并行处理与体系结构(3)
Parallel Processing and Architecture

2025-4-13

1

回顾：并行结构

- 并行性：同时做多件“事情”
- “事情”：指令，操作，任务
- 为什么采用并行结构
  - 绝对性能：主要目标，执行时间或任务吞吐量
    - 程序的执行时间由Amdahl定律控制
  - 功耗和能耗
    - 4N个单元工作在F/4频率下的功耗低于N个单元工作在F频率下
  - 提高成本效率和可扩展性，降低复杂性
    - 性能价格比
    - 很难设计出一个复杂单元能够有N个简单单元一样的执行效果
  - 提高可靠性：在空间上冗余执行
- 关键的使能因素
  - 半导体和互连网络技术的发展
  - 软件技术的发展

2

回顾：串行的硬件和软件

- 冯诺依曼结构
  - 几乎所有计算机都采用的控制流架构
  - 两个基本特征：存储程序和顺序执行指令
  - 冯诺依曼瓶颈
- 多任务处理
  - 进程和线程
- 改进冯诺依曼
  - 指令级并行
    - 流水线及其冒险
    - 乱序执行及精确异常
  - 分层存储结构
    - Cache
    - 虚拟存储
  - 投机执行
    - 动态分支预测
    - 动态内存歧义消解

3

Flynn分类法

- Mike Flynn, “Very High-Speed Computing Systems,” Proc. of IEEE, 1966
- 基于指令流、数据流的概念
- 根据硬件对指令流和数据流的支持不同来区分并行的组织
  - SISD: Single Instruction and Single Data Stream (uniprocessor) 单指令操作单个数据元素
  - SIMD: Single I and Multiple D Streams (GPUs) 单指令操作多个数据元素
    - 阵列处理机
    - 向量处理器
  - MISD: Multiple I and Single D Streams 多指令操作单个数据元素
    - 最接近的形式：脉动阵列处理器，流处理器
  - MIMD: Multiple I and Multiple D Streams (multicores) 多指令操作多个数据元素（多指令流）
    - 多处理器
    - 多线程处理器

4

基于模型的分类

- 共享内存（Shared-memory）
- 消息传递（Message-passing）
- 数据流（Dataflow）
- 脉动阵列（systolic）
- 数据并行（Data parallel）
- ……

共享内存（Shared-memory）

- 系统中的所有处理器可以直接访问系统中的所有存储器，这种结构提供了多处理器共享数据的便捷机制
  - 便捷性：位置透明；与单处理器类似
  - 廉价：与其它模型相比
- 存储器可以是集中式的也可以是分布式的
- 是一种单一地址空间的结构

共享内存（Shared-memory）

- 编程模型：
  - 可以很容易的支持各种并行模型：fork-join，任务队列，数据并行
  - 并行的线程通过共享内存来实现通信和同步
  - 通用模型，很容易模拟其它模型
- 这种结构的经典问题是可扩展性

消息传递（Message-passing）

- 处理器只可以直接访问本地存储器，所有的通信和同步通过消息机制实现
- 发送消息通常会带来额外开销：
  - 构造一个消息（增加消息头部），拷贝数据到缓冲区，发送数据，接收数据到缓冲区，拷贝数据到用户进程地址空间
  - 很多步骤需要操作系统参与
- 利用消息同步通常基于各种握手协议
- 一个最大的好处是容易扩展

消息传递（Message-passing）

- 支持多种编程模型：
  - Actor model，面向并发对象的编程
- 使用非常广泛
  - 集群系统
  - 云计算
  - 高性能计算

9

9

数据流（Dataflow）

- 数据流模型中，指令激活与否取决于指令中的操作数是否准备好
- 控制流模型中，计算是按照指令之间的显式或隐式顺序执行的
- 数据流模型的一个优点是，所有的数据依赖关系都清晰地表现在数据流图中，因此其并行性是显而易见的

10

10

脉动阵列（systolic）

- 基本原则
  - 用处理单元的规则阵列替代单个的处理单元，精心处理不同处理单元之间的数据流
  - 在不增加内存带宽的前提下获得高的吞吐量
- 与常规的流水线相比有显著的特色
- 多用于信号处理

11

11

数据并行（Data parallel）

- 编程模型假设每一个处理器都与一组数据相关联
- 所有处理器执行类似的操作处理不同的数据
- 在处理高并行的代码时非常有效
- 被图形处理器广泛采用

12

12

并行的类型

- **指令级并行**
  - 一个指令流中的不同指令可以并行执行
  - 流水线, 乱序执行, 投机执行, VLIW
  - 数据流
- **数据并行**
  - 数据的不同片段可以被并行的操作
  - SIMD: 向量处理, 阵列处理
  - 脉动阵列, 流处理器
- **任务级并行**
  - 不同的“任务/线程”可以被并行执行
  - 多线程
  - 多处理 (多核)

任务级并行: 生成任务

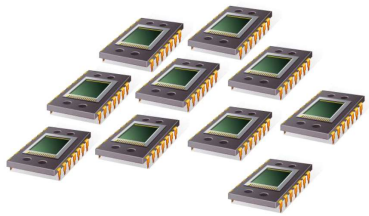
- 将一个问题分割成多个相关的任务(线程)
  - 显式: 并行编程
    - 当问题中的任务能够很自然地划分时
      - Web/数据库请求
    - 当任务的边界不那么清晰时
  - 透明/隐式: 线程级投机
    - 投机地分割单个线程
- 同时运行多个独立的任务(进程)
  - 当有多个进程时
    - 批处理的仿真, 不同用户的进程, 云计算的工作负载
  - 不能提升单个任务的性能

基于硬件的多线程

- **粗粒度多线程**
  - 基于“量子”
  - 基于事件
- **细粒度多线程**
  - 每周期
  - Thornton, “CDC 6600: Design of a Computer,” 1970.
  - Burton Smith, “A pipelined, shared resource MIMD computer,” ICPP 1978.
- **同时多线程**
  - 能够同时由多个线程分发指令
  - 有效提升执行单元的利用率

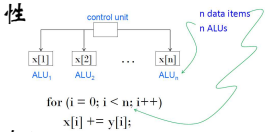
并行硬件

- 程序员可以通过代码来利用这些硬件



SIMD

- 通过在处理器之间划分数据来实现并行性
- 对多个数据项应用相同的指令
- 称为数据并行



- 如果我们没有数据项那么多的ALU怎么办？
- 迭代地划分工作和过程
- 例如m = 4个ALU，n = 15个数据项

ALU_1	ALU_2	ALU_3	ALU_4
X[0]	X[1]	X[2]	X[3]
X[4]	X[5]	X[6]	X[7]
X[8]	X[9]	X[10]	X[11]
X[12]	X[13]	X[14]	

- 缺点
  - 所有ALU都被要求执行相同的指令，或者保持空闲
  - 在经典设计中，它们也必须同步运行
  - ALU没有指令存储器
  - 对大数据并行问题有效，但对其他类型更复杂的并行问题无效

17

向量处理器

- 对数据的数组或向量进行操作，而传统CPU对单个数据元素或标量进行操作
- 向量寄存器
  - 能够存储操作数向量并同时对其内容进行操作
- 向量化和流水线功能单元
- 相同的操作应用于向量中的每个元素(或元素对)
- 向量指令
  - 对向量而不是标量进行操作
- 交叉存取
  - 多个内存“bank”，或多或少可以独立访问
  - 将向量的元素分布在多个存储体上，从而减少或消除读/写连续元素的延迟
- 跨步内存访问和硬件scatter/gather
  - 程序访问位于固定间隔的向量元素

18

向量处理器的优点和缺点

- 优点
  - 很快
  - 方便使用
  - 向量化编译器擅长识别要利用的代码
  - 编译器还可以提供关于无法向量化的代码的信息
    - 帮助程序员重新评估代码
  - 高内存带宽
  - Cache行中的每个数据都能被使用
- 缺点
  - 不能像其他并行架构一样处理不规则的数据结构
  - 处理更大问题的能力非常有限(可扩展性)

19

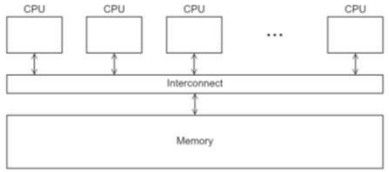
MIMD

- 支持在多个数据流上同时运行多个指令流
- 通常由一组完全独立的处理单元或核组成，每个处理单元或核都有自己的控制单元和ALU

20

共享内存系统

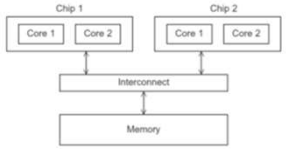
- 一组自主处理器通过互连网络连接到存储系统
- 每个处理器可以访问所有存储单元
- 处理器通常通过访问共享数据结构进行隐式通信
- 大多数广泛可用的共享内存系统使用一个或多个多核处理器
  - 单个处理器上的多个CPU或内核



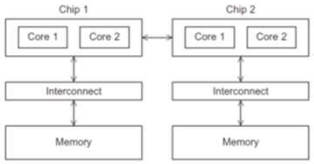
21

UMA VS. NUMA 多核系统

- UMA
  - 对于所有核，访问所有存储器位置的时间是相同的



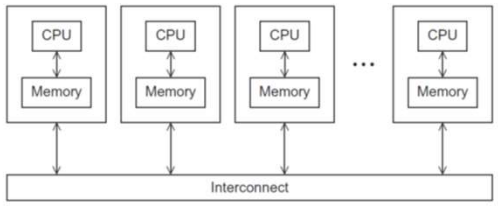
- NUMA
  - 与必须通过另一个芯片访问存储器相比，可以更快地访问核直接连接的存储器



22

分布式内存系统

- 集群(最常见的形式)
  - 商业系统的集合
  - 通过商业互连网络连接
- 集群的节点是由通信网络连接的独立计算单元
  - 又叫混合系统



23

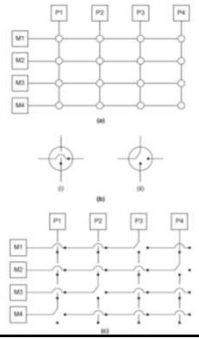
互连网络

- 影响分布式和共享内存系统的性能
- 可分为两类
  - 共享内存互连
  - 分布式存储互连

24

共享内存互连

- **总线互连**
  - 并行通信连线和一些控制总线访问的硬件的集合
  - 通信连线由连接到它的设备共享
  - 随着连接到总线的设备数量增加，对总线使用的争用增加，性能下降
- **交换互连**
  - 使用交换机来控制相连设备之间的数据路由
  - 交叉开关
    - 允许不同设备之间同时通信
    - 比总线快
    - 但是交换机和链路的成本相对较高



25

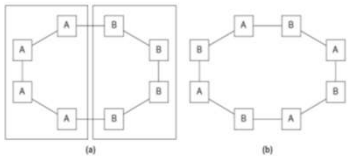
分布式存储互连

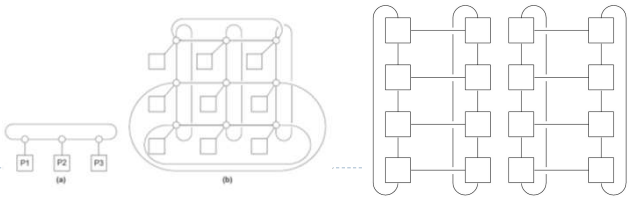
- **两类**
  - 直接互连
    - 每个交换机直接连接到一个处理器存储器对，并且这些交换机相互连接
  - 间接互连
    - 交换机可能不直接连接到处理器

26

直接互连

- **带宽**
  - 链路传输数据的速率
  - 通常以每秒兆位或兆字节为单位
- **对分带宽**
  - 衡量网络质量的一种方法
  - 衡量“同时通信的数量”或连接性的指标
  - 两半网络之间可以同时进行多少次通信
  - 不是计算连接两半的链路数量，而是计算链路带宽的总和

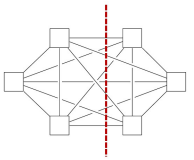




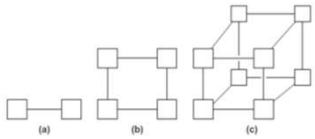
27

直接互连

- **全连接网络**
  - 每个交换机都直接连接到其他交换机
- **超立方体**
  - 高度连接的直接互连
  - 一维超立方体是具有两个处理器的全连接系统
  - 二维超立方体是由两个一维超立方体通过连接相应的开构成
  - 类似地，三维超立方体是由两个二维超立方体构成的



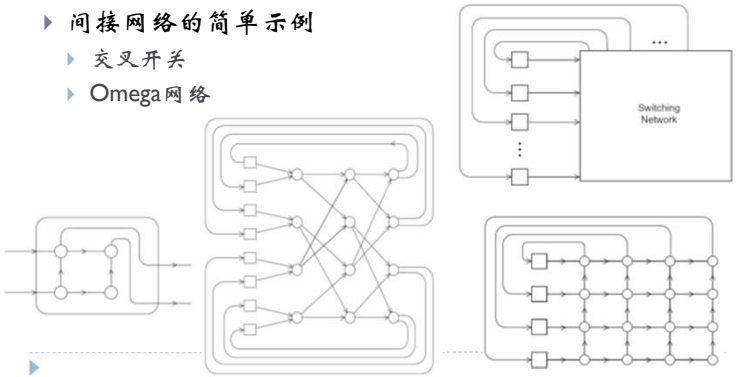
对分带宽= $p^2/4$



28

间接互连

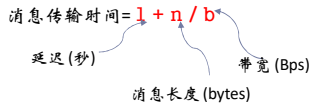
- 通常显示为单向链路和一组处理器，每个处理器都有一个输出链路和一个输入链路，以及一个交换网络
- 间接网络的简单示例
  - 交叉开关
  - Omega网络



29

通信的性能指标

- 每当数据被传输时，我们感兴趣的是数据到达目的地需要多长时间
- 延迟
  - 从源设备开始传输数据到目的设备开始接收第一个字节所经过的时间
- 带宽
  - 目标在开始接收第一个字节后接收数据的速率



30

多处理器的类型

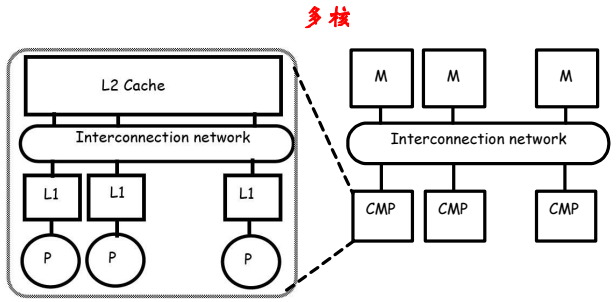
- 松耦合多处理器
  - 没有共享的全局存储地址空间
  - 多机网络
    - 基于网络的多处理器
  - 通常通过消息传递编程
    - 通过显式调用(send, receive)通信
- 紧耦合多处理器
  - 共享全局存储地址空间
  - 传统的多处理:对称多处理器(SMP)
    - 现有的多核处理器,多线程处理器
  - 编程模型与单处理器类似(多任务单处理器)，除了
    - 操作共享数据需要同步

31

31

紧耦合(共享内存)多处理器

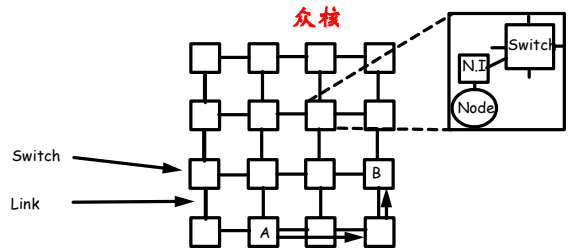
- 共享内存多处理器架构在今天随处可见
  - 由于多核/众核处理器的出现



32

32

紧耦合(共享内存)多处理器



这是片上多处理器（CMP）“未来”的样子

NODE= CORE+CACHE, CORE+CACHE+MEMORY, MEMORY, CACHE

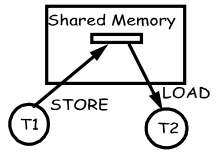
33

33

紧耦合(共享内存)多处理器

- CMP的处理器会共享部分内存
- 通信隐式的通过Load和store实现

共享内存是一种通信机制



对处理器的相对速度不做假设
=> 在需要的时刻进行同步

无论只有一个副本还是有多个副本，同步都是需要的
同步与一致性无关，除非它们直接有交互

34

34

紧耦合(共享内存)多处理器

高速缓存

- 需要Cache来保持核的高执行效率
- 在任何时候可能（在cache 中或在其它缓冲中）存在相同地址的多个拷贝

这些拷贝必须呈现出是一致的(也就是相当于只有一个拷贝)

- 当只有一个拷贝时，一致性自然得到保障
- 当同一个地址的多个拷贝总是相同时，一致性自然得到保障

在实践中这是不可能强制的

35

35

紧耦合(共享内存)多处理器

维持事件的序 — 也被称作内存一致性模型

- 虽然一致性需要面对的通常是一个单一地址
- 但是它也必须对不同的地址提供序的保障
- 考虑如下代码:

假设 A 和 flag 的初始值都是 0

P1	P2
...	...
A:=1;	while(flag==0)do nothing;
flag:=1;	print A;
...	...

- 虽然你可能会认为 P2 打印的值是 1
- 但是事实并不总是如此

它取决于ISA所假定的内存一致性模型

36

36

紧耦合(共享内存)多处理器的主要难点

- 共享存储同步
  - 锁，原子操作
- Cache 一致性
- 访存操作的序
  - 程序员希望硬件提供什么？
- 资源共享，竞争和分区
- 通信：互连网络
- 负载均衡

37

37

并行编程：问题选择

- 可以对两件事使用并行计算：
  - 加速现有应用程序
  - 提升应用程序的效果
- 应用程序应该是计算密集型的，除非可以实现显著的加速，否则并行化不值得努力
- 加速比和问题规模的扩展
  - 扩展模式：
    - Strong scaling: 保持问题规模不变
    - Weak scaling: 增加问题规模

38

38

并行加速比

- $a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$
- 假设每个操作占用1个周期，没有通信开销，每个操作可以在不同的处理器上执行
- 用单个处理器执行有多快？
  - 假设没有流水线或者对指令的并发执行
- 用3个处理器有多快？
  - 加速比(speedup)

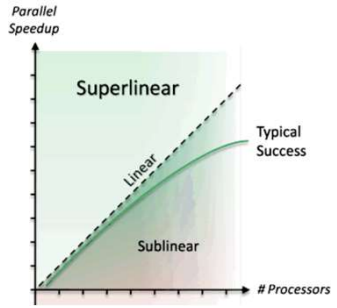
Horner, "A new method of solving numerical equations of all orders, by continuous approximation," Philosophical Transactions of the Royal Society, 1819.

39

39

超线性加速比？

- 当使用P个处理单元时能否获得大于P的加速比？
  - Cache 的影响
  - 工作集的影响
- 两种情况下：
  - 对比不公平
  - 访存的影响



40

40

利用率, 冗余度和效率

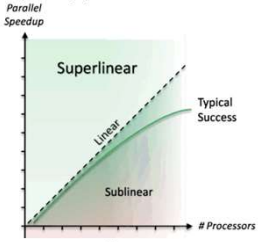
- 常用的指标
  - 假设所有P个处理器都满负荷参与并行计算
- 利用率：有多少处理能力被使用
  - U = (并行操作的数量) / (处理器 x 时间)
- 冗余度：并行处理时做了多少额外的工作
  - R = (并行操作的数量) / (用最佳的单处理器算法执行的操作数)
  - R总是≥1
- 效率：用了多少资源占能够获得多少资源的比例
  - E = (使用1个处理器花费的时间) / (处理器 x 使用P个处理器花费的时间)
  - E = U/R

41

41

真实的加速比

- 为什么真实的加速比是这样的？



$$\tau_p = \alpha \cdot \frac{\tau_1}{p} + (1-\alpha) \cdot \tau_1$$

单处理器程序中可以并行化的部分　　不可以并行化的部分

42

42

Amdahl定律

$$\text{加速比}_{P\text{个处理器}} = \frac{\tau_1}{\tau_p} = \frac{1}{\frac{\alpha}{p} + (1-\alpha)}$$

$$\text{加速比}_{P\to\infty} = \frac{1}{1-\alpha}$$

并行加速比的瓶颈

Amdahl定律的内涵：
1、当α<1时，增加越来越多的处理器，得到的收益(加速比)越来越少；
2、收益(加速比)不大，除非α≈1

Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," AFIPS 1967.

43

43

Amdahl定律的启示

- Amdahl定律的另一种表示
  - f: 程序可并行化的比例
  - N: 处理器数量

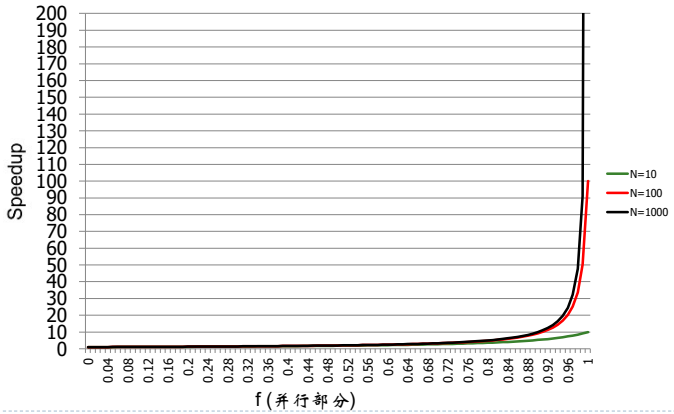
$$\text{加速比} = \frac{1}{1-f+\frac{f}{N}}$$

  - Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," AFIPS 1967.
- 最大化加速比受限于串行部分：串行瓶颈
- 并行部分通常也不是完美的并行
  - 同步开销 (比如，更新共享的数据)
  - 负载不均衡开销 (并行化不完美)
  - 资源共享开销 (N个处理器之间的竞争)

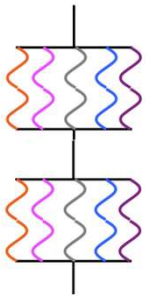
44

44

串行瓶颈



为什么串行是瓶颈?

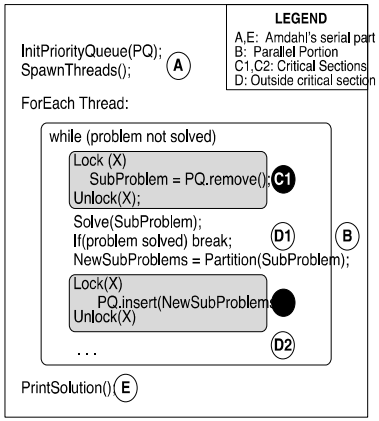


- 并行的机器有串行瓶颈
- 主要原因:有不能并行化的数据操作(比如,不能并行化的循环)

```
for ( i = 0 ; i < N; i++)
    A[i] = (A[i] + A[i-1]) / 2
```

- 数据准备是单线程的，而任务生成是并行的(通常任务本身又是串行的)

串行瓶颈的例子



并行部分的瓶颈

- **同步**:对共享数据的操作不能并行
  - 锁,同步互斥,栅障同步
  - 通信:Task任务之间可能需要互相的数据
  - 竞争共享数据时会造成线程串行
- **负载不均衡**:并行的任务可能有不同的长度
  - 由于并行化不理想或者微体系结构的影响
  - 在并行部分降低加速比
- **资源竞争**:并行任务会共享硬件资源,互相延迟
  - 为所有资源设计冗余(比如内存)成本太高
  - 每个任务单独运行时并没有额外的延迟产生

并行编程的困难

- 如果存在天然的并行性就不太难
  - “高度并行”的应用
  - 多媒体，物理模拟，图形图像处理
  - 大型 web 服务器，数据库？
- 困难在于
  - 让并行程序正确运行
  - 存在瓶颈时优化性能
- 并行计算机体系结构主要是关于
  - 如何设计计算机器以克服串行和并行瓶颈，获得高性能和高效率
  - 使程序员更容易开发正确并且高性能的并行程序

READING LIST

- 7 topics; 17 papers

ZY2457A03	范家诚
ZY2457A12	赛佳霖
ZY2457A14	盛一搏
ZY2457A16	张晓舟

- Topic 1
  ======
  Scalable Cache Coherence　　第9周（4月27日下午）

1a) Review: Chapter on Cache Coherence, "Parallel Computer Architecture A Hardware/Software Approach" by Culler and Singh, published by Morgan Kaufmann, 1997.
1b) D. Lenoski at al. "The Directory-Based Cache Coherence Protocol for the DASH Multiprocessor“ International Symposium on Computer Architecture 1990.
1c) A. Gupta et al.
"Reducing Memory and Traffic Requirements for Scalable Directory-Based Cache Coherence Schemes", International Conference on Parallel Processing 1990.
1d) J. Torrellas, M. Lam & J. Hennessy, "False Sharing and Spatial Locality in Multiprocessor Caches“, Transactions on Computers, June 1994.

READING LIST（2）

ZY2457420	王鲁新
ZY2457422	李泽昊
ZY2457424	苗雨阳
ZY2457509	姬夏迎

Topic 2
=====
Memory Consistency Models　　第9周（4月27日下午）

2a) K. Gharachorloo et al. "Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors“, International Symposium on Computer Architecture 1990.
2b) K. Gharachorloo et al. "Two Techniques to Enhance the performance of Memory Consistency Models“, International Conference on Parallel Processing 1991.
2c) S. Adve and K. Gharachorloo. "Shared Memory Consistency Models: A Tutorial“, DEC WRL Research Report 95/7, 1995

READING LIST（3）

ZY2457318	咸永飞
ZY2457328	于浩喆
ZY2457329	张超骏
ZY2457415	刘鑫

Topic 3
=====
Prefetching　　第12周（5月18日下午）

3a) T. Mowry et al. "Design and Evaluation of a Compiler Algorithm for Prefetching“, Architectural Support for Programming Languages and Operating Systems, 1992.
3b) Y. Solihin et al. "Using a User-Level Memory Thread for Correlation Prefetching“, International Symposium on Computer Architecture 2002.

READING LIST（4）

ZY2457308	王康宁
ZY2457311	王振伟
ZY2457312	王志飞
ZY2457316	文砚雷

Topic 4
=====
Synchronization　　第12周（5月18日下午）

4a) J. Goodman et al. "Efficient Synchronization Primitives for Large Scale Cache-Coherent Multiprocessors", Architectural Support for Programming Languages and Operating Systems, 1989.
4b) J. Mellor-Crummey and M. Scott. "Algorithms for Scalable Synchronization on Shared-Memory Multiprocessors", ACM Transactions on Computer Systems, 1991

53

53

READING LIST（5）

ZY2457213	李家兵
ZY2457216	李尚原
ZY2457229	邱文凯
ZY2457233	任星舟

Topic 5
=====
Multithreading　　第13周（5月25日下午）

5a) D. Tullsen et al. "Simultaneous multithreading: Maximizing On-Chip Parallelism ", International Symposium on Computer Architecture 1995.
5b) D. Tullsen et al. "Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor", International Symposium on Computer Architecture 1996.

54

54

READING LIST（6）

ZY2457118	程骁
ZY2457124	段泽邦
ZY2457129	郭燕
ZY2457206	沈紫静

Topic 6
=====
Multiple Processors on a Chip　　第13周（5月25日下午）

6a) K. Olukotun et al. "The Case for a Single-Chip Multiprocessor", Architectural Support for Programming Languages and Operating Systems, 1996
6b) G. Sohi et al. "Multiscalar Processors", International Symposium on Computer Architecture 1995.
6c) V. Krishnan and J. Torrellas. "A Chip Multiprocessor Architecture with Speculative Multithreading", IEEE Transactions on Computers 1999

55

55

READING LIST（7）

ZF2406102	盖一鸣
ZF2406108	罗辉
ZF2406114	张峻玮
ZY2457103	段欣然
ZY2457104	姜明月

Topic 7
=====
Speculative Parallelization and Execution　　第13周（5月25日下午）

7a) J. Steffan et al. "A Scalable Approach to Thread-Level Speculation" International Symposium on Computer Architecture 2000.
7b) J. Martinez et al. "Speculative Synchronization: Applying Thread-Level Speculation to Explicitly Parallel Applications", Architectural Support for Programming Languages and Operating Systems, 2002.

56

56

谢谢！

57

57