并行计算

——结构•算法•编程

主讲教师：谢磊

第二篇 并行算法的设计

第四章 并行算法的设计基础

第五章 并行算法的一般设计方法

第六章 并行算法的基本设计技术

第七章 并行算法的一般设计过程

第五章 并行算法的一般设计方法

Case Study:

（1）排序 （Sort）

（2）选择 （Select）

（3）搜索 （Search）

（4）匹配 （Matching）

第五章 并行算法的一般设计方法

5.1 串行算法的直接并行化

5.2 从问题描述开始设计并行算法

5.3 借用已有算法求解新问题

5.1 串行算法的直接并行化

5.1.1 设计方法描述

设计方法的描述

* 方法描述
  * 发掘和利用现有串行算法中的并行性，直接将串行算法改造为并行算法。
* 评注
  * 由串行算法直接并行化的方法是并行算法设计的最常用方法之一；
  * 不是所有的串行算法都可以直接并行化的；
  * 一个好的串行算法并不能并行化为一个好的并行算法；
  * 许多数值串行算法可以并行化为有效的数值并行算法。

5.1 串行算法的直接并行化

5.1.1 设计方法描述

5.1.2 快排序算法的并行化

5.1.3 枚举排序算法的并行化

Case Study:快排序算法的并行化

* 排序算法
  * 在计算机科学与数学中，一个排序算法（Sorting algorithm）是一种能将一串数据依照特定排序方式的一种算法。最常用到的排序方式是数值顺序以及字典顺序。有效的排序算法在一些算法（例如搜索算法与合并算法）中是重要的，如此这些算法才能得到正确解答。排序算法也用在处理文字数据以及产生人类可读的输出结果。

Case Study:快排序算法的并行化

* 快速排序及其串行算法
  * 快速排序（Quick Sort）是一种最基本的排序算法，它的基本思想是：在当前无序区R[1，n]中取一个记录作为比较的“基准”（一般取第一个、最后一个或中间位置的元素），用此基准将当前的无序区R[1，n]划分成左右两个无序的子区R[1，i-1]和R[i，n](1≤i≤n)，且左边的无序子区中记录的所有关键字均小于等于基准的关键字，右边的无序子区中记录的所有关键字均大于等于基准的关键字；当R[1，i-1]和R[i，n]非空时，分别对它们重复上述的划分过程，直到所有的无序子区中的记录均排好序为止。

Case Study:快排序算法的并行化

算法**5.1** 单处理机上快速排序算法

输入：无序数组data[1,n]

输出：有序数组data[1,n]

```
Begin
call procedure quicksort(data,1,n)
End
procedure quicksort(data,i,j)
Begin
(1)if (i<j) then
(1.1)r = partition(data,i,j)
(1.2)quicksort(data,i,r-1);
(1.3)quicksort(data,r+1,j);
end if
End
```

```
procedure partition(data,k,l)
Begin
(1)pivo=data[l]
(2)i=k-1
(3)forj=k tol-1 do
if data[j]≤pivothen
i=i+1
exchangedata[i] and data[j]
end if
end for
(4)exchangedata[i+1] and data[l]
(5)returni+1
End
```

Case Study:快排序算法的并行化

* 快速排序算法的性能主要决定于输入数组的划分是否均衡，而这与基准元素的选择密切相关。
* 在最坏的情况下，划分的结果是一边有n-1个元素，而另一边有0个元素（除去被选中的基准元素）。如果每次递归排序中的划分都产生这种极度的不平衡，那么整个算法的复杂度将是Θ（n^2）。
* 在最好的情况下，每次划分都使得输入数组平均分为两半，那么算法的复杂度为O（nlogn）。
* 在一般的情况下该算法仍能保持O（nlogn）的复杂度，只不过其具有更高的常数因子。

Case Study:快排序算法的并行化

* 快速排序的并行算法
  * 快速排序算法并行化的一个简单思想是，对每次划分过后所得到的两个序列分别使用两个处理器完成递归排序。
  * 例如对一个长为n的序列，首先划分得到两个长为n/2的序列，将其交给两个处理器分别处理；而后进一步划分得到四个长为n/4的序列，再分别交给四个处理器处理；如此递归下去最终得到排序好的序列。当然这里举的是理想的划分情况，如果划分步骤不能达到平均分配的目的，那么排序的效率会相对较差。

Case Study:快排序算法的并行化

算法5.2中描述了使用2^m个处理器完成对n个输入数据排序的并行算法.

算法**5.2** 快速排序并行算法

输入：无序数组data[1,n]，使用的处理器个数2^m

输出：有序数组data[1,n]

Begin

para_quicksort(data,1,n,m,0)

End

procedure para_quicksort(data,i,j,m,id)

Begin

(1)**if** (j-i)≤k **or** m=0 **then**

(1.1)P_id **call quicksort**(data,i,j)

else

(1.2)P_id: r=**partition**(data,i,j)

(1.3)P_id **send** data[r+1,j] to P_id+2^{m-1}-1

(1.4)**para_quicksort**(data,i,r-1,m-1,id)

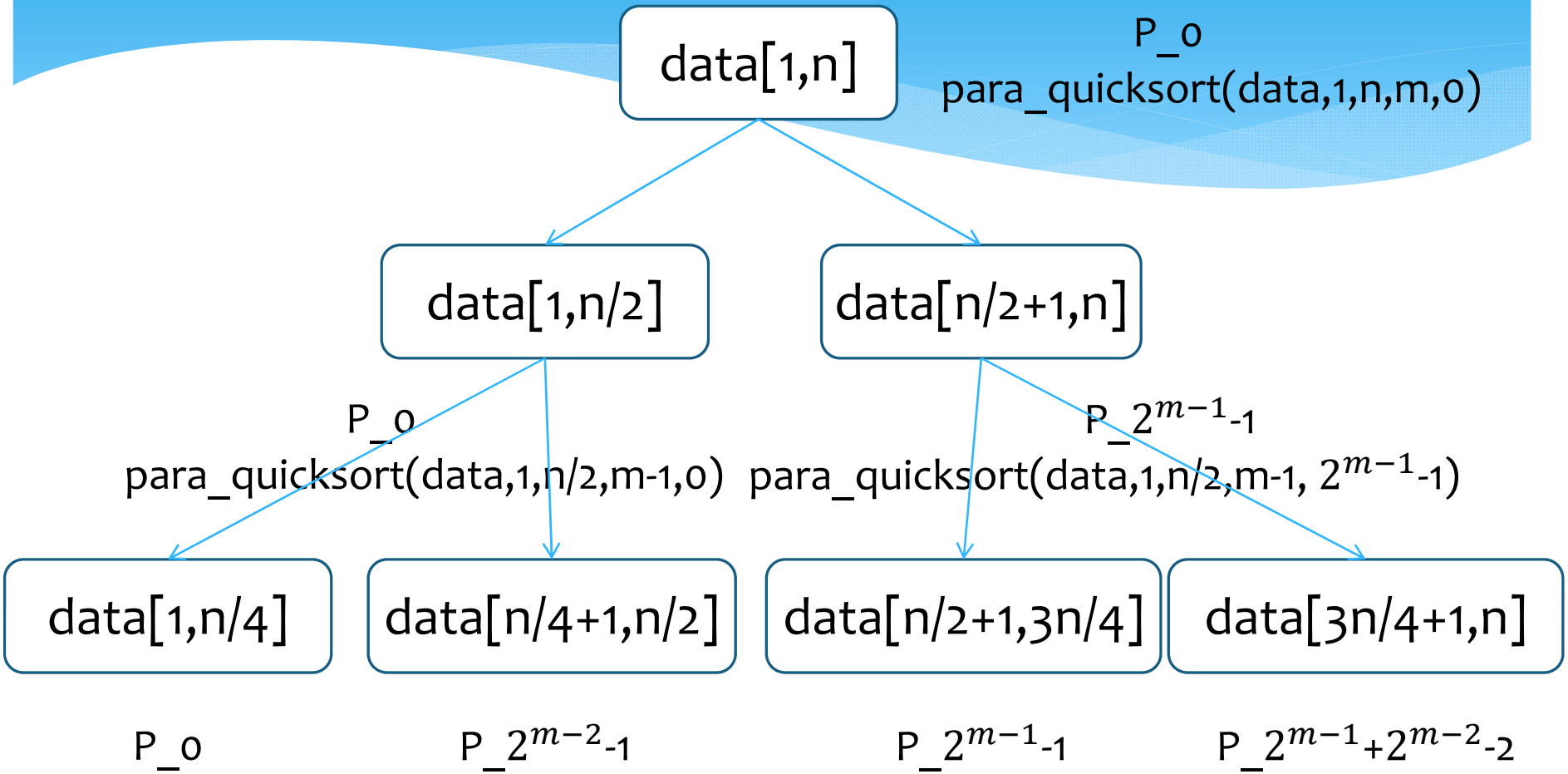
(1.5)**para_quicksort**(data,r+1,j,m-1,id+2^{m-1}-1)

(1.6)P_id+2^m-1 **send** data[r+1,j] back to P_id

end if

End

Case Study:快排序算法的并行化



Case Study:快排序算法的并行化

* 快速排序算法5.2的性能
  * 由于左右分支可以并行处理，所以总体执行时间T(n)=T(n/2)+n。
  * 根据Master定理，所以算法5.2的时间复杂度为Θ(n)。
  * 算法5.2是一种很自然的并行化方法，通过并行的调用快排序对两个所划分的子序列进行快排序。但是这种并行化的方法并未改变串行算法本身的属性，但是划分时间就是O(n)。要进一步提高效率，只有将划分步进行并行化才有可能得到成本最优的算法。

Case Study:快排序算法的并行化

* Master定理

Theorem A.3.68. Master Theorem
Let $a, b,$ and c be positive integers. Let

$$T(1) = 0,$$
$$T(n) = a \cdot T\left(\frac{n}{c}\right) + b \cdot n.$$

Then,

$$T(n) = \begin{cases} O(n) & \text{if } a < c, \\ O(n \cdot \log n) & \text{if } a = c, \\ O\left(n^{\log_c n}\right) & \text{if } c < a. \end{cases}$$

Case Study:快排序算法的并行化

* 算法5.3 PRAM-CRCW上为快排序构造二叉树算法
* 输入：序列(A1,…,An)和n个处理器
* 输出：供快速排序用的一棵二叉树

Begin

(1) For each processor i **do**

(1.1) root=I

(1.2) fi=root

(1.3) LCi=RCi=n+1

End

(2) Repeat for each processor i≠root **do**

If (Ai<Afi) or (Ai=Afi and i<fi) **then**

(2.1) LCfi=i

(2.2) **if** i=LCfi **then** exit **else** fi=LCfi **endif**

else

(2.3) RCfi=i

(2.4) **if** i=RCfi **then** exit **else** fi=RCfi **endif**

Endif

Endrepeat

End

Case Study:快排序算法的并行化

* 快速排序并行算法5.3的性能
  * 在算法每次迭代时，可在Θ(1)时间内构造一级树，而树高为Θ(logn),所以算法5.3的时间复杂度为Θ(logn)。
  * 在最坏情况下，构造出的树高为Θ(n)，所以算法5.3的时间复杂度为Θ(n)。

Case Study:枚举排序算法的并行化

* 枚举排序及其串行算法
  * 枚举排序（Enumeration Sort）是一种最简单的排序算法，通常也称为秩排序（Rank Sort）。该算法的具体思想是（假设按关键字递增排序），对每一个待排序的元素统计小于它的所有元素的个数，从而得到该元素最终处于序列中的位置。假定待排序的n个数存在a[1]…a[n]中。首先将a[1]与a[2]…a[n]比较，记录比其小的数的个数，令其为k，a[1]就被存入有序的数组b[1]…b[n]的b[k+1]位置上；然后将a[2]与a[1]，a[3]…a[n]比较，记录比其小的数的个数，依此类推。这样的比较操作共n(n-1)次，所以串行秩排序的时间复杂度为$O(n^2)$。

枚举排序串行算法

算法5.3 枚举排序串行算法

输入：无序数组a[1]…a[n]

输出：有序数组b[1]…b[n]

```
Begin
fori=1 ton do
(1)k=1
(2)forj=1 ton do
        if a[i]>a[j] then
        k=k+1
        end if
   end for
(3)b[k]= a[i]
end for
End
```

枚举排序的并行算法

对该算法的并行化是很简单的，假设对一个长为n的输入序列使用n个处理器进行排序，只需是每个处理器负责完成对其中一个元素的定位，然后将所有的定位信息集中到主进程中，由主进程负责完成所有元素的最终排位。该并行算法描述如下：

算法5.4 枚举排序并行算法

输入：无序数组a[1]…a[n]

输出：有序数组b[1]…b[n]

Begin

(1)P0播送a[1]…a[n]给所有Pi

(2)**for all** Pi**where** 1≤i≤n**para-do**

(2.1)k=1

(2.2)**for**j= 1 **to**n **do**

if (a[i] > a[j]) **or**(a[i] =a[j] **and**i>j) **then**

k = k+1

end if

end for

(3)P0收集k并按序定位

End

枚举排序的并行算法

* 在该并行算法中，使用了n个处理器，由于每个处理器定位一个元素，所以步骤(2)的时间复杂度为O（n）；步骤(3)中主进程完成的数组元素重定位操作的时间复杂度为O（n），通信复杂度分别为O（n）；同时(1)中的通信复杂度为O（n^2）；所以总的计算复杂度为O（n），总的通信复杂度为O（n^2）。

第五章 并行算法的一般设计方法

5.1 串行算法的直接并行化

5.2 从问题描述开始设计并行算法

5.3 借用已有算法求解新问题

5.2从问题描述开始设计并行算法

5.2.1 设计方法描述

从问题描述开始设计并行算法

* 方法描述
  * 从问题本身描述出发，不考虑相应的串行算法，设计一个全新的并行算法。
* 评注
  * 挖掘问题的固有特性与并行的关系；
  * 设计全新的并行算法是一个挑战性和创造性的工作。

5.2从问题描述开始设计并行算法

5.2.1 设计方法描述

5.2.2 字符串匹配算法的并行化

Case Study: 串匹配算法的并行化

* 字符串匹配
  * 在网络安全、信息检索等领域中，字符串匹配是一种使用很普遍的技术，例如杀毒软件和IDS中的特征码匹配，都需要用到字符串匹配。作为字符串匹配中的一种特殊情况，近似字符串匹配的研究也同样重要。这里将对字符串匹配算法进行简要分析和总结。
  * 精确字符串匹配
  * 近似字符串匹配

Case Study: 串匹配算法的并行化

* 精确字符串匹配

  字符串的精确匹配算法中，最著名的有KMP算法和BM算法。下面分别对几种常用的算法进行描述。

  * 1：KMP算法

    KMP算法，即Knuth-Morris-Pratt算法，是一种典型的基于前缀的搜索的字符串匹配算法。KMP算法的搜索思路应该算是比较简单的：模式和文件进行前缀匹配，一旦发现不匹配的现象，则通过一个精心构造的数组索引模式（失效函数）向前滑动相应的距离。这个算法相对于常规的逐个字符匹配的方法的优越之处在于，它可以通过数组索引，减少匹配的次数，从而提高运行效率。

Case Study: 串匹配算法的并行化

* 精确字符串匹配
  * 2：BM算法（Boyer-Moore算法）
    和KMP算法相反，BM算法采用的是后缀搜索。BM算法预先计算出两个函数值d1、d2，它们分别对应两种不同的情形。当进行后缀匹配的时候，如果模式最右边的字符和文本中相应的字符比较失败，则需要将模式向右移动。当遇到不匹配的字符并非模式最后字符时，则算法有所不同。通过最大安全移动距离来减少匹配的次数，从而提高运行效率。

Case Study: 串匹配算法的并行化

* 精确字符串匹配
  * 3：Shift-And算法

    顾名思义，这个算法是用位运算的。在brute force时的时候，在源串的每个位置，可能都要进行m(模式串长度)次比较，而SHIFT AND算法则是利用位运算提高这个过程。现在计算机的字长一般为32，64位也开始流行了。而一次比较的值为true or false,只需要一位就可以存储了。所以计算机是可以在一次运算里完成位长 次的比较的。通过这个思路可以把brute force的速度提高 位长 倍。

Case Study: 串匹配算法的并行化

* 近似字符串匹配

  近似字符串匹配主要有四种方法。第一种是动态规划方法，这是最古老，同时也是最灵活的近似匹配算法。第二种是基于自动机公式模拟文本搜索。第三种方式采用位并行方法来模拟其他方法，号称是“最成功”的一种方法。最后则是通过简单的过滤文本中不相关文本实现快速搜索。

Case Study: 串匹配算法的并行化

* 近似字符串匹配
  * 1：动态规划算法
  * 动态规划算法是基于“编辑距离”的概念实现近似字符串匹配。通俗地说，编辑距离表示将一个字符串变换成另一个字符串所需要进行的最少的编辑次数。这里的编辑操作包括添加、删除、替换。通过计算编辑距离矩阵，可以得出最佳匹配。
  * 编辑矩阵的初始化和计算是动态规划算法的关键。初始化数值直接决定是全局匹配还是局部匹配，而在计算公式中所采用的增量，则表示了各种操作的权值。

Case Study: 串匹配算法的并行化

* 近似字符串匹配
  * 2：基于自动机的算法

    关于基于自动机的方法，相关资料比较少，而中文资料则更少。在《柔性字符串匹配》一书中稍微提到的这种方法，但是并没有深入讲解其中的原理和实现步骤。

Case Study: 串匹配算法的并行化

* 近似字符串匹配
  * 3：位并行算法

    实际上，与其说位并行算法是一种近似搜索方法，倒不如说它是一种加速实现的手段。位并行算法是用来模拟经典算法的。在搜索中，通过并行模拟，可以加快经典算法的运行速度。位并行算法非常适合模式串比较短的情况。
  * 位并行实际上利用了计算机的并行性原理，将若干不同的值打包到一个长度为w的计算机字中，这样就可以利用一次操作或运算来实现原本需要若干次操作或运算才能完成的功能。

Case Study: 串匹配算法的并行化

* 近似字符串匹配
  * 4：文本快速过滤算法

    过滤算法是基于这种思想：判断文中某个位置的字串和模式串不匹配，可能比判断相匹配更容易。所以过滤算法的思路为：通过过滤算法过滤文本中不能产生成功匹配的区域，然后结合非过滤文本搜索算法，最终实现快速字符串匹配。当错误水平比较低时，文本过滤算法的运行效果很好，否则就很差。

Case Study：KMP算法

* **1.KMP算法简介**
  * KMP算法是模式匹配算法的一种改进算法，是D.E.Knuth与V.R.Pratt和J.H.Morris同时发现的，因此人们称它为克努特-莫里斯- 普拉特操作（简称KMP算法）。
* **2.算法核心**
  * 对字串进行预处理,计算出失效函数:依靠get_next函数计算出子串中每个字符对应的next[j]的值，从而减少子串回溯的距离，减少时间复杂度。
  * KMP算法的关键在于当模式与正文不匹配时，应如何利用已得到的“部分匹配”的结果，将模式向右滑动一段距离后再施行继续比较，这一距离可由失效函数来决定。

Case Study: KMP算法

* **3.算法性能**

  简单匹配算法的时间复杂度为O(m*n);可以证明KMP匹配算法的时间复杂度为O(m+n).

  4.计算子串的模式函数值.

  子串的模式函数值.next[j]有很多版本,这里介绍其中的一种.

  定义:

  (1)next[0]= -1 意义: 任何串的第一个字符的模式值规定为-1

  (2)next[j]= -1 意义: 模式串T中下标为j的字符, 如果与首字符相同,且j的前面的1-k个字符与开头的1-k个字符不等(或者相等但T[k]==T[j])(1≤k).

  (3)next[j]=k 意义: 模式串T中下标为j的字符, 如果j的前面k个字符与开头的k个字符相等, 且T[j] != T[k] (1≤k).

  (4)其他情况0.

Case Study: KMP算法

* **5. KMP算法的并行化**
  * KMP算法的精髓在于使用了失效函数，使用它来调整两指针j和k。这种指针来回移动的办法并不太容易并行化，因为并行算法设计的基本策略是试图将字符串分段并行处理，但上述KMP的顺序算法很难有效的使用在分段并行处理中。
  * 现有的串行算法恐难直接将其并行化，此时要从问题的描述出发，需求可能的新途径，设计出一个新的并行算法。实际上，两串是否能匹配，是与串的自身前缀有关，这种前缀特性就是串的周期性。所以**研究串的周期性是寻找并行化的一种可能的途径。**

Case Study: 串匹配算法的并行化

* 并行串匹配算法的基本思路
  * 既然串的周期特性对研究匹配是至关重要的，我们引入失配见证函数(Witness Function)来表征周期性，定义如下：
  * 定义：对于给定的j($1 \le j \le m/2$)，如果$P[j:m] \neq P[1:m-j+1]$，则存在某个w($1 \le w \le m-j+1$),使得$P(w) \neq P(s)$,其中s=j-1+w,记WIT(j)=w。
  * 可以根据WIT（j）函数来区分串是周期的还是非周期的：
    * 对于所有$2 \le j \le m/2$，当且仅当$WIT(j) \neq 0$时则串是非周期的；
    * 对于所有$2 \le j \le m/2$，若存在某个j使得$WIT(j) = 0$，则串是周期的。

Case Study: 串匹配算法的并行化

* 一旦确定了串的周期特性，我们可先研究**非周期串的匹配**，然后在此基础上再研究**周期串的匹配**。
* 非周期串匹配的研究
  * 目标：如何利用已计算出的WIT（i）快速地找出P在T中匹配的位置。
  * 为了减少P与T的比较次数，引入竞争函数duel(p,q)的概念。即当模式在某一位置p匹配时，则在另一位置q一定不匹配，这样就排除了q位置。

Case Study: 串匹配算法的并行化

* 非周期串匹配的研究
  * 可以设计一个算法来计算duel(p,q)。在duel(p,q)算法中，参数p，q与n，m相关，且p与q的选取应先从小范围，逐步到大范围，且在每个限定范围内可并行地求duel(p,q)，以确定竞争的幸存者。
  * 这样整个过程就像分组比赛一样，逐渐淘汰小组的获胜者，最终只可能保留少数几个幸存者，它们就是些可能的匹配位置号码，最后再进行一次正确性验证即可。

Case Study: 串匹配算法的并行化

* 并行串匹配算法（非周期性串）举例
  * 令T=abaababaabababaababaaba（n=23），P=abaababa(m=8)。根据模式P计算出WIT(1)=0，WIT(2)=1，WIT(3)=2，WIT(4)=4可知P是非周期串。
    * WIT(1)=0 abaababa= abaababa
    * WIT(2)=1 **b**aababa vs **a**baabab
    * WIT(3)=2 a**a**baba vs a**b**aaba
    * WIT(4)=4 aba**b**a vs aba**a**b
  * 为了计算此非周期串与T的匹配情况，先要计算P相对于T的WIT[1:n-m+1]之值，然后由其来计算duel(p,q)之值。

Case Study: 串匹配算法的并行化

* 并行串匹配算法（非周期性串）举例
  * 计算duel(p,q)时，将T与P由小到大划分成一些大小为$(2^1, 2^2, \ldots)$的块,在相同大小的各块内并行的计算duel(p,q)之值。其过程为:
  * 先将P与T各自划分为大小为2的一些块，这样模式块(ab)与正文块(ab)(aa)(ba)(ba)(ab)(ab)(aa)(ba)进行匹配，可知在位置1,4,6,8,9,11,14,16出现匹配（即duel(p,q) 的获胜者）；
  * 再将P与T各自划分为大小为4的一些块，这样模式块(abaa)与正文块(abaa)(baba)(abab)(aaba)进行匹配，可知在位置1,6,11和16出现匹配，而位置4,8,9和14被淘汰;

Case Study: 串匹配算法的并行化

* 并行串匹配算法（非周期性串）举例
  * 最后需用模式(ababbaba)在正文的位置1,6,11和16进行匹配检查，已验证其正确性。
  * 至于周期串的匹配，情况更为复杂，可参阅文献[191]。

第五章 并行算法的一般设计方法

5.1 串行算法的直接并行化

5.2 从问题描述开始设计并行算法

5.3 借用已有算法求解新问题

5.3 借用已有算法求解新问题

5.3.1 设计方法描述

5.3.2 利用矩阵乘法求所有点对间最短路径

设计方法的描述

* 方法描述
  * 找出求解问题和某个已解决问题之间的联系；
  * 改造或利用已知算法应用到求解问题上。
* 评注
  * 这是一项创造性的工作；
  * 使用矩阵乘法算法求解所有点对间最短路径是一个很好的范例。

5.3 借用已有算法求解新问题

5.3.1 设计方法描述

5.3.2 利用矩阵乘法求所有点对间最短路径

利用矩阵乘法求所有点对间最短路径

* 计算原理

有向图G=(V,E)，边权矩阵$W=(w_{ij})_{n\times n}$，求最短路径长度矩阵$D=(d_{ij})_{n\times n}$，d_{ij}为v_i到v_j的最短路径长度。假定图中无负权有向回路，记$d^{(k)}_{ij}$为v_i到v_j至多有k-1个中间结点的最短路径长，$D^k=(d^{(k)}_{ij})_{n\times n}$，则

(1) $d^{(1)}_{ij}=w_{ij}$　　当 $i\neq j$ (如果v_i到v_j之间无边存在记为∞)

$d^{(1)}_{ij}=0$　　当 $i=j$

(2) 无负权回路 ➔ $d_{ij}=d^{(n-1)}_{ij}$

(3) 利用最优性原理：$d^{(k)}_{ij}=\min_{1\leqslant l\leqslant n}\{d^{(k/2)}_{il}+d^{(k/2)}_{lj}\}$

视：”+” ➔ “×”，“min” ➔ “∑”，则上式变为

$d^{(k)}_{ij}=\sum_{1\leqslant l\leqslant n}\{d^{(k/2)}_{il}\times d^{(k/2)}_{lj}\}$

(4) 应用矩阵乘法：$D^1\rightarrow D^2\rightarrow D^4\rightarrow \ldots\rightarrow D^{2^{\log n}}$ $(=D^n)$

利用矩阵乘法求所有点对间最短路径

* SIMD-CC上的并行算法
  * 输入：A(0,j,k)=w_{jk},$0 \le j,k \le 1$
  * 输出：C(0,j,k)中是v_j到v_k的最短路径长度

Begin

(1)/*构筑矩阵D^1，并将其存入A和B的寄存器中*/

for j=0 **to** n-1 **par-do**

  for k=0 **to** n-1 **par-do**

  if j≠ k & A(0,j,k)=0 **then**

  B(0,j,k)=∞

  endif

  else

  B(0,j,k)=A(0,j,k)

  endfor

endfor

利用矩阵乘法求所有点对间最短路径

* SIMD-CC上的并行算法

(2)/*调用DNS算法构筑矩阵$D^2, D^4, \ldots, D^{n-1}$*/

for i=1 to [log(n-1)] **do**

DNS MULTIPLICATION(A,B,C) /*调用算法9.6*/

for j=0 to n-1 **par-do**

for k=0 **to** n-1 **par-do**

A(0,j,k)=C(0,j,k)

B(0,j,k)=C(0,j,k)

endfor

endfor

endfor

end