并行计算习题课

2024 年 5 月 14 日

课后作业 1

1. 调研并说明并行计算、分布式计算、云计算的异同

相同点

并行计算、分布式计算和云计算都是处理和执行大规模计算任务的技术。

不同点

分布式计算比并行计算具有更高的不确定性和独立性，具体表现在：缺乏全局状态信息、缺乏全局时间、缺乏统一的模式。云计算可以看作是分布式计算的一种特殊形式，它更注重服务的可用性、可扩展性和成本效率。同时它们在目的、设计和使用场景上有所不同。并行计算主要用于性能密集型任务；分布式计算解决的是规模和可靠性问题；云计算提供灵活、可扩展的服务，强调成本效益和资源的按需使用。

2. 并行计算应用调研

略（可考虑的应用：分子模拟、天气预报等）

3. 根据调研的应用需求，按预算 1000 万人民币购置计算设备，请给出采购清单（组件型号、数量、单价）

略（处理器、通信、存储等均需考虑）

课后作业 2

1. 课本第 139 页 4.2 12，140 页 4.11

4.2 两个 $N\times N$ 阶的矩阵相乘，时间复杂度为 $T_1=CN^3$ s，其中 C 为常数；在 n 节点的并行机上并行矩阵乘法的时间为 $T_n=(CN^3/n+bN^2/\sqrt{n})$ s，其中 b 是另一常数，第一项代表计算时间，第二项代表通信开销。

① 试求固定负载时的加速并讨论其结果。

② 试求固定时间时的加速并讨论其结果。

图 1: 4.2

1.1 固定负载时

（Amdahl 加速定律的基本出发点是: 对于很多科学计算, 实时性要求很高即在此类应用中时间是个关键因素, 而计算负载是固定不变的。为此在一定的计算负载下, 为达到实时性可利用增加处理器数来提高计算速度; 因为固定的计算负载是可分布在多个处理器上的, 这样增加了处理器就加快了执行速度, 从而达到了加速的目的。）

加速比为：$S = \frac{T_1}{T_n} = \frac{n}{1+\frac{b\sqrt{n}}{CN}}$

使用 Amdahl 定律公式：$S = \frac{P}{1+f(P-1)+\frac{W_o P}{W}} = \frac{n}{1+f(n-1)+\frac{b\sqrt{n}}{CN}}$

可得串行分量 f=0;

由加速比公式可以看出，并行程序性能受到通信开销影响，但是不受到串行分量影响。用于解决问题的处理器数目越多，加速比越大。

当问题规模较大（N 远大于 n）时，$\frac{b\sqrt{n}}{CN}$ 可以忽略不记，加速比与 n 成线性比例

当问题规模较小时，加速比与 $\sqrt{n}$ 成线性比例。

1.2 固定时间时

（Gustafson 加速定律的基本出发点是：1. 对于很多大型计算，精度要求很高，即在此类应用中精度是个关键因素，而计算时间是固定不变的。此时为了提高精度，必须加大计算量，相应地必须增加处理器数目才能维持时间不变。2. 在实际应用中没有必要固定工作负载而使计算程序运行在不同数目的处理器，增多处理器必须相应地增大问题规模才有实际意义。）

固定时间这种场景下，使用 Gustafson 定律来近似。

Gustason 加速比公式为：$S = \frac{f+p(1-f)}{1+\frac{W_o}{W}}$

处理器数目 $p = n$，串行分量 $f = 0$，负载 $W = CN^3$，额外开销 $W_o = \frac{bN^2}{\sqrt{n}}$，代入公式，得到 $S = \frac{n}{1+\frac{\frac{bN^2}{\sqrt{n}}}{CN^3}} = \frac{n}{1+\frac{b}{CN\sqrt{n}}}$

加速比与 n 成线性比例

固定时间内，用于解决问题的处理器数目越多，加速比越大。此时，并行程序的性能仅仅受到平均开销的限制（不存在串行瓶颈）。固定时间的加速比会随着开销的增大而降低。

4.11 一个在 p 个处理器上运行的并行程序加速比是 $p-1$，根据 Amdahl 定律，串行分量为多少？

图 2: 4.11

1.3 4.11，计算串行分量

根据 Amdahl 定律公式，可得等式：$S = \frac{1}{f+(1-f)/p} = p-1$，解得：$f = \frac{1}{(p-1)^2}$

2. A 是一个大小为 n 的布尔数组，欲求出 A[i] 为真的最小的下标 i，试设计一个常数时间的 PRAM-CRCW 并行算法。如果使用 PRAM-CREW 模型，运行时间如何?

算法 1 算法 PRAM-CRCW 上求最小下标

// 输入 A[1..n]，n 个不同元素

// M[1..n] 为中间处理用的布尔数组
begin

(1) for $1 \le i \le n$ par-do // 工作量 $O(n^2)$; 时间 $O(1)$
if A[i] then
M[i] = True
for $1 \le j < i$ par-do
if A[j] then M[i] = False
end for
else M[i] = False
end for

(2) for $1 \le i \le n$ par-do
if M[i] == 1 then res = i
end for
end

算法 2 算法 PRAM-CRCW 上求最小下标
// 输入 A[1..n]，n 个不同元素
// B[1..n], C[1..n][1..n], M[1..n] 为中间处理用的布尔数组
begin
(1) for $1 \le i \le n$ par-do // 工作量 $O(n)$; 时间 $O(1)$
if A[i]==True then B[i] = -i
else B[i] = -n
end for
(2) for $1 \le i, j \le n$ par-do // 工作量 $O(n^2)$; 时间 $O(1)$
if B[i] < B[j] then C[i, j] = 1
else C[i, j] = 0
end for
(3) for $1 \le i \le n$ par-do // 工作量 $O(n^2)$; 时间 $O(1)$，因为允许同时写
M[i] = 1
for $1 \le j \le n$ par-do
if C[i,j] == 0 then M[i] = 0
end for
(4) for $1 \le i \le n$ par-do
if M[i] == 1 then res = i
end

第（1）步将问题转化成求最大值，（2）（3）步为常数时间内求解最大值的算法。

算法 3 算法 PRAM-CRCW 上求最小下标

```
// 输入 A[1..n]，n 个不同元素
// B[1..n][1..n], C[1..n] 为中间处理用的布尔数组
begin
    (1) for 1 ≤ i, j ≤ n par-do // 工作量 O(n²); 时间 O(1)
        if i < j then B[i, j] = A[j] & !A[i] else B[i, j] = A[j]
    end for
    (2) for 1 ≤ i ≤ n par-do // 工作量 O(n²); 时间 O(1)，因为允许同时写
        (C[i] = B[1, i] & B[2, i] & ... & B[n, i] )
        C[i] = 1
        for 1 ≤ j ≤ n par-do
            if B[j,i] == 0 then C[i] = 0
    end for

    (3) for 1 ≤ i ≤ n par-do // 工作量 O(n)；时间 O(1)
        if C[i] then res = i
    end for
end
```

第（1）（2）步构造矩阵 C 对于位置 i，当且仅当先序位置不存在 True 且自身为 True 时，C[i] 为真。第（3）写入结果，只有一个 C 值为真。

使用 PRAM-CREW 时,由于不允许同时写,PRAM-CREW 可以 logn 倍模拟 PRAM-CRCW,其运行时间变为 O(logn)

3. 证明有向环基本并行着色算法正确性，给出该算法在 n 个处理器下的时间复杂度和工作量。

```
算法: SIMD-EREW模型
  //输入初始点着色c(i), 输出最终着色c'(i)
  begin
    for i=1 to n par-do
      (1)令k是c(i)和c(i的后继)的最低的不同二进制位
      (2)c'(i)= 2k+c(i)_k    //c(i)_k为c(i)的二进制第k位
    end for
  end
```

图 3: 基本并行着色算法

正确性证明

使用反证法。

假设存在节点 u 和 u 的后继节点 v 同色，即 $c'_u = c'_v$，记 v 的后继节点为 w

由算法可得，$2k_{uv} + c(u)_{k_{uv}} = 2k_{vw} + c(v)_{k_{vw}}$ (1)

即，$2(k_{uv} - k_{vw}) = -c(u)_{k_{uv}} + c(v)_{k_{vw}}$ (2)

又 $k_{uv} \in N, c(u)_{k_{uv}} \in 0, 1, c(v)_{k_{vw}} \in 0, 1$

所以 $-c(u)_{k_{uv}} + c(v)_{k_{vw}} \in -1, 0, 1$

所以要使等式 (2) 成立，则 $k_{uv} = k_{vw}, c(u)_{k_{uv}} = c(v)_{k_{vw}}$

即 $c(u)_{k_{uv}} = c(v)_{k_{uv}}$，与 k_{uv} 为 u 和 v 的最低的不同二进制位相矛盾，所以假设不成立，命题得证。

在 n 个处理器下的时间复杂度和工作量

时间复杂度：需要逐位进行比较，最多比较 logn 次，所以时间复杂度为 O(logn)

工作量：O(nlogn)

课后作业 3

1. 教材 P442 15.3

15.3 填写空白处，使下述两代码段完全等效。

① float data[1024];
MPI_Datatype floattype;
MPI_Type_vector(10,1,32,MPI_FLOAT,&floattype);
MPI_Type_commit(& floattype);
MPI_Send(data,1,floattype,dest,tag,MPI_COMM_WORLD);
MPI_Type_free(&floattype);

② float data[1024],buff[10];
for(_____;_____;i++) buff[i]=data[_____];
MPI_Send(buff,_____,MPI_FLOAT,_____,_____,_____);

图 4: 15.3

for(int i = 0; i < 10; i++) buff[i] = data[i*32];
MPI_Send(buff, 10, MPI_FLOAT, dest, tag, MPI_COMM_WORLD);

2. 教材 P447 15.13 • 不需要提交程序代码，性能高低不影响评分。• 自行编写程序，个人电脑运行，基于结果回答提问

15.13 (Buffon-Laplace 针问题)设想一个长为 l 的针掉在一个等距平行线网格上(如图 15.19 所示),每个格的长和宽分别是 a 和 b。针至少落在一根线上的概率为

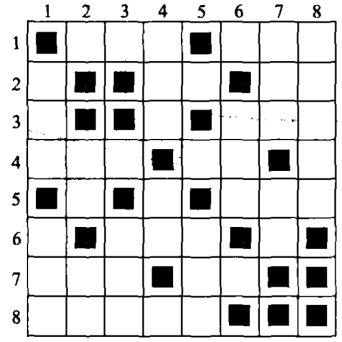

图 15.18 稀疏矩阵结构

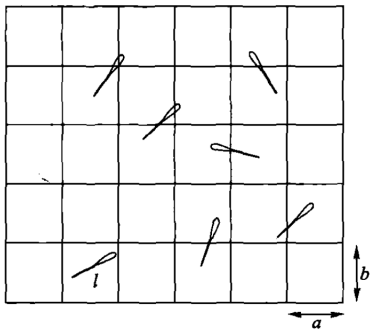

图 15.19 Buffon-Laplace 针问题

$$P(l,a,b)=\frac{2l(a+b)-l^2}{\pi ab}$$

我们可以用蒙特卡洛模拟法进行投针,从而来估计 π 的值。

① 用 C 语言写一个串行的 Buffon-Laplace 针问题的仿真程序。程序打印 π 的值。当针的数量是一百万时,运行模拟的时间是多少? π 的位数是多少?

② 写两个并行的 Buffon-Laplace 针问题的仿真程序,一个用 OpenMP,一个用 MPI。OpenMP 和 MPI 版本基于(1)中的 C 语言程序。运行程序,针的数量分别为 $10^4,\cdots,10^7$,处理器数分别为 2,4,6,8。你的代码规模如何? 对于针数量为 10^3 和 10^7,绘制运行时间与所用处理器数的关系图。可扩展性有什么不同? 并做出解释。

图 5: 5.13

2.1

$time = 0.713384s; \pi = 3.141615$

2.2

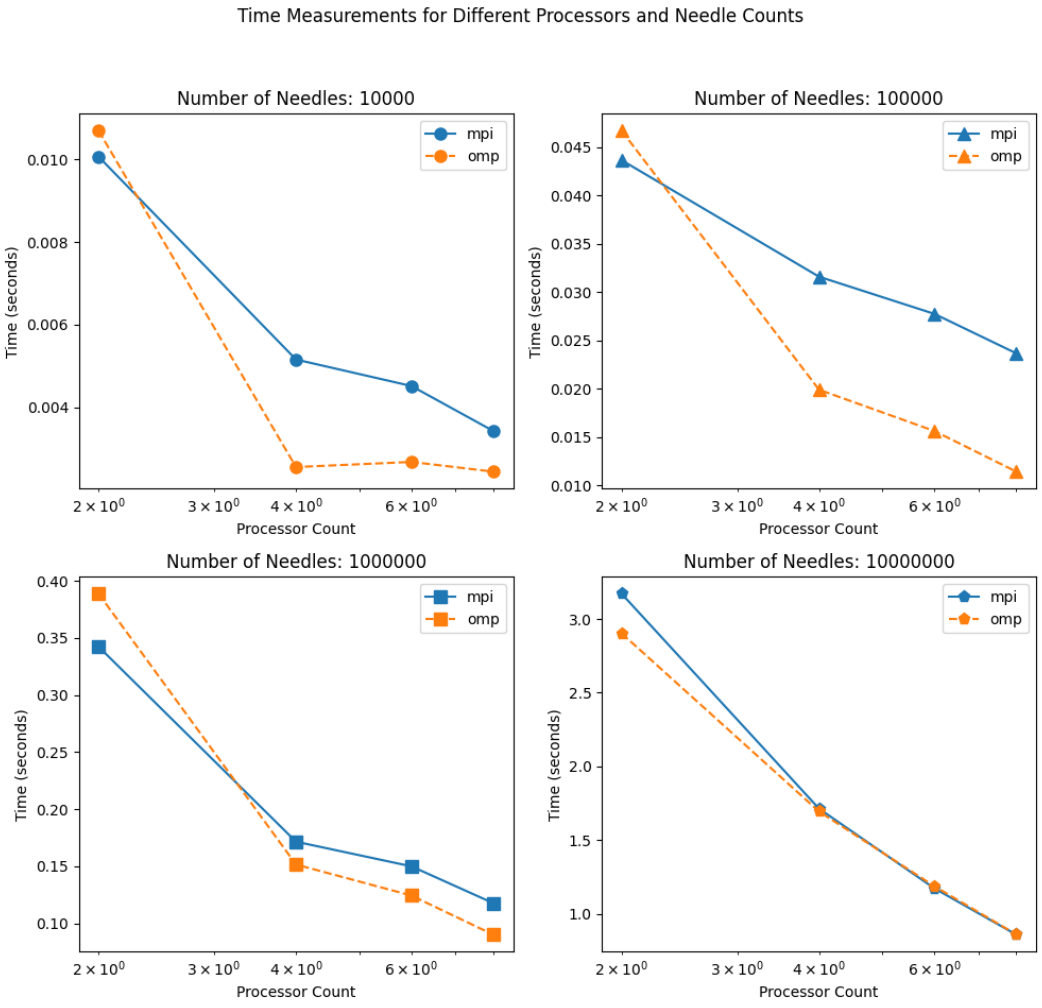

图 6: 关系图

可以看到在问题规模较小的情况下，omp 实现的性能高于 mpi 实现的性能，这是由于 omp 的线程开销较小。当问题规模较大的时，额外开销占比低，此时两种实现的性能相当。

课后作业 4

教材 P265 分析算法 10.6 时间复杂度

5. 超立方多计算机上高斯主元消去法

假定采用按行划分法分配处理器，具体实现方法如算法 10.6。其中，$marked_{1:n/p}$指明哪些行已是主元行，$pivot_{1:n/p}$指明迭代步数，value 表示主元值，winner 是控制主元行的处理器。

算法 10.6　超立方多计算机上求解稠密线性方程组高斯主元消去算法

输入：$\boldsymbol{A}_{n\times n}$，$\boldsymbol{b}=(b_1,\cdots,b_n)^{\mathrm{T}}$

输出：$\boldsymbol{x}=(x_1,\cdots,x_n)^{\mathrm{T}}$

Begin

for all P_{id}，**where** $1\leqslant id\leqslant p$ **do**

(1) **for** $i=1$ **to** n/p **do** /* 初始化；开始时所有行均非主元行 */

marked$_i$ = 0

endfor

(2) **for** $i=1$ **to** $n-1$ **do** /* 各处理器找主元候选者 */

(2.1) value = 0

(2.2) **for** $j=1$ **to** n/p **do**

if marked$_j$ = 0 & $|a_{ji}|$ > value **then**

value = $|a_{ji}|$

picked = j

endif

endfor

(2.3) winner = id /* 标记控制主元行的处理器 */

(2.4) MAX. TOURNAMENT (id, value, winner)

/* 调用 MAX 选全局最大主元 */

(2.5) **if** id = winner **then**

(i) $marked_{picked}=1$

(ii) $pivot_{picked}=i$

(iii) **for** $j=i$ **to** n **do**

temp. vector$_j$ = $a_{picked,j}$

endfor

(iv) temp. $vector_{n+1}=b_{picked}$

endif

(2.6) HC. BROADCAST(id, winner, temp. $vector_{i:n+1}$)

/* 主元处理器播送主元给其他处理器 */

(2.7) **for** $j=1$ **to** n/p **do** /* 未选中的行中消去列 i 之元素 */

图 7: 10.6.1

```
                    if marked_j = 0 then
                    (i) temp = a_ji / temp. vector_i
                    (ii) for k = i + 1 to n do
                            a_jk = a_jk - temp. vector_k × temp
                         endfor
                    (iii) b_j = b_j - temp. vector_{n+1} × temp
                    endif
                endfor
    endfor
    (3) for i = 1 to n/p do / * 定位从未被用作主元行的行 * /
            if marked_i = 0 then
              pivot_i = n
              break
            endif
        endfor
    endfor
End
MAX. TOURNAMENT(id, value', winner) / * 被主程序所调用的过程 * /
      Reference: id, value', winner
      局部变量: temp. value', temp. winner
Begin
    for i = 0 to log p - 1 do
      (1) partner = id ⊕ 2^i
      (2) [partner]temp. value' ← value'
      (3) [partner]temp. winner ← winner
      (4) if temp. value' > value' then
                value' = temp. value'
                winner = temp. winner
          endif
    endfor
End
```

图 8: 10.6.2

对各步时间分别分析:

(1) O(n/p)

(2) 运行 O(n) 次

(2.1) $O(1)$,

(2.2) $O(n/p)$,

(2.3) $O(1)$,

(2.4) 这一步处理器进行 $O(\log p)$ 次的一对一通讯,由于该通信均为和相邻节点间的通信,每次通信的开销为: $t_s + mt_w$, 其中消息大小为 2, 代入得这步的时间复杂度为: $O(t_s \log p + 2t_w \log p) = O(\log p)$

(2.5) $O(n)$,

(2.6) 超立方体中的一到多通信开销为: $(t_s + mt_w)logp$, 其中消息大小为 n, 代入得这步的时间复杂度为 $O((t_s + nt_w)\log p = O(n \log p)$,

(2.7) $O(n^2/p)$

所以, 第 (2) 步的时间复杂度为 $O(n/p+logp+n+nlogp+n^2/p)*O(n) = O(n^2logp+n^3/p)$

(3) $O(n/p)$

总的时间复杂度为: $O(n^3/p+n^2logp)$

教材 P309 11.5

11.5 参照算法 11.1，设计一个单处理机上时间为 $O(n\log n)$ 的离散傅氏逆变换算法，并以 $n=8$ 为例，画出其逆变换蝶式计算流图。

图 9: 11.5

串行FFT非递归算法

- 算法11.1: SISD上FFT迭代算法

输入: A=(a_0,a_1,⋯,a_{n-1}); 输出: B=(b_0,b_1,⋯,b_{n-1})

```
Begin
  (1)for k=0 to n-1 do ck=ak endfor      //初始化
  (2)for h=logn-1 to 0 do
       for k=0 to n-1 do
         (2.1)p=2^h   (2.2)q=n/p   (2.3)z=ω^(q/2)   //先算出z=ω^1,以后每次z=z×z
         (2.4)if ( k mod p = k mod 2p) then        //注意(i)和(ii)同时执行，读取相同的ck
               (i) ck=ck+ck+pz
               (ii)ck+p=(ck-ck+p)z^(k mod p)
             endif
       endfor
     endfor
  (3)for k=0 to n-1 do bk=cr(k) endfor      //r(k)为k的位序反
End
```

例如：n=8

h=2, p=4, q=2, z=ω^1

h=1, p=2, q=4, z=ω^2

h=0, p=1, q=8, z=ω^4

算法时间复杂度：T(n)=O(nlogn)

16

图 10: SISD FFT 算法

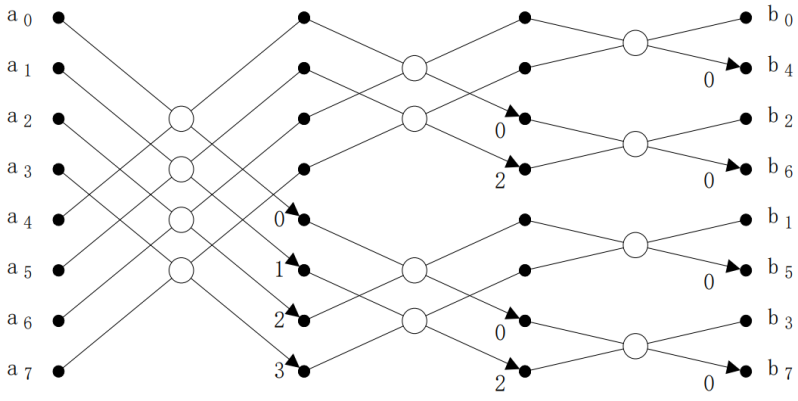

图 11: FFT 计算流图

仿照 SISD FFT 算法，将 $z = \omega^{q/2}$ 替换成 $z = \omega^{-q/2}$，并将第 (3) 步中 $b_k = c_{r(k)}$ 改为 $b_k = c_{r(k)}/n$。
计算流图同样对 FFT 的做相应的修改即可：将其中的数值取负，同时在最后一步除以 n。

随堂作业 1

a 给出 n-point DFT 流水线算法的时间复杂度和工作量

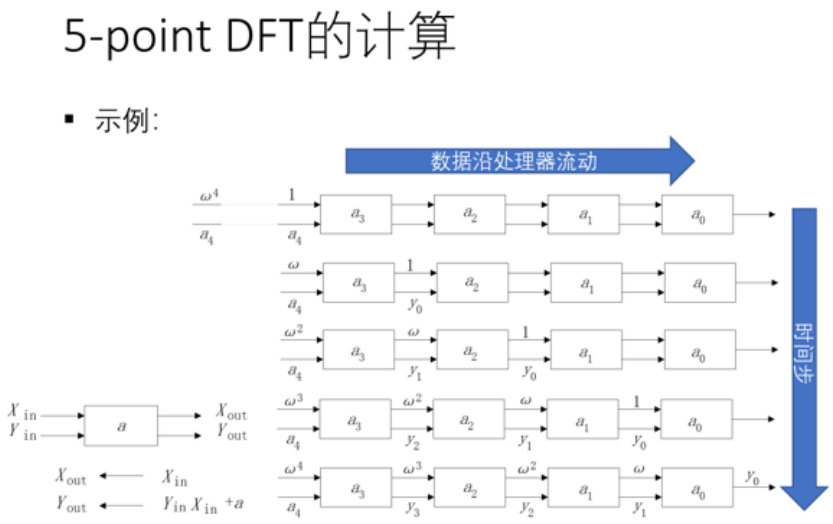

图 12: 5-point DFT

时间开销：$t(n) = O((t_s + 2t_w + 1)(n-1)) = O(n)$
工作量：$p(n) = (n-1), c(n) = p(n)t(n) = O(n^2)$

b 给出一维线性阵列上心动排序算法处理长为 n 的数组的时间复杂度和工作量

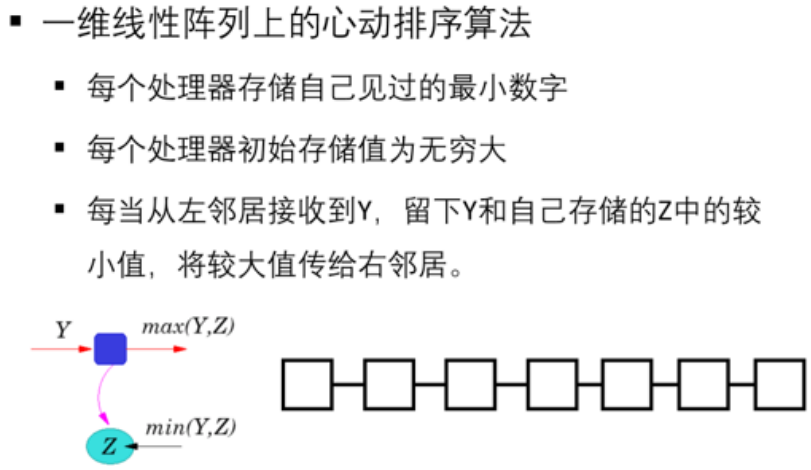

图 13: 心动排序算法

时间复杂度：$t(n) = O((t_s + t_w + 1)n) = O(n)$
工作量：$p(n) = n, c(n) = p(n)t(n) = O(n^2)$

随堂作业 2

a 给出二维环绕网孔上 CT 选路带状划分矩阵转置运行时间

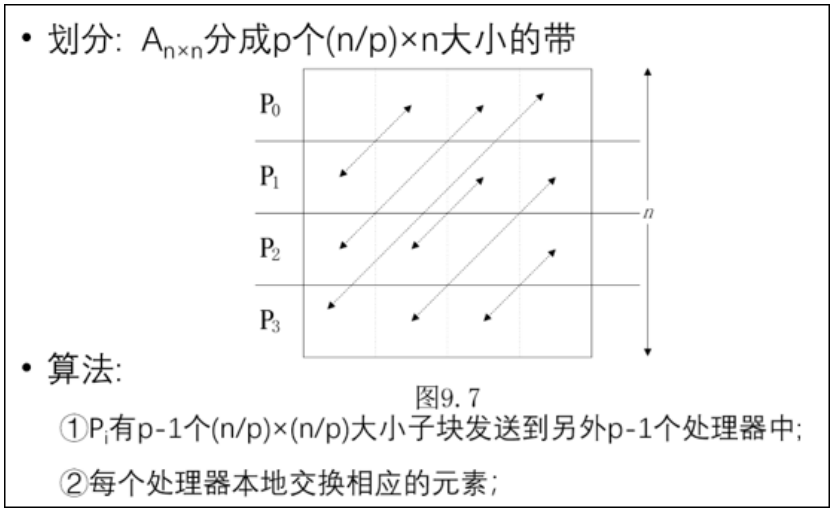

图9. 7

图 14: 矩阵转置

通信开销（二维环绕网络一对一传播：$t_s + mt_w$）：$(p-1)(t_s + (n/p)^2 t_w)$
计算开销：n^2/p
总运行时间：$(p-1)(t_s + (n/p)^2 t_w) + n^2/p$

b 给出二维环绕网孔上 CT 选路 Fox 乘法运行时间

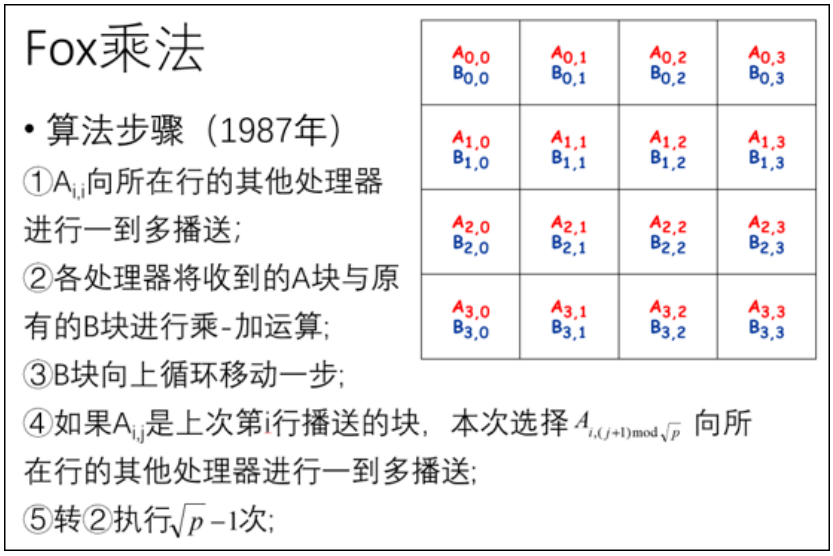

图 15: Fox 乘法

步骤 1 开销：$(t_s + t_w n^2/p) \log \sqrt{p}$
步骤 2 开销：$n^3/(p\sqrt{p})$
步骤 3 开销：一对一传播，$(t_s + t_w n^2/p)$
步骤 4 开销：$(t_s + t_w n^2/p) \log \sqrt{p}$
步骤 2-4 重复 $\sqrt{p}$ 次
总开销：$(t_s + t_w n^2/p)(1/2 \log p + \sqrt{p}(1 + 1/2 \log p)) + n^3/p$