并行计算复习————第三篇 并行计算理论基础：并行数值算法

第三篇 并行计算理论基础：并行数值算法

注：此篇较水，=。=

Ch9 稠密矩阵运算

9.1 矩阵的划分

矩阵的划分一般分为带状划分和棋盘划分，在此基础上又有循环划分的变体：

- 带状划分：把矩阵的若干行或若干列连续地划分给一个处理器
- 循环带状划分：把矩阵的若干行或若干列间断且等间隔地划分给一个处理器
- 棋盘划分：把方阵连续地划分成若干子方阵，每个处理器指派一个子方阵
- 循环棋盘划分：把方阵间断且等间隔地划分成若干子方阵，每个处理器指派一个子方阵

一般情况下，棋盘划分的划分方法能够开发出更高并行度的算法

注：下面讨论算法中的划分均是没有循环版本的划分

9.2 矩阵转置

矩阵转置串行执行时间是O(n2)O(n2)，我们可以用棋盘划分讨论网孔和超立方连接上的矩阵转置算法及其时间分析

（1）网孔上的矩阵转置

我们分析一下算法时间复杂度时都必须考虑第一篇第二章中的通信时间（通信时间和选路方式以及并行集群网络拓扑结构有关），若不考虑通信时间，比如这个矩阵转置，使用p=n2p=n2个处理器在O(1)O(1)的时间就可以完成，显然这过于理想化

考虑n×nn×n的方阵转置，且有p≤n2p≤n2个处理器，那么根据棋盘划分，每个处理器分得大小为(np√)×(np√)(np)×(np)的子矩阵，算法分成两步：

1. 子矩阵转置
2. 子矩阵内部转置

若不考虑节点延迟时间thth，那么有启动时间tsts，传输每个字节的时间twtw，则在相邻节点间传输一个子矩阵的时间是ts+tw×n2pts+tw×n2p

在p–√×p–√p×p的二维网孔上显然最长是2p–√2p，所以第一步并行转置时间是2p–√×(ts+tw×n2p)2p×(ts+tw×n2p)

子矩阵内部用串行转置算法，时间复杂度为n22pn22p，那么可以求得总运行时间为：

Tp=n22p+2p–√×(ts+tw×n2p)

Tp=n22p+2p×(ts+tw×n2p)

（2）超立方上的矩阵转置

超立方中可以采用递归转置的算法，每次四分块矩阵，p个处理器一直递归转置到矩阵为(np√)×(np√)(np)×(np)再用串行完成

不管什么网络拓扑，p个处理棋盘划分矩阵递归转置递归层数为log4p=logp2log4p=logp2

由于超立方连接拓扑特殊性质，每次存储转发需要2(ts+tw×n2p)2(ts+tw×n2p)，因此总运行时间为：

Tp=n22p+logp(ts+tw×n2p)

Tp=n22p+logp(ts+tw×n2p)

9.3 矩阵向量乘法

下面讨论n×nn×n的方阵和一个n×1n×1向量相乘的矩阵向量乘法，串行时间复杂度为O(n2)O(n2)

（1）带状划分的算法及其时间分析

行带状划分，处理器PiPi中存放xixi和ai,0,ai,1,...,ai,n−1ai,0,ai,1,...,ai,n−1，它负责计算出yi=∑n−1j=0ai,j×xjyi=∑j=0n−1ai,j×xj，很显然我们需要多到多播送每行自己的xixi

1.网孔

把上述情况推广到p≤np≤n的一般情形，那么则需要多到多播送n/pn/p个元素，查看第一篇中在二维环绕的p–√×p–√p×p采取SF多到多播送n/pn/p个元素时间为2ts(p–√−1)+n/p×tw(p−1)≈2ts(p–√−1)+ntw2ts(p−1)+n/p×tw(p−1)≈2ts(p−1)+ntw，每个处理器处理长度为n2pn2p的向量相乘需要n2pn2p的时间，则总时间为：

Tp=n2p+2ts(p–√−1)+ntw

Tp=n2p+2ts(p−1)+ntw

2.超立方

超立方只有通信时间不一样，查表可得总时间为：

Tp=n2p+tslogp+ntw

Tp=n2p+tslogp+ntw

（2）棋盘划分的算法及其时间分析

棋盘划分来完成这个问题并不直观且效果也不是很好（没有改进），直接给出棋盘划分在二维网孔上用SF选路的总执行时间：

Tp≈n2p+2tsp–√+3ntw

Tp≈n2p+2tsp+3ntw

9.4 矩阵乘法

矩阵乘法串行执行时间是O(n3)O(n3)，较为巧妙而复杂的Strassen分块矩阵乘法是O(n2.8)O(n2.8)，但总之采取串行算法无法达到O(n2)O(n2)及其以下时间，那么我们可以考虑并行矩阵乘法乘法来获取更快的时间

（1）简单并行分块算法

对于求两个n×nn×n的方阵A和B的乘积矩阵C，我们可以考虑分块策略：对于p个处理器的并行，
把An×nAn×n和Bn×nBn×n分割成(np√)×(np√)(np)×(np)的子矩阵，子矩阵之间用分块矩阵乘法完成计算，
虽然串行仍然是O(n3)O(n3)但分块给fenk我们并行带来了思路

p个处理器按p–√×p–√p×p的二维网孔排列，每个处理器Pi,jPi,j中存放子矩阵Ai,jAi,j和Bi,jBi,j，每个处理器完成Ci,j=∑p√k=0Ai,k×Bk,jCi,j=∑k=0pAi,k×Bk,j

考虑计算子矩阵乘法时间为：p–√×(np√)3=n3pp×(np)3=n3p，多到多播送数据量为(np√)2(np)2，那么有：

1.二维环绕网孔简单并行分块算法运行总时间为：

Tp=n3p+2p–√×(ts+tw×n2p)

Tp=n3p+2p×(ts+tw×n2p)

2.超立方简单并行分块算法运行总时间为：

Tp=n3p+logp×ts+2tw×n2p–√

Tp=n3p+logp×ts+2tw×n2p

注意到通信结束时每个处理器有2p–√2p个块，所以总的存储器要求
为2p–√×n2p×p=O(n2p–√)2p×n2p×p=O(n2p)，存储要求较高

（2）Cannon算法及其计算示例

Cannon算法是一个存储有效的并行矩阵分块乘法算法，它通过一系列巧妙的子矩阵循环移动来避免简单并行分块乘法算法中多到多播送所需要的高存储容量要求

Cannon算法分为两步：

1. 初始对准：通过初始循环移动子矩阵（交换处理器中子矩阵的内容）让每个处理器Pi,jPi,j都拥有一对子矩阵Ai,kAi,k和Bk,jBk,j（无所谓k初始是多少）
2. 乘加循环移动：每个处理器Pi,jPi,j对其子矩阵Ai,kAi,k和Bk,jBk,j做一次乘加运算，然后再循环移动保证每个处理器Pi,jPi,j都拥有一对子矩阵Ai,k′Ai,k′和Bk′,jBk′,j且k′k′不重复
3. 重复过程2直到完成所有乘加运算得到最终结果

具体的循环移动做法是

1. 初始对准：矩阵Ai,jAi,j向左循环移动i步，矩阵Bi,jBi,j向上循环移动i步
2. 乘加循环：每次矩阵Ai,jAi,j向左循环移动1步，矩阵Bi,jBi,j向上循环移动1步

Cannon算法十分巧妙也很直观，伪代码就不给出了，直接分析时间复杂度：

在超立方使用CT选路，初始对准循环移位时间为2(ts+tw×n2p+thlogp–√)2(ts+tw×n2p+thlogp)（我推出来不是这个，有点奇怪。。），sqrtpsqrtp次单步循环移位时间为2(ts+tw×n2p)×p–√2(ts+tw×n2p)×p，乘加总时间仍然是n3pn3p，那么有总时间：

Tp=n3p+2(ts+tw×n2p+thlogp–√)+2(ts+tw×n2p)×p–√≈n3p+2p–√(ts+tw×n2p)

Tp=n3p+2(ts+tw×n2p+thlogp)+2(ts+tw×n2p)×p≈n3p+2p(ts+tw×n2p)

同理可推知二维网孔上时间为：

Tp=n3p+4p–√(ts+tw×n2p)

Tp=n3p+4p(ts+tw×n2p)

（3）Fox算法及其计算示例

Fox算法和Cannon算法的思想一致，都是通过有效的循环移位来降低总存储空间，不过Fox算法采用行一到多播送，列循环单步上移的方法，算法如下：

1. 选择A的对角块Ai,iAi,i进行p–√−1p−1的行一到多播送，每行处理器都有一个该行的A的对角块Ai,iAi,i副本
2. 各处理器将收到的A子矩阵和各自的B矩阵做乘加运算
3. 若上一次一到多播送的Ai,jAi,j子矩阵，则这一次处理器Pi,(j+1)modp√Pi,(j+1)modp一到多播送Ai,(j+1)modp√Ai,(j+1)modp，转第2步直到完成计算

手工推演可以证明Fox的正确性，并且可以推得在超立方上用CT选路的运行时间为：

Tp=n3p+p–√×logp2(ts+tw×n2p)

Tp=n3p+p×logp2(ts+tw×n2p)

Ch10 线性方程组的求解

求解大规模线性方程组在很多科学领域有十分广泛需求，数值计算方法课程中介绍了许多串行化且可编程的线性方程组求解方法，用并行集群去求解大规模线性方程组需要将这些算法并行化

10.1 回代求解上三角形方程组的并行算法

n×nn×n的系数矩阵若是上三角阵，我们逐步串行回代需要O(n2)O(n2)的时间求解：

```
for i = n downto 1 do
    x[i] = b[i]/a[i][i]
    for j = 1 to i - 1 do
        b[j] = a[j][i] * x[i]
        a[j][i] = 0
    endfor
endfor
```

观察串行算法，第二重循环完全可以并行化，下面给出SIMD-CREW上的并行回代算法（p个处理器行循环带状划分）

```
for i = n downto 1 do
    x[i] = b[i]/a[i][i]
    for all Pk par-do
        for j = k to i - 1 step p do
            b[j] = a[j][i] * x[i]
            a[j][i] = 0
        endfor
    endfor
endfor
```

处理器足够多（p(n)=n）的话，复杂度降到O(n)O(n)

10.2 三对角方程组的奇偶规约求解法

算法略，p(n)=n2p(n)=n2的话，时间可以做到O(logn)O(logn)

10.3 稠密线性方程组Gauss-Seidel迭代法的并行化

Gauss-Seidel迭代法是利用上一轮迭代结果和本轮迭代已产生结果来进行线性方程组迭代的求解方法（详见《数值计算方法》），其迭代公式为：

$$x(k+1)i=-1aii[\sum j<iaijx(k+1)j+\sum j>iaijx(k)j-bi]$$

$$xi(k+1)=-1aii[\sum j<iaijxj(k+1)+\sum j>iaijxj(k)-bi]$$

Gauss-Seidel迭代法需要利用本轮迭代已产生的值，因此无法同步并行化，下面是MIMD上一步并行算法：

![p2](./data/p2.png =480x)

（没弄懂怎么异步开启进程）

稀疏线性方程组Gauss-Seidel迭代法的并行化两种算法：

- 小规模并行化算法(针对五点格式产生的线性方程组)
- 红黑着色并行算法(针对五点格式产生的线性方程组)

（也没搞懂，这部分就弃疗了吧）

Ch11 快速傅立叶变换FFT

11.1 离散傅里叶变换DFT

n个点(a0,a1,...,an−1)(a0,a1,...,an−1)的DFT(b0,b1,...,bn−1)(b0,b1,...,bn−1)的定义为：

bj=∑k=0n−1ωjkak,0≤j≤n−1

$$bj=\sum k=0n-1\omega jkak,0\leq j\leq n-1$$

其中ω=e2πi/nω=e2πi/n是单位n次元根，很显然DFT是O(n2)O(n2)的

11.2 串行FFT分治递归算法

FFT是一种O(nlogn)时间内计算序列DFT的快速算法，FFT主要利用了单位n次元根的对称性，即不需要计算出所有n2n2个单位n次元根

SISD上FFT递归算法伪代码如下：

```
Procedure RFFT(a,b) {
    if n == 1 {
        b[0] = a[0]
    } else {
        RFFT({a[0], a[2], ..., a[n-2]}, {u[0], u[1], ..., u[n/2-1]})
        RFFT({a[1], a[3], ..., a[n-1]}, {v[0], v[1], ..., v[n/2-1]})
        z = 1
        for j = 0 to n-1 {
            b[j] = u[j mod n/2] + z * v[j mod n/2]
            z = z * w
        }
    }
}
```

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

11.3 串行FFT蝶式分治算法

递归需要较大的栈开销，一旦数据量较大容易爆栈且函数调用会减慢运算时间，一般计算FFT采用迭代的蝶式FFT算法计算，并行计算书上的迭代版本如下：

```
for k = 0 to n-1 {
    c[k] = a[k]
}

for h = logn - 1 to 0 {
    p = 2^h
    q = n/p
    z = w^(q/2)
    for k = 0 to n-1 {
        if k mod p == k mod 2p {
            c[k] = c[k] + c[k+p]
            c[k+p] = (c[k] - c[k+p]) * z^(k mod p)
        }
    }

    for k = 1 to n-1 {
        b[r(k)] = c[k]
    }
}
```

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

11.4 SIMD-BF上的FFT算法及其时间分析

SIMD-BF上的FFT算法伪代码如下：

```
for i = 0 to n-1 par-do
    d[0][i] = a[i]
endfor

for r = 1 to log(n) do
    //j = exp(r,i)表示第r行第i列的w的指数部分
    //i二进制表示为t1t2...tr-1tr...tk则j的二进制表示为trtr-1...t1 0...0
    for 所有仅第r位不同且i的第r位取0的每对(i,j) par-do
        d[r][i] = d[r-1][i] + w^exp(r,i)*d[r-1][j]
        d[r][j] = d[r-1][i] + w^exp(r,j)*d[r-1][j]
    endfor

    for i = 0 to n-1 par-do
        计算exp(k,i) = j
        c[j] = d[k][i]
        b[i] = c[r(j)]
    endfor
endfor
```

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

PS：我并没有搞懂这个算法

蝶形网络连接无须考虑选录时间（why？），因为除了层次的for其他均是并行，时间复杂度为O(logn)