана, трябва да бъдат изчакани всички да приключат след това могат да бъдат пуснати нишки за другата четност. Ако всяка лента е с височина поне две клетки, това ни позволява да нямаме необходимостта от каквато и да е комуникация между отделните нишки, защото те не споделят памет по време на изпълнението си. Единствената синхронизация е, когато трябва да се изчакат всички нишки да бъдат готови в края на всяка полуитерация. Това ни гарантира, че всяка нишка заспива точно веднъж, когато е готова с работата си, или два пъти за цяла итерация. При предишния метод имаме поне едно заспиване при синхронизация на броят итерации.

Имайки предвид, че разпределението на рибите и акулите в океана е приблизително равномерно, може да се каже че броят на същества във всяка от линиите е приблизително равен, което обезсмисля използването на динамично балансиране.

Също, имайки предвид равномерното разпределение на съществата в океана, може с голяма точност да си гарантираме най-оптимални резултати при едра или средна грануларност. Разработената имплементация поддържа само едра грануларност, но това е достатъчно за целите на проекта.

Една възможна оптимизация е, да не се обхождат всички клетки на картата, а да се пази само или допълнително масив или свързан списък с позициите на риби и акули и по този начин не се обхождат безсмислено клетки в които няма нито риби, нито акули. Практически тестове показват обаче, че освен ако не изчезнат рибите и/или акулите, то след като популациите се стабилизират, то запълването на океана в 90% от времето е над 50% и е приблизително равномерно. Също, важна забележка е, че имплементацията която ще покажем тук, използва един

байт за съхранението на една клетка. Имайки всичко това предвид, нека да разгледаме възможните начини с техните недостатъци:

- **Да се съхраняват само масиви за всеки ред от океана с x кординатата на всяко същество в сортиран вид.**

  Този начин има максимална сложност O(w), където w е ширината на океана, за местене на същество, което е неприемливо.

- **Да се съхраняват допълнително масиви за всеки ред от океана с x кординатата на всяко същество в несортиран вид.**

  Ако трябваше да съхраняваме само тези масиви, то тогава проверката ни за свободна клетка щеше да е със сложност O(w). Затова трябва към картата да се съхраняват допълнително тези масиви. Дори да се използват само два байта за съхранението на координата (което ще огранизи ширината на океана), то взимайки предвид наблюденията, използването на памет ще ни се удвои, като тази памет ще се използва често за писане и за четене.

- **Да се съхраняват само едностранно свързани списъци за всеки ред от океана с x кординатата на всяко същество в сортиран вид. [1]**

  Всеки връх на свързания списък съхранява в себе си адрес който е поне четири байта, обикновено е осем байта. Дори и да се използва оптимизиран едностранно свързан списък, който си преалокира върховете и използва релативни адреси от два байта, заедно с двата байта за координата, стават общо четири байта. Това също ще означава използването на двойно повече памет. Освен това трябва да се вземе под внимание и че свързаните списъци са изключително cache unfriendly.

Имайки предвид, че RAM паметта е най-бавното нещо от една процесорна система, както и че допълнителното използване на памет ще увеличи използването на cache, може да хипотезираме, че който и метод на оптимизация да използваме, за достатъчно големи размери на океана, ще доведе до регресия.

4. Проектиране

Основната цел на този проект е да сравни производителността на две имплементации на „Wa-Tor“, една която взима под внимание NUMA архитектурата и една оптимизирана само за UMA архитектури. Ако имаме NUMA aware имплементация, лесно можем да получим UMA оптимизирана имплементация, като премахнем всички инструкции за CPU pinning и NUMA aware memory allocation. На тази идея се базира и текущият проект.

В началото, класът ExecutionPlanner получава информация за изискванията от потребителя (на колко ядра да се извършва симулацията и дали да се използват ядра които са hyperthreaded), получава информация за процесорната архитектура на системата (NUMA

node - ове и ядра които са разположени върху тях) и генерира списък с ядра които ще се използват за симулацията.

След това се генерира океана от клас WaTor::Map и се разделя на няколко поддомейна за всеки NUMA node, после поддомейните се разделят на още няколко поддомейни за всяко ядро от NUMA node - а, и накрая - се разделят на още два поддомейна - четен и нечетен. Алокацията на даден поддомейн (линия) се прави на NUMA node - който в последствие ще я обработва.

Особеност на имплементацията е, че една клетка на океана се съхранява в един байт, чрез използването на bit-field. Това се прави от класа WaTor::Tile. Един байт се разделя на два bit-field - а: age, lastAte. Ако и двете са нули, то това празна клетка, само с океан. Ако само lastAte е нула, то тогава в текущата клетка е риба, която е на възраст age-1. Възрастта се използва само за проверка за размножаване и заради това не е нужно да е точна. Ако и двете полета са ненулеви, то това е акула на възраст age-1, която е яла преди lastAte-1 хронона. Това позволява по-ефективно използване на кеша и readahead и по-висока производителност. Недостатъкът е, че в такъв случай, максималното време за размножаване на рибите и акулите и максималното време за смърт от глад на акулите е 14 включително.

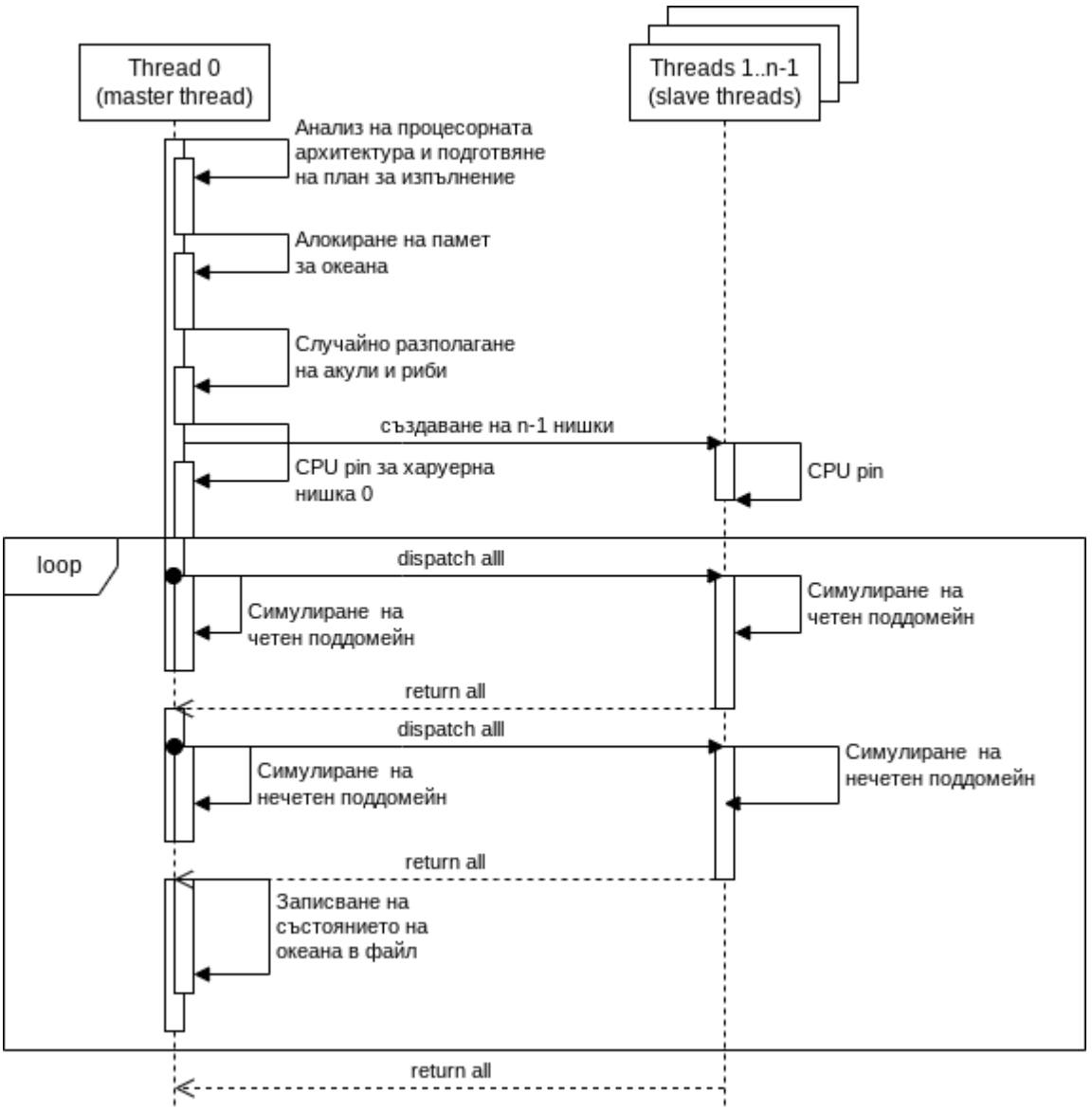
NUMA node	Hardware thread	Номер на поддомейн	Колона		
			0	1	2
0	0	0			
		1			
	1	2			
		3			
1	2	4			
		5			
	3	6			
		7			

Поставят се по случаен начин риби и акули (броят им зависи от аргументите, които са били подадени към програмата).

Накрая имаме master-slave архитектура с една master нишка и n-1 slave нишки. Първо четните поддомейни се симулират в паралел, а след като са приключили всички, се изпълняват нечетните поддомейни в паралел.

Имплементацията е коректна, спрямо дефиницията дадена по-горе, т. е. за един хронон всяко същество се мести точно веднъж. Това се гарантира от три битови маски, с дължина равна на широчината на океана, които попречват на дадено същество да бъде преместено. Една битова маска, която попречва на същество да бъде преместено повторно по време на обхождането, ако първоначално е преместено надясно или надолу. Както и още две, които попречват на същество да се премести повторно, ако съществото е дошло от горен или долен поддомейн.

След изпълнението на един хронон, състоянието на картата се записва в файл, където за всяка клетка отговарят 2 бита. Това се прави дори и потребителя да не иска файл с историята на симулацията, тогава просто файлът в който се записва е /dev/null, за да се попречи на компилатора, оптимизирайки да премахне част от симулацията. Друга характерна особеност е, че имплементацията не стартира всеки път нишка, която впоследствие join - ва, а стартира веднъж всички нишки и впоследствие всяка нишка чака с condition variable работа. Накрая на програмата се извеждат информация за времената и средната тактова честота на процесора.



5. Тестови резултати

Тестовата система е 2 x Intel Xeon E5-2660 v3 с 2 x 8 ядра и 2 x 16 нишки.

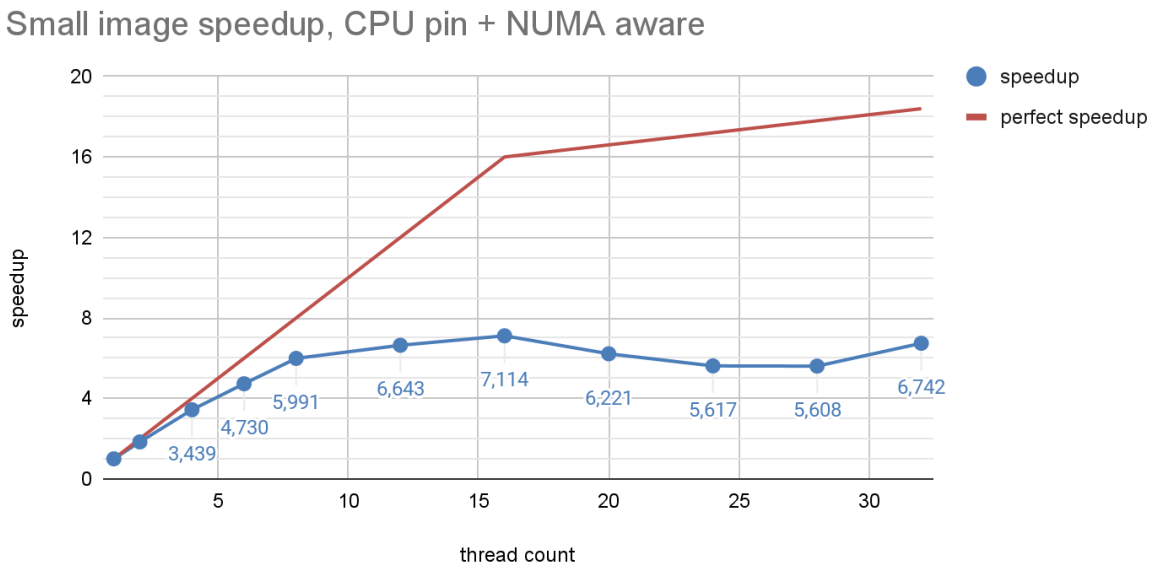
Параметрите на симулацията при тестване са:

- Малък океан: 800x400, 5000 хронона
- Голям океан: 4000x2000, 500 хронона
- Много голям океан: 32000x16000, 200 хронона

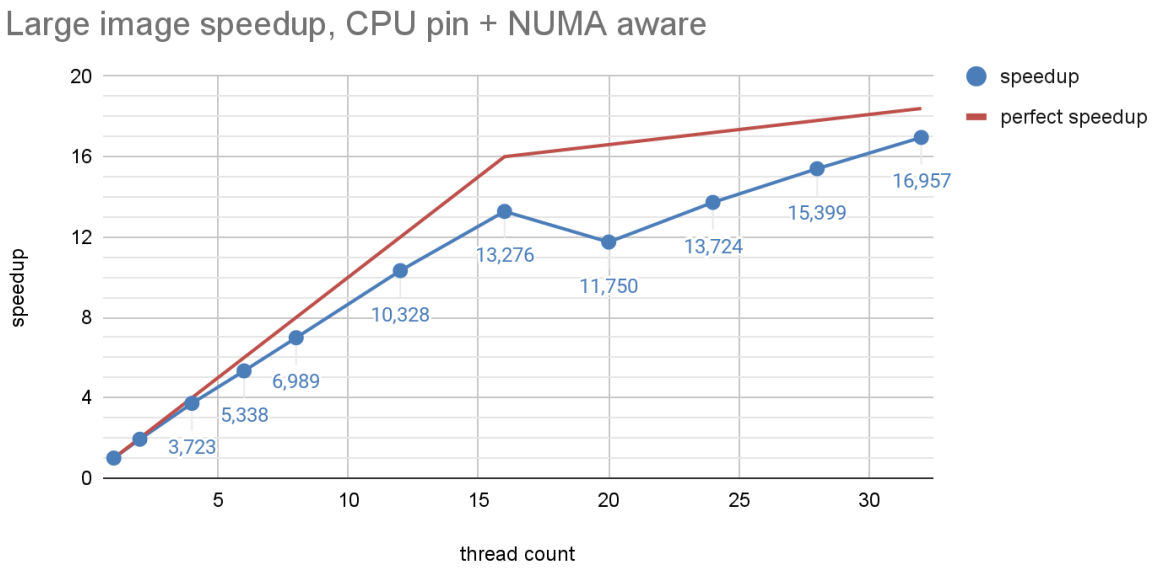
За всички тестове останалите параметри са: за 3 хронона се размножават рибите, за 10 хронона се размножават акулите и за 3 хронона акулите умират от глад.

Забележка 1: При системите с hyperthreading няма как да бъде определено точно ускорението. Затова в конкретните опити, при смятането на перфектно ускорение, hyperthread - ната нишка се брои за 0,15 ядро.

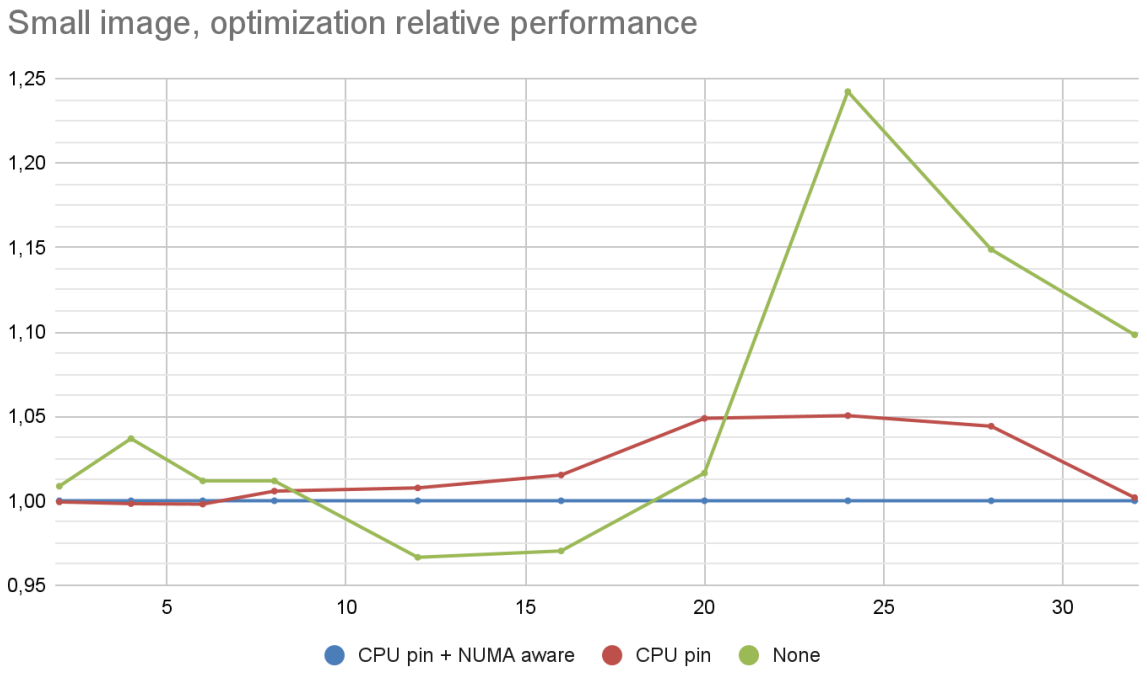
- **Графика на ускорение, малък океан, cpu pin + NUMA allocate**



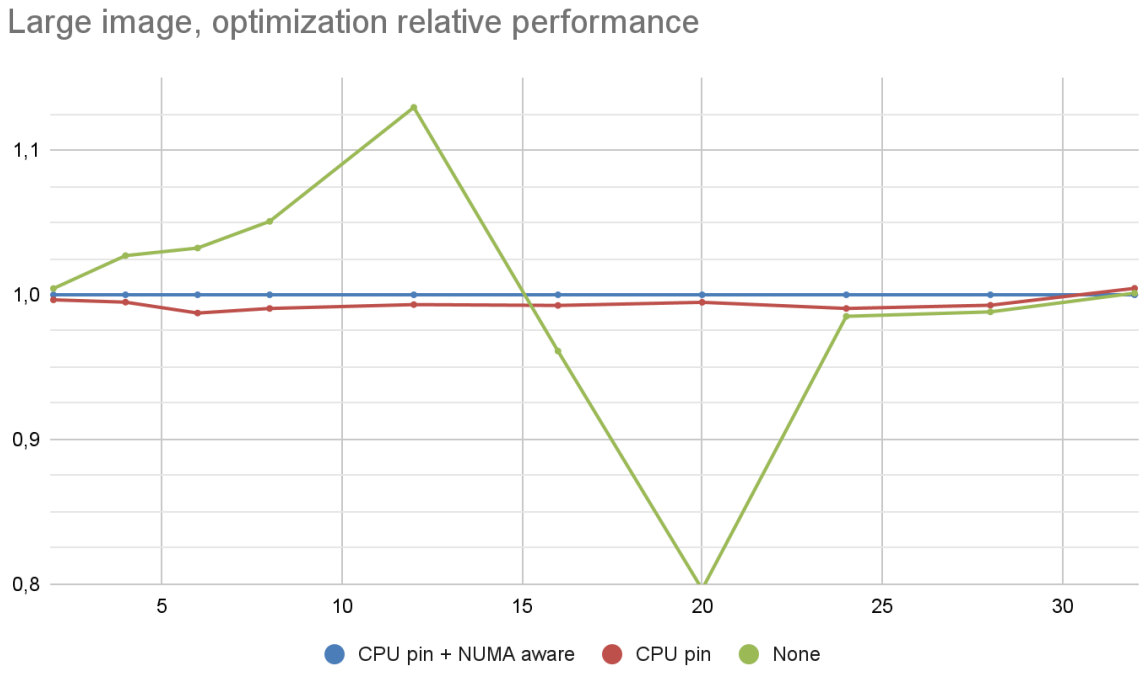
- **Графика на ускорение, голям океан, cpu pin + NUMA allocate**



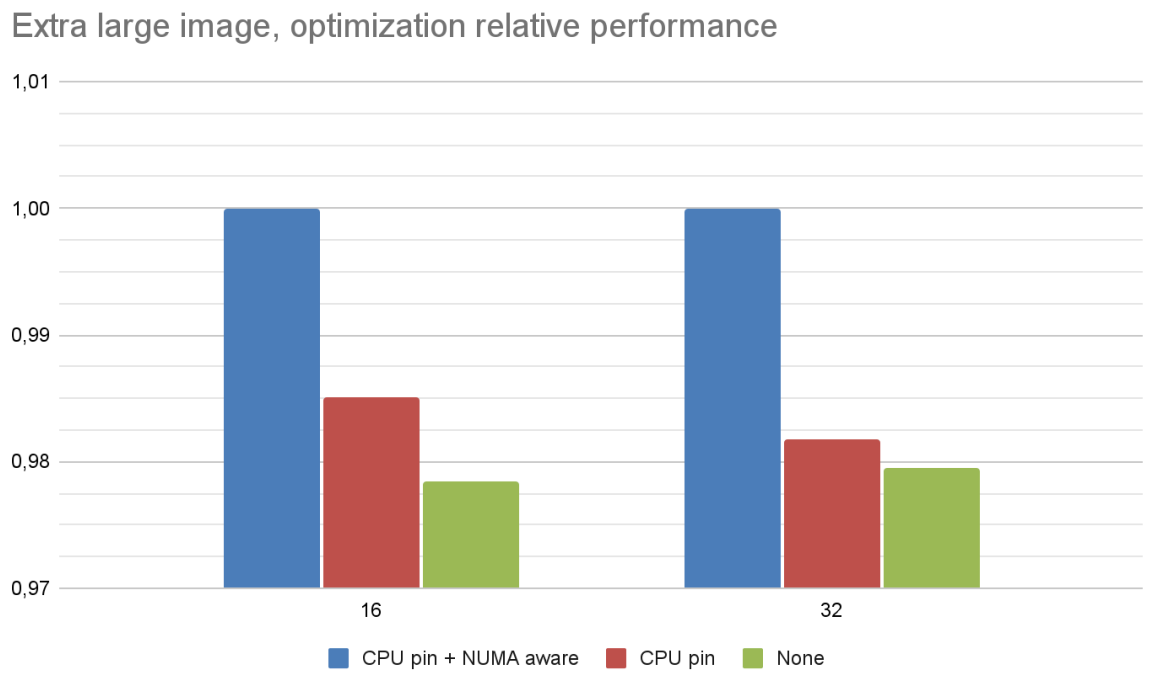
- **Графика на релативна скорост, малък океан**



- **Графика на релативна скорост, голям океан**



- **Графика на релативна скорост, много голям океан**



- **Коментар по резултатите**

  Ефективността на cpu pin и numa allocation е малка. Наблюдава се <1% ускорение в по-голямата част от резултатите, а и в немалка част от резултатите се наблюдава и деградация. За много големи океани се вижда оптимизация малко над 2%. Причините за това са всевъзможни, но някои от тях са: комуникацията между NUMA node - овете е достатъчно бърза, по време на оптимизация, компилатора генерира инструкции, които биха достъпили памет на чужд node и/или просто бъг в имплементацията. Четвърта възможна причина е, че цялата карта за даден NUMA node се побира в L3 cache - а на самият NUMA node и така RAM почти не се използва по време на симулацията, но това не обяснява защо при много големи океани подобрението е малко над 2%.

- **Таблица с тестови резултати:**

  Таблицата с тестовите резултати може да бъде намерена тук:

  https://docs.google.com/spreadsheets/d/1o0KnSR2gPHebNcg5sukDk1G9vdSQslt9UI00CE6uhL8/edit?usp=sharing

- **Github с имплементацията:**

  https://github.com/pacodinev/parwator/

  Към момента, последната версия се намира на клон refactor.

6. Последващо развитие:

Допълнителен анализ за регресии с numaprof.

Да се добави поддръжката на други грануларности, освен 1.

7. Използвани източници:

- [1] F. Baiardi, A. Bonotti, L. Ferrucci, L. Ricci, and P. Mori, „Load balancing by domain decomposition: the bounded neighbour approach“, January 2003, (https://www.researchgate.net/publication/236854644_Load_balancing_by_domain_decomposition_the_bounded_neighbour_approach)
- [2] Benno J. Overeinder, Peter M. A. Sloot, „Application of time warp to parallel simulations with asynchronous cellular automata“, European simulation symposium 1993, pp. 397-402, (https://www.researchgate.net/publication/2272430_Application_Of_Time_Warp_To_Parallel_Simulations_With_Asynchronous_Cellular_Automata)
- [3] A. K. Dewdney, „Sharks and fish Wage an ecological War on the toroidal planet Wa-Tor“, Scientific American. pp. 14-22, December 1984, (http://cs.gettysburg.edu/~tneller/cs107/wator_dewdney.pdf)