题型

- 单选题
- 填空题
  - 基本概念：不是死记硬背
  - 计算题
- 简答题
  - 一题一问
  - 不要废话多，简明扼要
  - 复习case study
- 问答题
  - 一题多问

考试内容

结构、算法。不考编程细节。

1-8章、MapReduce。

知识点

O、总结图片

静态互连网络特性比较

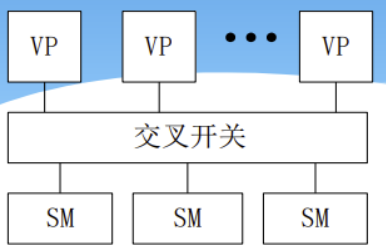
网络名称	网络规模	节点度	网络直径	对剖宽度	对称	链路数
线性阵列	N	2	$N-1$	1	非	$N-1$
环形	N	2	$\lfloor N/2 \rfloor$（双向）	2	是	N
2-D网孔	$(\sqrt{N}\times\sqrt{N})$	4	$2(\sqrt{N}-1)$	$\sqrt{N}$	非	$2(N-\sqrt{N})$
Illiac网孔	$(\sqrt{N}\times\sqrt{N})$	4	$\sqrt{N}-1$	$2\sqrt{N}$	非	$2N$
2-D环绕	$(\sqrt{N}\times\sqrt{N})$	4	$2\lfloor\sqrt{N}/2\rfloor$	$2\sqrt{N}$	是	$2N$
二叉树	N	3	$2(\lceil\log N\rceil-1)$	1	非	$N-1$
星形	N	$N-1$	2	$\lfloor N/2 \rfloor$	非	$N-1$
超立方	$N=2^n$	n	n	$N/2$	是	$nN/2$
立方环	$N=k\cdot 2^k$	3	$2k-1+\lfloor k/2 \rfloor$	$N/(2k)$	是	$3N/2$

动态互连网络比较

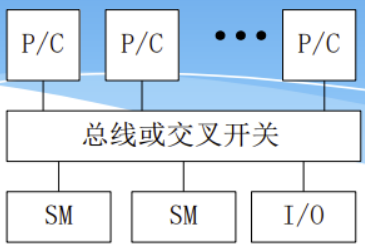
动态互连网络的复杂度和带宽性能一览表			
网络特性	总线系统	多级互连网络	交叉开关
开关复杂度	$O(n)$	$O(n\log_k n)$	$O(n^2)$
每个处理器带宽	$O(w/n)$～$O(w)$	$O(w)$～$O(nw)$	$O(w)$～$O(nw)$
报道的聚集带宽	SunFire服务器中的Gigaplane总线：2.67GB/s	IBM SP2中的512节点的HPS：10.24GB/s	Digital的千兆开关：3.4GB/s

* n,节点规模　w，数据宽度

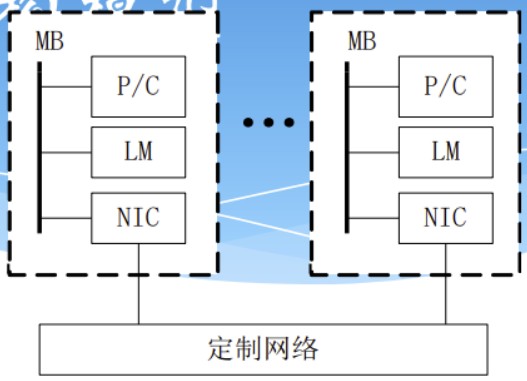
并行计算机结构模型



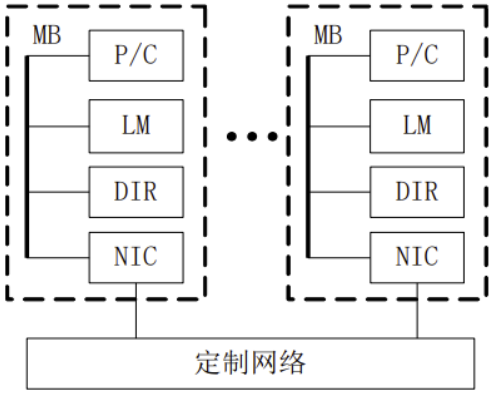
(a) PVP



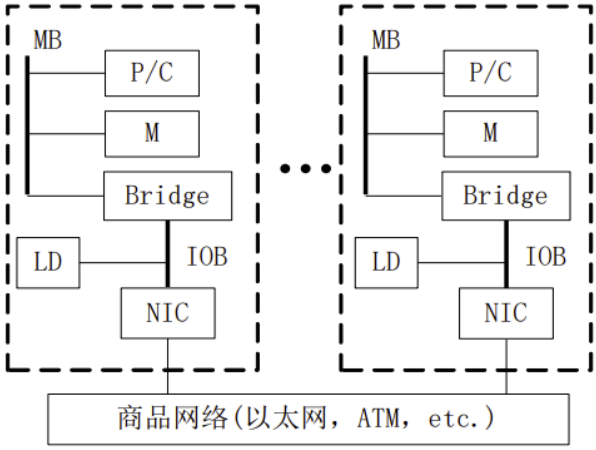
(b) SMP



(c) MPP



(d) DSM

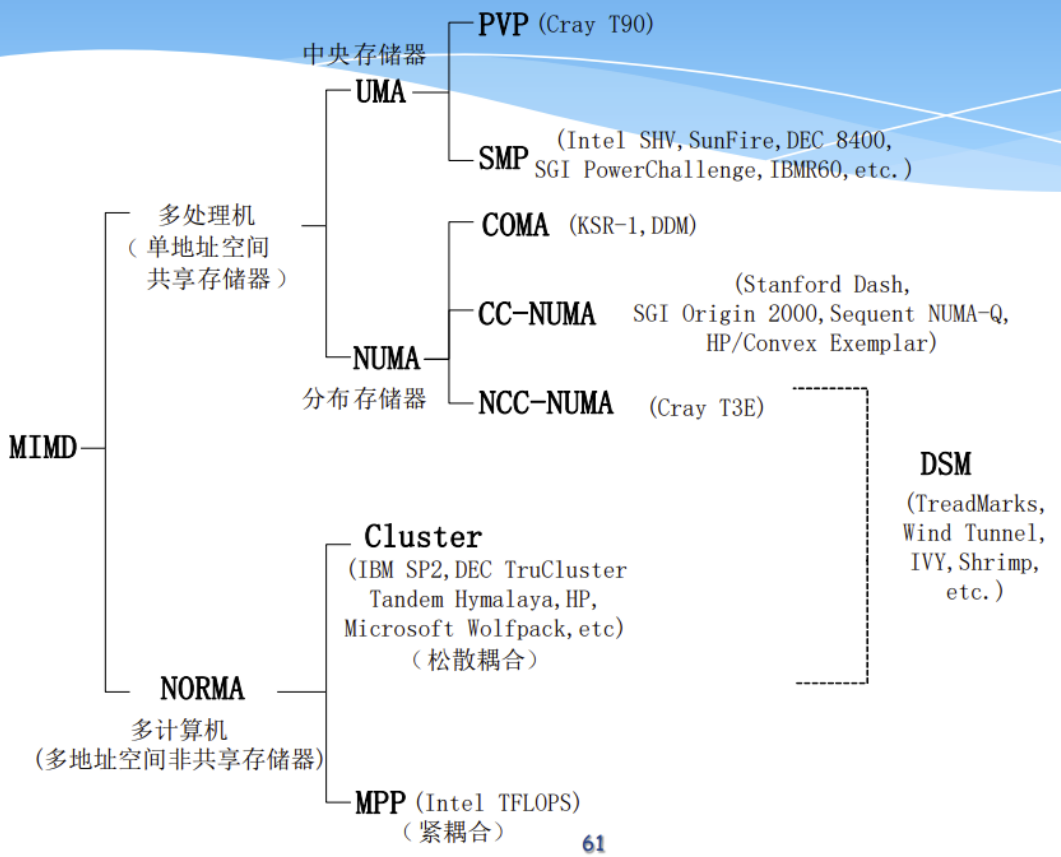


(e) COW

五种结构特性一览表

属性	PVP	SMP	MPP	DSM	COW
结构类型	MIMD	MIMD	MIMD	MIMD	MIMD
处理器类型	专用定制	商用	商用	商用	商用
互连网络	定制交叉开关	总线、交叉开关	定制网络	定制网络	商用网络（以太ATM）
通信机制	共享变量	共享变量	消息传递	共享变量	消息传递
地址空间	单地址空间	单地址空间	多地址空间	单地址空间	多地址空间
系统存储器	集中共享	集中共享	分布非共享	分布共享	分布非共享
访存模型	UMA	UMA	NORMA	NUMA	NORMA
代表机器	Cray C-90, Cray T-90, 银河1号	IBM R50, SGI Power Challenge, 曙光1号	Intel Paragon, IBMSP2, 曙光	Stanford DASH, Cray T 3D	Berkeley NOW, Alpha Farm

构筑并行机系统的不同存储结构



SMP\MPP\机群比较

系统特征	SMP	MPP	机群
节点数量(N)	≤O(10)	O(100)-O(1000)	≤O(100)
节点复杂度	中粒度或细粒度	细粒度或中粒度	中粒度或粗粒度
节点间通信	共享存储器	消息传递 或共享变量（有DSM时）	消息传递
节点操作系统	1	N(微内核) 和1个主机OS(单一)	N（希望为同构）
支持单一系统映像	永远	部分	希望
地址空间	单一	多或单一（有DSM时）	多个
作业调度	单一运行队列	主机上单一运行队列	协作多队列
网络协议	非标准	非标准	标准或非标准
可用性	通常较低	低到中	高可用或容错
性能/价格比	一般	一般	高
互连网络	总线/交叉开关	定制	商用

小结

并行计算模型综合比较一览表

	PRAM	APRAM	BSP	logP
体系结构	SIMD-SM	MIMD-SM	MIMD-DM	MIMD-DM
计算模式	同步计算	异步计算	异步计算	异步计算
同步方式	自动同步	路障同步	路障同步	隐式同步
模型参数	1 （单位时间步）	d,B d:读/写时间 B:同步时间	p, g, l p:处理器数 g:带宽因子 l:同步间隔	l, o, g, p l:通信延迟 o:额外开销 g:带宽因子 p:处理器数
计算粒度	细粒度/中粒度	中粒度/大粒度	中粒度/大粒度	中粒度/大粒度
通信方式	读/写共享变量	读/写共享变量	发送/接收消息	发送/接收消息
编程地址空间	全局地址空间	单一地址空间	单地址/多地址空间	单地址/多地址空间

38

一、并行计算机系统及结构模型

1.1 并行计算

科学发现的三大支柱：理论科学、实验科学、计算科学。计算机科学是计算科学的核心和重要组成部分。要养成计算思维。

- 考虑可计算性和计算复杂性
- 应用什么样的方法实现具体的计算
- 如何高效在计算机上求解问题

并行计算是在并行机上进行计算，以解决科学与工程问题的需求。需求类型是计算密集、数据密集、网络密集的。

1.2 并行计算机系统互连

系统互连。

只讨论到LAN，因为构造工作站集群COW时会用到。

- 系统域网络SAN：将短距离内的不同节点连接起来形成单一系统
- 局域网LAN：SAN可用于，范围约为一栋楼、校园内的局域网相连构建一个完整的系统
- 都域网MAN：覆盖整个城市
- 广域网WAN：覆盖全球

总线：

- 处理器总线
- 局部（本地）总线
- 存储器总线：连接存储器和处理器
- IO总线：连接IO设备
- 系统总线：连接其他设备

网络性能指标：

- 节点度：射入或射出一个节点的边数
- 网络直径：网络中任何两个节点之间的最长距离
- 对剖宽度：对分网络所必须删去的最少边数
- 对剖带宽：每秒钟内在最小的对剖平面上通过所有连线的最大信息数（bit/byte）
- 对称的：如果从各个节点观察网络，看到的都是一样

静态互联网络：拥有固定的连接，程序执行期间的拓扑结构保持不变

- 一维线性阵列
  - 二近邻连接
  - N节点，N-1条边
- 二维网孔
  - 四近邻，与上下左右连接
  - 变种：
    - Illiac：竖直方向带环绕，水平方向蛇环绕。直径$\sqrt{N}-1$。这是因为蛇状的结构可以允许一次同时向上向左的移动。
    - 二维环绕：两个方向都带环绕。直径$2\lfloor\frac{\sqrt{N}}{2}\rfloor$，也即两倍的半个直线距离。
- 树连接
  - 三近邻，内部节点只和父节点和两个子节点相连
  - 直径$2(\lceil logN\rceil-1)$
  - 可以变为X-tree以减小直径，极端情况节点度变为N-1，退化为星形连接
  - 问题是根容易成为性能瓶颈，引入胖树，通路从叶到根逐渐变宽
- 超立方网络

    - n-超立方由2^n个节点组成
    - 直径 = 节点度 = n，对剖宽度为$\frac{N}{2}$
- 立方环
    - 超立方的节点被一个环代替
    - $N = k2^k$，k-立方环
    - 对剖宽度$\frac{N}{2k}$，直径$2k - 1 + \lfloor \frac{k}{2} \rfloor$
- 洗牌交换网
- 蝶形网络

嵌入：用膨胀系数描述一个嵌入的质量，膨胀系数等于被嵌入网络中一条链路在嵌入网络中的最大链路数。为1则是完美嵌入。

- 环网可以完美嵌入到2-D环绕网
- 3-超立方完美嵌入到2-D环绕网

动态互联网络：用交换开关构成的，可以按照应用程序的要求动态地改变连接状态

- 总线
    - 连接处理器、存储器、IO外围设备等的一组导线和插座，用于主设备和从设备之间的数据传输
    - 多级层次结构
- 交叉开关
    - 高带宽
    - 由程序控制每个交叉点开关的状态，提供从源到目的地之间的动态连接
    - 应用：
        - 处理器之间的通信
        - 处理器与存储器之间的存取
    - 成本为n^2，限制了在大型系统中的应用
- 多级互联网络
    - 单级交叉开关级联起来形成MIN
    - 用在SIMD和MIMD中
    - Ω网络：完美洗牌系统连接，每一级$\frac{N}{2}$个2*2开关单元，一共有log_2N级。
    - 蝶形网络。每一级$\frac{N}{2}$个2*2开关单元，一共有log_2N级。
    - 允许输入连接到任意输出，一对一、一对多都可以，不能多对一否则冲突
    - 级间互联：均匀洗牌、蝶形、交叉开关、立方连接……

标准互联网络：科普

1.3 并行计算机系统结构

- SISD：同一时刻只有一个控制流和一个数据流，单指令单数据流

- MIMD：多指令多数据流
- SIMD：单指令多数据流，如向量计算单元
- MISD：多指令单数据流，各个处理单元组成一个线性阵列，分别执行不同的指令流，而同一个数据流则顺次通过这个阵列中的各个处理单元。这种系统结构只适用于某些特定的算法。

SIMD和MISD模型更适合于专用计算。在商用并行计算机中，MIMD模型最为通用，SIMD次之，而MISD最少用。

并行计算机结构模型：

- PVP：并行向量处理机
  - 多个向量处理单元VP通过交叉开关与多个共享存储SM相连
  - 交叉开关专门设计高带宽
  - 不适用高速缓存，而是使用大量向量寄存器和指令缓冲器
- SMP：对称多处理机
  - 多个处理器通过总线或者交叉开关与SM相连
  - 对称性带来较高的并行度
  - SM使得处理器不能太多，难以扩展
- MPP：大规模并行处理机
  - MB：包含处理机、本地存储
  - 多个MB通过专门设计的高带宽低延迟定制网络相连
  - 可扩放性，规模很大
  - 各做各的
- DSM：分布式共享存储处理机
  - 比MPP的MB中多一个DIR：高速缓存目录，支持分布高速缓存的一致性
  - 也就是说有物理上分布在各个MB的共享存储
  - 单地址空间
  - 分类：
    - CC-NUMA
    - NCC-NUMA
    - COMA
    - 软件实现
- COW：工作站集群
  - 每个节点都是完整的工作站（可以是PC、SMP）
  - 通过低成本的商品网络相连

以上后四个已经趋于一致，就是大量的节点通过高速网络互联。遵循Shell结构：用专门定制的Shell电路将商用微处理器和节点的其他部分连接起来。

并行计算机访存模型：

- UMA：均匀存储访问
  - 物理存储器被所有处理器均匀共享
  - 所有处理器访问任何存储单元的时间均相同
  - 每台处理器可以有私有的高速缓存
  - 外围设备以一定方式共享
- NUMA：非均匀存储访问
  - 共享存储器在物理上分布在所有处理器中，所有本地存储器形成全局的单地址空间
  - 访问本地LM、群内共享存储器CSM快，访问外地存储器或者全局存储器GSM慢
  - 后两条同上
- COMA：全高速缓存存储访问
  - NUMA的特例
  - 没有存储层次结构，全部高速缓存组成全局地址空间
  - 使用高速缓存目录D进行远程的访问
  - 可以随意初始化，缓存会在运行时被搬运到需要的地方
- CC-NUMA：高速缓存一致性非均匀存储访问
  - 使用基于目录的高速缓存一致性协议，不需要软件来实现高速缓存的一致性
  - 保留SMP易于编程的优点，改善SMP的可扩放性
  - 实际上是分布式共享存储的DSM
  - 它最显著的优点是程序员无需明确地在节点上分配数据，系统的硬件和软件开始时自动在各节点分配数据，在运行期间，高速缓存一致性硬件会自动地将数据迁移至要用到它的地方
- NORMA：非远程存储访问
  - 所有存储器都是私有的

二、当代并行机系统

编程、资金、通信、硬件、结构、数据、存储、资源、扩放、通用

2.1 共享存储多处理机系统

对称多处理机SMP：采用商用微处理器，通常有片上和片外Cache，基于总线连接，集中式共享存储UMA

- 优点：
  - 对称性
  - 单地址空间，易编程，动态负载平衡，无需担心数据分配
  - 高速缓存及其一致性，数据局部性，硬件维持一致性
  - 低通信延迟，通信使用简单的内存读写指令
- 问题：

- 欠可靠：总线、OS、存储器失效都会导致崩溃
- 通信延迟（相对于CPU），竞争会加剧延迟
- 慢速增加的带宽：带宽的发展跟不上存储器
- 不可扩放性：总线不可扩放

2.2 分布存储多计算机系统

大规模并行机MPP:NORMA，由可变化可扩放的成百上千处理器组成

- 可扩放性，但是要平衡处理能力、存储、IO的能力
- 系统成本低来源于每个微处理器的低成本
- 需要通用性高、可用性高（大量节点中肯定难免失灵）
- 通信要求：要快，否则使用LAN的COW会取代之
- 存储器和IO能力

2.3 机群系统

机群系统：通过松散集成的计算机软件和硬件连接起来高度紧密地协作完成计算工作。但是可以看作是一台计算机。分为异构和同构两种，主要区别在于组成集群系统的计算机之间的体系结构是否相同。

- 高性能计算集群：
  - 高性能计算集群采用将计算任务分配到集群的不同计算节点而提高计算能力
  - 比较流行的HPC集群特别适合于在计算中各计算节点之间发生大量数据通讯的计算作业，比如一个节点的中间结果或影响到其它节点计算结果的情况
- 负载均衡集群
  - 一般通过一个或者多个前端负载均衡器，将工作负载分发到后端的一组服务器上，从而达到整个系统的高性能和高可用性
- 高可用性集群
  - 一般是指当集群中有某个节点失效的情况下，其上的任务会自动转移到其他正常的节点上
  - 还指可以将集群中的某节点进行离线维护再上线，该过程并不影响整个集群的运行
- 网格计算
  - 连接一组相关并不信任的计算机，更像一个公共计算设施
  - 针对有许多独立作业的工作任务作优化，在计算过程中作业间无需共享数据
  - 管理独立执行工作的计算机之间的作业分配

工作站机群COW：分布式存储，MIMD，每个节点是完整的计算机，而MPP中只有微内核

- 优点：
  - 投资风险小，每个单独的工作站至少可用，哪怕系统不可用
  - 编程方便，每个节点的工作单独执行，不需要并行化编程

    - 系统结构灵活：异构性
    - 性能价格比高
    - 充分利用分散的计算资源
    - 可扩放性好
- 问题：
    - 通信性能
    - 并行编程环境

可用性计算。

- 串行的server则是相乘
- 并行的server是$1-(1-p)^n$

2.4 并行计算机存储组织

- 层次存储技术
- 高速缓存一致性
    - 写策略：
        - 写通过：一改就写
        - 写回：放回的时候写
    - 高速缓存不一致的原因
        - 共享可写数据
        - 进程迁移
        - 绕过高速缓存的IO操作
    - 保证高速缓存的一致性
        - 监听总线协议
        - 基于目录的协议

三、并行计算性能评测

3.1 并行机的一些基本性能指标

CPU的性能指标：

- 工作负载
    - 执行时间、浮点运算数、指令数目
- 并行执行时间
    - $T_n = T_{compute} + T_{paro} + T_{comm}$
    - 总时间=计算时间+并行开销时间+通信时间

- 并行开销包含进程管理、组操作等
- 通信时间包含同步、通信、聚合等

并行与通信开销

- 相对于计算时间来说很大
- 测量：
  - 乒乓方法：我发一个m字节消息，你收到m个之后，立即发回给我。总时间除以2就是点到点的通信时间。
  - 可以一般化为热土豆法，1->2->3->...->n
- 开销$t(m) = t_0 + \frac{m}{r}$
  - t_0：启动时间
  - r：传送无限长消息的通信速率
  - $m_{\frac{1}{2}}$：半峰值长度，达到一半渐进带宽所要的消息长度
  - π_0：特定性能，表示短消息带宽
    - $t_0 = m_{\frac{1}{2}}/r = \frac{1}{\pi_0}$
- 整体通信：n个处理器，每人发m字节
  - Broadcasting 广播
  - Gather 收集：某个处理器接受了m*n个字节
  - Scatter 散射：某处理器发给所有人每人m字节
  - Total Exchange 全交换：每两人彼此发送m字节，总通信量mn^2
  - circular-shift 循环移位：每人给自己后面的人发

机器成本、利用率也要考虑

- 机器成本价格
- 机器性能/价格比
- 利用率 = 可达到的速度 / 峰值速度

3.2 加速比性能定律

并行系统的加速比：对于一个给定的应用，并行算法的执行速度相对于串行算法的执行速度加快了多少倍。

- P：处理器数
- W：问题规模
  - $W_s = W_1$：程序中的串行分量，f为串行分量比例
  - W_p：程序中可并行化部分，$1 - f$为并行分量比例
  - $W = W_p + W_s$
- $T_s = T_1$：串行执行时间

- T_p：并行执行时间
- S：加速比
- E：效率

Amdahl定律：

出发点：

- 固定不变的计算负载，实时性要求高
- 固定的计算负载分布在多个处理器上
- 增加处理器加快执行速度，从而达到加速的目的

$$S=\frac{W_s+W_p}{W_s+\frac{W_p}{p}}=\frac{f+(1-f)}{f+\frac{1-f}{p}}=\frac{p}{1+f(p-1)}$$

当p趋向于极限的时候，$S=\frac{1}{f}$。相当于并行部分不消耗时间。但是加速比被f所限制。

如果加上额外开销：

$$S=\frac{W_s+W_p}{W_s+\frac{W_p}{p}+W_o}=\frac{W}{fW+\frac{W(1-f)}{p}+W_o}=\frac{p}{1+f(p-1)+\frac{W_op}{W}}$$

当p趋近于无限，极限为$S=\frac{1}{f+\frac{W_o}{W}}$

Gustafson定律：

出发点：

- 计算时间固定不变，精度是一个关键的因素
- 除非学术研究，否则固定工作负载增多处理器没有意义，最好同时相应地增大问题规模

$$S=\frac{W_s+pW_p}{W_s+\frac{pW_p}{p}}=\frac{W_s+pW_p}{W_s+W_p}=f+p(1-f)=p+f(1-p)=p-f(p-1)$$

说明加速比与p呈现线性关系，是一个乐观的结论。

Sun-Ni定律：

基本思想：

- 只要存储空间许可，应尽量增大问题规模以得到更好更精确的解
- $G(p)$表示存储容量增加到p倍时并行工作负载的增加量

$$S=\frac{fW+(1-f)G(p)W}{fW+(1-f)G(p)\frac{W}{p}}=\frac{f+(1-f)G(p)}{f+(1-f)\frac{G(p)}{p}}$$

- G(p)=1即固定工作负载的Amdahl定律

- G(p)=p即固定时间的Gustafson定律
- G(p)>p时，负载比存储增加的快，Sun-Ni定律加速比高

加速比讨论：

- 经验加速公式：$\frac{p}{logp} \leq S \leq p$
- 可达线性加速的应用问题有诸如矩阵相加、内积运算，几乎没有通信开销
- $\frac{p}{logp}$加速比：分治类应用问题
- 通信密集类：$\frac{1}{C(p)}$
- 超线性加速：
  - 多个方向同时搜索，找到解之后通知所有处理器停止
  - 高速缓存补偿通信时间
- 绝对加速：最佳串行算法时间/并行算法
- 相对加速：同一算法在单处理器时间/多处理器

3.3 可扩放性评测标准

在确定的应用背景下，计算机系统、算法、程序等，性能随处理器数量的增加而按比例提高的能力。

影响因素：

- 串行分量
- 额外开销：通信、竞争、等待、冗余操作、同步
- 加大的处理器数超过了算法中的并发程度

增加问题规模有利于提高加速的因素：

- 较大的问题规模提供较高并发度
- 额外开销的增加慢于有效计算的增加
- 串行分量的比例不是固定不变的，随着规模的增大而减小

可扩放性研究的主要目的：

- 确定针对某类问题使用何种的算法和并行体系结构，可以有效利用大量处理器
- 对于某种体系结构的并行机上的某种算法移植到大规模处理机上之后运行的性能
- 对固定的问题规模，确定在某类并行机上最优的处理器数和最大可获得的加速比
- 指导改进并行算法和并行机体系结构，使算法充分利用处理器资源

等效率度量标准

- $T_e = \sum_{i=0}^{p-1} t_e^i$，第i个处理器的有用计算时间
- T_o，总的额外开销

- $T_p = t_e^i + t_o^i$
- $T_e + T_o = pT_p$，T_p是p个处理器运行算法的时间
- 工作量$W = T_e$
- $S = \frac{T_e}{T_p} = \frac{p}{1+\frac{T_o}{T_e}}$
- $E = \frac{S}{p} = \frac{1}{1+\frac{T_o}{T_e}}$

为了维持效率需要在增加处理器数的同时增大问题规模。定义等效率函数$f_E(p)$为问题规模W随p变化的函数。这个函数是线性的，则可扩放；斜率越小可扩放性越好。

等速度度量标准

- 并行计算的速度$V = \frac{W}{T}$
- p个处理器的平均速度$\bar{V} = \frac{V}{p} = \frac{W}{pT}$

如果在增加处理器的时候增加一定的工作量，能维持平均速度不变，可扩放。

$\Phi(p, p') = \frac{W/p}{W'/p'} = \frac{p'W}{pW'}$。越大表示可扩放性越好。平均速度严格不变时，$\Phi(p, p') = \frac{T}{T'}$。

平均延迟度量标准

- 速度衡量的优点：
  - 速度很直观提现效率
  - 速度是不同结构的机器之间可比的量
  - 包含了计算和延迟而影响
  - 容易测量
- 定义运行时延迟L_i，此外还有启动前延迟、停止后延迟
- 系统平均延迟$\bar{L}(W, p) = \sum_{i=1}^{p}(T_{para} - T_i + L_i)/p$
- $pT_{para} = T_o + T_{seq}$
- $T_o = p\bar{L}(W, p)$
- $\bar{L}(W, p) = T_{para} - T_{seq}/p$

在保持效率E不变的前提下，用平均延迟的比值来标志随处理器个数p增加需要增加的工作量W。

四、并行算法的设计基础

CASE STUDY

求最大值

- 输入为$n = 2^m$存放在A(n,2n-1)中，求得的最大值置于A(1)

- 树状求最大值
- 两层for循环，内层的并行计算
- 时间O(m)

计算前缀和

- 两个辅助数组B、C
- 树状结构，B计算局部和
- B将局部和播送给C作为结果

4.1 并行算法的基础知识

并行算法：一些可同时执行的诸进程的集合，这些进程互相作用和协调动作从而达到给定问题的求解。

复杂性度量指标（度量“最坏情况复杂度”“期望复杂度”）：

- 运行时间t(n)：计算时间和通讯时间
- 处理器数p(n)
- 并行算法成本c(n)=t(n)p(n)
  - 并行算法成本如果在数量级上等于最坏情况下串行求解此问题所需的执行步数，则称此并行算法是成本最优的
- 总运算量W(n)
- 令W(n)是某并行算法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

同步： 并行算法中使用同步上锁，实现互斥访问。

通信：

- 共享存储：global read/write
- 分布存储：send/receive

4.2 并行计算模型

PRAM模型：并行随机存储机器。又称共享存储的SIMD模型：SIMD-SM。可以分为：

- PRAM-EREW：互斥读写
- PRAM-CREW：同时读，互斥写
- PRAM-CRCW：同时读写
  - 同时写是不现实的，需要规约
  - CPRAM-CRCW：Common，只允许所有的处理器同时写相同的数

- PPRAM-CRCW：Priority，只允许最优先的处理器先写
- APRAM-CRCW：Arbitrary，任意的

其中，CRCW性能最好，EREW最差，可以logP倍地模拟CRCW和CREW：$T_{EREW} = O(logP \cdot T_{CRCW}) = O(logP \cdot T_{CREW})$

PRAM模型：

- 优点：适合并行算法地表示和复杂性分析，易于使用，隐藏了并行机的通信、同步等细节
- 缺点：不适合MIMD并行机，忽略了SM的竞争、通讯延迟

异步APRAM模型：

- 每个处理器有局部存储器、局部时钟、局部程序，但是没有全局时钟，程序异步执行；
- 处理器之间通过SM进行通讯；
- 处理器之间的依赖关系需要显式地加入同步路障。
- 指令类型分为：全局读、全局写、局部操作、同步。
- 计算过程由同步障分开的全局相组成，同步是为了使得指令的完成时间差异不影响整体的计算任务。

时间计算：

- $T = \sum t_{ph} + B \times$ 同步障次数。其中t_{ph}是全局相内个处理器执行指令时间最长的。
- d：全局读写平均时间，随着p增加而增大
- B：同步路障时间，是关于p的非降函数
- $2 \leq d \leq B \leq p$
- $B(p) \in O(dlogp)$或者$B(p) \in O(dlogp/logd)$

优缺点：

- 比PRAM模型接近现实，但是仍相差较远
- 其上并行算法非常有限，也不适合MIMD-DM模型

BSP模型：块同步模型。异步的MIMD-DM模型。支持消息传递系统，块内异步并行，块间显式同步。

- p：处理器数
- l：同步障时间
- g：带宽因子：传送h条消息时间为hg
- 每个超级步内各自进行计算，然后全局同步
- 一个超级步计算成本 $= max_{processes}\{$进程i的局部计算时间$\} + max\{h_i g_i\} + L$

优缺点：

- 强调计算和通讯的分离，提供一个可编程环境（如果计算和通讯合适地平衡），易于程序复杂性分析
- 需要显式同步机制（硬件支持），限制消息传递数目、超级步的长度

logP模型：分布存储的、点到点通讯的多处理机模型，通讯由一组参数描述，实行隐式同步。

- L：消息传递延迟
- o：overhead，处理器收发一条消息的额外开销
- g：gap，处理器可连续进行消息收发的最小时间间隔
- p：processor个数

优缺点：

- 捕捉MPC通讯瓶颈，隐藏并行机的网络拓扑、路由、协议
- 可以应用到SM、消息传递、数据并行的编程模型中
- 难以进行算法描述设计分析

BSP是大同步模型，所有处理器一起同步，便于算法设计分析；而如果分散为子集同步可以提升性能，如果把子集缩小为成对的处理器，则变为了logP模型，更好地对资源进行控制。

五、并行算法的一般设计方法

5.1 串行算法的直接并行化

发现串行算法中的并行性，直接改造。

快排并行化

- 树状的并行，总体的执行时间为$T(n)=T(n/2)+n$，根据主定理复杂度为$\Theta(n)$
- 如果将快速排序视为构造一棵二叉搜索树，再中序遍历就可以得到有序序列。核心：同时写，线性化最后一个写入成功的就是被选中的。复杂度为树高，因为每一层都是并发构造的，$\Theta(logn)$

枚举排序并行化

- n个处理器，每个人负责一个数字，O(n)

5.2 从问题描述开始设计并行算法

从问题本身出发，不考虑对应的串行算法，因为不可并行化。设计全新算法。

字符串匹配算法并行化

- 给定正文串text和目标串pat，找到pat在text出现的位置
- 精确字符串匹配：
  - KMP、BM、Shift-And算法
- 近似字符串匹配：
  - 动态规划
  - 基于自动机
  - 位并行
  - 文本快速过滤法

KMP算法使用next数组灵活调整P、T上的指针，这是一种顺序的算法，不适合将字符串拆分进行并行化。

以上都是串行算法。要设计并行算法。

- 从周期性入手
- 定义失配见证函数Wit(j)=w，如果对于给定的 `j<m/2` ，P[j:m]!=P[1:m-j+1]，而w使得P(w)!=P(j-1+w)。
  - 对所有 `2<=j<=m/2` ，当且仅当WiT(j)!=0时，串是非周期的。
- 由WiT()函数计算竞争函数duel(p,q)，当模式在某一位置p匹配时，另一位置q一定不匹配，减少匹配次数
  - 计算时在相同大小的块内进行
  - 每块大小为2^k
- 然后用模式在找到的位置进行匹配

5.3 借用已有算法求解新问题

找出求解的问题和现有已解决问题之间的联系。

利用矩阵乘法求所有点对间最短路径

六、并行算法的基本设计技术

6.1 划分设计技术

- 将问题分成几乎等尺寸的子问题
- 用多个处理器解决子问题

关键在于如何均匀划分。

均匀划分技术：PSRS排序。

方根划分技术：

- 流程
  - 方根划分A、B为p、q段
  - 段间比较，将A的划分元插入B中哪个对应段
  - 段内比较，将A的划分元插入B中对应段哪个位置
  - 递归归并，B按照插入的划分元重新划分，形成段对，递归解决
- 需要处理器数$k = \lfloor \sqrt{pq} \rfloor$
- 时间分析：
  - $\lambda_0 = p$
  - $\lambda_i \leq \lfloor \sqrt{\lambda_{i-1}} \rfloor \leq \cdots \leq \lfloor p^{2^{-i}} \rfloor$
  - 算法在$\lambda_i = C$时停止递归，则$\lfloor p^{2^{-i}} \rfloor \geq C$
  - $i \leq loglogp + C_1$

对数划分技术：归并问题视为寻找A中的元素在$A \cup B$中的位置。复杂度logn

功能划分技术：n个元素划分为等长的p组，每组满足某种特性。

在(m,n)选择问题中，找到n个元素中前m个最小的。

- 功能划分：分成g=n/m组，每组m个元素
- 局部排序
- 两两比较
- 排序-比较

6.2 分治设计技术

分治侧重于子问题的归并上，而划分侧重于在原问题的划分技术。

双调归并网络

- 双调序列：先增再减或者先减再增。
- Batcher定理：给定一个双调序列，将其按照中心对称反转，依然是双调序列
- 双调归并：将双调序列归并为非降序列
  - 中心对称地找到集合s、l分别代表较小的和较大的值
  - 递归求解MIN=f(s)，MAX=f(l)

6.3 平衡树设计技术

思想：以树的叶节点为输入，中间结点为处理结点，由叶向根或由根向叶逐层并行处理。

就是之前的两个CASE STUDY，求最大值和前缀和。

6.4 倍增设计技术

又称指针条约技术，适合于处理链表或有向树之类的数据结构。递归调用时，需要处理的数据距离逐步加倍，k步之后可以完成距离为2^k地所有数据计算。

表序问题。

求森林的根。

6.5 流水线设计技术

将算法流程划分为p个前后衔接的任务片段，每个片段的输出作为下一个的输入。所有任务片段按同样的速率产出结果。

七、并行算法的一般设计过程

7.1 PCAM设计方法学

四个阶段：

- 划分：分解成小的任务，开拓并发性和可扩放性
- 通讯：子任务间的数据交换，检测划分的合理性
- 组合：根据任务的局部性，组合成更大的任务
- 映射：将每个任务分配到处理器上，提高算法性能

7.2 划分

先进行数据分解（域分解），然后进行功能分解。划分阶段忽略处理器数目和目标机器的体系结构。

域分解

- 划分数据，成为大致相等的小数据片
- 考虑数据上的相应操作
- 如果要使用别的任务中的数据，则会产生任务间的通讯

功能分解

- 划分计算
- 划分后研究不同任务所需的数据，如果不相交则划分成功；有大量重叠，则产生很多通信，要重来

示例：搜索树。

划分判据

- 灵活性：划分出的任务书高于目标机上的处理器数一个量级
- 避免冗余计算和冗余存储，为了扩放性
- 任务的尺寸大致相当，为了工作量均衡
- 问题尺寸增加应当使得任务数增加而不是单个任务量
- 采用多种划分方法，提高灵活性

7.3 通讯

划分生成的任务一般不能完全独立执行，需要进行数据交流。

功能分解确定了数据流，而数据流限制了子任务的并发性。

通讯模式：

- 局部/全局通讯
  - 每个人之间都有可能通讯吗
- 结构化/非结构化通讯
  - 任务之间形成规整结构
- 静态/动态通讯
  - 通信伙伴会不会变化
- 同步/异步通讯

通讯判据

- 所有任务执行的通信是否大致相当，可扩放性
- 每个任务是否尽可能局部通讯，避免全局通信，设法结构化
- 通讯操作能否并行
- 计算能否并行

7.4 组合

- 由抽象到具体，将组合的任务能在一类并行机上有效执行。
- 合并小尺寸任务，减小任务数。
- 增加任务粒度和重复计算，减少通讯成本。
- 保持映射和扩展的灵活性。

表面-容积效应：

- 通讯量与任务子集的表面成正比
- 计算量和任务子集的体积成正比
- 增加重复计算减少通讯量。

重复计算:
重复计算减少通讯量，但增加了计算量，应保持恰当的平衡。

示例：蝶式结构求全和。

组合判据

- 增加粒度是否减少了通讯成本
- 重复计算要权衡
- 保持灵活性和可扩放性
- 任务数和问题尺寸成比例
- 组合是否保持类似的通讯和计算
- 在不增加一切成本的前提下，能否进一步减少任务数
- 考虑修改串行代码的成本

7.5 映射

目的：减少算法总的运行时间

- 并发的任务放到不同处理器，增强并行度
- 需要通信的任务置于同一个处理器，提高局部性

负载平衡算法:

- 静态的：事先确定
- 概率的：随即确定
- 动态的：执行期间动态负载
- 基于域分解：
    - 递归对剖：计算全局信息
    - 局部算法：使用局部信息
    - 概率方法：随机分配
    - 循环映射：轮流分配

任务调度算法:

- 经理-雇员模式（集中）
- 非集中模式

映射判据

- 采用集中式负载平衡，master节点是否存在通讯瓶颈
- 动态负载平衡，调度策略的成本如何?
- 概率、循环方法，是否有足够多的任务分配?

八、基本通讯操作

8.0 预备知识

- t_s：启动时间，准备包头等
- t_h：节点延迟时间，包头穿越相邻节点的时间
- t_w：字传输时间，传输每个字的时间
- l：链路数
- m：信包大小

8.1 选路方法与开关技术

分类：

- 最短路径/非最短路径
  - 贪心、随机
- 确定选路/自适应选路
  - 视网络情况而定

维序选路：

- X-Y选路
- E-立方选路：通过异或，逐维度选路

如果使用存储转发选路，消息都被分为包。问题是每个结点必须对整个消息和包进行缓冲，缓冲器大；网络时延与发送消息历经的节点数成正比。$t_{SF} = t_s + l(mt_w + t_h) = O(ml)$

可以考虑切通选路，传递消息之前建立物理通道，占用。$t_{CT} = t_s + mt_w + lt_h = O(m + l)$

由于t_h很小，有时候可以忽略。

8.2 单一信包一到一传输

距离l计算：

- 一维环形：$l \leq \lfloor p/2 \rfloor$
- 带环绕：$l \leq 2\lfloor \sqrt{p}/2 \rfloor$
- 超立方：$l \leq logp$

8.3 一到多播送

- SF：
  - 环：$t = (t_s + mt_w)\lceil p/2 \rceil$
  - 环绕网孔：$t = 2(t_s + mt_w)\lceil \sqrt{p}/2 \rceil$
  - 超立方：$t = (t_s + mt_w)logp$
- CT：
  - 环：
    - 先发给p/2的，再同时发给p/4的，……
    - $t = \sum_{i=1}^{logp}(t_s + mt_w) = (t_s + mt_w)logp$
  - 网孔：
    - 先行再列
    - $t = 2\sum_{i=1}^{log\sqrt{p}}(t_s + mt_w) = (t_s + mt_w)logp$
  - 超立方：
    - 从低维向高维播送
    - $t = (t_s + mt_w)logp$

8.4 多到多播送

- SF:
  - 环：
    - 同时向一个方向发送
    - $t = (t_s + mt_w)(p-1)$
  - 环绕网孔：
    - 先行再列
    - $t = (t_s + mt_w)(\sqrt{p}-1) + (t_s + m\sqrt{p}t_w)(\sqrt{p}-1)$
  - 超立方：
    - 依次按维
    - $t = \sum_{i=1}^{logp}(t_s + 2^{i-1}mt_w) = t_s logp + mt_w(p-1)$
- CT：和SF一样，但是会造成链路竞争

MapReduce

MapReduce是一种编程模型，用于处理和生成大数据集，通过在集群上运行并行分布式算法。这个模型主要由两个阶段组成：Map和Reduce。

- Map阶段：在这个阶段，任务被分解为一系列简单的任务进行处理。每个Map操作都会对输入的数据进行某种形式的转换或过滤，然后生成一组键值对。
- Reduce阶段：在这个阶段，Map阶段的输出被进一步处理，通常是通过汇总或合并操作，以生成最终的结果。

MapReduce的主要优势在于其可扩展性和容错能力，使其能够有效地处理大规模数据。

- 并发性：map和reduce操作可以各自并发执行，但是在所有map操作结束之后才能开始reduce。
- 鲁棒性：worker失效由master重新分配任务；master失效则重启。