



南京大學

NANJING UNIVERSITY

并行处理系统

殷亚凤

智能软件与工程学院

苏州校区南雍楼东区225

yafeng@nju.edu.cn , <https://yafengnju.github.io/>



并行处理系统

- 并行处理系统概述
- 多处理器系统
- 多计算机系统
- 向量处理技术和SIMD技术
- 并行处理编程模式简介





并行处理的主要技术问题

- **互连**

- 并行处理：将多个计算模块和存储模块互连，通过控制这些它们的并行工作来提高处理速度。

- **数据一致性**

- 在不同的计算模块中会设置共享高速缓存和主存，因此，在并行处理系统中存在复杂的数据一致性问题。

- **同步控制**

- 共享存储时，多个计算模块访问同一块数据需解决数据的互斥访问。
- 分布计算和存储时，需要对各个模块进行同步控制。

- **任务划分**

- 大任务分解成子任务并行执行，包含算法分解和数据划分两个方面。

- **并行程序设计**

- 对系统中运行的程序进行并行化处理描述，以说明哪些处理逻辑段可并行执行、哪些有先后顺序关系，以及处理的数据可以怎样划分等。



并行处理的主要技术问题

- **资源调度和管理**
 - 计算资源的调度和存储资源的管理，比在串行处理系统中复杂得多。
- **容错性和安全性**
 - 预防由于单个节点失效可能带来的数据丢失、程序出错或系统崩溃等。
 - 要求系统必须考虑良好的**可靠性设计**、**失效检测**和**恢复机制**。
- **性能分析与评估**
 - 并行处理性能通常用**加速比**来度量
 - 串行系统与并行系统中执行时间比值
 - 并行系统与串行系统的作业吞吐量比值
 - 评价并行系统的指标包括**可用性**、**可扩展性**、**负载均衡**、**可靠性**等





并行处理系统的分类

- 按指令和数据处理方式划分（Flynn分类）
 - **SISD（单指令流单数据流）**：指令级并行（超流水、超标量、动态调度）
 - **SIMD（单指令流多数据流）**：数据级并行（如Intel的MMX、SSE、AVX等）
 - **MIMD（多指令流多数据流）**：多计算机和多处理器系统
- 按地址空间的访问方式划分（MIMD）
 - **多计算机系统**：计算节点有各自独立编址的存储器，通过消息传递方式访问其他节点的存储器，也称**消息传递系统**
 - **多处理器系统**：各计算节点共享统一编址的存储器，可通过LOAD和STORE指令访问系统中的存储器，也被称为**共享存储多处理器（Shared memory multiProcessor, SMP）系统**
- 按访存时间是否一致划分（**多处理器系统**）（Non-Uniform Memory Access）
 - **一致性访存（UMA）**：每个处理器访存时间一致，通常共享一个存储器
 - **非一致性访存（NUMA）**：处理器的访存时间不一致，与存储器位置有关
 - **非一致性内存访问（CC-NUMA）**：Cache-Coherent NUMA，支持cache一致性的NUMA



并行处理系统的分类

• 按处理单元的位置及其互连方式划分

- **多核**：一个CPU芯片中含多个（2、4、8等）核，共享LLC和主存 **属多处理器**
- **对称多处理器（Symmetric MultiProcessor, SMP）**：相同类型CPU通过总线互连，并等同地位共享所有存储资源。即多个CPU对称工作。可见SMP就是一种UMA结构多处理器。PC、工作站和服务器等多采用SMP结构。
- **大规模并行处理机（MPP）**：以**专用内联网络**连接数量众多处理单元而构成的并行计算系统。例如，可通过专用互连网络（如Mesh、交叉开关）将数量达几百甚至几千个的SMP服务器连接成MPP，SMP服务器之间协同工作，以完成同一个任务。
- **集群（Cluster）**：以**高速网卡**将若干PC或SMP服务器或工作站连接而成的并行计算系统，其中每个节点有各自的独立编址的内存和磁盘，属于紧耦合异构多计算机系统（消息传递机制）。
- **网格（Grid）**：以**因特网等广域网**将远距离分布的一组异构计算机系统连接而成的分布式并行处理系统，属于松耦合异构多计算机系统。 **属多计算机**
- **众核**：一个GPU芯片中含几百个简单核，众多并行线程同时执行
- **APU**：**CPU+GPU融合**





系统互联及输入输出组织

- 并行处理系统概述
- **多处理器系统**
- 多计算机系统
- 向量处理技术和SIMD技术
- 并行处理编程模式简介

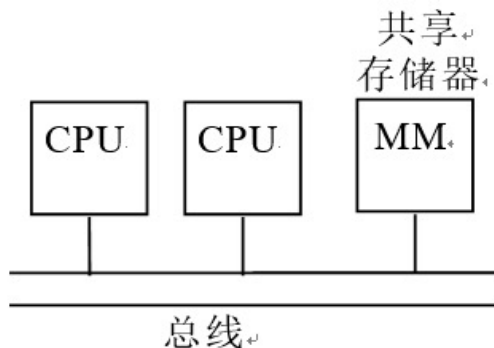




多处理器系统：UMA多处理器结构

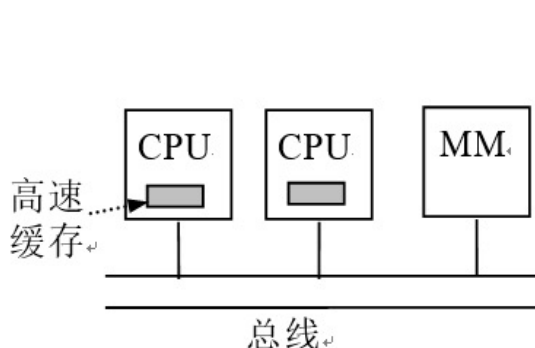
• 基于总线连接的UMA系统

共享MM：存放
可写的共享变量



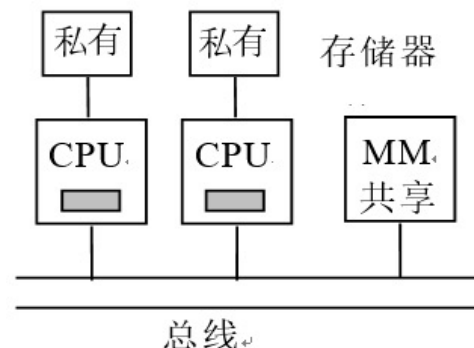
(a) 没有高速缓存

编译器、链接器和操作系统等各种
系统软件的功能需进行相应调整



(b) 有高速缓存

私有MM：存放程序、
非共享变量、只读数
据、栈内信息等



(c) 有高速缓存和私有存储器

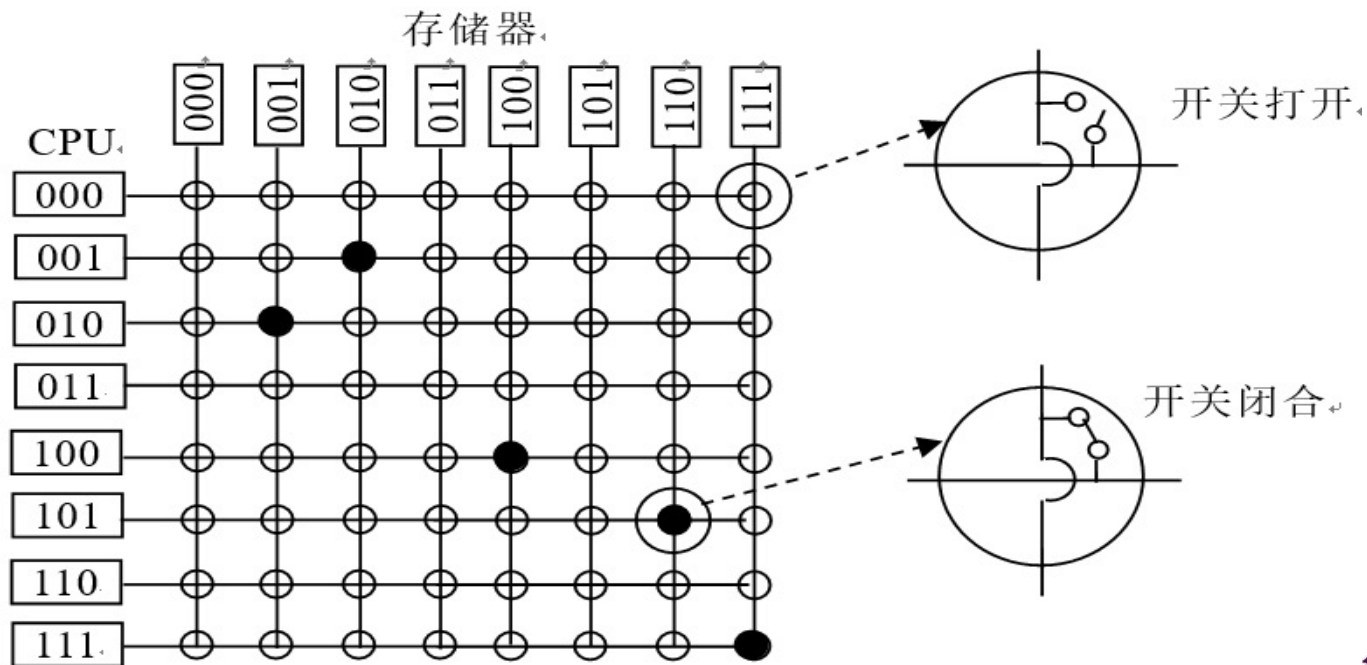
有Cache时：需Cache一致性协议，最常用的是侦听协议，如MESI协议，采用写无效策略（当侦听到一个写操作时，使其他副本无效）

缺点：总线繁忙，所连接模块数受限，因此有系统考虑基于交叉开关网络的UMA系统



多处理器系统：UMA多处理器结构

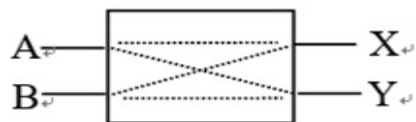
- 基于交叉开关网络的UMA系统：连接 m 个CPU模块到 n 个存储器模块的最简单电路是交叉开关连接网路。交叉开关的数量为 $m \times n$ ，故**规模受限于开关数量**。



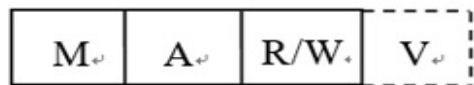


多处理器系统：UMA多处理器结构

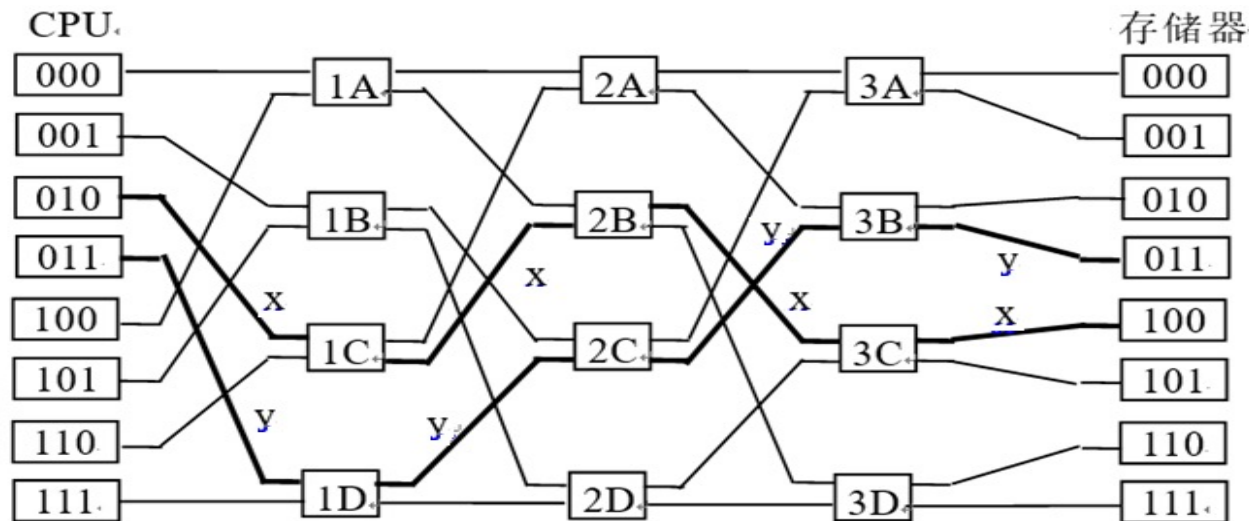
- 基于多级交换网络的UMA系统：在CPU模块和存储器模块之间用**多级小规模交叉开关**（如 2×2 ）网络连接，在数据访问过程中需要携带消息，消息中带有存储模块号、地址、读/写操作类型。



(a) 2×2 交叉开关



(b) 消息格式



(c) Omega 交换网络



多处理器系统：NUMA多处理器结构

- **UMA系统的缺点：**

- 要保证每个处理器的访存时间与存储模块位置无关，因而其连接的**处理器规模受到一定的限制**。

(要构建更大规模的多处理器系统，须打破访存时间一致的限制，在保证单一地址空间的前提下允许访存时间不一致，因此，出现了非一致内存访问（NUMA）多处理机。)

- **NUMA系统的特点：**

- 与UMA一样，所有**存储器统一编址**，都可用LOAD和STORE指令访问，因此，UMA程序可在NUMA系统上执行。
- NUMA系统中，**本地访问快于非本地访问**。
- 所连接**处理器规模比UMA更大**。
- **在节点互连、并行编程、cache一致性方面与UMA多处理器不同**。

- **处理器中带有**一致性高速缓存**时，系统被称为CC-NUMA。**





多处理器系统：NUMA多处理器结构

基于目录的CC-NUMA系统

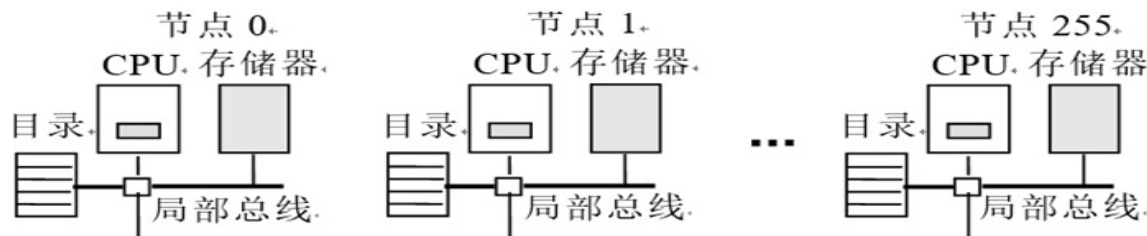
例：节点18执行LOAD指令，
其MMU将虚拟地址转换为
物理地址480000C8H
48H=72，**C8H=200(3,8)**

含义：从节点72的本地存储器
第200单元读信息

查72#目录中第3项，
 $V=0$ (miss)。

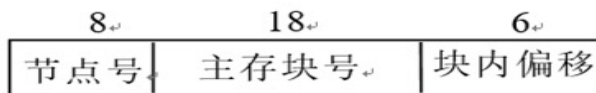
从72#的200号单元读出，
送节点18中某Cache行，改
第3项的 $V=1$ ，节点为18

目录：记录本地存储模块中各主存块的副本在哪个节点的Cache中，都不存在，则有效位 $V=0$ 。



互连网络长什么样？

(a) CC-NUMA 互连结构。



(b) 主存地址划分。

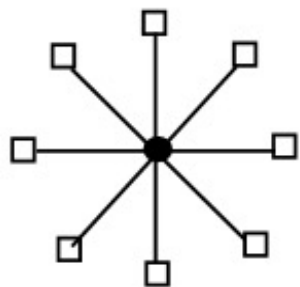
$2^{18}-1$		
		\vdots
4	1	35
3	0	
2	1	72
1	0	
0	0	

(c) 节点 72 中的目录。

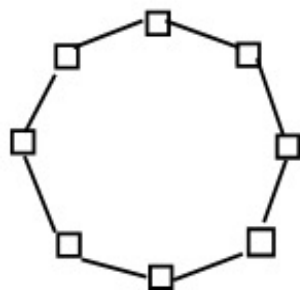


多处理器系统：NUMA多处理器结构

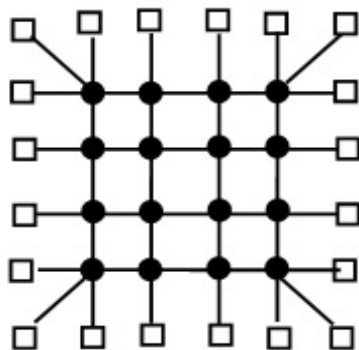
• NUMA互连网络结构



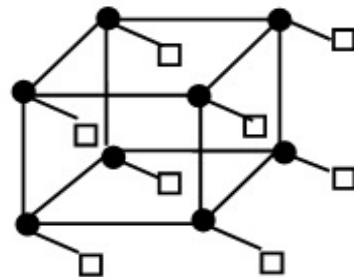
(a) 星形



(b) 环形



(c) 网格



(d) 立方体

方块表示一个CPU-存储器节点，黑圆点表示交换器



多处理器系统

- **片级多处理器**

- **功耗墙**： $P=CV^2f$ （P-功耗，C-时钟跳变时门电路电容，与集成度成正比；V是电压；f是主频）。功耗与集成度和主频成正比。集成度和主频的大幅度提升导致了功耗急剧增大。
- **指令级并行墙**：ILP技术发展到了极限
- 2004年，Intel 4GHz主频处理器未能解决散热问题，标志升频技术时代的终结。
- 2005年，Intel宣布从单处理器性能提升战略转向多核微处理器架构战略。
- **多核**：一个CPU芯片包含多个简单处理器核（Core），通过多核并行计算提高性能。也称为**片级多处理器计算机（Chip-level MultiProcessors, CMP）**。

- **硬件多线程**（从CPU的角度看：一个线程就是一个指令序列）

- **多线程并行**：为每个线程提供单独的通用寄存器组、PC以及线程切换机制
- **细粒度**：在多个线程之间轮流交叉执行指令，能在每个时钟周期切换线程
- **粗粒度**：在某线程出现较大阻塞开销时才切换，例如，cache缺失时切换
- **同时多线程（如Intel的超线程）**：在单处理器或单核中设置多套线程状态部件而共享cache和功能部件。能在实现ILP的同时实现线程级并行，即：同一时钟内在不同发射槽中发射不同线程中的指令。



共享存储器的同步控制

- 为什么需要同步控制？

- 每个处理器核有一个或多个指令流（硬件多线程时）在执行。若多个核都要访问共享存储器，则由于每个核在执行程序时存在流水线阻塞、功能部件冲突、指令顺序调度等各种随机情况，**多个核的程序之间访存顺序存在不确定性**，从而造成执行结果的不确定。

- 如何进行同步控制？

- ISA必须规定所用的存储器一致性模型（Memory Consistency Model），简称存储器模型，并提供一套配套同步指令。

- 三种代表性存储器模型

- 顺序（Sequential）一致性模型**：每个核完全按照程序指定的访存顺序执行，**不可改变访存顺序**。对存储器的访问都应该是串行的，效率低。
- 宽松（Relaxed）一致性模型**：允许在一定的条件下**改变核内程序对于不同存储单元的访存顺序**。通常用一个存储器屏障指令，设置一个同步点，前面的访存不能在屏障后进行、后面的也不能越过屏障提前进行。即**双向限制**！
- 释放（Release）一致性模型**：**只限定一个方向**的访存指令执行顺序。采用获取-释放机制实现，可用获取（Acquire）指令拦截后面的访存指令，或用释放（Release）指令拦截前面的访存指令。



共享存储器的同步控制

- 举例说明存储器一致性模型的使用

要求：核0将一块数据写入存储器中某个区域，然后，通知核1读取此数据块。

多核应用程序主要实现思想如下：

- (1) **核0**和**核1**约定一个共享的全局变量作为同步标志。全局变量在主存中分配一个存储单元，两个核都可以访问该存储单元，以设置或读取其值；
- (2) **核0**完成了写数据块的操作后，就将一个“特殊的值”写入共享变量单元中作为旗语；
- (3) **核1**不断监测此共享变量的值，一旦检测到“特殊的值”，则认为可以安全地读取数据块。

因此，两个核上的程序可以各自抽象为以下操作序列。

—核0：写入数据块→设置旗语。

—核1：监测旗语→检测到“特殊的值”→读取数据块。

显然，核0的“写入数据块”和“设置旗语”的操作一定不能改变顺序

核1的“监测旗语”和“读取数据块”的操作也一定不能改变顺序

试分别描述顺序一致性模型、宽松一致性模型和释放一致性模型的实现。





共享存储器的同步控制

- 在顺序一致性模型的多核系统中，按以下操作序列进行。

- 核0：写入数据块→设置旗语。
- 核1：监测旗语→检测到“特殊的值”→读取数据块。

核0和核1完全按照各自的顺序来访存，可能的顺序有：

- (1) 写入数据块→设置旗语→监测旗语→检测到“特殊的值”→读取数据块。
- (2) 写入数据块→监测旗语→设置旗语→检测到“特殊的值”→读取数据块。

显然，以上两种顺序都没有问题。

- 在宽松一致性模型的多核系统中，由于数据块和共享变量所在存储地址不同，若不用屏障指令，则编译器或处理器可能会进行指令顺序调度优化，使程序最终执行的结果不满足程序员的预期。因此需要在程序中插入屏障指令FENCE。

- 核0：写入数据块→FENCE指令→设置旗语。
- 核1：监测旗语→检测到“特殊的值”→FENCE指令→读取数据块。

显然，屏障指令（前、后不越屏障）保证了程序员所要求的访存顺序。

- 若将“FENCE指令”和“设置旗语”合成“释放旗语”指令；将“监测旗语”和“FENCE指令”合成“获取旗语”指令，就是释放一致性模型中的方案。



共享存储器的同步控制

- 在**释放一致性模型的多核系统中**，按以下操作序列进行。
 - **核0**：写入数据块→释放旗语。
 - **核1**：获取旗语→检测到“特殊的值”→读取数据块。
“释放旗语”指令拦截前面的访存，以保证前面的访存不会在其之后执行；
“获取旗语”指令拦截后面的访存，以保证后面的访存不会在其之前执行。
显然，上述操作序列满足程序员要求。
- **总结比较：**
 - **顺序一致性模型**：绝对保证访存顺序，不能进行指令调度优化操作，效率低。
 - **宽松一致性模型**：不同存储地址的访存需要用屏障指令保证前、后两个方向的访存顺序。没有顺序要求时，可不用屏障指令，因而可进行指令调度优化。
 - **释放一致性模型**：不同存储地址的访存操作可用“释放”或“获取”指令单方向拦截访存操作。没有顺序要求时，可不用“释放”和“获取”指令，因而可进行指令调度优化。比宽松一致性模型更加宽松！
 - 还有更多存储器一致性模型；共享存储器同步控制问题不仅限于多核处理器系统，在共享存储器多处理器系统中以及硬件多线程共享存储器时也一样。





系统互联及输入输出组织

- 并行处理系统概述
- 多处理器系统
- **多计算机系统**
- 向量处理技术和SIMD技术
- 并行处理编程模式简介

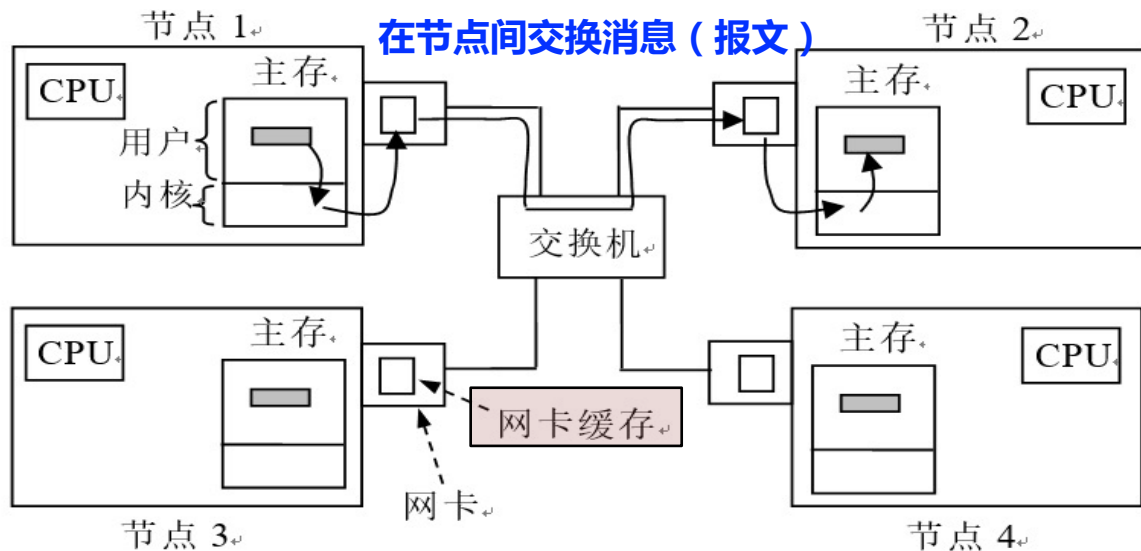




多计算机系统

• 集群 (Cluster)

- **互连方式**：局域网连接普通PC、通过消息传递进行通信，比多处理器系统中的互连网络慢，**集群中信息访问为微秒级，多处理器中为纳秒级**
- **体积**：集群在一个或多个房间甚至更大空间中，而多处理器通常在一个机箱中
- **应用广泛**：数据库系统、文件服务器、Web服务器等



网卡中通常包含多个**DMA**通道或专门的I/O处理器.

交换机可采用**分组交换** (将消息划分成更小的“分组”) 和**电路交换**两种机制





多计算机系统

- 网络 (Grid)

- **互连方式**：利用互联网把分散在不同地理位置的多个异构计算机组成一台逻辑上的“虚拟超级计算机”
- **体积**：物理上可能位于世界各地
- **构建思路**：不同的管理系统上运行相应的中间件，以使用户和程序可以通过方便的、一致的方式访问所有资源
- **基本原理**：在网格中的每个计算机中运行一个特殊的程序，这个程序可以用来管理计算机并使计算机加入到网格中。因此，这个程序通常需要处理用户认证及远程登录、资源发布与发现、作业调度与分配等。当网格中的某个用户需要计算机完成某个任务时，网格软件决定何处有空闲的硬件、软件和数据资源，然后将作业迁移到有资源的计算机处，安排执行并收集处理结果返回给用户。
- **预期目标**：实现可靠的、具有较高容错和容灾性的系统，并且可以节省资源，实现资源共享
- **应用困难**：还存在许多技术问题，设想难以实现！



系统互联及输入输出组织

- 并行处理系统概述
- 多处理器系统
- 多计算机系统
- **向量处理技术和SIMD技术**
- 并行处理编程模式简介





向量处理机简介

- **向量处理机的基本思想**

- 面向向量型数据的并行计算：各分量彼此无关、各自独立
- 主要用于求解大型问题，须具有集中式大容量高吞吐量的主存和向量寄存器，以源源不断地供给操作数和取走运算结果

- **举例：考察一个C语言循环程序（大约需运行700--800条标量指令）**

for (i=0; i<64; i++) c[i]=a[i]+b[i];

- 在向量处理机（32个向量寄存器，每个存放64个32位数据字，装入/存储指令（v_lw和v_sw）可从内存连续256个单元装入数据到一个向量寄存器或将向量寄存器内容存到内存连续单元中，运算指令（如v_add）可同时对两个向量寄存器中的64个32位字并行计算，并将结果存到另一个向量寄存器），则上述C语言循环程序段对应的指令只有4条。

- **v_lw \$v_t1, 0(\$s1) #将a[0]到a[63]取到\$v_t1中**
- **v_lw \$v_t2, 0(\$s2) #将b[0]到b[63]取到\$v_t2中**
- **v_add \$v_t3, \$v_t1, \$v_t2 #计算a[i]+b[i]，存到\$v_t3中**
- **v_sw \$v_t3, 0(\$s3) #将a[i]+b[i]的值存到c[i]中**



Intel架构中的SIMD技术

- **IA-32的浮点处理架构有两种：**
 - **浮点协处理器x87架构 (x87 FPU)**
 - ✓ 8个80位寄存器ST(0) ~ ST(7)（采用栈结构），栈顶为ST(0)
 - **由MMX发展而来的SSE架构**
 - ✓ **MMX指令**使用8个64位寄存器MM0~MM7，借用8个80位寄存器ST(0)~ST(7)中64位尾数所占的位，可同时处理8个字节，或4个字，或2个双字，或一个64位的数据
 - ✓ MMX指令并没带来3D游戏性能的显著提升，故相继推出SSE指令集，它们都采用**SIMD（单指令多数据，也称数据级并行）技术**
 - ✓ **SSE指令集**将80位浮点寄存器扩充到128位多媒体扩展通用寄存器XMM0~XMM7，可同时处理16个字节，或8个字，或4个双字（32位整数或单精度浮点数），或两个四字的数据，而且从SSE2开始，还支持128位整数运算或同时并行处理两个64位双精度浮点数





Intel架构中的SIMD技术

• 用简单的例子来比较普通指令与数据级并行指令的执行速度

- ✓为使比较结果不受访存操作影响，下例中的运算操作数在寄存器中
- ✓为使比较结果尽量准确，例中设置的循环次数较大: $0x4000000 = 2^{26}$
- ✓例子只是为了说明指令执行速度的快慢，并没有考虑结果是否溢出

以下是普通指令写的程序

所用时间约为22.643816s

080484f0 <dummy_add>:

```
80484f0: 55                push %ebp
80484f1: 89 e5             mov %esp, %ebp
80484f3: b9 00 00 00 04    mov $0x4000000, %ecx
80484f8: b0 01             mov $0x1, %al
80484fa: b3 00             mov $0x0, %bl
80484fc: 00 c3             add %al, %bl
80484fe: e2 fc             loop 80484fc <dummy_add+0xc>
8048500: 5d                pop %ebp
8048501: c3                ret
```

循环400 0000H = 2^{26} 次，每次只有一个数（字节）相加





Intel架构中的SIMD技术

以下是SIMD指令写的程序

所用时间约为1.411588s

08048510 <dummy_add_sse>:

8048510: 55

8048511: b8 00 9d 04 10

8048516: 89 e5

8048518: 53

8048519: bb 20 9d 04 14

804851e: b9 00 00 40 00

8048523: 66 0f 6f 00

8048527: 66 0f 6f 0b

804852b: 66 0f fc c8

804852f: e2 fa

8048531: 5b

8048532: 5d

8048533: c3

push %ebp

mov \$0x10049d00, %eax

mov %esp, %ebp

push %ebx

mov \$0x14049d20, %ebx

mov \$0x400000, %ecx

movdqa (%eax), %xmm0

movdqa (%ebx), %xmm1

paddb %xmm0, %xmm1

loop 804852b <dummy_add_sse+0x1b>

pop %ebx

pop %ebp

ret

22.643816s/ 1.411588s

≈16.041378,与预期结果一致!

SIMD指令并行执行效率高!

} SIMD指令

dqa : 两个对齐四字

循环400000H=2²²次, 每次同时有128/8=16个数(字节)相加



南京大学
NANJING UNIVERSITY



GPU简介

- **CPU中的MMX、SSE以及AVX等采用SIMD技术**
 - 针对绘图运算、3D游戏加速、视频编码等图形处理能力的增强而提出
 - 包含在Intel CPU中，但CPU不专用于图形处理，需从CPU中分离出来
- 随着计算机游戏产业的不断发展，形成了**专门的图形处理功能部件：GPU (Graphics Processing Unit) 芯片**
 - 1999年NVIDIA发布第一款GPU
 - 早期GPU主要用于3D图形渲染等图形处理
 - 随着以CUDA为代表的GPU通用计算API的普及，GPU被广泛应用于石油勘测、天文计算、流体力学模拟、分子动力学仿真、生物计算等科学计算领域
 - **GPU的含义也从原来专门的图形处理器转变为GPGPU (General-Purpose computing on GPU)**



系统互联及输入输出组织

- 并行处理系统概述
- 多处理器系统
- 多计算机系统
- 向量处理技术和SIMD技术
- 并行处理编程模式简介





并行处理编程模式简介

- **共享存储变量方式（多线程并行程序设计）**

- 用于共享存储器多处理器系统
- 通常将一个任务分解成若干个处理逻辑段（称为一个线程）
- 线程间通过共享存储变量进行数据交换
- **Pthread**：在串行程序中加入Pthread函数调用实现多线程并行处理。
- 所有Pthread线程含有：
 - ✓ 一个线程标识符（有专门的线程创建、线程终止等Pthread函数）
 - ✓ 包括程序计数器PC的一组寄存器（用于存放各线程的现场信息）
 - ✓ 一组存储在结构（struct）类型变量中的属性，这些属性包括栈大小、调度参数以及线程所需的其他信息
- **OpenMP**：在串行程序中插入编译指导命令指示哪个程序段可并行执行

例如：在for语句前加编译指导语句“`#pragma omp parallel for`”，告诉编译器将for语句编译成多线程并行代码。在编译指导语句中需指定并行线程个数、线程中私有变量、对哪个变量进行规约以及如何规约等。OpenMP中还提供API函数，包括运行环境设置函数、锁操作函数和时间操作函数等





并行处理编程模式简介

• 消息传递方式

- 用于分布式存储器访问的并行处理系统（如集群）
- 通过消息传递（Message Passing）方式进行数据的交换。（采用分布式存储访问，无法通过执行LOAD/STORE指令访问另一节点私有存储器，因此，无法通过共享存储变量来交换数据）
- 每个子任务作为一个进程在各自独立的计算节点上并行执行，因此也可以被狭义地理解为多进程并行程序设计方式
- 最典型的是MPI（Message Passing interface，消息传递接口）标准
- MPI：在串行程序中加入MPI并行编程接口函数，实现多进程并行处理，所有节点运行同一个程序。并行程序段部分由不同计算节点处理不同数据
- 由专门的API接口函数来指定哪一段是并行程序段、共有多少计算节点/进程数目、如何进行同步通信以传递消息等



并行处理编程模式简介

- **MapReduce大规模并行计算框架**

- 由谷歌提出的一种**面向大规模数据处理**的并行处理模式和方法
- 可构建数百甚至数千个节点的基于集群的高性能并行计算平台
- 程序员仅需要编程实现Map和Reduce两个基本操作接口
- **Map操作**主要负责对一组数据记录进行某种重复处理
- **Reduce操作**主要负责对Map操作的中间结果进行某种规约并输出结果，进行reduce处理之前, 必须等到所有map函数完成，故进入reduce前需有一个**同步障(barrier)**; 这个阶段也负责对map的中间结果数据进行**收集整理(aggregation & shuffle)处理**, 以便reduce能够计算最终结果，
- 最终汇总所有reduce的输出结果即可获得最终结果





并行处理编程模式简介

• CUDA并行编程模型

- 用于众核GPU+多核CPU的异构系统
- 对C/C++语言扩展了三类功能：层次结构线程组、同步栅和共享内存
- 应用“任务—数据块—数据元素”
- GPU “TPC—SM—SP (SFU、DFU、LD/ST) ”
- CUDA “网格 (grid) —线程块 (thread block) —线程” ← **层次结构线程组**
- 可在串行程序中调用一个并行执行的内核 (kernel) 函数程序，网格就是执行相同内核函数程序的一组线程块，相当于处理一个子任务
- **举例：一个应用包含两个顺序执行的子任务：**
kernelA<<<6,10>>>(params)：含6个一维线程块，各线程块有10个线程
kernelB<<<(6,4), (8,3)>>>(params)：含6×4的二维线程块，各线程块含8×3个线程
必须在A和B之间设置同步栅





提问

Q & A

殷亚凤

智能软件与工程学院

苏州校区南雍楼东区225

yafeng@nju.edu.cn , <https://yafengnju.github.io/>



南京大學
NANJING UNIVERSITY