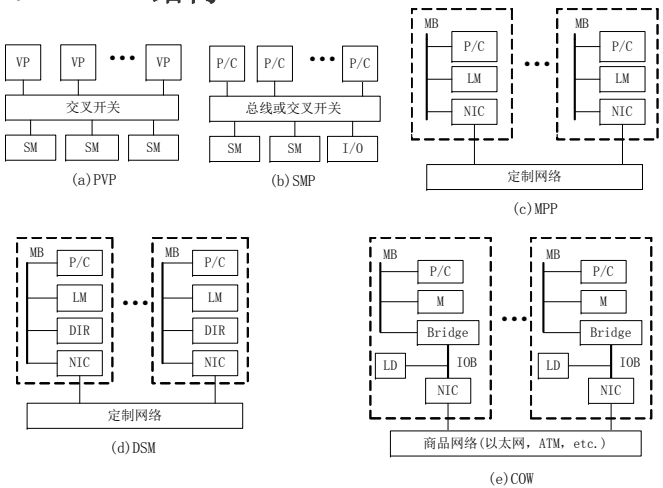
2.2 MIMD结构



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	IBMR50, SGI Power Challenge，曙光1号	Intel Paragon，IBMSP2，曙光2000	Stanford DASH，Cray T 3D	Berkeley NOW，Alpha Farm

Dichotomy of Parallel Computing Platforms

- An explicitly parallel program must specify concurrency and interaction between concurrent subtasks.
- The former is sometimes also referred to as the control structure and the latter as the communication model.

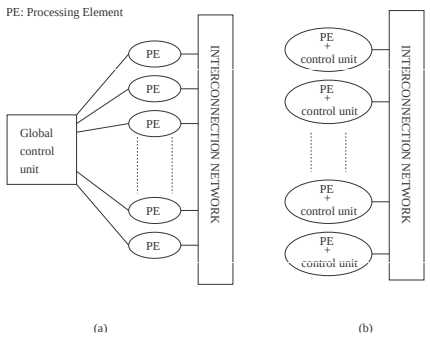
Control Structure of Parallel Programs

- Parallelism can be expressed at various levels of granularity - from instruction level to processes.
- Between these extremes exist a range of models, along with corresponding architectural support.

Control Structure of Parallel Programs

- Processing units in parallel computers either operate under the centralized control of a single control unit or work independently.
- If there is a single control unit that dispatches the same instruction to various processors (that work on different data), the model is referred to as single instruction stream, multiple data stream (SIMD).
- If each processor has its own control unit, each processor can execute different instructions on different data items. This model is called multiple instruction stream, multiple data stream (MIMD).

SIMD and MIMD Processors

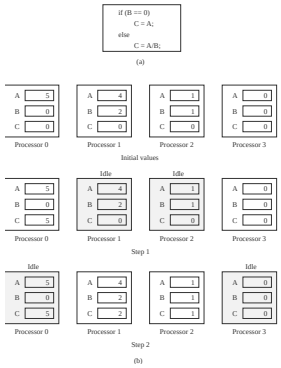


A typical SIMD architecture (a) and a typical MIMD architecture (b).

SIMD Processors

- Some of the earliest parallel computers such as the Illiac IV, MPP, DAP, CM-2, and MasPar MP-1 belonged to this class of machines.
- Variants of this concept have found use in co-processing units such as the MMX units in Intel processors and DSP chips such as the Sharc.
- SIMD relies on the regular structure of computations (such as those in image processing).
- It is often necessary to selectively turn off operations on certain data items. For this reason, most SIMD programming paradigms allow for an ``activity mask'', which determines if a processor should participate in a computation or not.

Conditional Execution in SIMD Processors



Executing a conditional statement on an SIMD computer with four processors: (a) the conditional statement; (b) the execution of the statement in two steps.

MIMD Processors

- In contrast to SIMD processors, MIMD processors can execute different programs on different processors.
- A variant of this, called single program multiple data streams (SPMD) executes the same program on different processors.
- It is easy to see that SPMD and MIMD are closely related in terms of programming flexibility and underlying architectural support.
- Examples of such platforms include current generation Sun Ultra Servers, SGI Origin Servers, multiprocessor PCs, workstation clusters, and the IBM SP.

SIMD-MIMD Comparison

- SIMD computers require less hardware than MIMD computers (single control unit).
- However, since SIMD processors ae specially designed, they tend to be expensive and have long design cycles.
- Not all applications are naturally suited to SIMD processors.
- In contrast, platforms supporting the SPMD paradigm can be built from inexpensive off-the-shelf components with relatively little effort in a short amount of time.

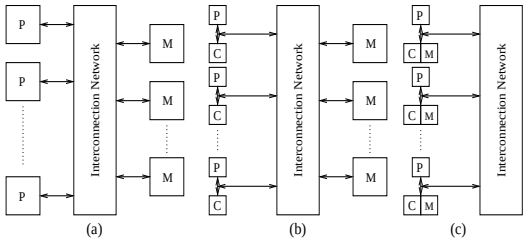
Communication Model of Parallel Platforms

- There are two primary forms of data exchange between parallel tasks - accessing a shared data space and exchanging messages.
- Platforms that provide a shared data space are called shared-address-space machines or multiprocessors.
- Platforms that support messaging are also called message passing platforms or multicomputers.

Shared-Address-Space Platforms

- Part (or all) of the memory is accessible to all processors.
- Processors interact by modifying data objects stored in this shared-address-space.
- If the time taken by a processor to access any memory word in the system global or local is identical, the platform is classified as a uniform memory access (UMA), else, a non-uniform memory access (NUMA) machine.

NUMA and UMA Shared-Address-Space Platforms



Typical shared-address-space architectures: (a) Uniform-memory access shared-address-space computer; (b) Uniform-memory-access shared-address-space computer with caches and memories; (c) Non-uniform-memory-access shared-address-space computer with local memory only.

NUMA and UMA Shared-Address-Space Platforms

- The distinction between NUMA and UMA platforms is important from the point of view of algorithm design. NUMA machines require locality from underlying algorithms for performance.
- Programming these platforms is easier since reads and writes are implicitly visible to other processors.
- However, read-write data to shared data must be coordinated (this will be discussed in greater detail when we talk about threads programming).
- Caches in such machines require coordinated access to multiple copies. This leads to the cache coherence problem.
- A weaker model of these machines provides an address map, but not coordinated access. These models are called non cache coherent shared address space machines.

Shared-Address-Space vs. Shared Memory Machines

- It is important to note the difference between the terms shared address space and shared memory.
- We refer to the former as a programming abstraction and to the latter as a physical machine attribute.
- It is possible to provide a shared address space using a physically distributed memory.

Message-Passing Platforms

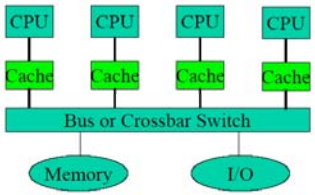
- These platforms comprise of a set of processors and their own (exclusive) memory.
- Instances of such a view come naturally from clustered workstations and non-shared-address-space multicomputers.
- These platforms are programmed using (variants of) send and receive primitives.
- Libraries such as MPI and PVM provide such primitives.

Message Passing vs. Shared Address Space Platforms

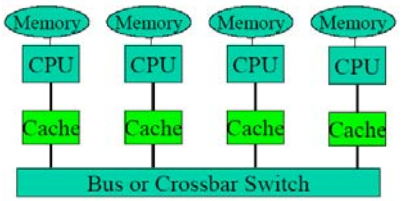
- Message passing requires little hardware support, other than a network.
- Shared address space platforms can easily emulate message passing. The reverse is more difficult to do (in an efficient manner).

SMP architecture

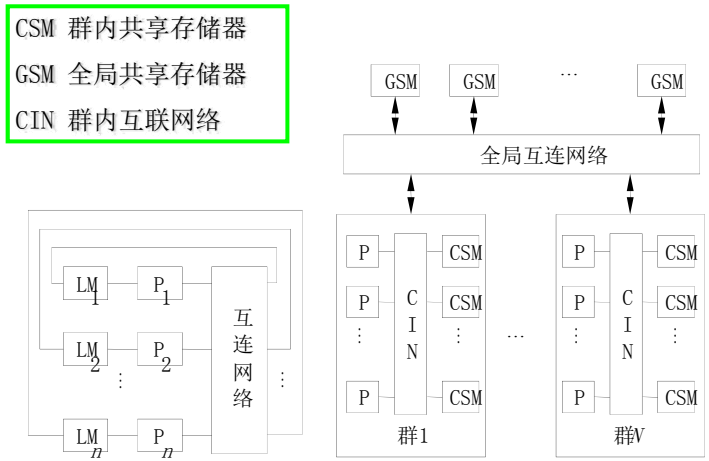
- **SMP uses shared system resources (memory, I/O) that** can be accessed equally from all the processors
- **Cache coherence is maintained**



NUMA architecture

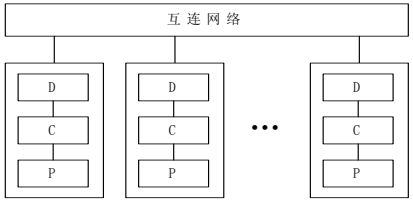


- Shared address space
- Memory latency varies whether you access local or remote memory
- Cache coherence is maintained using an hardware or software protocol

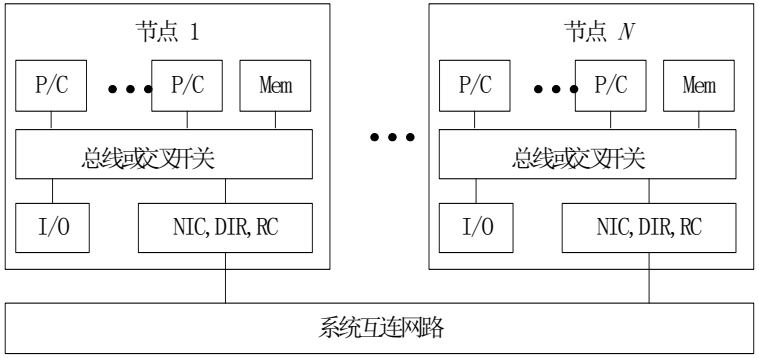


(a)共享本地存储模型 (b)层次式机群模型

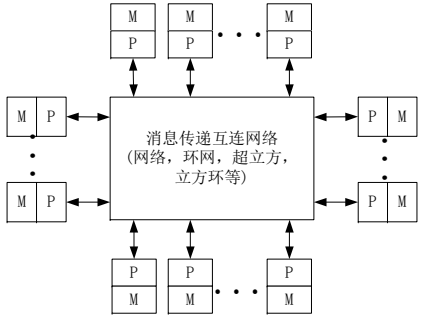
- COMA(Cache-Only Memory Access)模型是全高速缓存存储访问的简称。其特点是：
  - **各处理器节点中没有存储层次结构，全部高速缓存组成了全局地址空间；**
  - **利用分布的高速缓存目录D进行远程高速缓存的访问;**
  - **COMA中的高速缓存容量一般都大于2 级高速缓存容量；**
  - **使用COMA时，数据开始时可任意分配，因为在运行时它最终会被迁移到要用到它们的地方。**



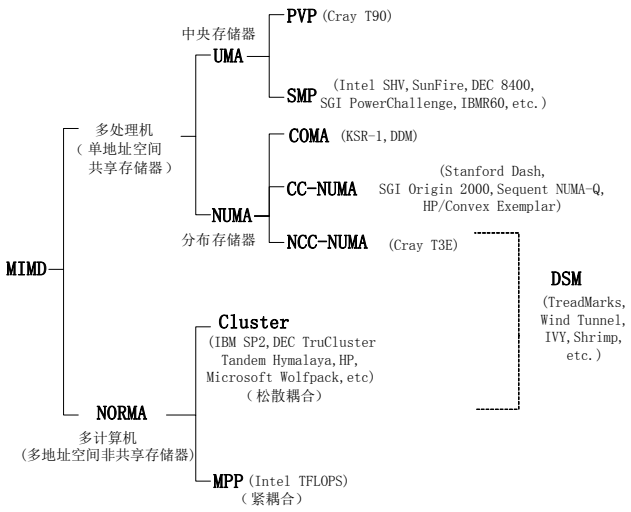
- CC-NUMA（Coherent-Cache Nonuniform Memory Access）模型是高速缓存一致性非均匀存储访问模型的简称。其特点是：
  - **大多数使用基于目录的高速缓存一致性协议；**
  - **保留SMP结构易于编程的优点，也改善常规SMP的可扩放性；**
  - **CC-NUMA实际上是一个分布共享存储的DSM多处理机系统；**
  - **它最显著的优点是程序员无需明确地在节点上分配数据，系统的硬件和软件开始时自动在各节点分配数据，在运行期间，高速缓存一致性硬件会自动地将数据迁移至要用到它的地方。**



- NORMA（No-Remote Memory Access）非远程存储访问模型—消息传递
  - 所有存储器是私有的；
  - 绝大数NUMA都不支持远程存储器的访问；
  - 在DSM中，NORMA就消失了。



构筑并行机系统的不同存储结构



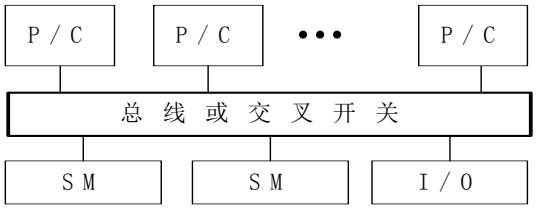
第二章 当代并行机系统

- 2.1 共享存储多处理机系统
  - **2.1.1 对称多处理机SMP结构特性**
- 2.2 分布存储多计算机系统
  - **2.2.1 大规模并行机MPP结构特性**
- 2.3 机群系统
  - **2.3.1 大规模并行处理系统MPP机群SP2**
  - **2.3.2 工作站机群COW**

Asanovic K, Bodik R, Demmel J, et al. A view of the parallel computing landscape[J]. Communications of the ACM, 2009, 52(10): 56-67.

对称多处理机SMP(1)

- 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结构
- 例子：SGI Power Challenge, DEC Alpha Server,Dawning 1



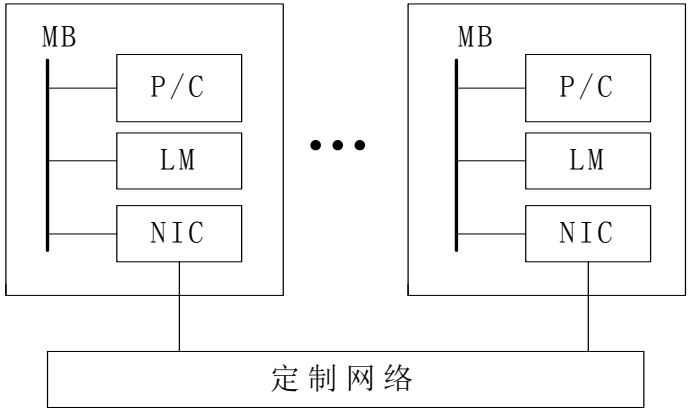
对称多处理机SMP(2)

- 优点
  - **对称性**
  - **单地址空间，易编程性，动态负载平衡，无需显示数据分配**
  - **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，Load/Store完成**
- 问题
  - **欠可靠，BUS,OS,SM**
  - **通信延迟（相对于CPU），竞争加剧(数百至数千指令周期)**
  - **慢速增加的带宽（MB double/3年,是平均值一半;IOB更慢）**
  - **不可扩放性=>(Bus, Crossbar, CC-NUMA)**

大规模并行机MPP

- 成百上千个处理器组成的大规模计算机系统，规模是变化的。
- NORMA结构，高带宽低延迟定制互连。
- 可扩放性：Mem, I/O,平衡设计
- 系统成本：商用处理器，相对稳定的结构，SMP节点,分布
- 通用性和可用性：不同的应用，PVM,MPI,交互，批处理，互连对用户透明，单一系统映象，高可用技术
- 通信要求
- 存储器和I/O能力(total capacity)
- 例子：Intel Option Red(p=9216)

  IBM SP2, Dawning 1000



典型MPP系统特性比较

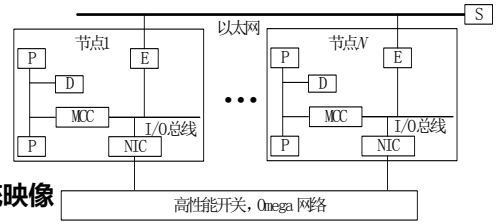
MPP模型	Intel/Sandia ASCI Option Red	IBM SP2	SGI/Cray Origin2000
一个大型样机的配置	9072个处理器，1.8Tflop/s(NSL)	400个处理器，100Gflop/s(MHPCC)	128个处理器，51Gflop/s(NCSA)
问世日期	1996年12月	1994年9月	1996年10月
处理器类型	200MHz, 200Mflop/s Pentium Pro	67MHz，267Mflop/s POWER2	200MHz，400Mflop/s MIPS R10000
节点体系结构和数据存储器	2个处理器，32到256MB主存，共享磁盘	1个处理器，64MB到2GB本地主存，1GB到14.5GB本地磁盘	2个处理器，64MB到256MB分布共享主存和共享磁盘
互连网络和主存模型	分离两维网孔，NORMA	多级网络，NORMA	胖超立方体网络，CC-NUMA
节点操作系统	轻量级内核（LWK）	完全AIX（IBM UNIX）	微内核Cellular IRIX
自然编程机制	基于PUMA Portals的MPI	MPI和PVM	Power C， Power Fortran
其他编程模型	Nx，PVM，HPF	HPF，Linda	MPI，PVM

MPP所用的高性能CPU特性比较

属性	Pentium Pro	PowerPC 602	Alpha 21164A	Ultra SPARC II	MIPS R10000
工艺	BiCMOS	CMOS	CMOS	CMOS	CMOS
晶体管数	5.5M/15.5M	7M	9.6M	5.4M	6.8M
时钟频率	150MHz	133MHz	417MHz	200MHz	200MHz
电压	2.9V	3.3V	2.2V	2.5V	3.3V
功率	20W	30W	20W	28W	30W
字长	32位	64位	64位	64位	64位
I/O	8KB/8KB	32KB/32KB	8KB/8KB	16KB/16KB	32KB/32KB
高速缓存					
2级高速缓存	256KB (多芯片模块)	1~128MB (片外)	96KB (片上)	16MB (片外)	16MB (片外)
执行单元	5个单元	6个单元	4个单元	9个单元	5个单元
超标量	3路(Way)	4路	4路	4路	4路
流水线深度	14级	4~8级	7~9级	9级	5~7级
SPECint 92	366	225	>500	350	300
SPECfp 92	283	300	>750	550	600
SPECint 95	8.09	225	>11	N/A	7.4
SPECfp 95	6.70	300	>17	N/A	15
其它特性	CISC/RISC 混合	短流水线长 L1高速缓存	最高时钟频率最大片上2级高速缓存	多媒体和图形指令	MP机群总线可支持4个CPU

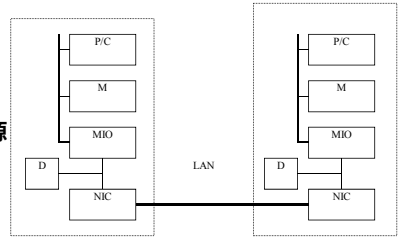
机群型大规模并行机SP2

- 设计策略：
  - **机群体系结构 (RS6000, AIX)**
  - **标准环境**
  - **标准编程模型**
  - **系统可用性**
  - **精选的单一系统映像**
- 系统结构：
  - **高性能开关 HPS 多级Ω网络**
  - **宽节点、窄节点和窄节点2**



工作站机群COW

- 分布式存储，MIMD，工作站+商用互连网络，每个节点是一个完整的计算机，有自己的磁盘和操作系统，而MPP中只有微内核
- 优点：
  - **投资风险小**
  - **系统结构灵活**
  - **性能/价格比高**
  - **能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**



- 例子：Berkeley NOW，Alpha Farm, FXCOW

典型的机群系统

典型的机群系统特点一览表	
名称	系统特点
Princeton:SHRIMP	PC商用组件，通过专用网络接口达到共享虚拟存储，支持有效通信
Karsruhe:Parastation	用于分布并行处理的有效通信网络和软件开发
Rice:TreadMarks	软件实现分布共享存储的工作站机群
Wisconsin:Wind Tunnel	在经由商用网络互连的工作站机群上实现分布共享存储
Chica、Maryl、Penns:NSCP	国家可扩放机群计划：在通过因特网互连的3个本地机群系统上进行元计算
Argonne:Globus	在由ATM连接的北美17个站点的WAN上开发元计算平台和软件
Syracuse:WWVM	使用因特网和HPCC技术，在世界范围的虚拟机上进行高性能计算
HKU:Pearl Cluster	研究机群在分布式多媒体和金融数字库方面的应用
Virgina:Legion	在国家虚拟计算机设施上开发元计算软件

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

第三章 并行计算性能评测

- 3.1 并行机的一些基本性能指标
- 3.2 加速比性能定律
  - **3.2.1 Amdahl定律**
  - **3.2.2 Gustafson定律**
  - **3.2.3 Sun和Ni定律**
- 3.3 可扩放性评测标准
  - **3.3.1 并行计算的可扩放性**
  - **3.3.2 等效率度量标准**
  - **3.3.3 等速度度量标准**
  - **3.3.4 平均延迟度量标准**

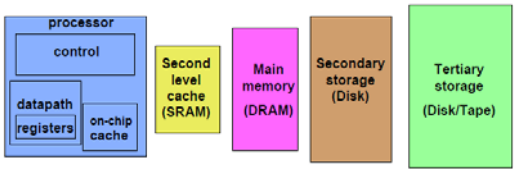
CPU的某些基本性能指标

- 工作负载
  - **执行时间（特定计算机系统上给定应用的Elapsed time）**
  - **浮点运算数(加减乘、赋值、比较、类型转换1；除法、开平方4；正余弦8)**
  - **指令数目（百万条指令，MIPS）**
- 并行执行时间$T_n = T_{comput} + T_{paro} + T_{comm}$，
  T_{comput}为计算时间，
  T_{paro}为并行开销时间(进程管理、组操作、进程查询)
  T_{comm}为相互通信时间（同步、通信、聚合）
- 例：估计APRAM模型下执行时间

$$\max\left(\frac{T_1}{n}, T_\infty\right) \le T_n \le \frac{T_1}{n} + T_\infty$$

Memory Hierarchy

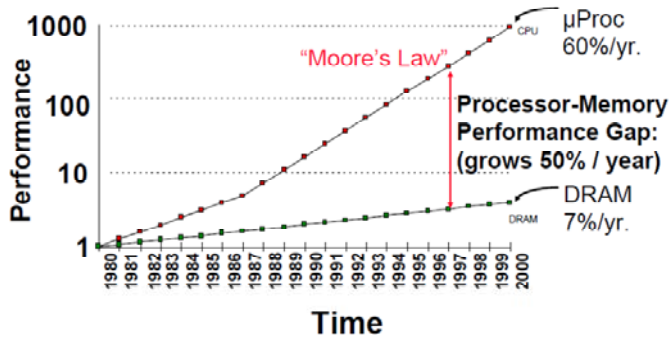
- Most programs have a high degree of locality in their accesses
  - **spatial locality:** accessing things nearby previous accesses
  - **temporal locality:** reusing an item that was previously accessed
- Memory hierarchy tries to exploit locality



Speed	1ns	10ns	100ns	10ms	10sec
Size	B	KB	MB	GB	TB

Processor-DRAM Gap (latency)

- Memory hierarchies are getting deeper
  - Processors get faster more quickly than memory



Approaches to Handling Memory Latency

- Bandwidth has improved more than latency
- Approach to address the memory latency problem
  - Eliminate memory operations by saving values in small, fast memory (cache) and reusing them
    - need temporal locality in program
  - Take advantage of better bandwidth by getting a chunk of memory and saving it in small fast memory (cache) and using whole chunk
    - need spatial locality in program
  - Take advantage of better bandwidth by allowing processor to issue multiple reads to the memory system at once
    - concurrency in the instruction stream, eg load whole array, as in vector processors; or prefetching

存储器性能

- 存储器的层次结构(C,L,B)
  - 容量C：各个物理存储器件能保存多少字节的数据
  - 延迟L：读取各个物理器件中一个字的时间
  - 带宽B：1秒内各层物理器件中能传送多少字节
- 估计存储器的带宽
- RISC add r1,r2,r3 r 8bytes 100MHz(加法单拍完成)
- B = 3*8*100*10^6 B/s= 2.4GB/s

并行与通信开销

- 并行（Tparo）和通信开销（Tcomm）：相对于计算很大。
  - **PowerPC (每个周期 15ns 执行4flops;**
    创建一个进程1.4ms 可执行372000flops)
- 开销的测量：
  - **乒--乓方法（Ping-Pong Scheme）节点0发送m个字节给节点1；节点1从节点0接收m个字节后，立即将消息发回节点0。总的时间除以2，即可得到点到点通信时间，也就是执行单一发送或接收操作的时间。**
  - **可一般化为热土豆法（Hot-Potato），也称为救火队法（Fire-Brigade) 0——1 —— 2 —— … —— -n-1 —— 0**

hot potato routing

Last modified: Friday, February 13, 2004

_ _ **(n.)** A form of routing in which the nodes of a network have no buffer to store packets in before they are moved on to their final predetermined destination. In normal routing situations, when multiple packets contend for a single outgoing channel, packets that are not buffered are dropped to avoid congestion. But in hot potato routing, each packet that is routed is constantly transferred until it reaches its final destination because the individual communication links can not support more than one packet at a time. The packet is bounced around like a "hot potato," sometimes moving further away from its destination because it has to keep moving through the network. This technique allows multiple packets to reach their destinations without being dropped. This is in contrast to "store and forward" routing where the network allows temporary storage at intermediate locations. Hot potato routing has applications in optical networks where messages made from light can not be stored in any medium.

Ping-Pong Scheme

```
if ( my _node _id =0 )  then /*发送者*/
   start _time =second (  )
            send an m-byte message to node 1
            receive an m-byte message from node 1
   end_time = second (  )
   total_time = end_time – start_time
      communication_time[i] = total_time/2
       else if ( my_node_id = 1 )  then  /*接收者*/
            receive an m-byte message from node 0
            send an m-byte message to node 0
   endif
```

并行开销的表达式：点到点通信

- 通信开销 $t(m) = t_0 + m/r_\infty$
  - **通信启动时间 t_0**，是 一个由系统决定的参数而与m无关
  - **渐近带宽r_∞：传送无限长的消息时的通信速率**，即系统在传送大量数据(忽略启动时延的影响)时的最大持续宽带
  - **半峰值长度$m_{1/2}$：达到一半渐近带宽所要的消息长度**
  - **特定性能π_0：表示短消息带宽**
  - **这四个参数的关系**

$$t_0 = m_{1/2} / r_\infty = 1/\pi_0$$

并行开销的表达式：整体通信

- 典型的整体通信有：
  - **播送（Broadcasting）：处理器0发送m个字节给所有的n个处理器**
  - **收集（Gather）：处理0接收所有n个处理器发来在消息，所以处理器0最终接收了m n个字节；**
  - **散射（Scatter）：处理器0发送了m个字节的不同消息给所有n个处理器，因此处理器0最终发送了m n个字节；**
  - **全交换（Total Exchange）：每个处理器均彼此相互发送m个字节的不同消息给对方，所以总通信量为mn^2个字节；**
  - **循环移位（Circular-shift）：处理器i发送m个字节给处理器i+1，处理器n-1发送m个字节给处理器0，所以通信量为m n个字节。**

路障和通信开销比较表

表 2.2 SP_2机器的整体通信和路障同步开销表达式一览表

整体通信操作	表 达 式
播 送	52logn + (0.029logn)m
收集 / 散射	(17logn +15) + (0.025n-0.02)m
全 交 换	80logn + (0.03n $^{1.29}$)m
循环移位	(6logn +60) + (0.003logn + 0.04)m
路障同步	94logn + 10

机器的成本、价格与性/价比

- 机器的成本与价格
- 机器的性能/价格比 Performance/Cost Ratio ：系指用单位代价（通常以百万美元表示）所获取的性能（通常以MIPS或MFLOPS表示）
- 利用率（Utilization）：可达到的速度与峰值速度之比

算法级性能评测

- 加速比性能定律
  - **并行系统的加速比是指对于一个给定的应用，并行算法（或并行程序）的执行速度相对于串行算法（或串行程序）的执行速度加快了多少倍。**
  - **Amdahl 定律**
  - **Gustafson定律**
  - **Sun Ni定律**
- 可扩放性评测标准
  - **等效率度量标准**
  - **等速度度量标准**
  - **平均延迟度量标准**

Speedup

- The *speedup* of a parallel application is

  Speedup(p) = Time(1)/Time(p)
- Where
  - **Time(1) = execution time for a single processor and**
  - **Time(p) = execution time using p parallel processors**
- If Speedup(p) = p we have *perfect speedup* (also called *linear scaling*)
- As defined, speedup compares an application with itself on one and on p processors, but it is more useful to compare
  - **The execution time of the best serial application on 1 processor**

  versus
  - **The execution time of best parallel algorithm on p processors**

Efficiency

- The *parallel efficiency* of an application is defined as

  Efficiency(p) = Speedup(p)/p
  - **Efficiency(p) <= 1**
  - **For perfect speedup Efficiency (p) = 1**
- We will rarely have perfect speedup.
  - **Lack of perfect parallelism in the application or algorithm**
  - **Imperfect load balancing (some processors have more work)**
  - **Cost of communication**
  - **Cost of contention for resources, e.g., memory bus, I/O**
  - **Synchronization time**
- Understanding why an application is not scaling linearly will help finding ways improving the applications performance on parallel computers.

Example: Speedup

- Adding *n* numbers using *n* processing elements

$$T_p = \Theta(\log n)$$
$$T_1 = n$$
$$S = \Theta\left(\frac{n}{\log n}\right)$$

Example: Speedup

- A serial bubble sort of 10^5 records takes 150 seconds
- A serial quick sort of 10^5 records takes 30 seconds
- Parallel version of bubble sort (odd-even sort) takes 40 seconds on four processing elements
- Speedup:
  - **150/40=3.75**
  - **30/40=0.75**

Superlinear Speedup

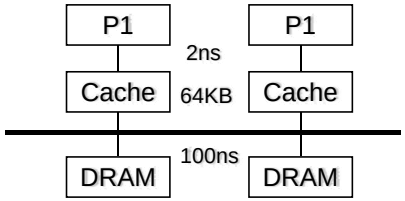
Question: can we find "*superlinear*" speedup, that is

Speedup(p) > p ?

- Choosing a bad "baseline" for T(1)
  - **Old serial code has not been updated with optimizations**
  - **Avoid this, and always specify what your baseline is**
- Shrinking the problem size per processor
  - **May allow it to fit in small fast memory (cache)**
- Application is not deterministic
  - **Amount of work varies depending on execution order**
  - **Search algorithms have this characteristic**

Example: Superlinear Speedup

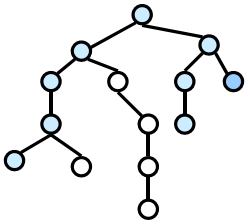
- Problem size: W, cache hit rate: 80%
  - **Effective memory access time=2*0.8+100*0.2=21.6ns**
  - **Processing rate=1/21.6GFLOPS=46.3MFLOPS, assume one FLOP/memory access**



- Problem size: W/2, cache hit rate: 90%
  - **Effective memory access time=2*0.9+100*0.08+400*0.02=17.8ns**
  - **Processing rate=2/17.8GFLOPS=2*56.18MFLOPS=112.36**
  - **Speedup=112.36/46.3=2.43**

Example: Superlinear Speedup

- Search Problem
  - **$14t_c/5t_c=2.8$**



Amdahl's Law

- Suppose only part of an application runs in parallel
- Amdahl's law
  - **Let f be the fraction of work done serially,**
  - **So $(1 - f)$ is fraction done in parallel**
  - **What is the maximum speedup for P processors?**

Speedup(p) = T(1)/T(p)

T(p) = (1-f)*T(1)/p +f*T(1)

= T(1)*((1-f) + p*f)/p

Speedup(p) = p/(1 + (p-1)*f)

assumes perfect speedup for parallel part

Even if the parallel part speeds up perfectly, we may be limited by the sequential portion of code.

Amdahl定律

$$S = \frac{W_s + W_p}{W_s + \frac{W_p}{p}}$$

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

$$\lim_{p \to \infty} S = \frac{1}{f}$$

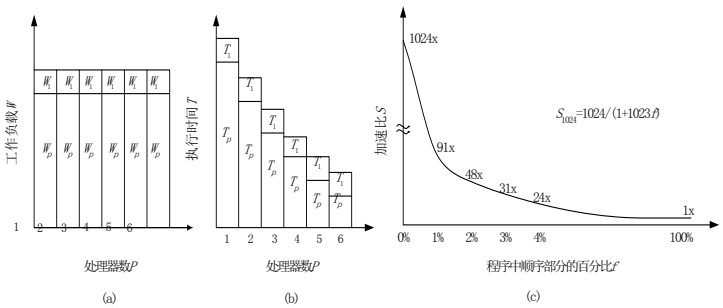
Amdahl定律

$$S = \frac{W_s + W_p}{W_s + \frac{W_p}{p} + W_o}$$

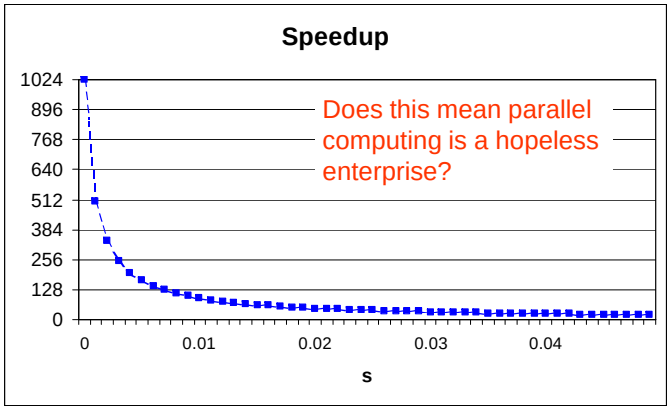
$$= \frac{W}{fW + \frac{W(1-f)}{p} + W_o}$$

$$= \frac{p}{1 + f(p-1) + W_o p / W}$$

Amdahl定律



Amdahl's Law (for 1024 processors)



See: Gustafson, Montry, Benner, "Development of Parallel Methods for a 1024 Processor Hypercube", SIAM J. Sci. Stat. Comp. 9, No. 4, 1988, pp.609.

Scaled Speedup

- Speedup improves as the problem size grows
  - **Among other things, the Amdahl effect is smaller**
- Consider
  - **scaling the problem size with the number of processors (add problem size parameter, n)**
  - **for problem in which running time scales linearly with the problem size: T(1,n) = T(1,1)*n**
  - **let n=p (problem size on p processors increases by p)**

ScaledSpeedup(p,n) = T(1,n)/T(p,n)

T(p,n) = (1-*f*)*n*T(1,1)/p +*f* *T(1,1)

= (1-*f*)*T(1,1) + *f* *T(1,1)=T(1,1)

ScaledSpeedup(p,n) = n = p

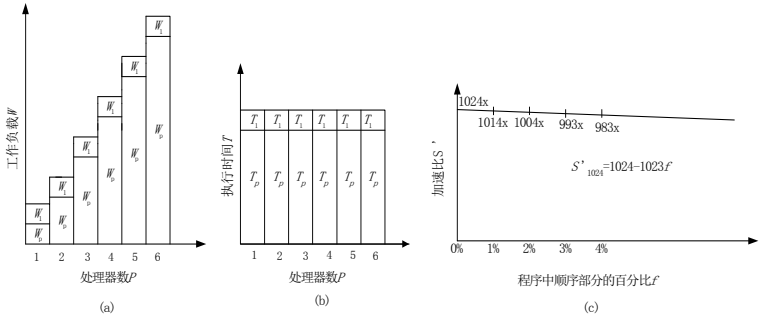
assumes serial work does not grow with n

Gustafson定律

$$S' = \frac{W_s + pW_p}{W_s + pW_p / p} = \frac{W_s + pW_p}{W_s + W_p}$$

$$S' = f + p(1-f)$$
$$= p + f(1-p)$$
$$= p - f(p-1)$$

Gustafson定律（续）



Gustafson定律

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

Sun 和 Ni定律

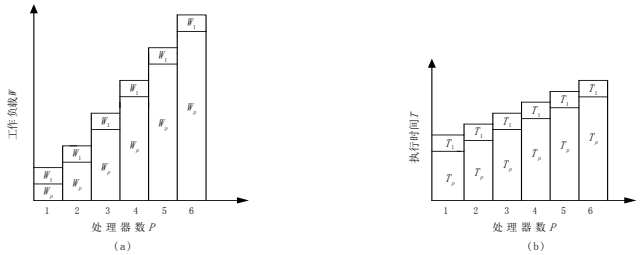
- **基本思想：**
  - 只要存储空间许可，应尽量增大问题规模以产生更好和更精确的解（此时可能使执行时间略有增加）
  - 假定在单节点上使用了全部存储容量M并在相应于W的时间内求解之，此时工作负载$W = fW + (1-f)W$
  - 在p 个节点的并行系统上，能够求解较大规模的问题是因为存储容量可增加到pM。令因子$G(p)$反映存储容量增加到p倍时并行工作负载的增加量，所以扩大后的工作负载 $W = fW + (1-f)G(p)W$
- **存储受限的加速公式 ：**

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

- **并行开销W_o:**

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

Sun 和 Ni定律(续)



- G(p)=1时就是Amdahl加速定律；
- G(p)=p 变为 $f + p(1-f)$，就是Gustafson加速定律
- G(p)>p时，相应于计算机负载比存储要求增加得快，此时 Sun和 N i 加速均比 Amdahl 加速和 Gustafson 加速为高。

Scaled Efficiency

- Previous definition of *parallel efficiency* was

  Efficiency(p) = Speedup(p)/p
- We often want to scale problem size with the number of processors, but scaled speedup can be tricky
  - **Previous definition depended on a linear work in problem size**
- May use alternate definition of efficiency that depends on a notion of throughput or rate, R(p):
  - **Floating point operations per second**
  - **Transactions per second**
  - **Strings matches per second**
- Then

  Efficiency(p) = R(p)/(R(1)*p)
- May use a different problem size for R(1) and R(p)

Three Definitions of Efficiency: Summary

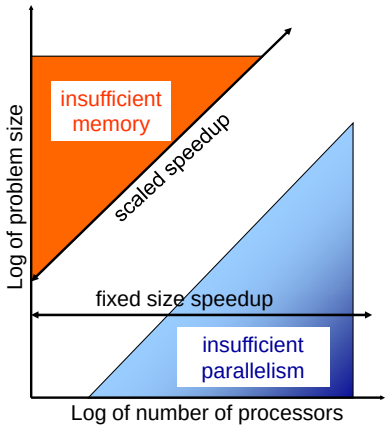
- People use the word "efficiency" in many ways
- Performance relative to advertised machine peak

  Flop/s in application / Max Flops/s on the machine
  - **Integer, string, logical or other operations could be used, but they should be a machine-level instruction**
- Efficiency of a fixed problem size

  Efficiency(p) = Speedup(p)/p
- Efficiency of a scaled problem size

  Efficiency(p) = R(p)/(R(1)*p)
- All of these may be useful in some context
- Always make it clear what you are measuring

Performance Limits



3.3 可扩放性评测标准

- 并行计算的可扩放性（Scalability）也是主要性能指标
  - **可扩放性最简朴的含意是在确定的应用背景下，计算机系统（或算法或程序等）性能随处理器数的增加而按比例提高的能力**
- 影响加速比的因素：处理器数与问题规模
  - **求解问题中的串行分量**
  - **并行处理所引起的额外开销（通信、等待、竞争、冗余操作和同步等）**
  - **加大的处理器数超过了算法中的并发程度**
- 增加问题的规模有利于提高加速的因素：
  - **较大的问题规模可提供较高的并发度；**
  - **额外开销的增加可能慢于有效计算的增加；**
  - **算法中的串行分量比例不是固定不变的（串行部分所占的比例随着问题规模的增大而缩小）。**
- 增加处理器数会增大额外开销和降低处理器利用率，所以对于一个特定的并行系统（算法或程序），它们能否有效利用不断增加的处理器的能力应是受限的，而度量这种能力就是可扩放性这一指标。

可扩放性评测标准（续)

- 可扩放性:调整什么和按什么比例调整
  - **并行计算要调整的是处理数p和问题规模W，**
  - **两者可按不同比例进行调整，此比例关系（可能是线性的，多项式的或指数的等）就反映了可扩放的程度。**
- 可扩放性研究的主要目的：
  - **确定解决某类问题用何种并行算法与何种并行体系结构的组合，可以有效地利用大量的处理器；**
  - **对于运行于某种体系结构的并行机上的某种算法当移植到大规模处理机上后运行的性能；**
  - **对固定的问题规模，确定在某类并行机上最优的处理器数与可获得的最大的加速比；**
  - **用于指导改进并行算法和并行机体系结构，以使并行算法尽可能地充分利用可扩充的大量处理器**
- 目前无一个公认的、标准的和被普遍接受的严格定义和评判它的标准

等效率度量标准

- 令t_e^i 和t_o^i 分别是并行系统上第i个处理器的有用计算时间和额外开销时间（包括通信、同步和空闲等待时间等）

$$T_e = \sum_{i=0}^{p-1} t_e^i \qquad T_o = \sum_{i=0}^{p-1} t_o^i$$

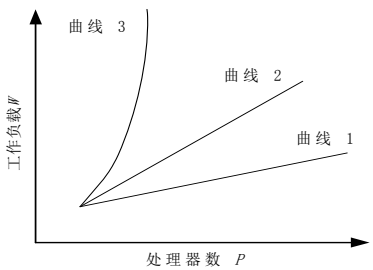
- T_p 是p个处理器系统上并行算法的运行时间，对于任意i显然有 $T_p = t_e^i + t_o^i$ ，且 $T_e + T_o = pT_p$
- 问题的规模W为最佳串行算法所完成的计算量$W=T_e$

$$S = \frac{T_e}{T_p} = \frac{T_e}{\frac{T_e + T_o}{p}} = \frac{p}{1+\frac{T_o}{T_e}} = \frac{p}{1+\frac{T_o}{W}} \qquad E = \frac{S}{P} = \frac{1}{1+\frac{T_o}{T_e}} = \frac{1}{1+\frac{T_o}{W}}$$

- 如果问题规模W 保持不变，处理器数p增加，开销T_o增大，效率E下降。为了维持一定的效率（介于0与1之间），当处理数p增大时，需要相应地增大问题规模W的值。由此定义函数f_E（p）为问题规模W随处理器数p变化的函数，为等效率函数（ISO-efficiency Function）（Kumar1987）

等效率度量标准（续）

- 对于某算法若随着p增加$f_E(p)$增加较小则称其为可扩放的。
- 曲线1表示算法具有很好的扩放性；曲线2表示算法是可扩放的；曲线 3表示算法是不可扩放的。
- 优点：简单可定量计算的、少量的参数计算等效率函数
- 缺点：如果T_o无法计算出（在共享存储并行机中）



等速度度量标准

- p 表示处理器个数，W表示要求解问题的工作量或称问题规模（在此可指浮点操作个数），T为并行执行时间，定义并行计算的速度V为工作量W除以并行时间T
- p个处理器的并行系统的平均速度定义为并行速度V除以处理器个数 p：

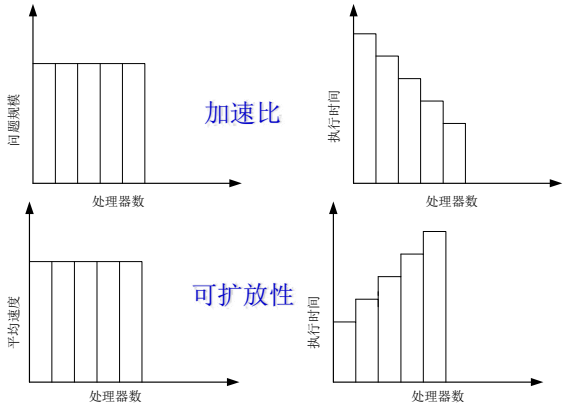
$$\overline{V} = \frac{V}{p} = \frac{W}{pT}$$

- W是使用p个处理器时算法的工作量，令W'表示当处理数从p增大到p'时，为了保持整个系统的平均速度不变所需执行的工作量，则可得到处理器数从 p到p'时平均速度可扩放度量标准公式 （值越大表示可扩放性越好）

$$\Psi(p,p') = \frac{W/p}{W'/p'} = \frac{p'W}{pW'}$$

等速度度量标准（cont’d）

- 优点：直观地使用易测量的机器性能速度指标来度量
- 缺点：某些非浮点运算可能造成性能的变化



平均延迟度量标准

- T_i为P_i的执行时间，包括延迟L_i，P_i的总延迟时间为“L_i+启动时间+停止时间”。定义系统平均延迟时间为

$$\bar{L}(W,p)=\sum_{i=1}^{p}\left(T_{para}-T_i+L_i\right)\Big/p$$

- $pT_{para}=T_o+T_s$ $\qquad T_o = p\bar{L}(W,p)$

$$\bar{L}(W,p)=T_{para}-T_{seq}/p$$

-
- $\bar{L}(W,p)$ 在p个处理器上求解工作量为W问题的平均延迟
- $\bar{L}(W',p')$在p’个处理器上求解工作量为W’问题的平均延迟当处理器数由p变到p’，而推持并行执行效率不变，则定义平均延迟可扩放性度量标准为

$$\Phi(E,p,p')=\frac{\bar{L}(W,p)}{\bar{L}(W',p')}$$

平均延迟度量标准（续）

- 优点：平均延迟能在更低层次上衡量机器的性能
- 缺点：需要特定的软硬件才能获得平均延迟