Трафик

17 март 2006 г.

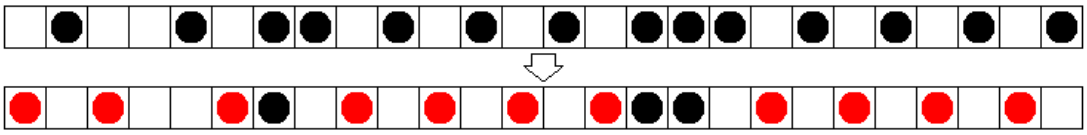
Целта на упражнението е да се разработи паралелен алгоритъм, който решава силно опростен модел на автомобилен трафик по тясна еднопосочна улица.

1. Модел

Разглеждаме изключително опростен модел на улично движение. Еднопосочното улично платно представяме като набор от клетки, всяка от които с дължина, равна на средната дължина на пътните превозни средства. Всяка клетка може да бъде свободна или заета от не повече от едно превозно средство, както е показано на следната диаграма:

Фиг. 1. Начално състояние на "уличното платно"

На всяка стъпка превозните средства се придвижват с една клетка надясно, но само ако съседната клетка е вече свободна:



Фиг. 2. Стъпка от симулацията. Автомобилите, които са се преместили, са означени в червено. Забележете как най-десният автомобил преминава в най-лявата клетка поради периодичните гранични условия.

Считаме, че двете крайни клетки са съседни помежду си, т.е. улицата представлява разгърнат модел на затворен път (напр. автомобилна писта). Така дефиниран

моделът може да се използва за грубо описание на движението по регулирана улица, където автомобилите се движат с приблизително еднаква скорост.

Дефинираме *скорост на придвижване* v като отношение на броя на преместилите се автомобили към общия брой на автомобилите в симулацията:

$$v = \frac{n}{N_{auto}}$$

Първоначалната конфигурация генерираме по случаен начин, като единствен входен параметър е *коефициентът на задръстване* p, дефиниран като отношението на броя на автомобилите към броя на клетките в улицата:

$$p = \frac{N_{auto}}{N_{str}}$$

При така дефинираните две числени характеристики на модела можем да изследваме поведението на скорстта на придвижване като функция от коефициента на задръстване.

2. Алгоритмична реализация

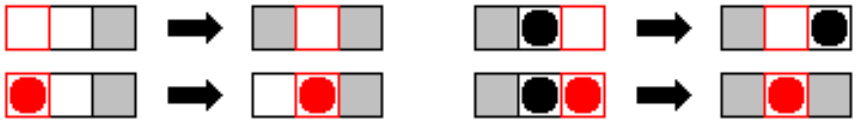
Описаният модел се реализира тривиално алгоритмично като клетъчен автомат. Достатъчно е да забележим следното просто правило:

Ако на дадена стъпка клетка a_i е празна, то състоянието ѝ на следващата стъпка се определя само от състоянието на клетка a_{i-1}. Ако клетка a_i е пълна, то състоянието ѝ на следващата стъпка се определя само от състоянието на клетка a_{i+1}.

Правилото е:

$$a_i(t+1) = \begin{cases} a_{i-1}(t), & \text{ако } a_i(t) = 0 \\ a_{i+1}(t), & \text{ако } a_i(t) = 1 \end{cases}$$

и е илюстрирано на следната диаграма:



Фиг. 3. Придвижването на автомобилите като правила на клетъчен автомат. Съответствието на клетките преди и след стъпката е показано в червено.

Всички премествания стават едновременно, и затова се налага използването на временен масив, в който да се съхрани новото състояние на автомата след прилагане на стъпката.

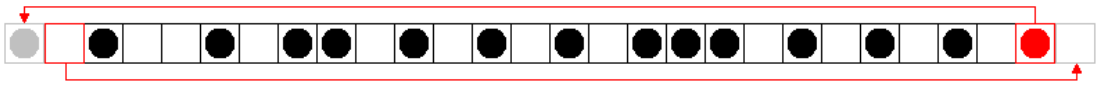
2.1. Сериен алгоритъм

Серийната реализация на алгоритъма може да се опише с псевдокода Алгоритъм 1.

Algorithm 1 Сериен алгоритъм.

```
запълване на street[1..N]
do
  street(0) := street(N)
  street(N+1) := street(1)
  v := 0
  do i := 1 to N
    if street(i) = 0 then
      newstreet(i) := street(i-1)
    else
      newstreet(i) := street(i+1)
    end if
    if newstreet(i) <> street(i) then
      v := v + 1
    end if
  end do
  street := newstreet
  v := v / (2*N)
end do
```

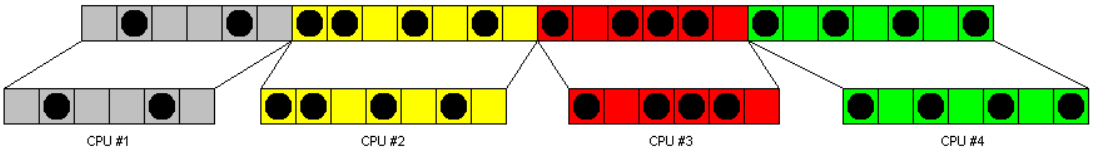
За да избегнем специална обработка на първата и последната клетка в масива, както и за да реализираме периодичните гранични условия, разширяваме с по една клетка масива *street* от двете страни. Преди извършване на стъпката последният реален елемент от *street* се копира в клетката преди първия реален елемент и обратно. Ако не желаем граничните условия да са периодични (симулация на сегмент от отворена улица), просто инициализираме двата допълнителни елемента по подходящ начин на всяка стъпка. Допълнителните елементи се наричат *хало*.



Фиг. 4. Хало. Допълнителните елементи са показани в сиво, а елементите, които се копират - в червено. Стрелките указват посоката на копирането.

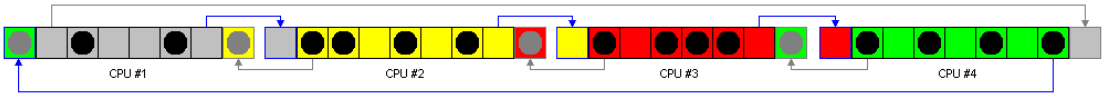
2.2. Паралелен алгоритъм

Паралелизацията на алгортъма се постига чрез разделяне на масива, представляващ улицата, на няколко подмасива, като всеки подмасив се обработва от различен процесор:



Фиг. 5. Разпределяне на масива между няколко процесора. Подмасивите, които отделните процесори обработват, са оцветени в различен цвят.

Тъй като за пресмятане на състоянието на всяка клетка е необходимо да се познава състоянието на съседите ѝ отляво и отдясно, то всеки процесор трябва да се информира за състоянието на крайните клетки на съседните му процесори. Отново прибягваме до механизма на халото:



Фиг. 6. При паралелната версия всеки подмасив има собствено хало. Хало елементите се обновяват в началото на всяка стъпка на автомата посредством *размяна на халота*.

На фигура 6 е показана операцията *размяна на халота* - изключително базова операция при по-голямата част от паралелните алгоритми с декомпозиция на проблемната област. Операцията се извършва на две стъпки:

1. Предаване на необходимата информация на съседа вдясно и получаване на информацията от съседа вляво. Стъпката се нарича *изместване вдясно*. На фигурата е показано в син цвят.
2. Предаване на необходимата информация на съседа вляво и получаване на информацията от съседа вдясно. Стъпката се нарича *изместване вляво*. На фигурата е показана в сив цвят.

Всяка стъпка включва една операция по изпращане на данни и една операция по получаване. Тъй като всички процеси извършват изместването по един и същи начин (първо изпращат и след това очакват данни или обратно), то можем да се подходи по два начина:

1. Главният процес (напр. процесът с ранк 0) първо изпълнява операцията по изпращане към десния му съсед, след което минава в режим на очакване на данни от левия посредством операция по получаване. Всички останали процеси първо очакват данните от левия си съсед, след което изпращат данни на десния си съсед. Тази схема е изключително неефективна, тъй като води до блокиране на главния процес в режим на изчакване докато обмяната на информацията между всеки два съседни процеса се разпространи като щафета надясно, чак до последния процес. Ще наречем този метод *синхронно*

Algorithm 2 Синхронно изместване вдясно

if $rank = 0$ **then**
  send($lstreet(N_0)$, $nrank$)
  receive($lstreet(0)$, $prank$)
else
  receive($lstreet(0)$, $prank$)
  send($lstreet(N_{rank})$, $nrank$)
end if

Algorithm 3 Асинхронно изместване вдясно

ireceive($lstreet(0)$, $prank$, $handle$)
send($lstreet(N_{rank})$, $nrank$)
wait($handle$, $status$)

изместване. Псевдокодът е показан в Алгоритъм 2. В алгоритъма се изплзват следните променливи:
$rank$ - ранк на текущия процес
$lstreet[1..N_i]$ - локалната част от масива $street$,която даденият процес обработва
N_i - брой на елементите в подмасива, който процесът с ранк i обработва
$nrank$ - ранк на процеса вдясно
$prank$ - ранк на процеса вляво

2. Всички процеси стартират неблокираща операция по приемане на информация от левите си съседи, след което започват блокиращо предаване на информация към десните си съседи. След приключване на предаването преминават в режим на изчакване до завършване на неблокиращото приемане. Методът ще наречем *асинхронно изместване.* Псевдокодът е показан в Алгоритъм 3. Използват се следните променливи:
$handle$ - манипулатор на неблокиращата операция
$status$ - статус на неблокиращата операция след приключването й

Независимо от избрания метод на изместване, същият се повтаря още веднъж и за операцията по изместване вляво.

2.3. Разпределяне на работата

Паралелното преместване на колите по улицата е само половината от решението на задачата. Всеки процес трябва първоначално да се снабди със своята част от масива, представляващ цялата улица, както и да предава, от време на време, на главния процес данните, необходими му да реконструира цялостната улица (напр. за да разпечатаме на изхода последователните стъпки с цел проверка на алгоритъма). Също така всеки процес следва да съобщи на главния и бройката на преместилите се в неговия подмасив автомобили, за да може да се формира сумарния брой преместени автомобили и да се изчисли скоростта v.

Algorithm 4 "Ръчно" разпределяне на работата и сумиране на броя преместили се автомобили

; първоначално разпращане на масива street на останалите процеси
if $rank = 0$ **then**
  do $srank := 1$ **to** $size$-1
    изчисляване на $begin_{srank}$ и end_{srank} за съответния ранк
    send($street(begin_{srank}{:}end_{srank})$, $srank$)
  end do
else
  receive($street(begin_{rank}{:}end_{rank})$, 0)
end if
.
.
; в края на всяка итерация улицата се сглобява от главния процес
$moved$:= брой преместили се коли в подмасива на процеса
if $rank = 0$ **then**
  do $srank := 1$ **to** $size$-1
    изчисляване на $begin_{srank}$ и end_{srank} за съответния ранк
    receive($street(begin_{srank}{:}end_{srank})$, $srank$)
    receive($smoved$, $srank$)
    $moved := moved + smoved$
  end do
else
  send($street(begin_{rank}{:}end_{rank})$, 0)
  send($moved$, 0)
end if

За целите на първото от двете, а имено комуникацията на масива с колите, може да се подходи по два различни начина. Първият от тях се реализира с псевдокода, показан в Алгоритъм 4.

С цел опростяване на програмата всеки процес притежава локално масив *street* с пълната големина на улицата (разширен с двата хало елемента), като използва само част от него за своите цели. *size* е големината на комуникатора, в който участват процесите (MPI_COMM_WORLD в случая). В алгоритъма е включено и изчисляването на пълния брой автомобили, които са се преместили.

Вторият, значително по-прост и ефикасен начин, се реализира със *scather/gather операции*. За целта всеки процес конструира вектор $offsets$ с отместванията в *street* на началата на подмасивите, които всеки процес следва да получи/предаде, както и вектор *lengths* с дължините на съответните подмасиви. Векторите включват и частта, която и главният процес следва да "получи"/"предаде" (говорейки си сам на себе си, поради което и кавичките). Псевдокодът е показан като Алгоритъм 5.

Изчисляването на общия брой преместени автомобили се прави най-ефикасно и най-елементарно от програмна гледна точка с примитивите на MPI за глобална

Algorithm 5 Разпределяне на работата с примитивите на MPI

do $srank := 0$ **to** $size$-1
  изчисляване на $begin_{srank}$ и end_{srank} за ранк $srank$
  $offsets(srank) := begin_{srank}$
  $lengths(srank) := end_{srank} - begin_{srank} + 1$
end do
; начално разпращане на подмасивите от процес с ранк 0
scatterv($street$, $street(begin_{rank}{:}end_{rank})$, $offsets$, $lengths$, 0)
.
.
; в края на всяка итерация улицата се събира в процес 0
gatherv($street$, $street(begin_{rank}{:}end_{rank})$, $offsets$, $lengths$, 0)

Algorithm 6 Сумиране с глобална редукция

$moved$:= брой преместили се коли в подмасива на процеса
reduce($totalmoved$, $moved$, MPI_SUM, 0)

редукция. В случая е необходим един единствен ред код, показан като Алгоритъм 6.

След прилагането му сумарния брой преместили се автомобили се намира в променливата $totalmoved$ в процес с ранк 0. Стойността на същата променлива в другите процеси не е дефинирана.

3. Задачи за упражнение

1. Реализирайте серийния Алгоритъм 1. Убедете се, че програмата работи правилно.
2. Реализирайте паралелна версия на програмата, използваща синхронна обмяна на халота и "ръчно" разпределяне на работата (алгоритми 2 и 4). Убедете се, че паралелната версия работи правилно. Сравнете резултатите със серийната версия при една и съща засявка на случайния генератор.
3. Променете паралелната версия да използва асинхронна обмяна на халота по Алгоритъм 3. Убедете се, че програмата работи правилно и сравнете с предишните две версии.
4. Използвайте механизмите на MPI за разпределяне на работата и за глобална редукция за да реализирате алгоритми 5 и 6. Сравнете резултатите с предишните версии.
5. Тествайте латентността на различните механизми за обмяна на халота, разпределяне на работата и глобално сумиране, като реализирате различни комбинации от тях (помислете за модулно структуриране на кода още в предните задачи).