第9章 算法设计

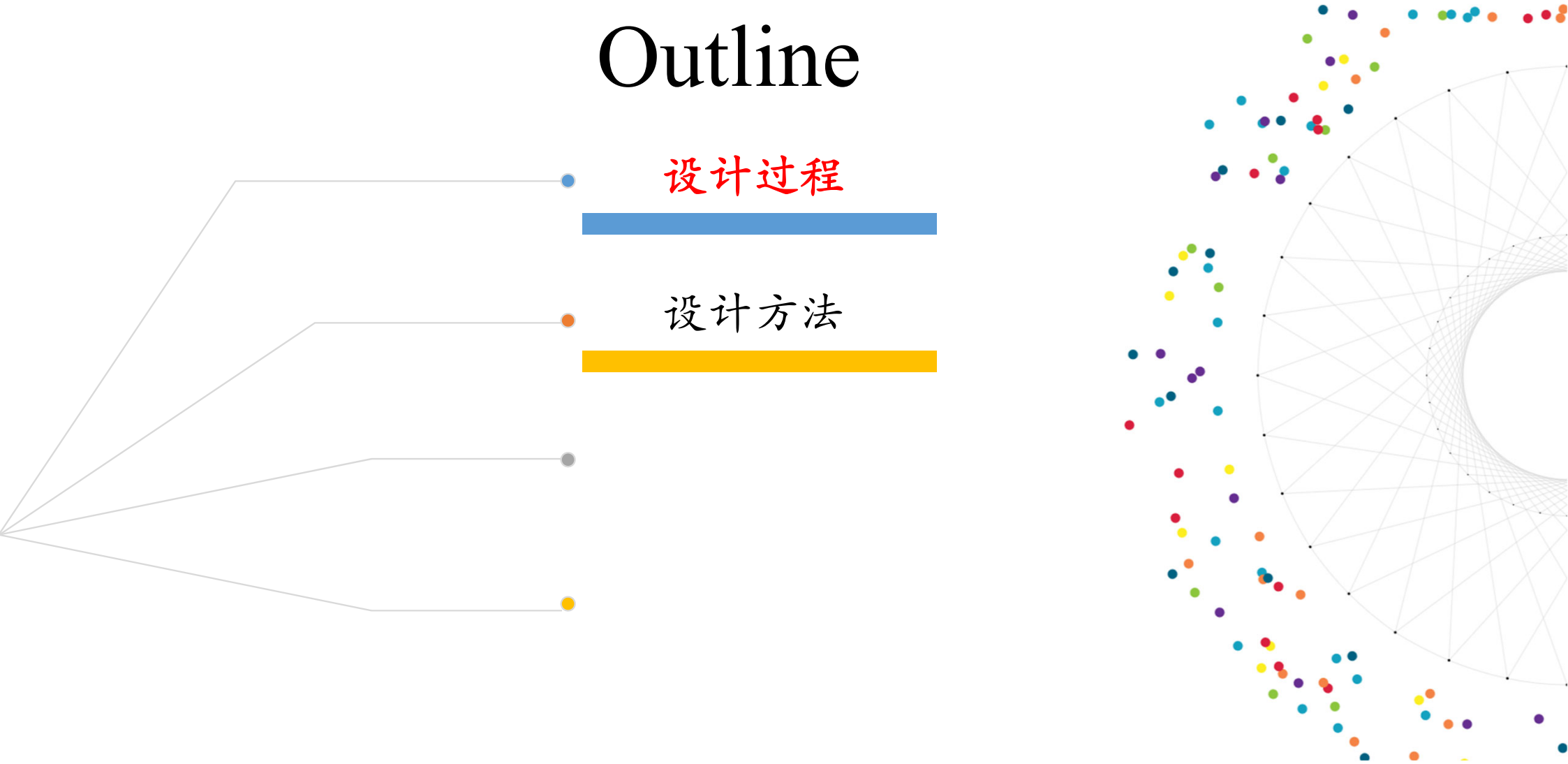
授课教师：胡俊成

jchu@jlu.edu.cn

Outline

设计过程

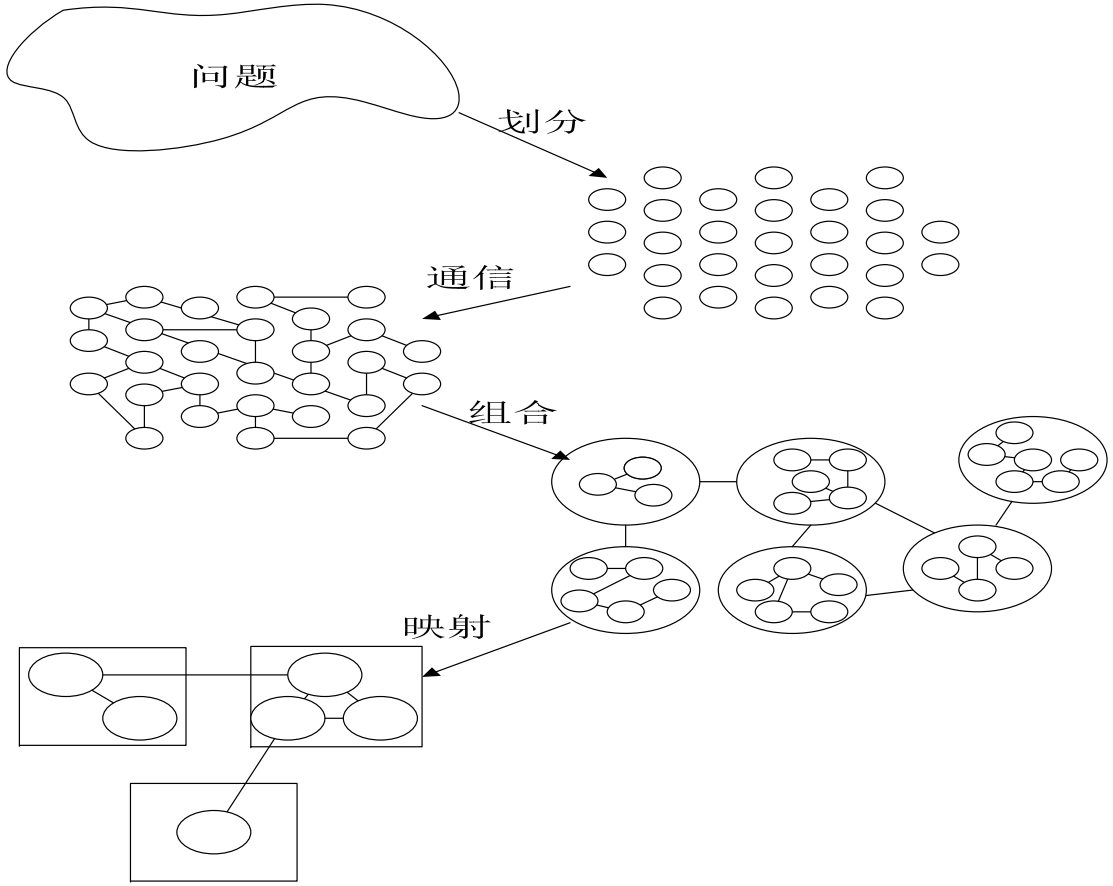
设计方法



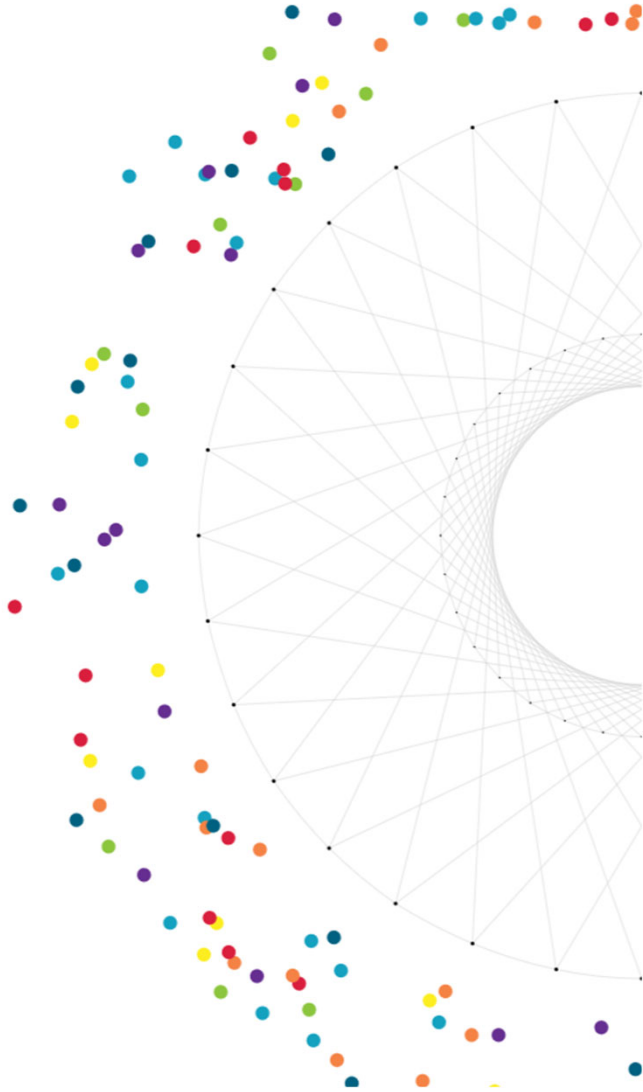
PCAM设计过程

- 并行算法设计过程的四个阶段
- 划分(Partitioning)将大任务分解成小任务，尽量开拓并发性；
- 通讯(Communication) 确定诸任务间的数据交换，监测任务划分的合理性；
- 组合(Agglomeration) 依据任务的局部性优化通信成本或提高性能，必要时将一些小任务组合成更大的任务
- 映射(Mapping) 将每个任务分配到处理器上执行，监测其执行性能以备下一轮迭代优化。

PCAM生命周期



Outline



任务的划分

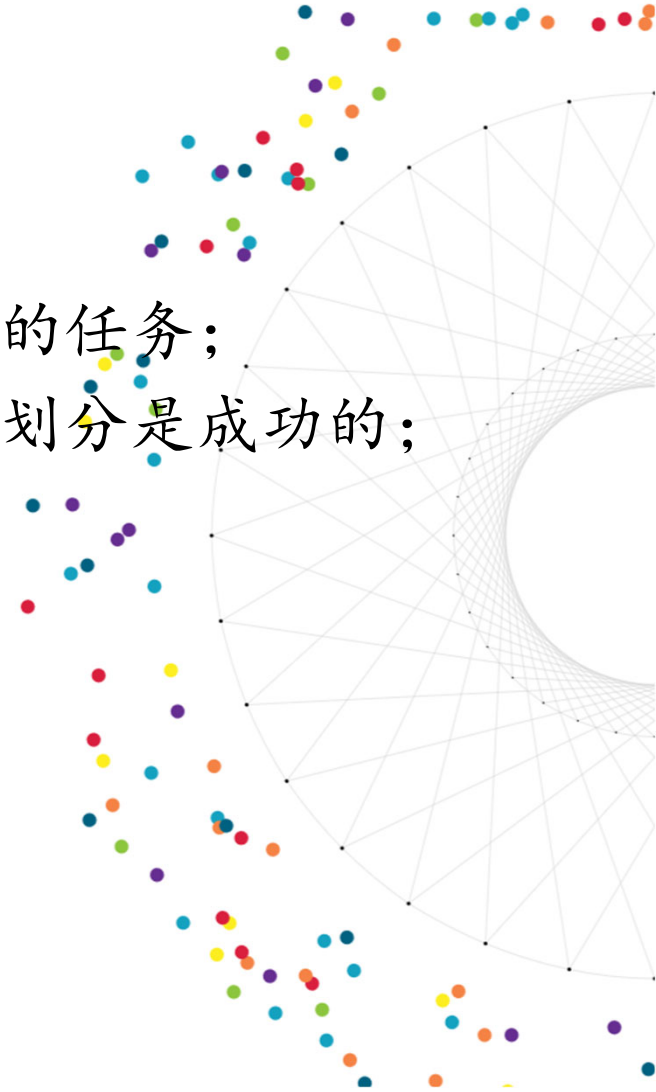
- 就是将原始计算问题分割成一些小的计算任务，以充分开拓算法中存在的并行性；
- 先进行数据分解(域分解domain decomposition)，再进行计算功能的分解(功能分解functional decomposition)
- 划分的要点是力图避免数据复制和计算复制，使数据集和计算集互不相交；
- 划分阶段忽略处理器数目和目标机器的体系结构；

域分解

- 划分的对象是数据，可以是算法的输入数据、中间处理数据和输出数据；
- 1优先集中划分最大的数据，将数据分解成大致相等的小数据片；
- 2划分时考虑数据上的相应计算操作，在计算的不同阶段，可能需要对不同的数据结构进行操作，或者需要对同一数据结构做不同的分解；
- 3如果一个任务需要别的任务中的数据，则会产生任务间的通讯；
- 4 划分的主要原理是时空局部性原理，注意时空窗口的粒度；

功能分解

- 划分的对象是计算，将所有计算过程划分为不同的任务；
- 划分后，如果不同任务所需的数据不相交的，则划分是成功的；



划分判据

- 划分是否具有灵活性？（任务数多倍于处理器数）
- 划分是否避免了冗余计算和存储？（划分至少满足局部极小）
- 划分任务尺寸是否大致相当？（负载均衡）
- 任务数与问题尺寸是否成比例？（问题尺寸的增加应该引起任务数量的增加，而非任务尺寸的增加）

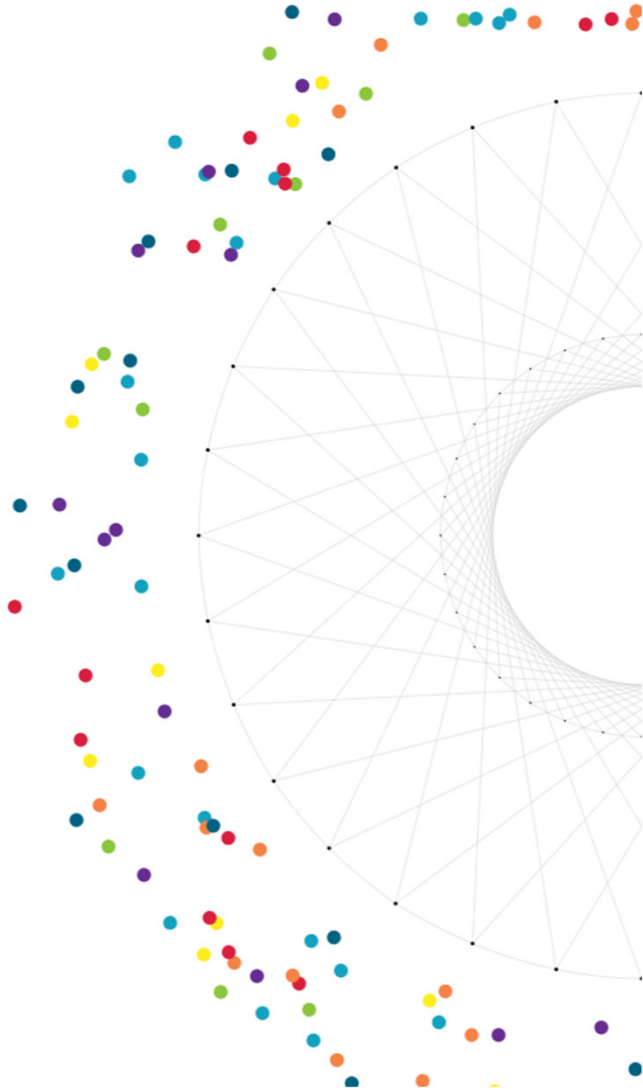
Outline

1 划分

2 通信

3 组合

4 映射

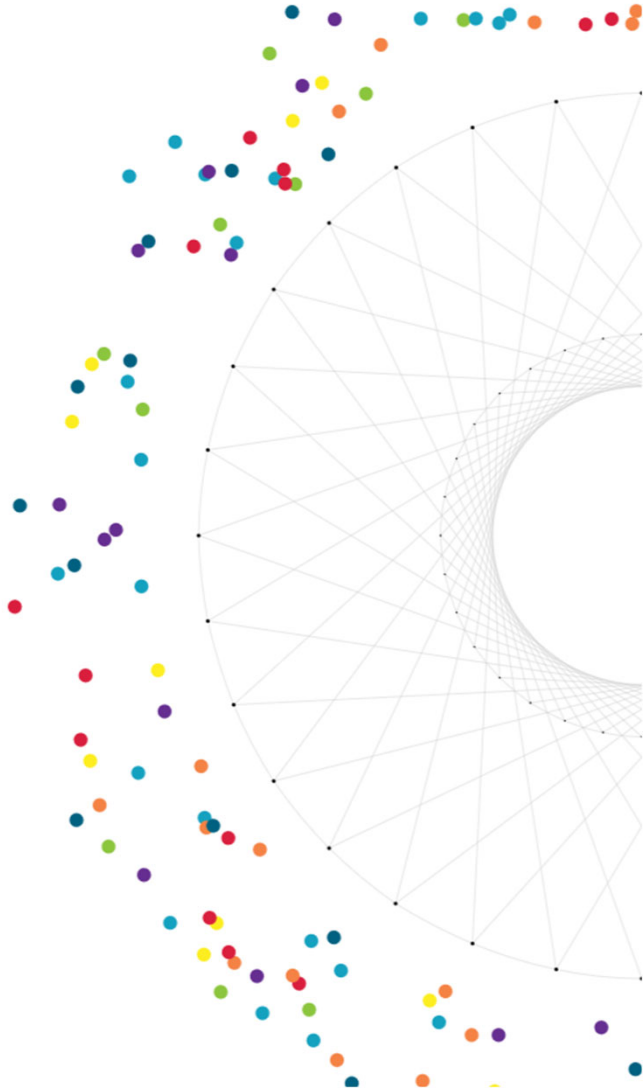


任务的通信

- 通信，就是为了实现并行计算，诸任务之间所进行的数据传输。
- 划分产生的诸任务，一般不能完全独立执行，需要在任务间进行数据交流，从而产生了通信；
- 通信通常是数据从“生产者”向“消费者”的流动，“生产”和“消费”都是操作，而且具有时序关系，因而通信操作的划分无法通过域分解来确定，只能通过功能分解来确定；
- 诸任务本是并发执行的，通讯则限制了这种并发性；

通信模式

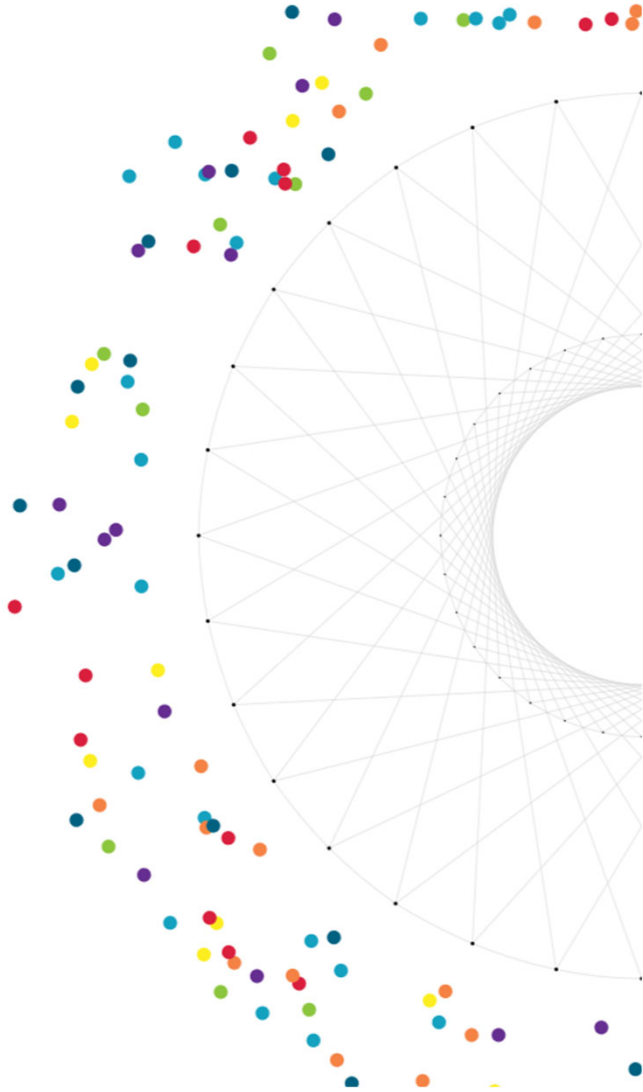
- 局部/全局通讯（空间局部性）
- 结构化/非结构化通讯（拓扑）
- 静态/动态通讯（身份和角色）
- 同步/异步通讯（是否阻塞）



通信判据

- 所有任务是否执行大致相当的通信量?
- 是否尽可能的将全局通信化成局部通信?
- 各个通信操作是否能并行执行?
- 通信操作与同步点的距离是否合适？便于异步并行执行

Outline



任务的组合

- 组合是由抽象到具体的过程，使将组合的任务能在一类并行机上有效的执行；
- 合并小尺寸任务，减少任务数。如果任务数恰好等于处理器数，则也完成了映射过程；
- 通过增加任务的粒度和重复计算，可以减少通信成本；
- 保持映射和扩展的灵活性，降低软件工程成本，注意负载均衡；

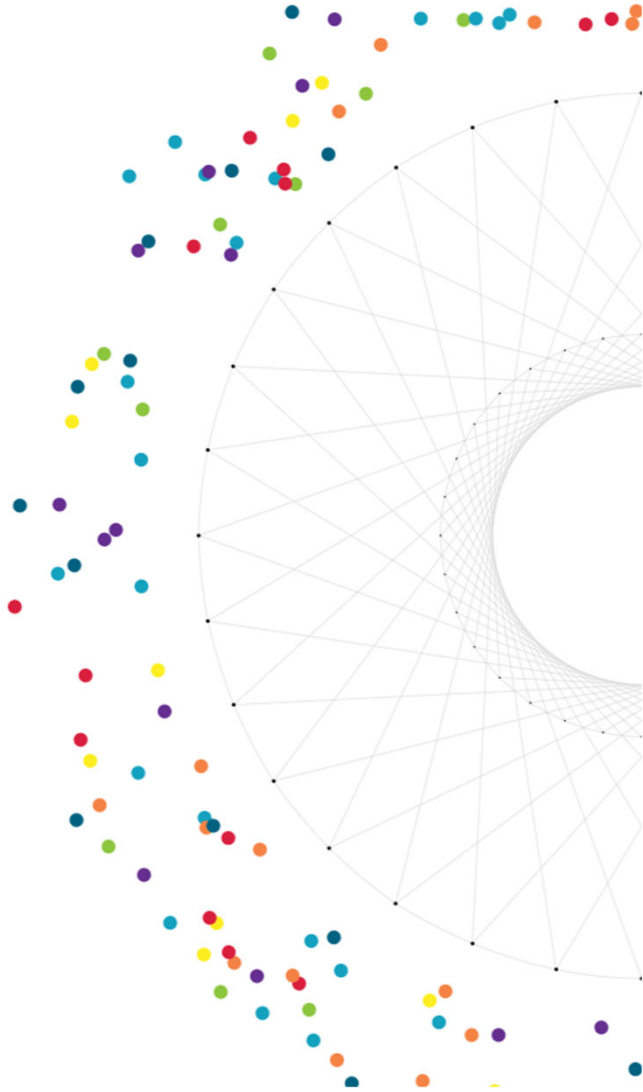
粒度控制

- 大量细粒度的任务未必能产生有效的并行算法，可能反而增加通信代价和任务调度代价。
- 表面-容积效应：一个任务的通信需求与它所操作的数据子域的表面积成正比，而这个任务的计算需求与它所操作的体积（=数据子域表面积*计算操作的深度）成正比；
- 体积不变的前提下，增加计算操作深度（又称重复计算或者冗余计算），能够减少子域表面积，进而减少通信量；
- Tradeoff：T总=T计算+T通信；

组合判据

- 通过组合增加粒度是否减少了通讯成本?
- 用重复计算换取通信代价是否已权衡了其得益?
- 组合后是否保持了灵活性和可扩放性?
- 组合的任务数是否与问题尺寸成比例?
- 组合后的各个任务是否保持了类似的计算和通讯代价?
- 有没有减少并行执行的机会?

Outline

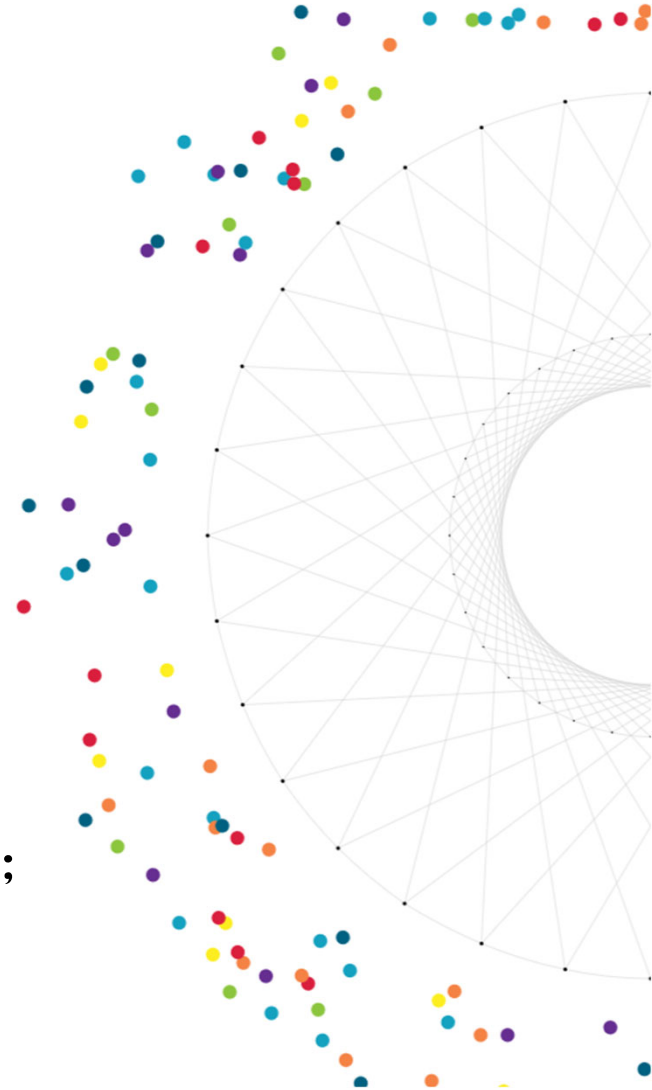


任务的映射

- 映射的目标：每个任务要映射到具体的处理器，减少算法的执行时间；映射实际是一种权衡，属于NP完全问题；
- 任务数大于处理器数时，域分解引入负载平衡问题，功能分解引入任务调度问题；
- 并发的任务放在不同的处理器，频繁通信的任务放在同一处理器

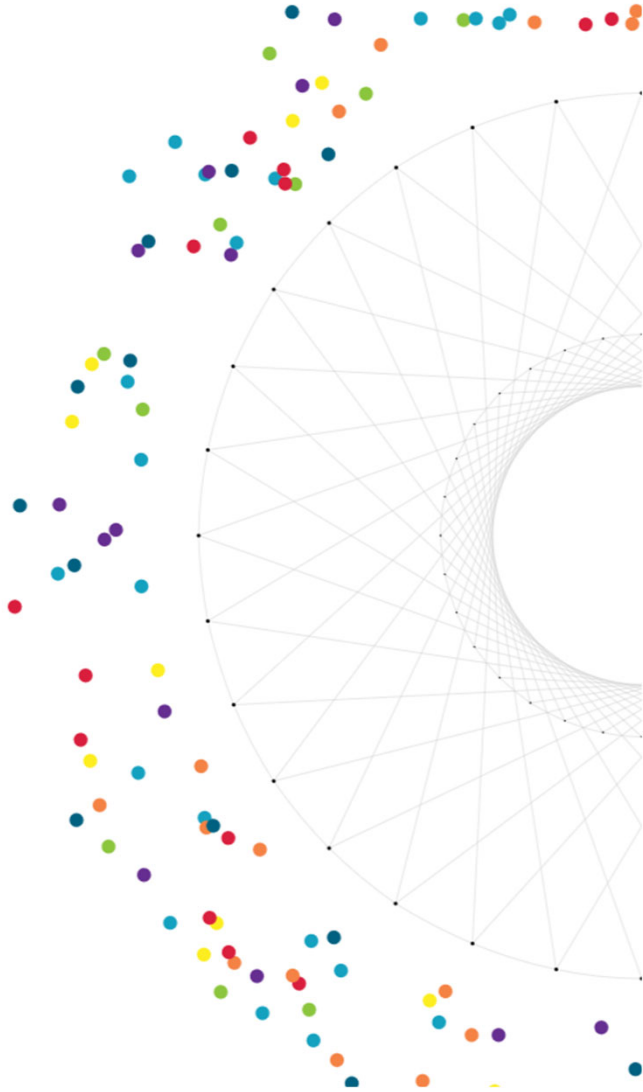
负载平衡算法

- 负载平衡分类
  - 静态的：事先确定；
  - 概率的：随机确定；
  - 动态的：动态确定；
- 基于域分解的：
  - 递归对剖：多维空间上递归的选一维进行对剖；
  - 局部算法：与邻居比较决定是否把任务迁给邻居；
  - 概率方法：满足某概率条件的随机数发生器进行投放；
  - 循环映射：等概率的特例；



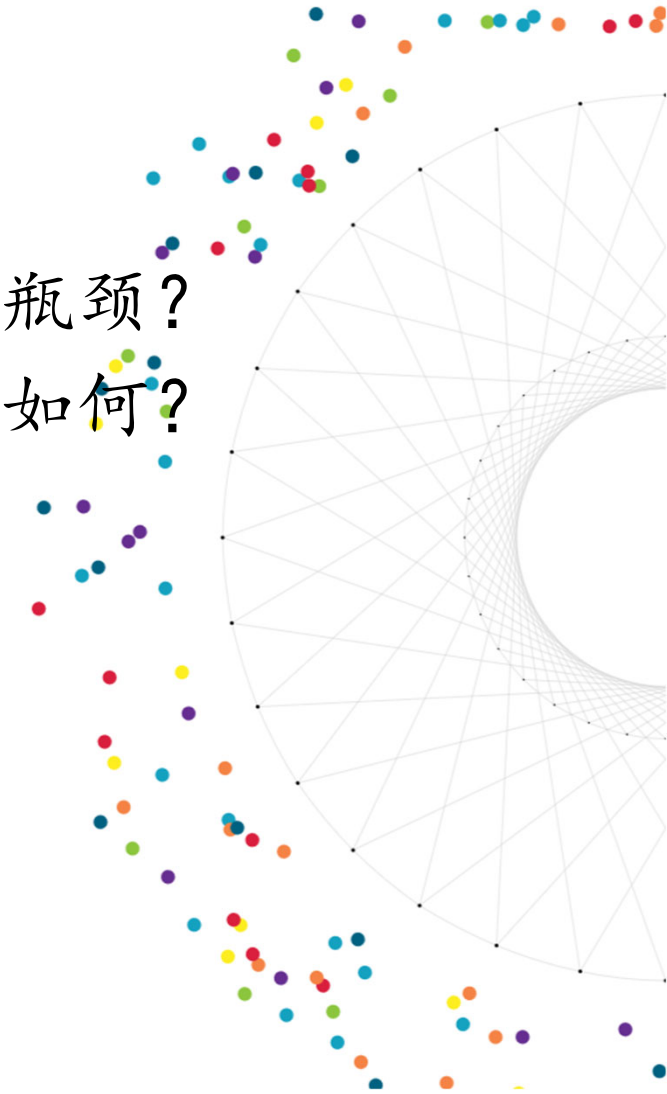
任务调度算法

- 集中式任务池调度
  - 经理/雇员模式，总经理/区域经理/雇员模式
- 分布式任务池调度
  - 企业联合，各自异步，跨池交互，流水线



映射判据

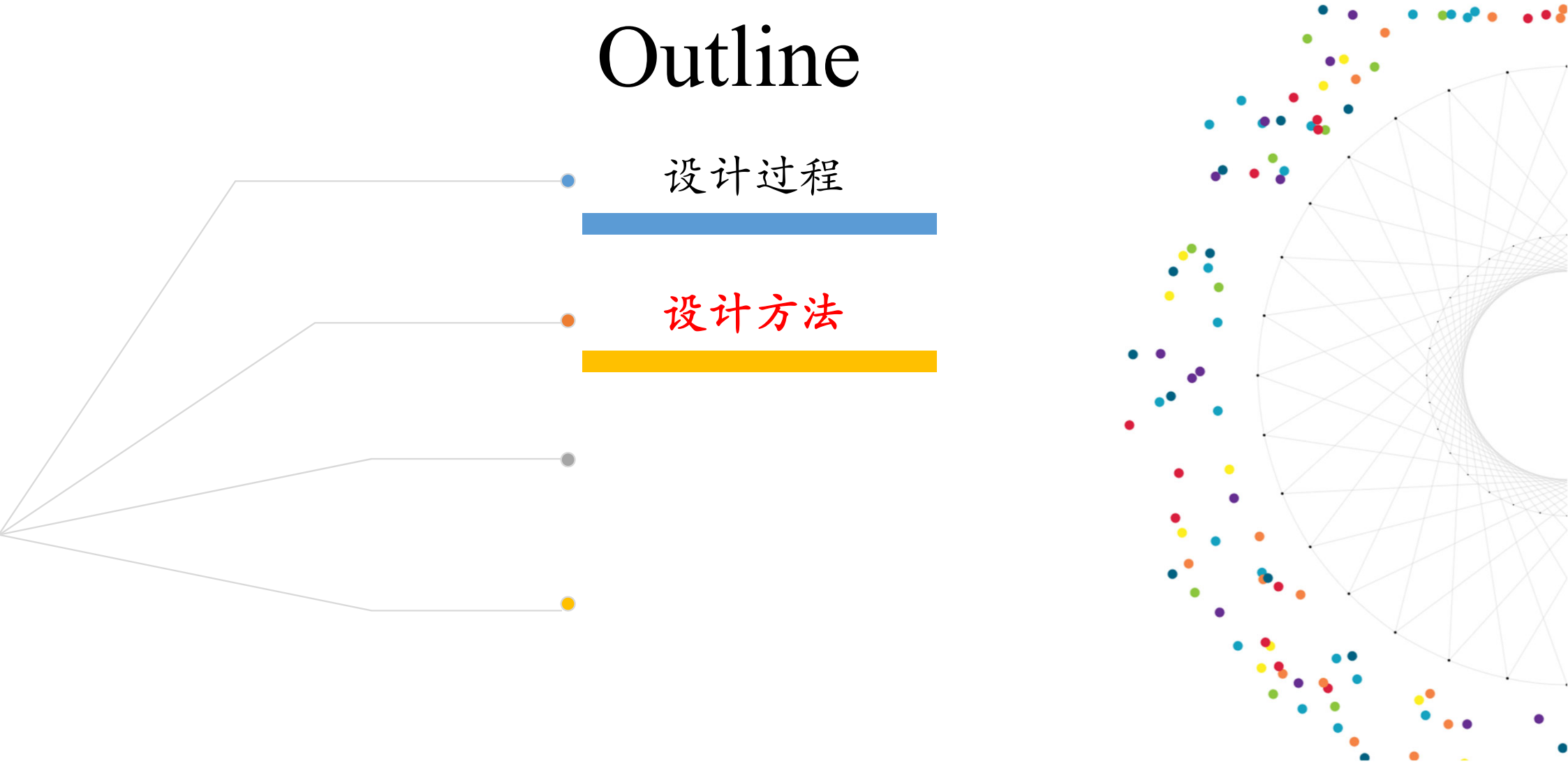
- 采用集中式负载平衡方案，是否存在经理瓶颈？
- 采用动态负载平衡方案，调度策略的成本如何？



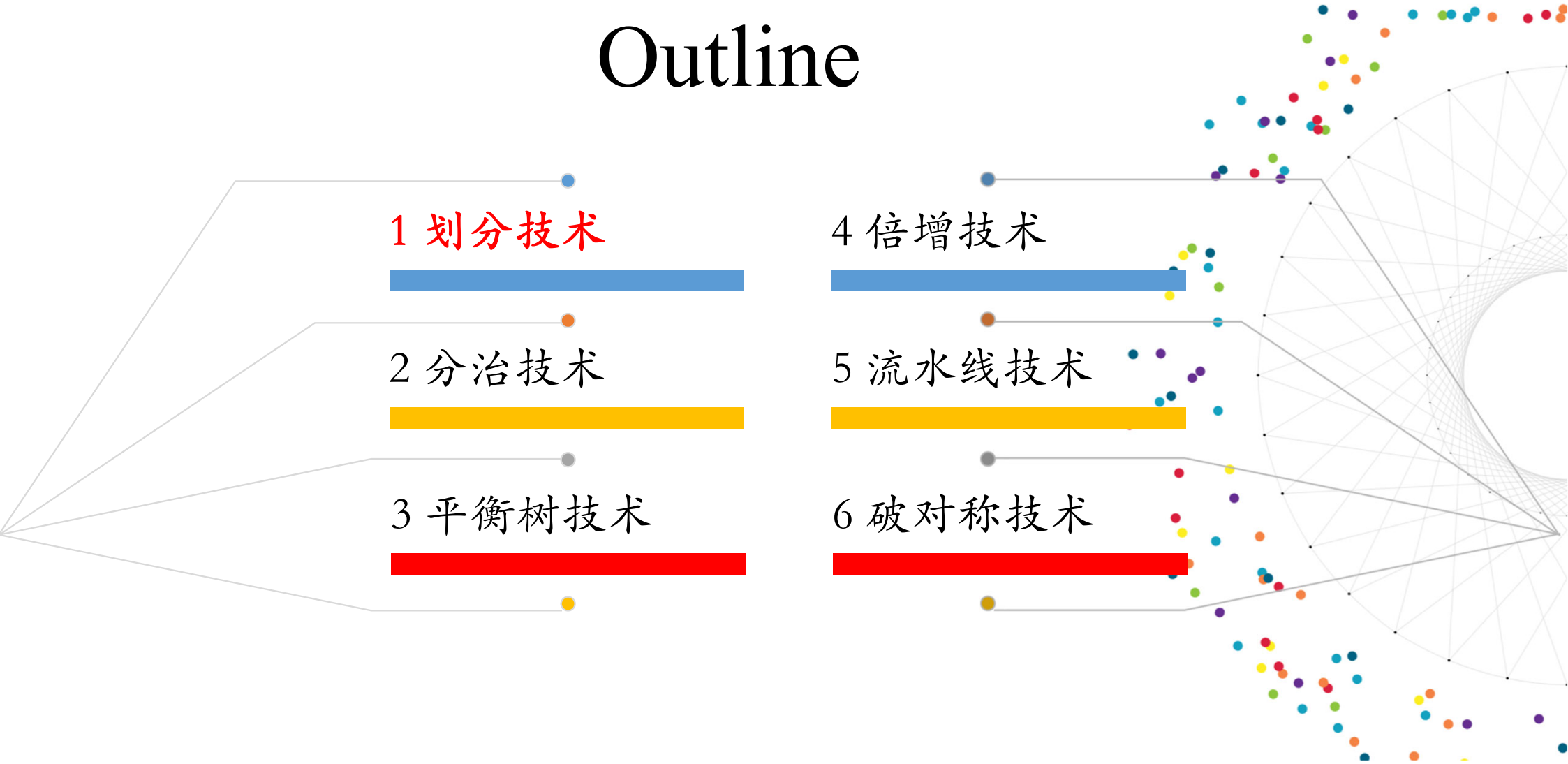
Outline

设计过程

设计方法

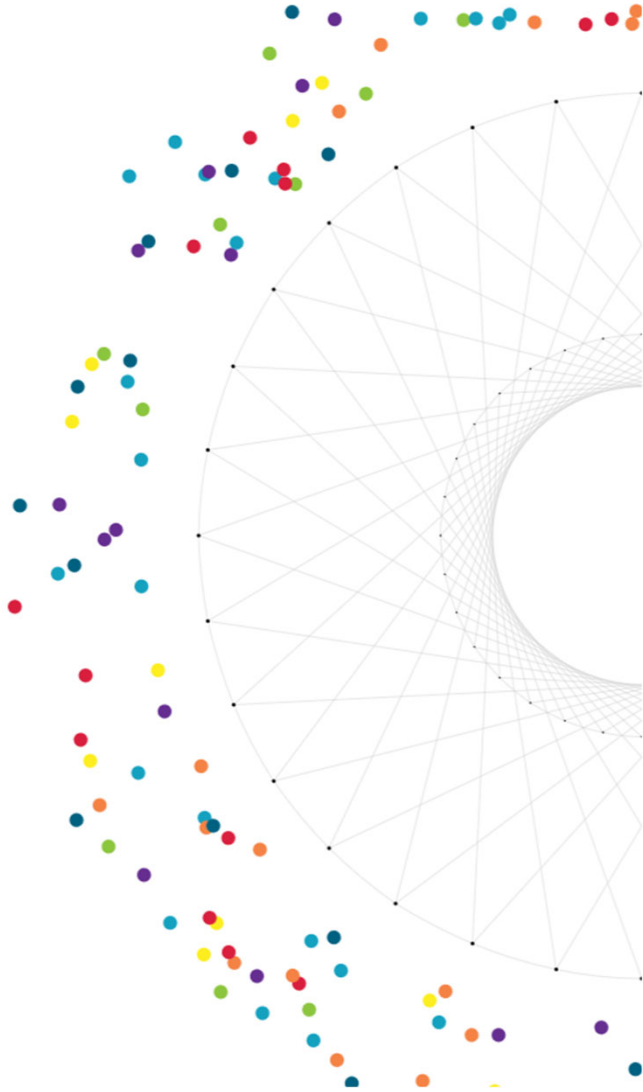


Outline



1 划分技术

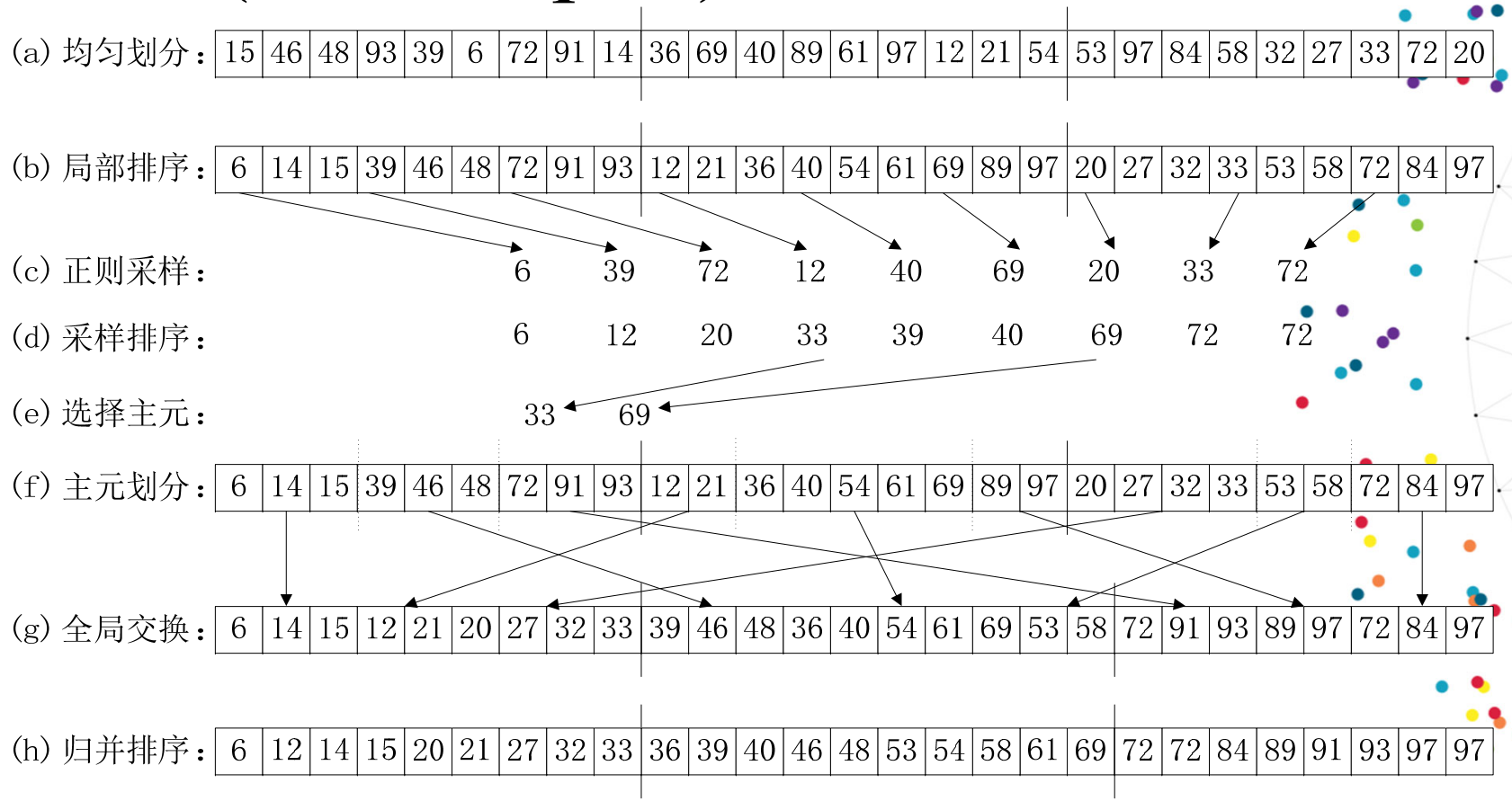
- 1.1 均匀划分法
- 1.2 平方根划分
- 1.3 对数划分
- 1.4 功能划分



1.1均匀划分

- n个元素分成p组，每组n/p个元素在一个处理器上处理。
- 均匀划分示例：数组A[1..n]在MIMD-SM模型上进行PSRS（Parallel Sorting by Regular Sampling）并行排序
- (1)均匀划分：将n个元素均匀划分成p段，每个处理器处理n/p个元素；
- (2)局部排序：每个处理器调用串行排序算法，排序n/p个数；
- (3)正则采样：每个处理器从自己的有序段中选取p个样本元素；
- (4)样本排序：用一台处理器对p*p个样本元素进行串行排序；
- (5)选择主元：用一台处理器从第(4)步排好序的样本序列中选取p-1个主元，并广播给所有处理器；
- (6)主元划分：每个处理器按照这p-1个主元将各自早已排好序的有序段划分成p段
- (7)全局交换：每个处理器将自己管理的p个段，通过全局交换按段号进行重组。
  - 段1@U1##段1@U2##...##段1@Up--> U1
  - 段2@U1##段2@U2##...##段2@Up--> U2
  - …
  - 段p@U1##段p@U2##...##段p@Up--> Up
- (8)归并排序：每个处理器对各自收到的重组后的元素，执行归并排序。

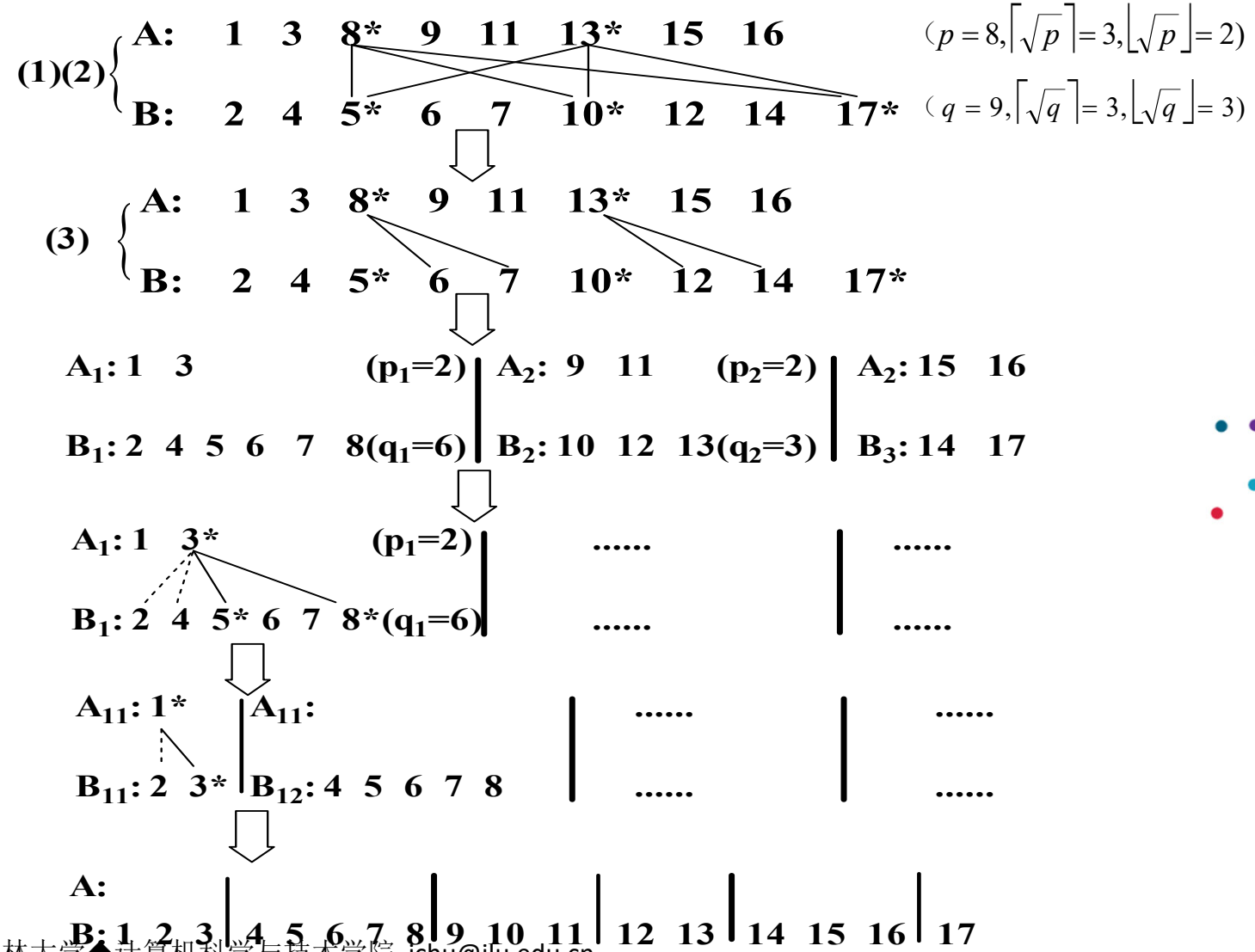
PSRS(N=27，p=3)排序过程如下：



1.2平方根划分

- n个元素分成√n个区段，每个区段√n个元素在一个处理器上处理。
- 平方根划分示例：两个长度为n的有序段在SIMD-CREW模型上进行Valiant归并排序
- (1)方根划分：将A和B分别划分成√n个区段；第i*√n个元素都标记为划分元素；
- (2)段间比较：保持B不动，逐个取A中的划分元素同B中的所有划分元素比较，用于确定每个A划分元素将会对应插入B中的哪一区段；
- (3)段内比较：针对每个A划分元，将其与B对应区段内所有元素进行比较，并插入区段内适当位置，标记插入的每个A划分元；
- (4)递归归并：使用所有A划分元再次将B重新分成多个区段，A中剩余的所有元素及其区段保持不变，则B中区段个数和A中区段个数相等，两两配对，每一对A区段-B区段递归执行(1)~(3)，直至A中剩余元素个数为0，此时B中元素个数为2n，递归结束。

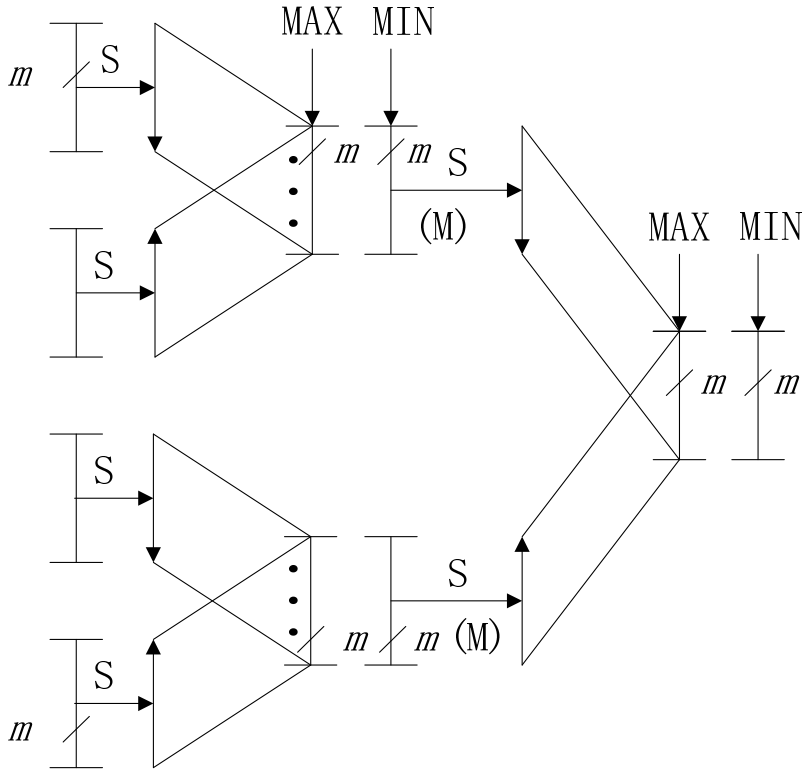
■ 示例: A={1,3,8,9,11,13,15,16},p=8; B={2,4,5,6,7,10,12,14,17},q=9



1.3功能划分

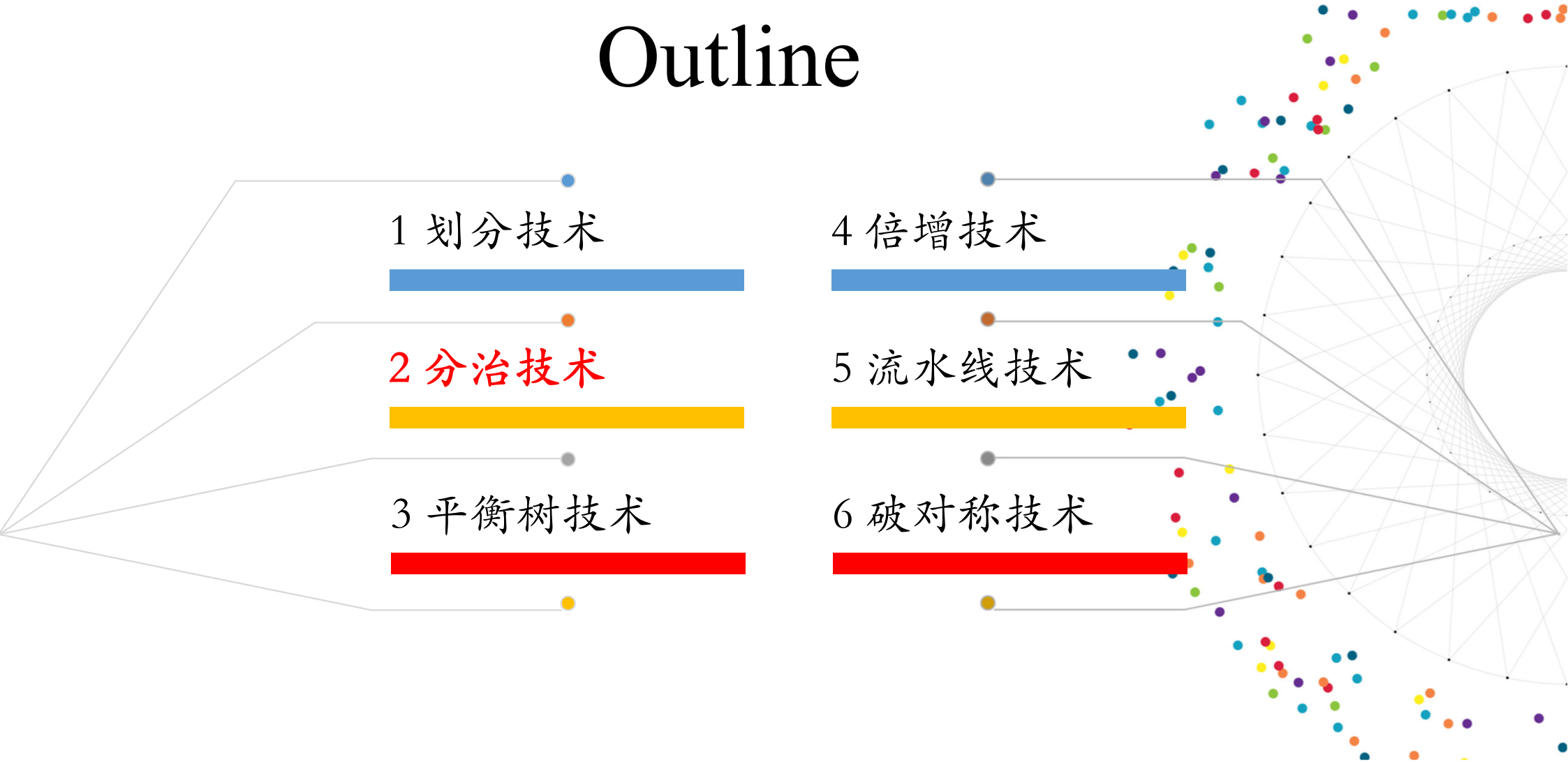
- N个元素分成N/m个区段，每个区段m个元素在一个处理器上处理。
- 功能划分示例：(m, n)选择问题(求出n个元素的序列A[1..n]中前m个最小者)
- (1) 功能划分：将A划分成g=n/m组，每组含m个元素；
- (2) 局部排序：使用Batcher排序网络将各组并行进行排序；
- (3) 两两比较：将所排序的各组两两进行比较，从而形成MIN序列；
- (4) 排序-比较：对各个MIN序列，重复执行第(2)和第(3)步，直至选出m个最小者。

功能划分示例（m,n）选择



S表示排序，M表示归并

Outline



分治技术的思想

- 分治策略是一种问题求解的方法学，其思想是将原来的大问题分解成若干个特性相同的子问题分而治之。若得到的子问题仍然偏大，可以反复使用分治策略直到很容易求解的子问题为止。使用分治技术时，分解后的子问题通常和原来问题的类型相同，则很容易使用递归过程求解。

分治v.s.划分

- 侧重点不同：划分是面向求解问题的需要或过程而进行的；分治是面向求解问题的简单，规范化而进行的。
- 难点不同：划分的难点是划分点的确定问题；分治的难点是问题间的同步通信和（递归）结果的合并问题。
- 子问题规模不同：划分是根据求解需要进行的，结果不一定是等分；分治一般是以1/k进行的等分。

并行分治法的步骤

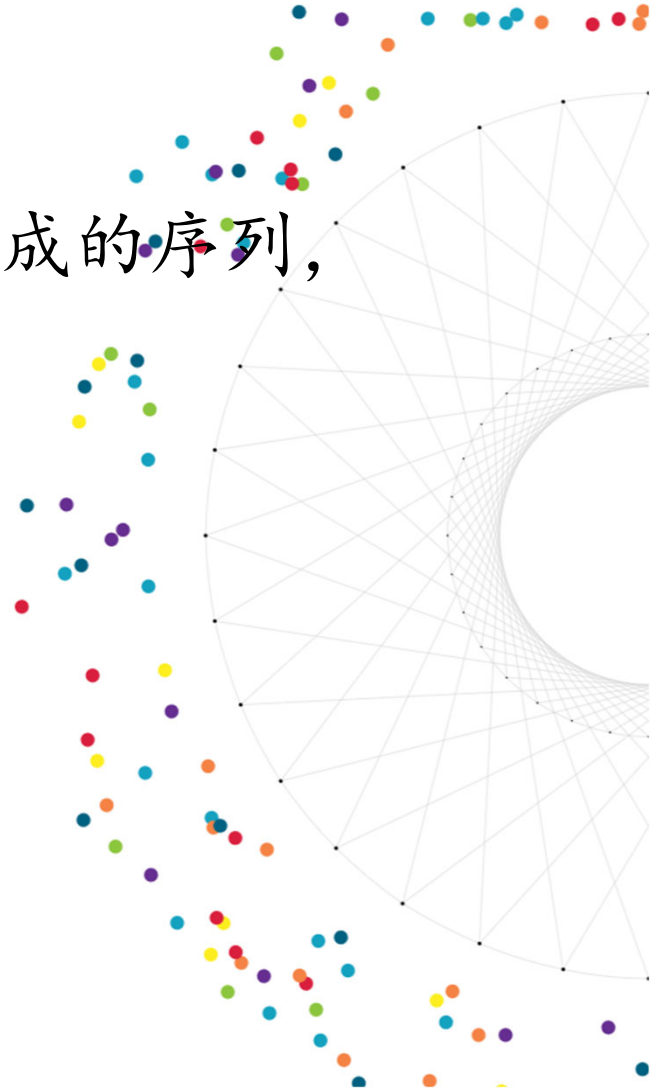
- （1）将输入划分成若干个规模相等的子问题；
- （2）同时(并行地)递归求解这些子问题；
- （3）并行地合并子问题的解，直至得到原问题的解。

并行分治法示例：双调归并网络

- 什么是双调序列？
- 一个序列a1,a2,…,an是双调序列(Bitonic Sequence)，如果：
- (1)存在一个ak(1≤k≤n), 使得a1≥…≥ak≤…≤an成立；或者
- (2)序列能够循环移位满足条件(1)

2 双调序列的例子

- 由一个非严格增序列X和非严格减序列Y构成的序列，
- (23,10,8,3,5,7,11,78)
- (1,3,5,7,8,6,4,2,0)
- (8,7,6,4,2,0,1,3,5)
- (1,2,3,4,5,6,7,8)



3 双调序列的特殊性质

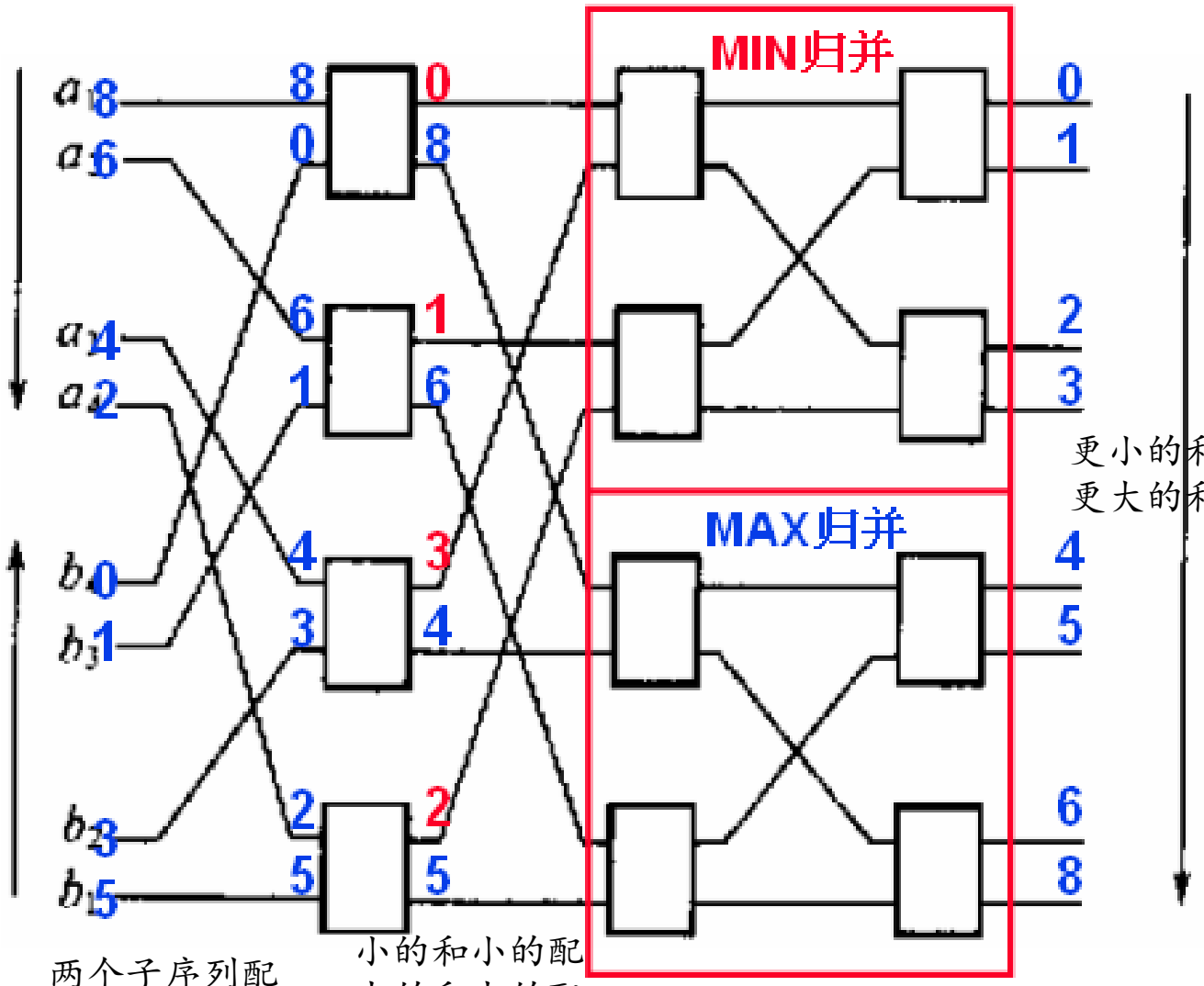
- 将任意一个长为2n的双调序列A分为等长的两半X和Y，将X中的元素与Y中的元素一一按原序比较，即a[i]与a[i+n](i<n)比较，将较大者放入MAX序列，较小者放入MIN序列。
- 则得到的MAX和MIN序列仍然是双调序列，并且MAX序列中的任意一个元素不小于MIN序列中的任意一个元素。（Batcher's Theorem）

4 双调归并的硬件基础

- 可以用若干个Batcher比较器组成比较器网络（Comparator Network）来实现。每个比较器是一个具有双输入和双输出的比较交换单元，它可将输入中的小者置于上输出端，而将大者置于下输出端。

5 双调归并过程

- 根据Batcher定理，可以将输入2n元素双调序列首先通过洗牌比较操作得到一个MAX序列和一个MIN序列，然后通过两个n阶双调归并器处理就可以得到一个有序序列。
- Procedure BITONIC_MERG(X)
- (1)for i=1 to n par-do
- {Si=min{xi,xi+n}； Li=max{xi,xi+n}； }
- end for
- (2)Recursive Call:
- {BITONIC_MERG(MIN=(S1,…,Sn));
- BITONIC_MERG(MAX=(L1,…, Ln)； }
- (3)output sequence MIN##MAX


MIN归并
MAX归并
a_8 8
a_6 6
a_4 4
a_2 2
b_0 0
b_1 1
b_3 3
b_5 5
8 0
0 8
6 1
1 6
4 3
3 4
2 2
5 5
0
1
2
3
4
5
6
8
更小的和更小的配
更大的和更大的配
两个子序列配
小的和小的配
大的和大的配
两两比较
2个(2,2)双调归并网络

Outline

1 划分技术

2 分治技术

3 平衡树技术

4 倍增技术

5 流水线技术

6 破对称技术

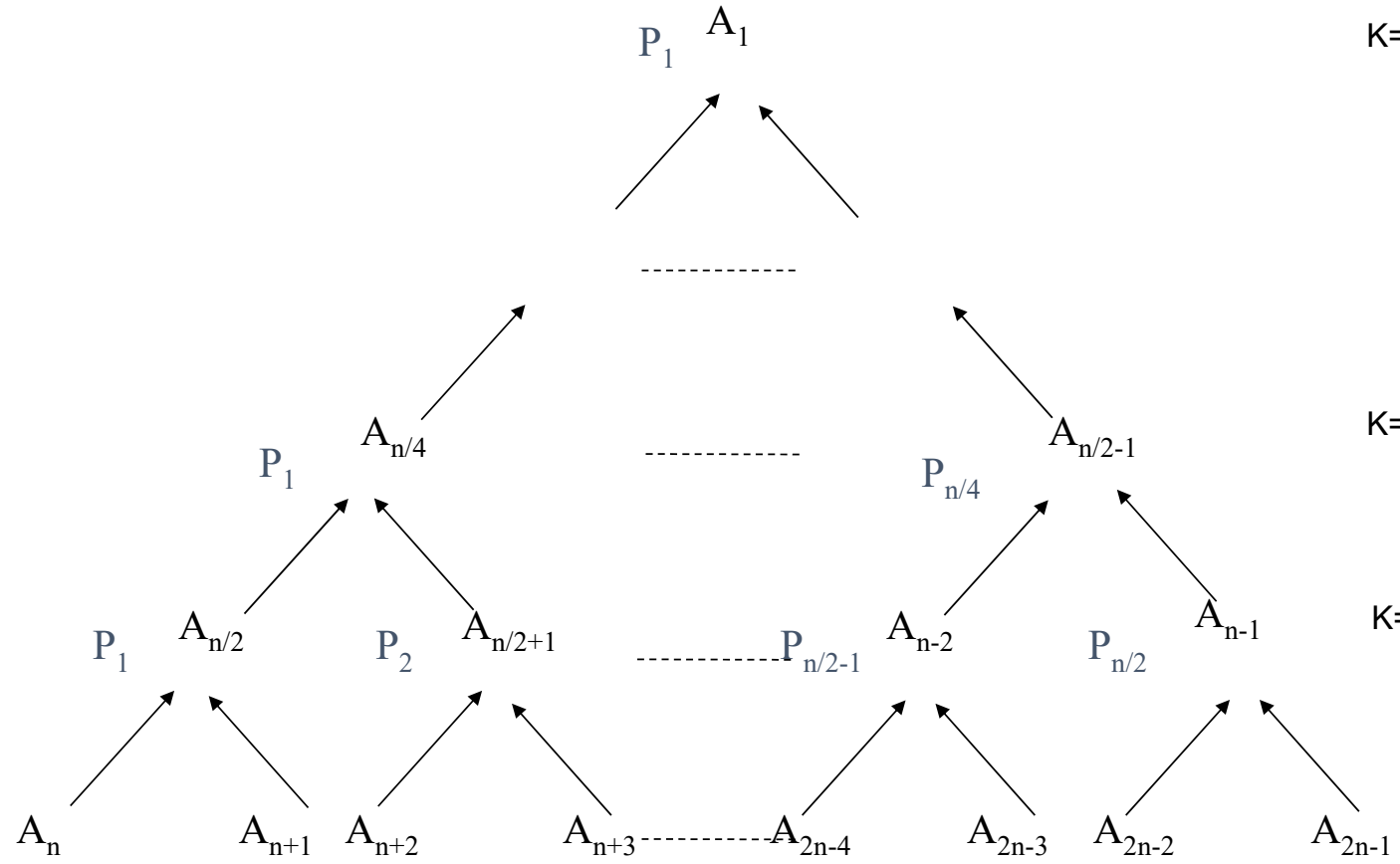
平衡树技术

- 平衡树（Balanced Tree）方法是将输入元素作为叶节点构筑一棵平衡二叉树，中间结点为处理结点，由叶向根或由根向叶逐层往返遍历，在树的同一深度上各节点并行计算。平衡二叉树可以推广到平衡多叉树。

平衡树实例1：平衡树求最大值

- 长度为n=2^m的数组A[n:(2n-1)]使用平衡树求最大值（A[1:(n-1)]存中间结果）
  - Begin
  - 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]}
  - end for
  - end for
  - end
- 复杂度分析
  - 并行算法复杂度t(n)=m×O(1)=O(logn)
  - 实际比较次数O(n)
  - 最大处理器数p(n)=n/2

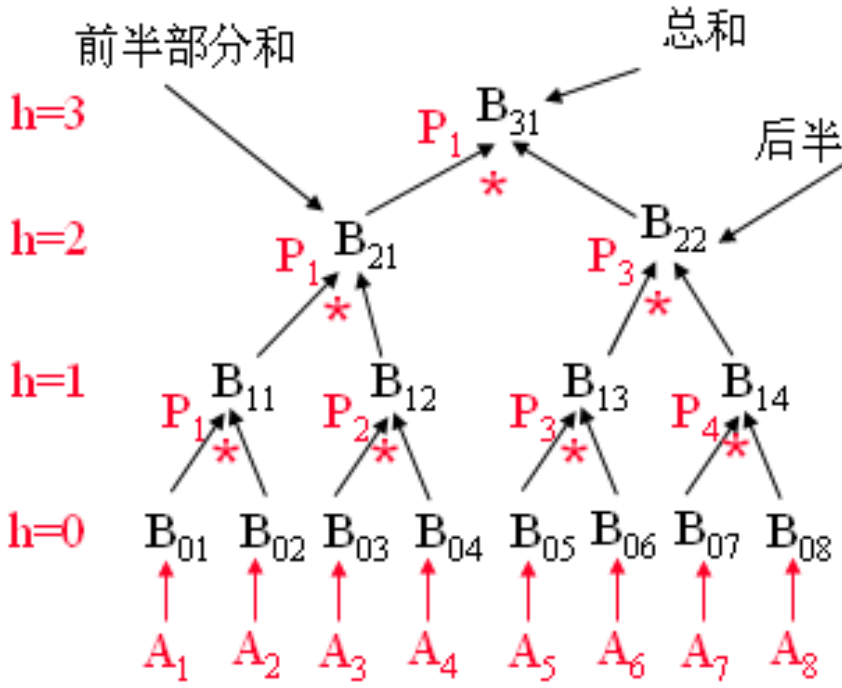
平衡树实例1：平衡树求最大值



平衡树实例2：平衡树求前缀和

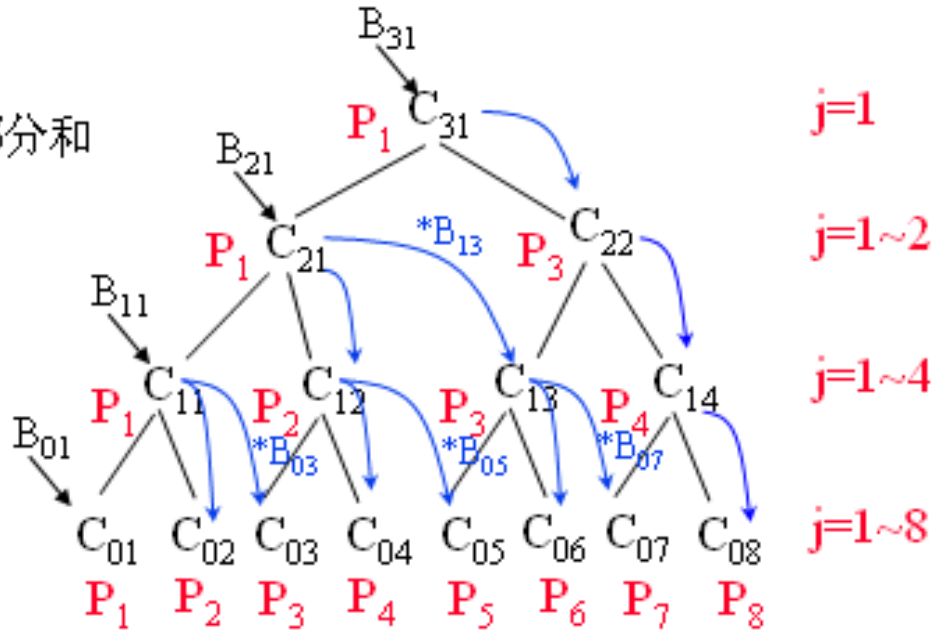
- 根据X[1:n]求S[1:n]，其中Si=x1*x2*…*xi，*可以是＋或×
- 串行算法： S[i]=S[i-1]*x[i]　　计算时间为 O(n)
- 并行算法：
  - 用X初始化数组B的叶节点部分；
  - 二维数组B记录由叶到根正向遍历树中各结点的信息(计算局部和)；
  - 二维数组C记录由根到叶反向遍历树中各结点的信息(播送前缀和)；
  - 输出C的叶节点部分到S;

平衡树实例2：平衡树求前缀和



计算：B[h,j]=B[h-1,2j-1]*B[h-1,2j]

正向遍历(求和)



计算：C[h,1]=B[h,1] j=1

C[h,j]=C[h+1, j/2] j=even

C[h,j]=C[h+1,(j-1)/2]*B[h, j] j=odd>1

反向遍历(播送前缀和)

Outline

1 划分技术

2 分治技术

3 平衡树技术

4 倍增技术

5 流水线技术

6 破对称技术

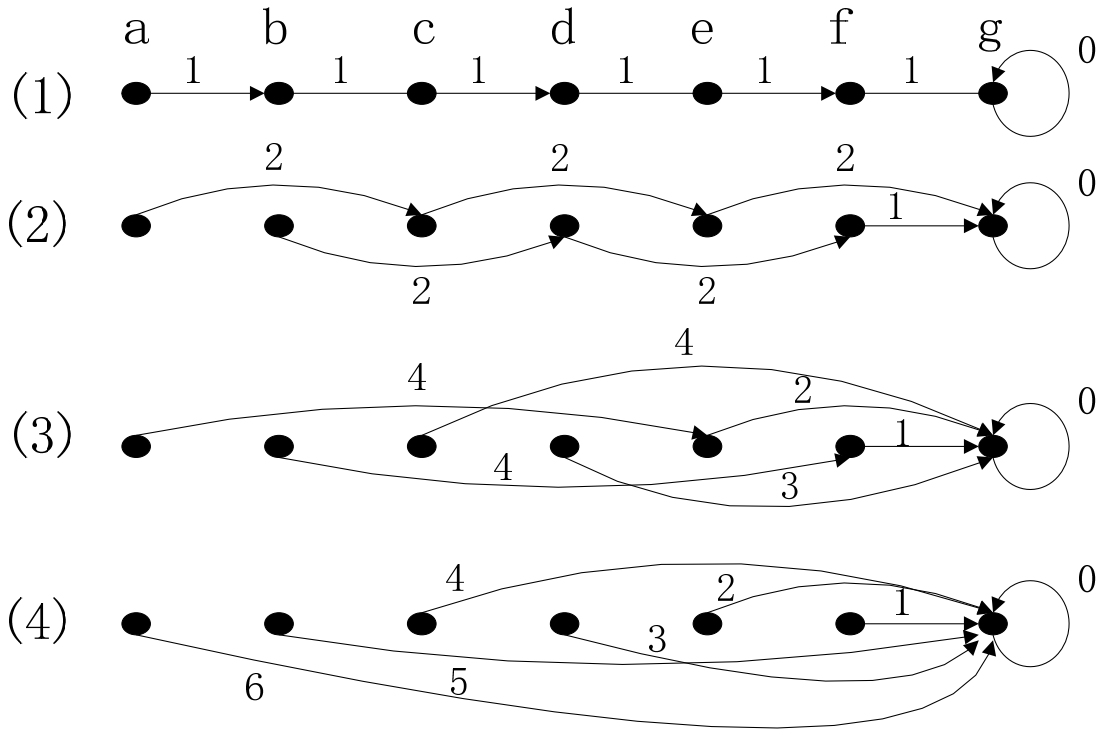
倍增技术

- 倍增（Doubling）技术又称指针跳跃(pointer jumping)技术，当递归调用时，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。特别适合于处理链表或有向树之类的数据结构.

倍增实例：

- 在SIMD-EREW上求数组L[1:n]中每个元素Lk在表列中的位序Rk=Rank(Lk)。
- 中间数据结构P[1:n]用于存放每个元素Lk的后继的标号。Rk可视为元素Lk至表尾的距离。
- for all k∈1 par_do
- p(k)←next(k) //p(k)初始化
- if p(k)≠k , then D(k) ←1, else D(k) ←0
- repeat「logn」time
- for all k∈1 par_do
- If p(k)≠p(p(k)) D(k) ←D(k)+D(p(k)), p(k) ←p(p(k))
- for all k∈1 par_do
- R(k) ←D(k)

倍增实例：



(1)初始化

p[a]=b, p[b]=c, p[c]=d, p[d]=e, p[e]=f, p[f]=g, p[g]=g

r[a]=r[b]=r[c]=r[d]=r[e]=r[f]=1, r[g]=0

(2)第一轮迭代

p[a]=c, p[b]=d, p[c]=e, p[d]=f, p[e]=p[f]=p[g]=g

r[a]=r[b]=r[c]=r[d]=r[e]=2, r[f]=1, r[g]=0

(3)第二轮迭代

p[a]=e, p[b]=f, p[c]=p[d]=p[e]=p[f]=p[g]=g

r[a]=4, r[b]=4, r[c]=4, r[d]=3, r[e]=2, r[f]=1, r[g]=0

(3)第三轮迭代

p[a]=p[b]=p[c]=p[d]=p[e]=p[f]=p[g]=g

r[a]=6, r[b]=5, r[c]=4, r[d]=3, r[e]=2, r[f]=1, r[g]=0

Outline

1 划分技术

2 分治技术

3 平衡树技术

4 倍增技术

5 流水线技术

6 破对称技术

流水线技术

- 流水线（Pipelining）通过时间上重叠和空间上并行的方式，将一个计算任务T分成n个前后衔接的子任务T[1:n]，使得tk的输出作为t(k+1)的输入，且tk完成后t(k+1)就可立即开始，并以同样的速度进行计算。
- 流水线示例：一维脉动阵列上的离散傅里叶变换

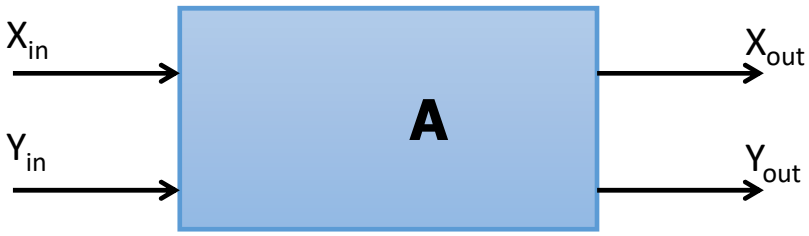
离散傅里叶变换

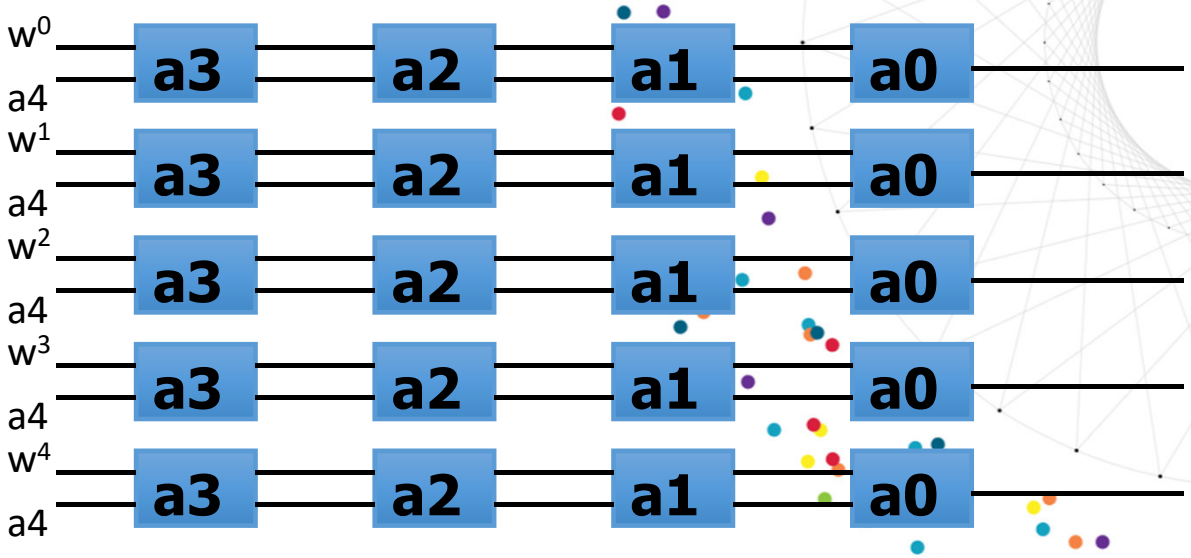
- 离散傅里叶变换（Discrete Fourier Transform，DFT）
- 将A=[a0,a1,…,a(n-1)]变换为B=[b0,b1,…,b(n-1)]，其中
- bj=a0*w^(0*j) + a1*w^(1*j) + a2*w^(2*j) + a3*w^(3*j)…
- Horner规则将直接求解转换为步进求解，可见转换后表达式规律性非常强，可使用专用硬件（处理器）进行处理

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

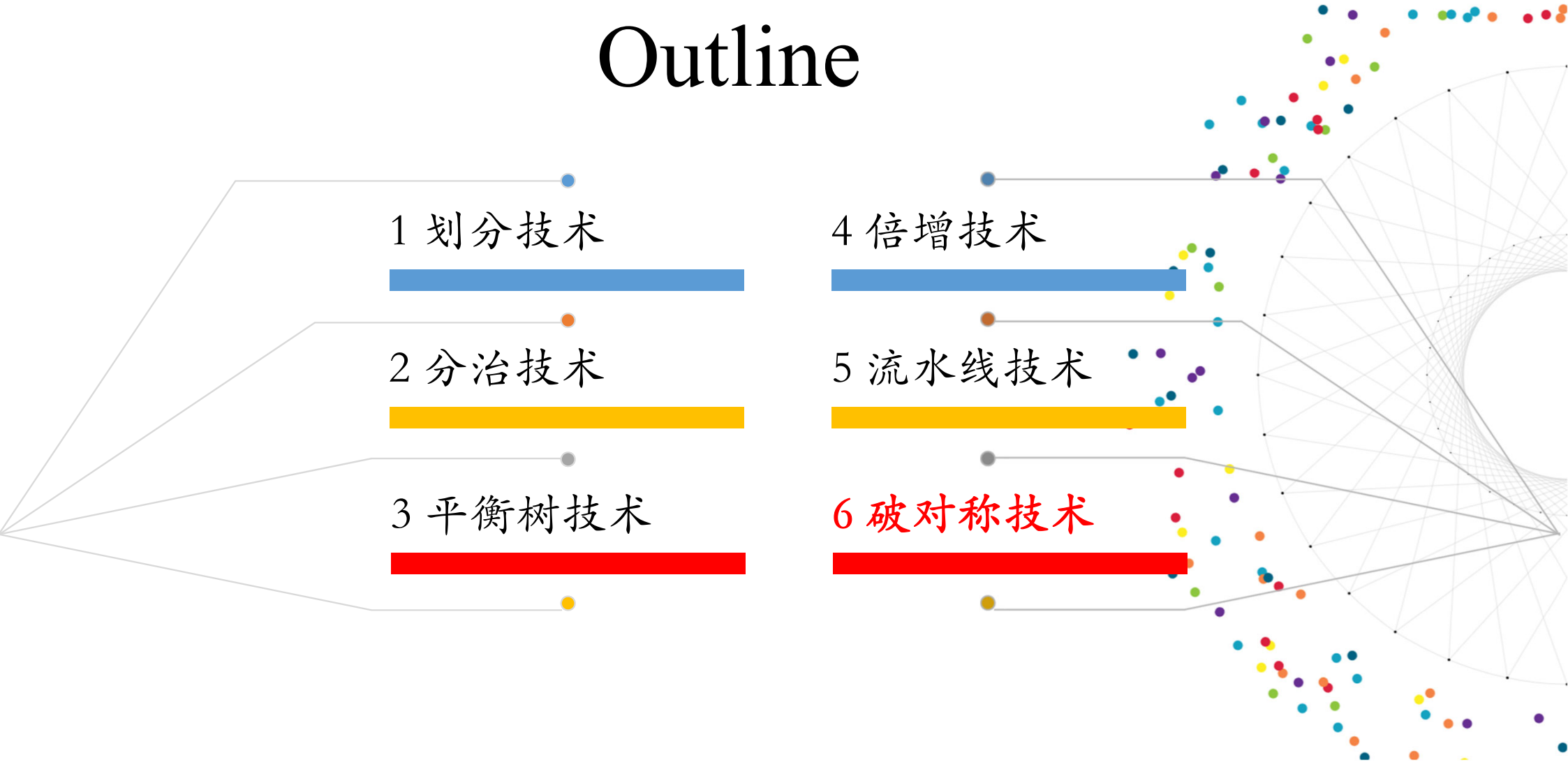
脉动阵列(Systolic Array，SA)

- 基础表达式 Y=B*C + A → Yout = Xin * Yin + A
- 按此制作一个脉动式乘加器单元，根据方程组的需要，将多个这样的处理单元连接起来构成脉动阵列。





Outline

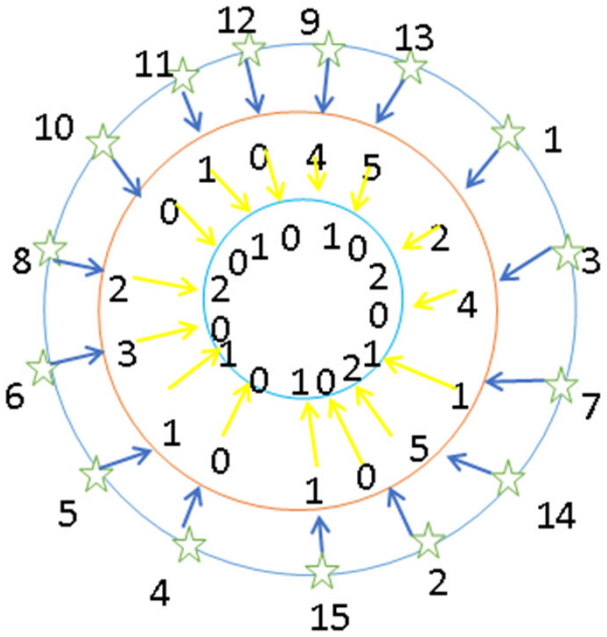


破对称技术

- 破对称（Symmetry Breaking）就是要打破某些问题的对称性，常用于图论和随机算法问题。
- 破对称示例：SIMD-EREW上n个节点的有向环图的顶点着色算法（从n种颜色降低到3种颜色）

有向环图的顶点着色

- 有向环图G=(V,E)，V表示顶点集合，E表示路径集合，S表示求当前点的后继，P表示求当前点的前驱，则有S(i)=j，P(j)=i。
  - 用数组c[1:n]存储圆弧上每个顶点的颜色，
  - 初始化每个顶点的颜色c(i)=i，并将c(i)用二进制表示，c(i)0表示c(i)的最低位二进制值，c(i)1表示c(i)的次低位二进制值，c(i)k表示c(i)的从低向高第k位二进制值。
  - For i=1 to n par_do
    - 比较第i个顶点的颜色c(i)和该顶点后继节点的颜色c(j)，对两个二进制串同时由低位向高位遍历，假设第k位两者开始出现不同，记录数值k，更新颜色值C’(i)←2k+c(i)k。
- 如果顶点对称，则算法不收敛，如果顶点破对称，算法收敛到解。



顶点v(i)	初始色C(i)	k	C(i)k	C'(i)
1	0001	1	0	2->2
3	0011	2	0	4->0
7	0111	0	1	1->1
14	1110	2	1	5->2
2	0010	0	0	0->0
15	1111	0	1	1->1
4	0100	0	0	0->0
5	0101	0	1	1->1
6	0110	1	1	3->0
8	1000	1	0	2->2
10	1010	0	0	0->0
11	1011	0	1	1->1
12	1100	0	0	0->0
9	1001	2	0	4->1
13	1101	2	1	5->0