并行计算复习————第二篇 并行计算理论基础：并行算法设计_JCGuo的专栏-CSDN博客

并行计算复习

第二篇 并行计算理论基础：并行算法设计

Ch5 并行算法与并行计算模型

5.1 并行算法的基础知识

1.并行算法的表达

（1）par-do

n个节点并行完成for循环（每个节点不同，和i相关）：

```
for i = 1 to n par-do
    ...
endfor
```

- 1
- 2
- 3

（2）for all

所有节点都执行相同语句：

```
for all Pi, where 0 <= i <= k do
    ...
endfor
```

- 1
- 2
- 3

2.并行算法的复杂度

- 运行时间t(n)：求解问题的时间，包括计算时间和通信时间
- 处理器数目p(n)：求解给定问题所用的处理器数目
- 成本c(n)：c(n) = t(n)*p(n)
- 成本最优——并行算法的成本在数量级上等于最坏情况下串行求解次问题所需要的执行步数
- 工作量W(n)：并行算法完成的总的操作数量
- 工作量最优：功耗低、环保

并行算法的WT表示——Brent定理：

令W(n)是并行算法A在运行时间T(n)内所执行的运算量，则A使用p台处理器可在t(n) = O(W(n)/p + T(n))时间内执行完毕
1

5.2 并行计算模型

1.PRAM模型（SIMD-SM）

PRAM（Parallel Random Access Machine）并行随机存取机器，是一种抽象并行计算模型，它假设：

- 存在容量无限大的SM
- 有限或无限个功能相同的处理器，且均有简单算术运算和逻辑判断功能
- 任何时刻各处理器可通过SM交换数据

根据并发访问机制，又分为：

- 不允许同时读和同时写的PRAM-EREW
- 允许同时读但不允许同时写的PRAM-CREW
- 允许同时读和同时写的PRAM-CRCW

PRAM-CRCW又分为：

- 只允许同时写相同的数CPRAM-CRCW
- 只允许优先处理器先写PPRAM-CRCW
- 允许任意处理器自由写APRAM-CRCW

PRAM优点：

- 适合并行算法表达、分析和比较
- 使用简单，屏蔽了通信、存储管理、进程同步等并行细节
- 易于修改算法设计以适应不同并行机

PRAM缺点：

- PRAM是同步模型，同步锁费时
- 不适用于MIMD和DM
- 假设任何处理器可在单位时间内访问任何处理单元而不考虑竞争和带宽，不现实

2.异步APRAM模型（MIMD-SM）

异步APRAM模型假设：

- 每个处理器有LM、局部时钟、局部程序
- 处理器通信经过SM
- 无全局时钟，各处理器异步执行各自指令
- 处理器之间的指令相互依赖关系必须显式加入同步障
- 一条指令可以在非确定但有界时间内完成

指令类型有：

- 全局读：从SM读到LM

- 局部操作：LM操作存入LM
- 全局写：LM写入SM
- 同步：各处理器需等到其他处理器到达后才能继续执行

APRAM比PRAM更加接近实际并行机

3.BSP模型（MIMD-DM）

BSP（Bulk Synchronous Parallel）大同步并行机（APRAM算作轻量）是一个分布式存储的MIMD模型，它的计算由若干全局同步分开的、周期为L的超级步组成，各超级步中处理器做LM操作并通过选路器接收和发送消息；然后做一次全局检查，以确定该超级步是否已经完成（块内异步并行，块间显式同步）

参数：处理器数p、选路器吞吐率g、全局同步间隔L、一个超级步中一个处理器至多发送或接收h条消息

4.LogP模型：MIMD-DM，点到点通讯

LogP模型是分布式存储、点到点通信的MIMD模型

LogP采取隐式同步，而不显式同步障

Ch6 并行算法基本设计策略

6.1 串改并

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

最常用的设计思路但并不普适，好的串行算法一般无法并行化（数值串行算法可以）

快速排序的串改并

SISD上串行执行，最坏情况下O(n^2)，理想情况下O(nlogn)

思路：将O(n)的划分（Partition）并行化是关键，算法：

- 构造一棵二叉排序树，主元是根
- 小于等于主元素处于左子树，大于主元素处于右子树
- 左右子树均是二叉排序树

在CRCW模型上，用伪代码描述如下：

```
//A[1...n]排序用n个处理器，处理器i中存有A[i]
//f[i]中存有其元素是主元素的处理器号
//LC[1...n]和RC[1...n]分别记录给定主元素的左儿子和右儿子

for each processor i par-do
    root = i    //所有处理器竞争，只有一个写入root
    f[i] = root //所有处理器都把root写入自己的f[i]
    LC[i] = RC[i] = n + 1   //初始值
endfor

repeat for each processor i != root do
    if (A[i] < A[f[i]]) || (A[i] == A[f[i]] && i < f[i])
        LC[f[i]] = i    //并发写，所有满足条件的i只有一个写入LC[f[i]]作为左子树的根，也就是下一次循环的主元
素
        if i == LC[f[i]] then
            exit        //若当前处理器写入则什么也不做
        else
            f[i] = LC[f[i]] //若当前处理器没有写入，那么它只能当LC[f[i]]的字节点了
        endif
    else
        RC[f[i]] = i    //并发写，所有满足条件的i只有一个写入RC[f[i]]作为右子树的根，也就是下一次循环的主元
素
        if i == RC[f[i]] then
            exit        //若当前处理器写入则什么也不做
        else
            f[i] = RC[f[i]] //若当前处理器没有写入，那么它只能当RC[f[i]]的字节点了
        endif
    endif
endrepeat
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25
- 26
- 27
- 28

每次迭代构造一层排序二叉树花费O(1)时间，在基本平衡的情况下树高O(logn)，则算法复杂度为O(logn)

6.2 全新设计

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

有向环K着色算法的并行化设计

问题：有向环顶点着色，一共K种颜色，相邻顶点不允许同色

串行算法（3着色）：交替2种颜色，若顶点是奇数则需要用到第3种颜色（难以并行化）

SIMD-EREW上的并行K着色算法：

```
//初始随机着色为c[i]，每个顶点着色不同
//输出着色方案为nc[i]
for i = 1 to n par-do
    k = c[i]和c[i的后继]的最低的不同二进制位
    nc[i] = 2 * k + c[i]的二进制第k位
endfor
```

- 1
- 2
- 3
- 4
- 5
- 6

O(1)时间完成，需要算法正确性证明

6.2 借用法

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

利用矩阵乘法求所有点对间最短路径

d[k][i][j]表示从vi到vj**至多**经过k-1个中间顶点时的最短路径，d[k][i][j] = min{d[k/2][i][l]+d[k/2][l][j]}

那么可以用矩阵乘法的改进（乘变加，求和变min）做logn次矩阵乘法即可

思路是这样，具体伪代码略，需要O(n^3)个节点，时间复杂度为O(log^2(n))

Ch7 并行算法常用设计技术

6.1 划分设计技术

使用划分法把问题求解分成两步：

1. 把给定问题划分成p个几乎等尺寸的子问题
2. 用p台处理器并行求解子问题

（1）均匀划分(PSRS排序)

长度为n的待处理序列均匀划分给p个处理器，每个处理器处理n/p个元素

MIMD机器上的PSRS排序算法：

(1)均匀划分:将n个元素A[1..n]均匀划分成p段，分配给p个处理器
(2)局部排序:pi调用串行排序算法对A[(i-1)n/p+1..in/p]排序
(3)选取样本:pi从其有序子序列A[(i-1)n/p+1..in/p]中选取p个样本元素
(4)样本排序:用一台处理器对p2个样本元素进行串行排序
(5)选择主元:用一台处理器从排好序的样本序列中选取p-1个主元,并播送给其他pi
(6)主元划分:pi按主元将有序段A[(i-1)n/p+1..in/p]划分成p段
(7)全局交换:各处理器将其有序段按段号交换到对应的处理器中（一定保证均匀划分？？）
(8)归并排序:各处理器对接收到的元素进行归并排序

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8

（2）方根划分(Valiant归并排序)

长度为n的待处理序列，取i*sqrt(n)为划分元，将元素划分成若干段交给处理器处理

SIMD-CREW机器上的Valiant归并排序：

(1)方根划分：把A和B（长n有序段）的第i*sqrt(n)元素作为划分元，把A和B分成若干段
(2)段间比较：A中的所有划分元和B中的所有划分元比较，确定A中划分元应插入B中的哪一段
(3)段内比较：A中的划分元和B中相应段比较并确定插入位置，这些插入位置又将B重新划分成了若干段
(4)段组合并：插入A划分元后，又得到若干有序段组需要归并，递归直到有一组（A）的长度为0

- 1
- 2
- 3
- 4
- 5

使用n个处理器可以在O(loglogn)内完成

（3）对数划分(并行归并排序)

取i*logn作为划分元划分

定义位序rank(x,X)为X中小于等于x的元素个数，则有PRAM-CREW上的对数划分：

```
//非降有序的A={a[1],...,a[n]}和B={b[1],...,b[m]}归并
//假设log(m)和k=m/log(m)均为整数
j[0]=0
j[k]=n

for i = 1 to k - 1 par-do
    j[i] = rank(b[ilogm], A)
endfor

for i = 1 to k - 1 par-do
    Bi = {b[ilogm+1], ..., b[(i+1)logm]}
    Ai = {a[j[i]+1], ..., a[j[i+1]]}
endfor

//将原问题转化为子序列组Bi和Ai的归并，那么同样可以递归调用完成整个序列的归并
//对数划分保证Bi和Ai中的元素均大于Bi-1和Ai-1中的元素
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17

（4）功能划分((m,n)-选择)

功能划分是根据特定问题而把序列分成p个等长组，每组符合问题特性的一种划分方法

(m,n)-选择问题是将长为n的序列中选取前m个较小的元素，利用功能划分来实现并行化的(m,n)-选择问题求解：

```
(1)功能划分：将A划分成g=n/m组，每组含m个元素
(2)局部排序：使用Batcher排序网络将各组并行排序
(3)两两比较：将所排序的各组两两比较，从而形成MIN序列
(4)排序-比较：对各个MIN序列，重复执行第(2)和第(3)步，直至选出m个最小者
```

- 1
- 2
- 3
- 4

6.2 分治设计技术

分治将复杂问题划分成较小规模特性相同的子问题，且子问题类型和原问题类型相同，通常用递归完成分治算法

（1）双调归并网络

双调序列：若序列x[0],…,x[n-1]是双调序列，则存在一个0<=k<=n-1使得x[0]>=…>=x[k]<=x[k+1]<=…<=x[n-1]或序列循环移动可以得到此关系

Batcher定理：双调序列对0<=i<=n/2-1，比较所有x[i]和x[i+n/2]得到的小序列MIN和大序列MAX仍然是双调序列

Batcher双调归并排序算法：

```
//输入双调序列X得到非降有序序列Y
procedure Bitonic-Merge(x) {
    for i = 0 to n/2 - 1 par-do
        s[i] = min(x[i], x[i+n/2])
        l[i] = max(x[i], x[i+n/2])
    endfor

    Bitonic-Merge(s)
    Bitonic-Merge(l)

    output s followed by l
}
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12

6.3 平衡树设计技术

以树的叶结点为输入，中间结点为处理结点，由叶向根或由根向叶逐层进行并行处理

（1）求最大值

SIMD-TC(SM)上O(logn)的树上求最大值算法：

```
//带求最大值的n个元素放在A[n, ..., 2n-1]中，最大值放在A[1]中
//n=2^m
for k = m-1 to 0 do
    for j = 2^k to 2^(k+1)-1 par-do
        A[j]=max(A[2j],A[2j+1])
    endfor
endfor
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7

SIMD－CRCW上常数时间求最大值算法：

```
//输入A[1...p]共p个元素
//B[1..p][1..p],M[1..p]为中间处理用的布尔数组, 如果M[i]=1, 则A[i]为最大值

for i = 1 to p, j = 1 to p par-do   //W(n)=O(p^2)换取时间O(1)
    if A[i] >= A[j] then
        B[i,j] = 1
    else
        B[i,j] = 0
    endif
endfor

for i = 1 to p par-do
    M[i] = B[i,1]∧ B[i,2]∧... ∧B[i,p]
    //向量操作保证O(1)完成，则该并行段也是O(1)
    //若M[i]=1，则是A[i]最大值
endfor
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16

（2）计算前缀和

定义二元结合运算*，则n个元素x[1,…,n]的前缀和是如下定义的n个部分和：

s[i]=x[1]*x[2]*x[i]，i=1,…,n

串行计算n个元素前缀和需要O(n)的时间（利用s[i] = s[i-1] * x[i]）

下面给出SIMD-TC上的求前缀和O(logn)的算法：

```
for j = 1 to n par-do
    B[0, j] = A[j]  //平衡树底层是n个元素
endfor                                  //O(1)

for h = 1 to logn do                    //正向遍历，父节点存所有字节点之和
    for j = 1 to n/2^h par-do
        B[h,j]=B[h-1,2j-1]*B[h-1,2j]    //父节点存子节点之和
    endfor                              //O(1)
endfor                                  //共logn层，则总时间O(logn)

for h = logn to 0 do
    for j = 1 to n/2^h par-do
        if j % 2 == 0 then      //该节点是右儿子
            C[h,j]=C[h+1,j/2]   //右儿子等于父节点的值
        else if j == 1 then
            C[h,j]=B[h,1]       //每一层第一个节点等于B树上对应值
        else                    //该节点是左儿子
            C[h,j]=C[h+1,(j-1)/2]*B[h,j]    //叔节点和\*自身和
        endif
    endfor
endfor
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21

6.4 倍增设计技术

递归调用时，所要处理数据之间的距离逐步加倍, 经过k步后即可完成距离为2^k的所有数据的计算

（1）表序问题

n个元素的序列L，每个元素分配一个处理器，给定k求出L[k]的rank(k)值，rank(k)等于其到表尾的距离

每个元素有一个指向下一个元素的指针next(k)

下面给出SIMD-EREW上求元素表序的算法：

```
for k in L par-do
    p(k) = next(k)
    if p(k) != k then
        distance(k)=1
    else
        distance(k)=0
    endif
endfor                  //O(1)完成distance初始化

repeat t = ceil(logn) times
    for k in L par-do
        if p(k)!=p(p(k)) then //不是到数第2个
            distance(k) += distance(p(k))   //把next的dis加到自己上来
            p(k)=p(p(k))    //next倍增
        endif
    endfor

    for k in L par-do
        rank(k)=distance(k)     //为什么每次都去赋值rank不最后赋值一次？
    endfor

endrepeat               //O(logn)
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23

（2）求森林的根

森林是一组有根有向树，找森林的根就是求出所有节点所在的树的根节点

可以用倍增技术去修改每个节点的后继（后继即是父节点），下面是SIMD-CREW上的求森林根算法：

```
for i = 1 to n par-do
    s[i] = p[i]
    while s[i] != s[s[i]] do    //该节点不是根
        s[i] = s[s[i]]          //指向父节点
    endwhile                    //这个while不需要同步吗？
endfor                          //O(logn)
```

- 1
- 2
- 3
- 4
- 5
- 6

6.5 流水线技术

将算法路程分成p个前后衔接的任务片段，一个任务片段完成之后其后继任务片段可以立即开始，那么久可以引入流水线的思想来处理多条数据

（1）五点的DFT计算

计算DFT时引入Horner法则把任务划分成流水段：

$$\begin{cases} y_0 = b_0 = a_4\omega^0 + a_3\omega^0 + a_2\omega^0 + a_1\omega^0 + a_0 \\ y_1 = b_1 = a_4\omega^4 + a_3\omega^3 + a_2\omega^2 + a_1\omega^1 + a_0 \\ y_2 = b_2 = a_4\omega^8 + a_3\omega^6 + a_2\omega^4 + a_1\omega^2 + a_0 \\ y_3 = b_3 = a_4\omega^{12} + a_3\omega^6 + a_2\omega^4 + a_1\omega^2 + a_0 \\ y_4 = b_4 = a_4\omega^{16} + a_3\omega^6 + a_2\omega^4 + a_1\omega^2 + a_0 \end{cases} \Rightarrow \begin{cases} y_0 = (((a_4\omega^0 + a_3)\omega^0 + a_2)\omega^0 + a_1)\omega^0 + a_0 \\ y_1 = (((a_4\omega^1 + a_3)\omega^1 + a_2)\omega^1 + a_1)\omega^1 + a_0 \\ y_2 = (((a_4\omega^2 + a_3)\omega^2 + a_2)\omega^2 + a_1)\omega^2 + a_0 \\ y_3 = (((a_4\omega^3 + a_3)\omega^3 + a_2)\omega^3 + a_1)\omega^3 + a_0 \\ y_4 = (((a_4\omega^4 + a_3)\omega^4 + a_2)\omega^4 + a_1)\omega^4 + a_0 \end{cases}$$

O(n)的时间即可完成DFT计算

（2）4流水线编程实例

老师举出一个可流水化的矩阵赋值的例子，意义不是很大，略