内容

- 1. 引言
  - 1.1 并行计算的发展历史与现状
  - 1.2 并行计算平台
  - 1.3 并行计算的性能度量
- 2. 并行程序设计的基本性质
- 3. 并行程序实现环境

1.1 并行计算的发展历史与现状

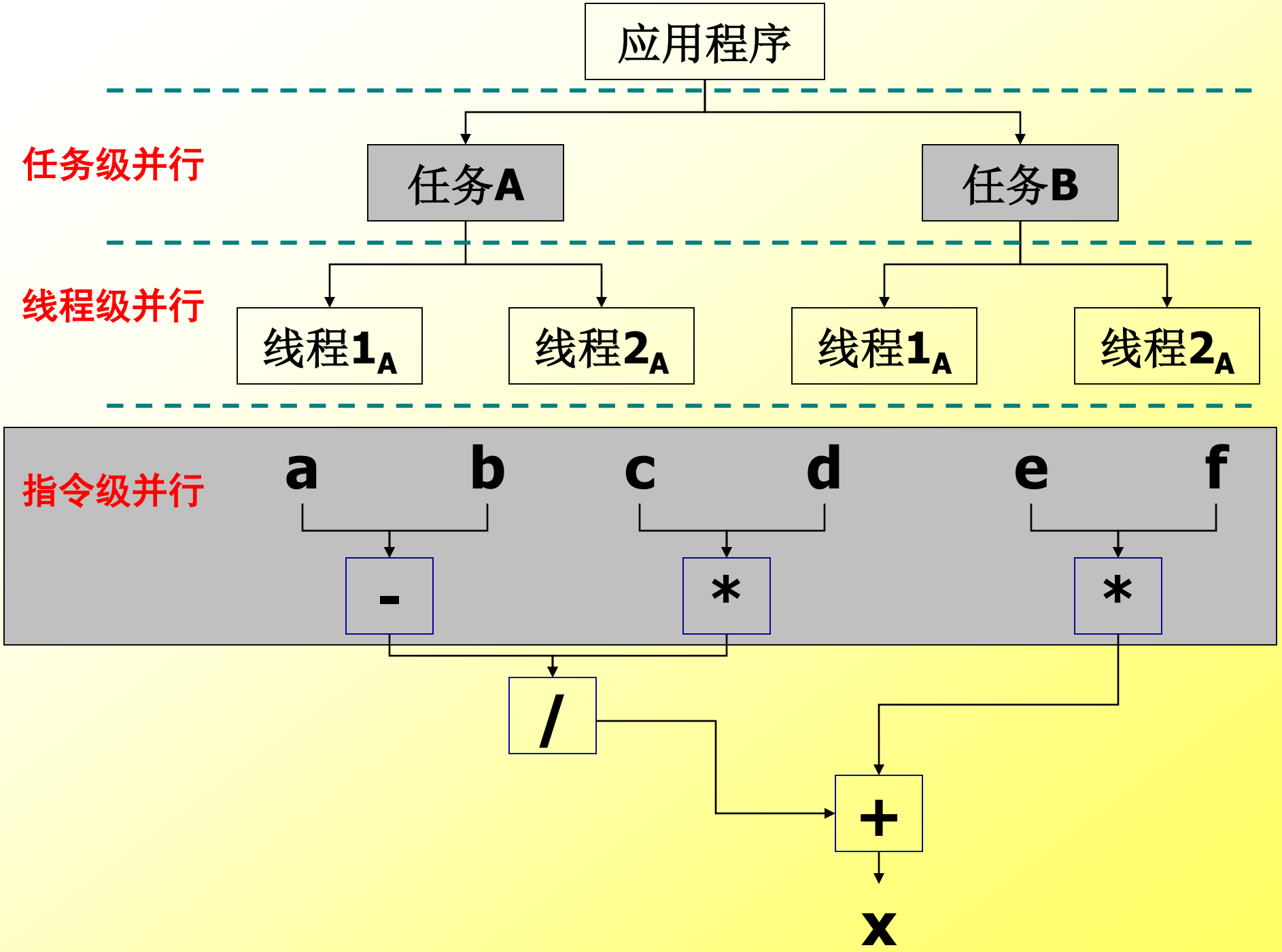
- 1945年，数学家约翰.冯.诺依曼
  - 计算模型：“存储程序”（冯.诺依曼体系结构）
    - 程序中的指令**以线性和单线程的模式串行执行**。
- 20世纪60年代
  - 分时共享操作系统，首次引入了程序**并发执行**的概念
- 1976年至今（高性能计算机）
  - 科学研究手段：理论+物理实验+计算
    - http://www.top500.org/ （世界上最快的计算机，每年发布两次）---**“天河一号”，2010年后，曾获第1位！**
    - 我国是继美国、日本之后第三个具有研制高性能计算机能力的国家。
    - 向量并行→多处理机并行→大规模并行→集群技术→定制技术

1.1 并行计算的发展历史与现状

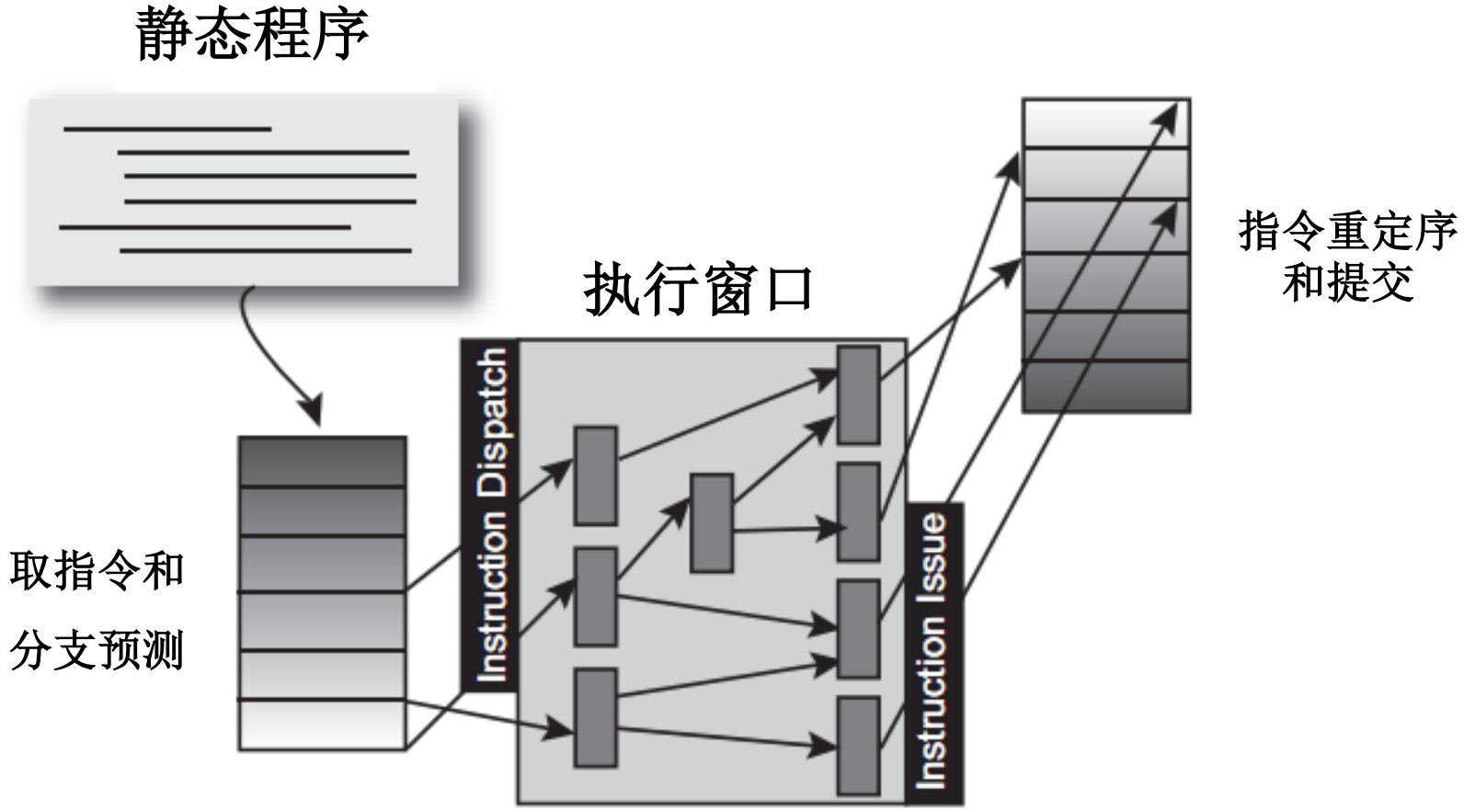
- 微处理器中的并行计算（三个里程碑）
  - (1)1993年Intel Pentium**超标量处理器**
    - 能够在单个时钟周期内执行多条指令的处理器。
      - 采用**指令级并行技术**（Instruction-Level Parallelism）

指令级并行技术

即动态执行或乱序（out of order）执行技术，使得CPU能够以更加优化的方式对指令进行重新定序），从而达到消除流水线停顿，进而保持处理器的执行单元尽量处于工作状态。


应用程序
任务级并行
任务A
任务B
线程级并行
线程1A
线程2A
线程1A
线程2A
指令级并行
a
b
c
d
e
f
-
*
*
/
+
x

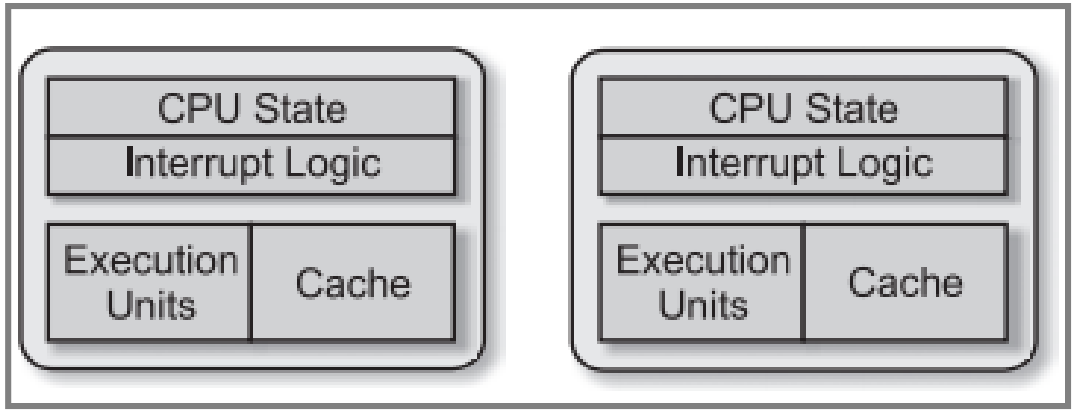
1.1 并行计算的发展历史与现状



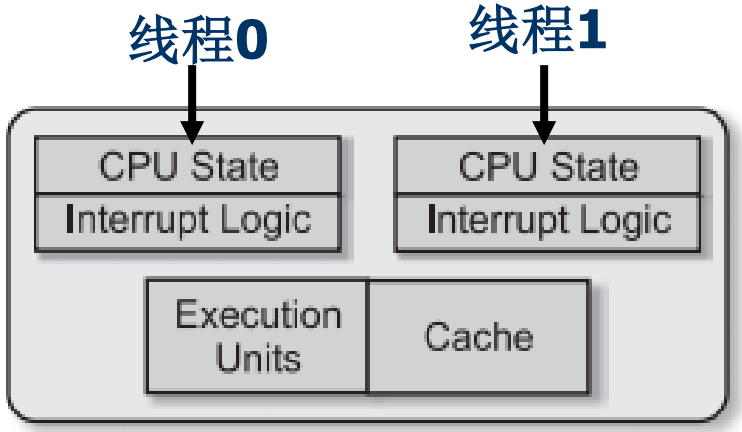
超标量处理器内部的基本执行流程

1.1 并行计算的发展历史与现状

- 微处理器中的并行计算（三个里程碑）
  - (2)2000年 超线程技术(Hyper-Threading Technology)
    - 支持超线程技术的处理器首次采用同时多线程技术
      - 即：SMT技术（Simultaneous Multi-Threading）
    - Intel实现的SMT技术就是超线程技术。



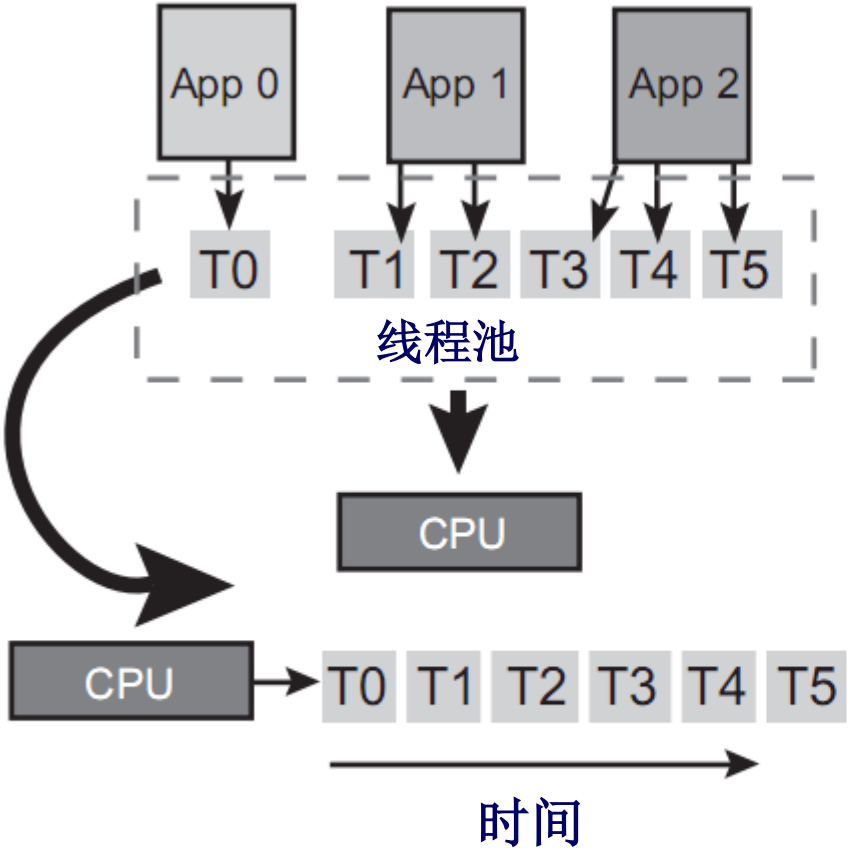
（a）多处理器结构



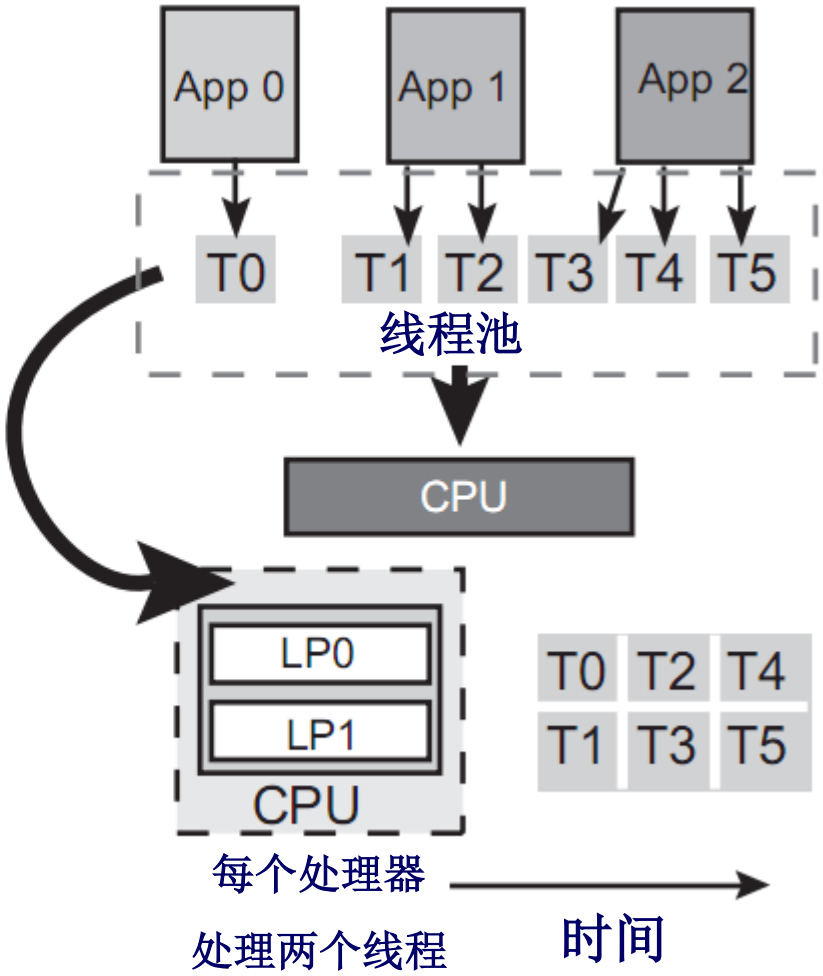
（b）超线程技术

1.1 并行计算的发展历史与现状

多线程



超线程技术



1.1 并行计算的发展历史与现状

- 微处理器中的并行计算（三个里程碑）
  - （3）2005年 **多核处理器**（双核处理器）
    - 采用单芯片多处理器（CMP，Chip Multiprocessor）设计
    - 在单个处理器芯片内实现两个或更多的“**执行核**”。这些核都是相互独立的处理器，只是位于同一块芯片上而已。
    - 执行核有自己的体系结构资源，并且可以与SMT技术结合。

Sun的Niagara系列

IBM的Power系列

AMD的Opteron（皓龙系列）

Intel的Xeon（至强系列）

Tilera的TILE-Gx100

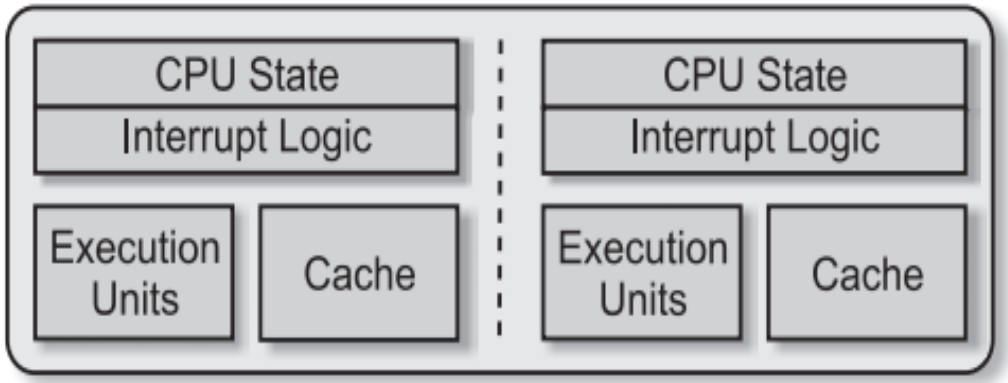
GPGPU

- **Nvidia的Tesla**
- **AMD的Firestream**
- **Intel的Larabee**

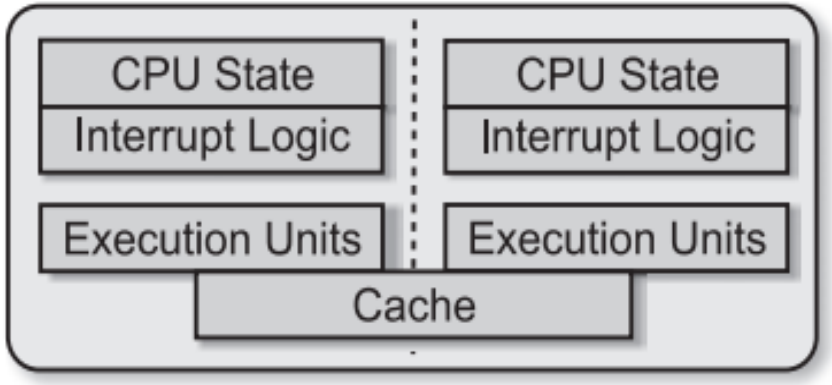
Cell处理器

FPGA

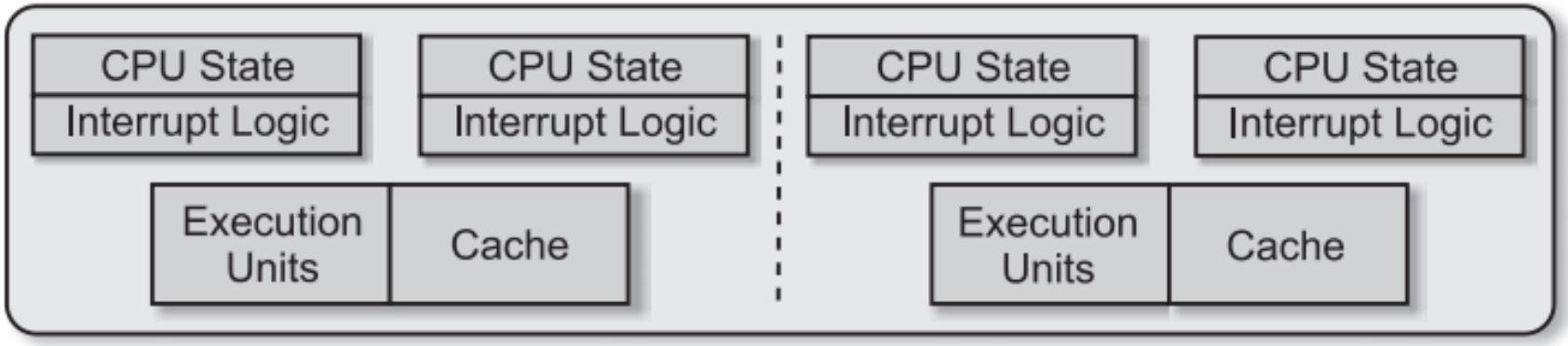
1.1 并行计算的发展历史与现状



（a）多核体系结构



（b）共享cache的多核体系结构

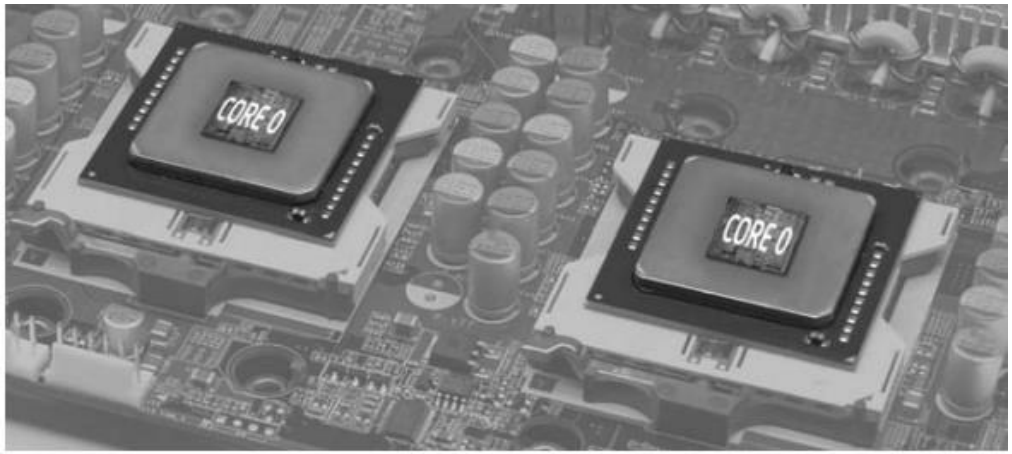


（c）采用超线程技术的多核体系结构

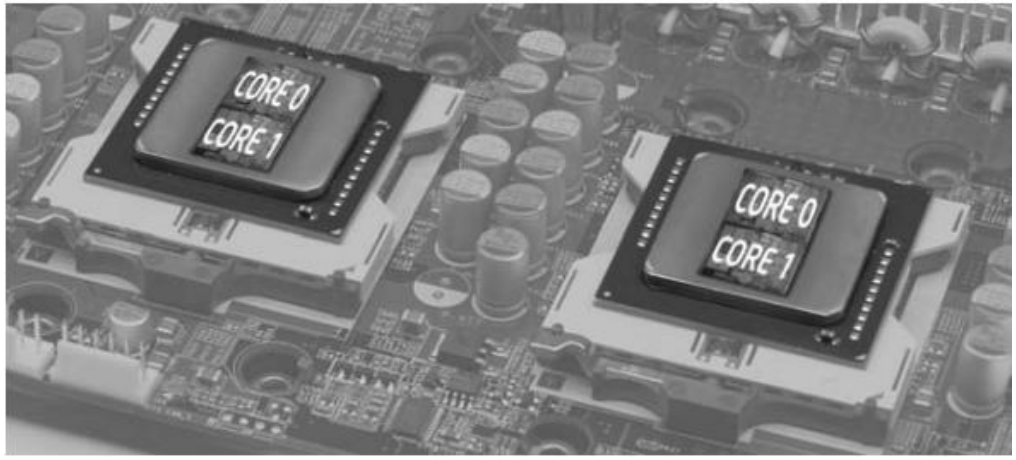
1.1 并行计算的发展历史与现状

- 基本概念

多处理器与多核处理器



（a）由单核处理器构成的多处理器系统



（b）由多核处理器构成的多处理器系统

1.1 并行计算的发展历史与现状

- 背景知识
  - 1965年（摩尔定律）
    - 半导体厂商能够集成在芯片中的晶体管数量大约每18个月翻一番。
  - 硬件的发展却逐渐面临极限：
    - 单位面积下可以容纳的晶体管数量趋于极限，直接影响到处理器的主频。
    - 功耗问题

摩尔定律指出的发展趋势已经变缓

处理器主频正在和摩尔定律分道扬镳…

1.1 并行计算的发展历史与现状

- 推荐阅读文献
  - http://www.gotw.ca/publications/concurrency-ddj.htm（译文发表于《程序员》2006.11 ）
    - The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software（*By Herb Sutter 2005年*）

免费大餐不久就将结束。对此，你有何打算，做好下一步准备了么？

对主要的处理器厂商以及架构，包括**Intel**、**AMD**和**Sparc**、**PowerPC**来说，改善**CPU**性能的传统方法，如提升时钟速度和指令吞吐量，基本已走到尽头，现在开始向超线程和多核架构靠拢。而且这两个特性（特别是多核）已经在部分芯片实现，如**PowerPC**和**Sparc IV**；**Intel**和**AMD**也将在**2005**年内赶上。**2004**年**In-Stat/MDR**秋季处理器论坛的主题就是多核设备，很多公司都展示了改进和新研发的多核处理器。不过，要将**2004**年称为多核年，显然还不够理直气壮。

多核将引领软件研发发生基础性变化，特别对接下来几年里那些面向一般应用、运行在**PC**和低端服务器上的应用软件（在今天已经销售出去的软件里占有很大比例）而言。在这篇文章里，我想就多核为何突然对软件产生重要影响，以及并发巨变如何影响我们和我们未来编写软件方式的问题展开讨论。

我可以这么说：免费大餐已经结束一两年了，但我们现在才开始意识到这个问题。

内容

- 1. 引言
  - 1.1 并行计算的发展历史与现状
  - 1.2 并行计算平台
  - 1.3 并行计算的性能度量
- 2. 并行程序设计的基本性质
- 3. 并行程序实现环境

1.2 并行计算平台

- 计算机体系结构的分类（Flynn分类法-1972年）
  - 分类依据一：
    - 计算机在单个时间点能够处理的指令流（instruction streams）的数量。
  - 分类依据二：
    - 计算机在单个时间点能够处理的数据流（data streams）的数量。

1.2 并行计算平台

- （1）单指令单数据流机器（SISD）
  - 传统的串行计算机，其硬件不支持任何形式的并行。
  - 标量计算机（scalar computer）
  - 每个时钟周期只处理一条指令，一个数据流。

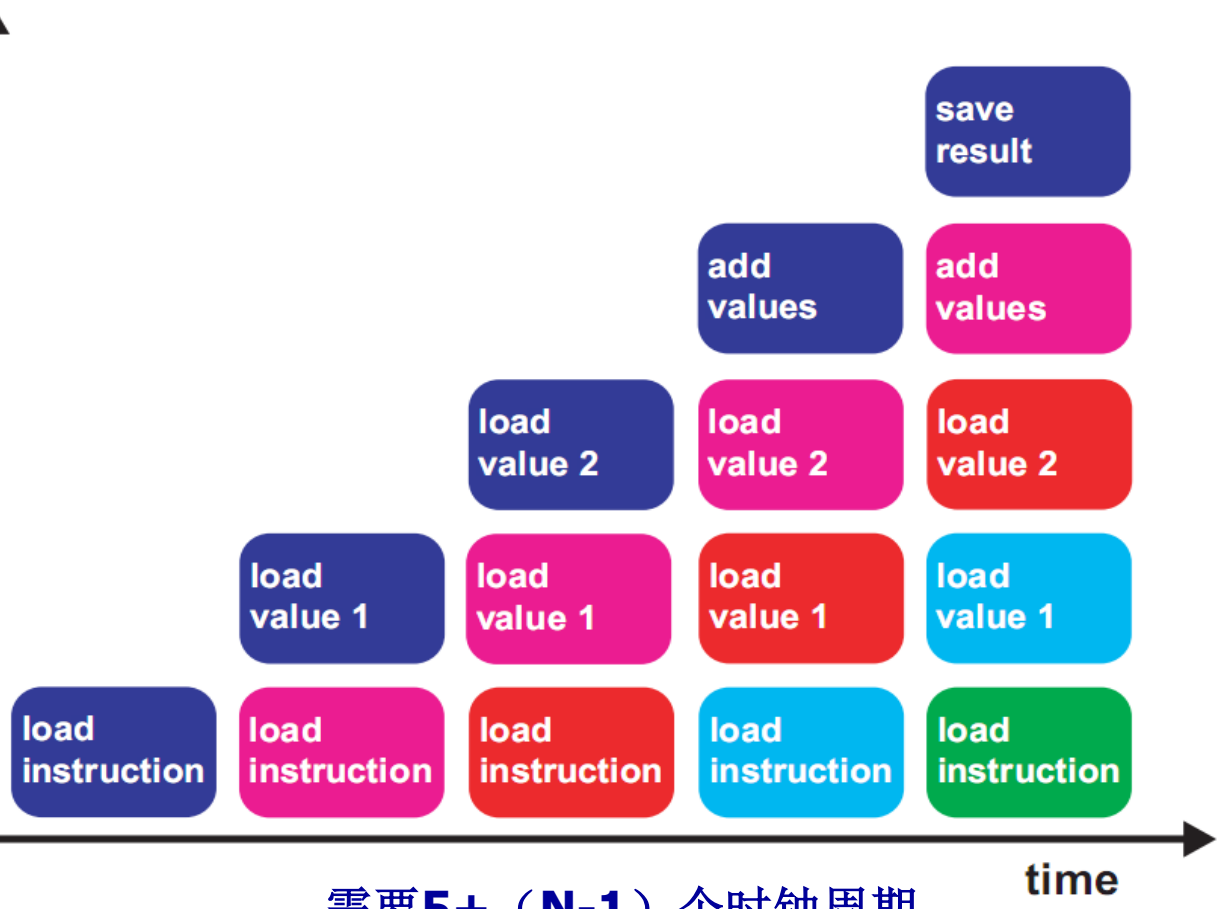
两个数值相加

需要5个时钟周期

1.2 并行计算平台

- SISD的缺点：处理器资源利用效率低下
  - 解决方案：流水线处理（pipelining）

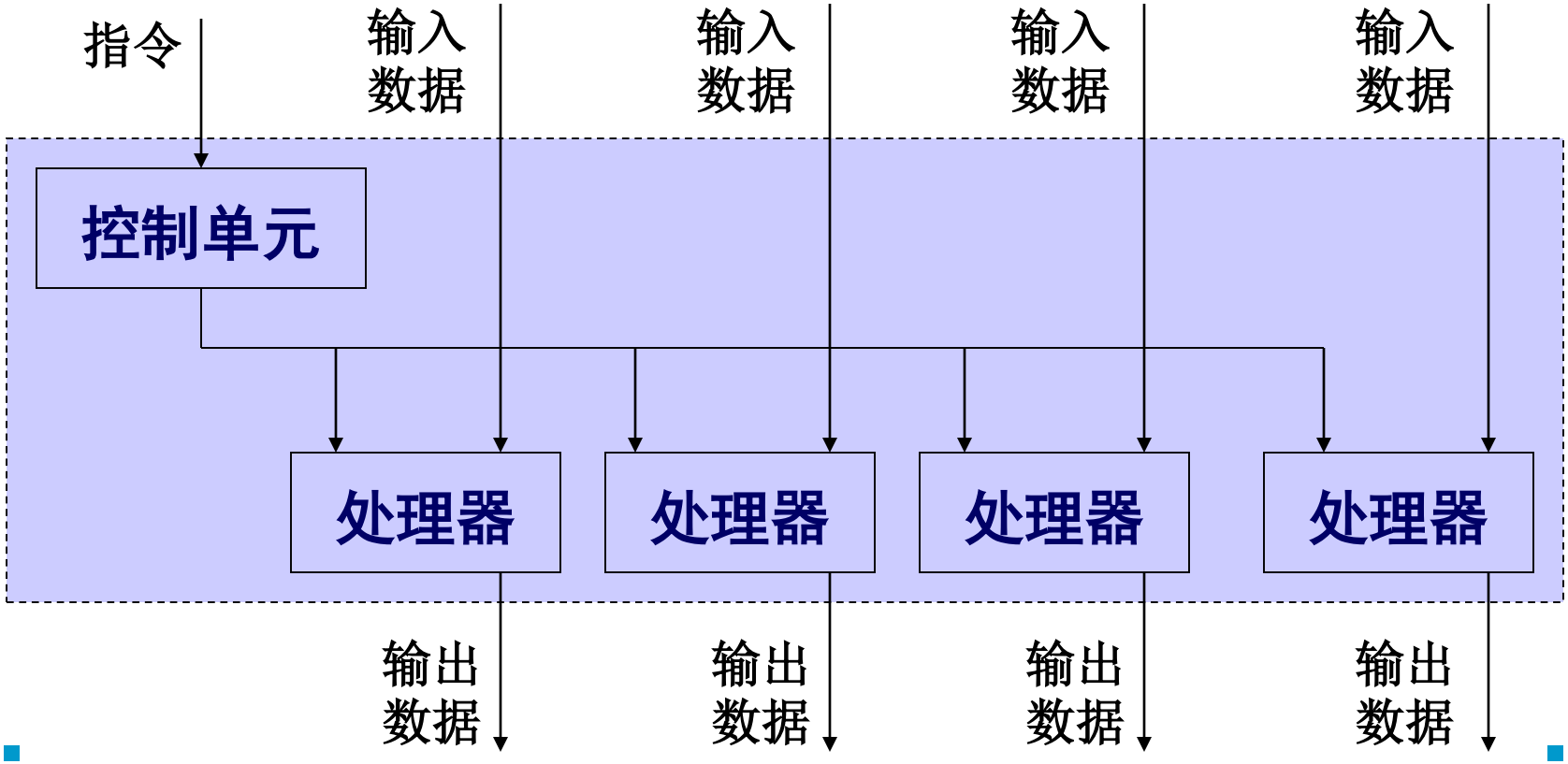
五对数值相加



需要5+（N-1）个时钟周期

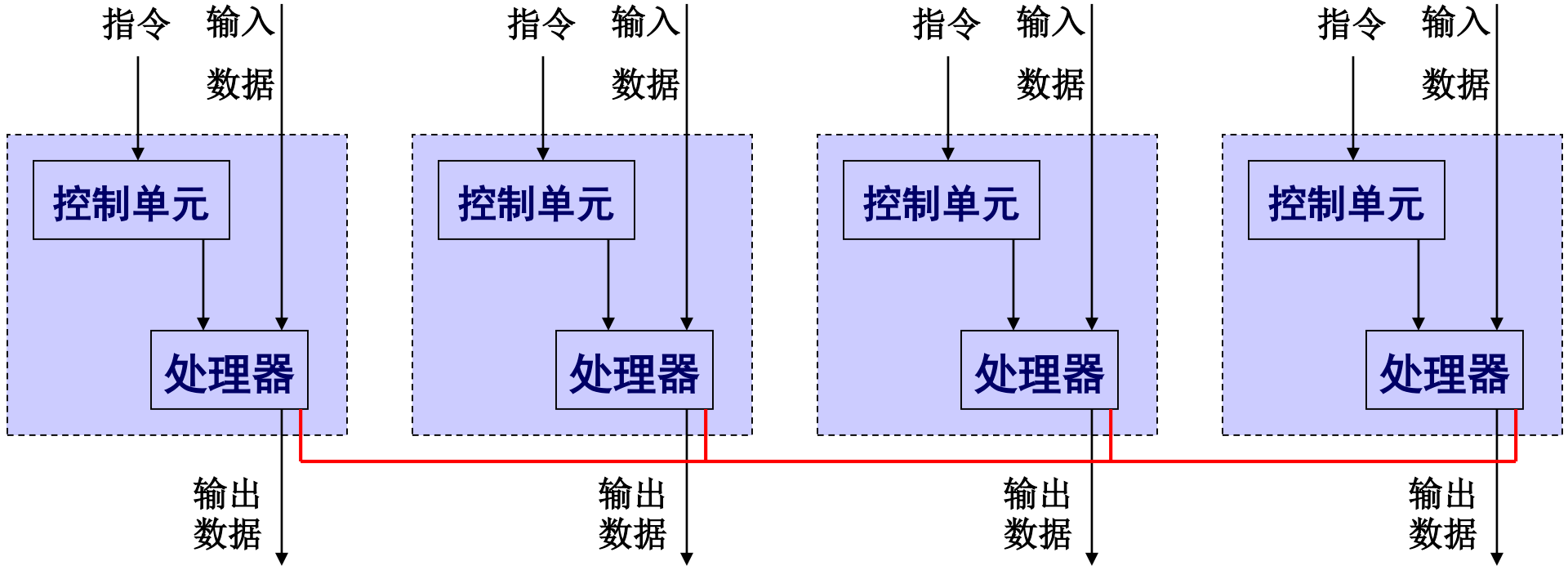
1.2 并行计算平台

- （2）单指令多数据流机器（SIMD）
  - 一个指令流被并发地广播给多个处理器，每一个具有它自己的数据流。



1.2 并行计算平台

- （3）多指令多数据流机器（MIMD）
  - 能够同时执行多个指令流，这些指令流分别对不同的数据流进行操作。
  - 目前最流行的并行计算平台（多核计算平台是MIMD）

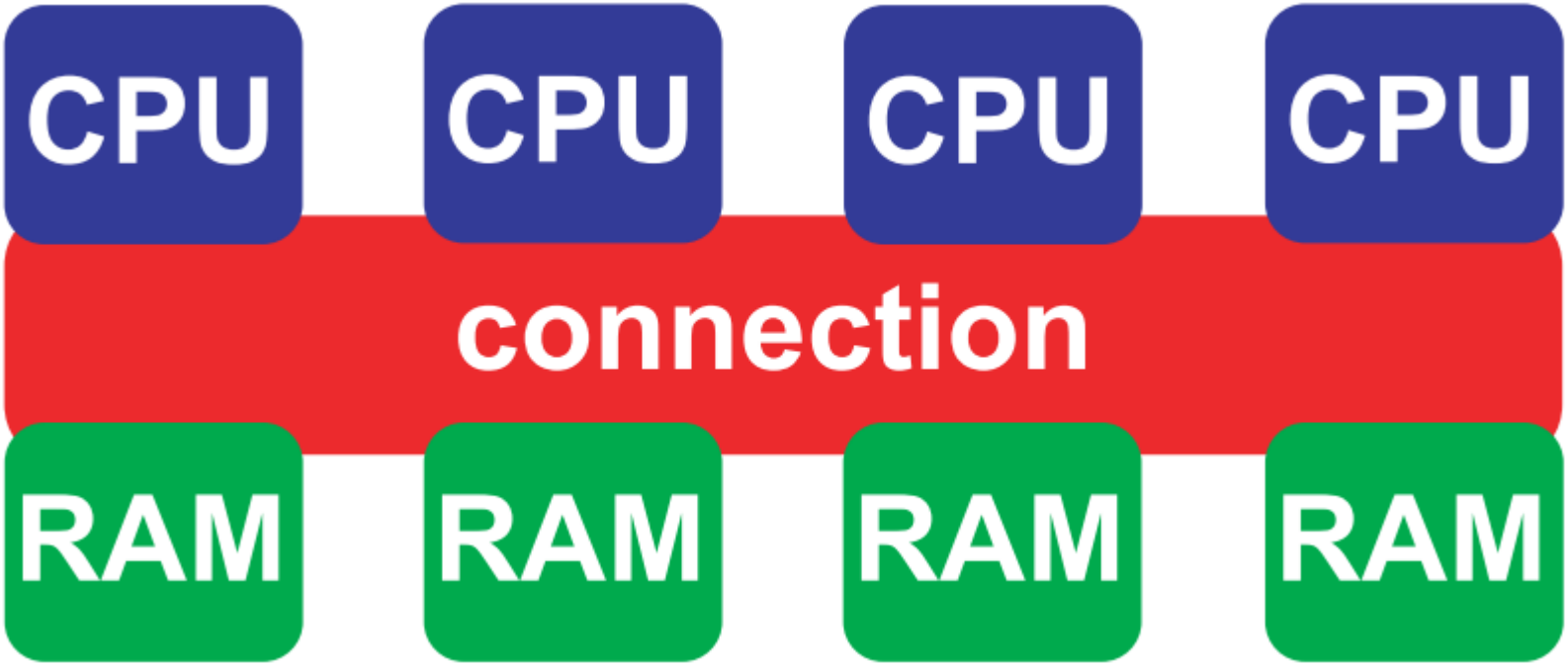


1.2 并行计算平台

- MIMD的进一步分类，分类依据是：
  - **存储器的组织方式**
    - 共享存储器
      - 对称多处理器（**SMP**）
      - 非均匀存储器访问（**NUMA**）
    - 分布式存储器
      - 大规模并行处理机（**MPP**）
      - 集群（**Cluster**）
      - 网格（**Grid**）
    - 混合系统

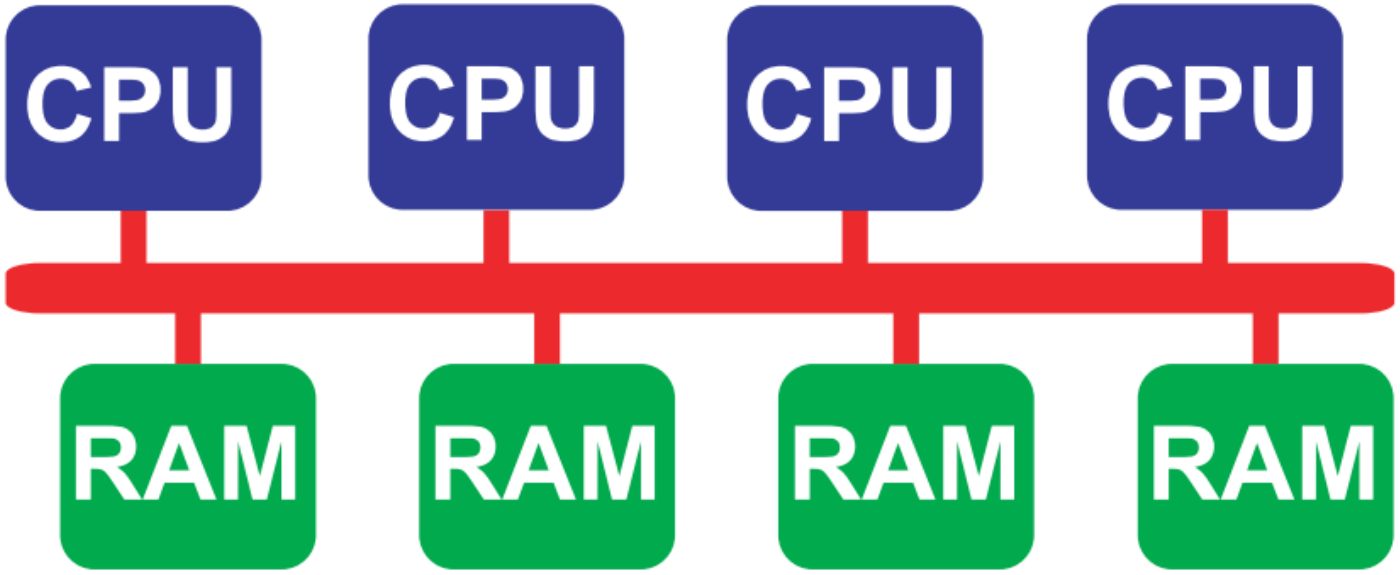
1.2 并行计算平台

- （1）共享存储器（SM，Shared Memory）
  - SM-MIMD（MIMD systems with shared memory）
    - 在一个共享存储器系统中，所有的进程共享一个地址空间，并通过读写共享变量来相互通信。



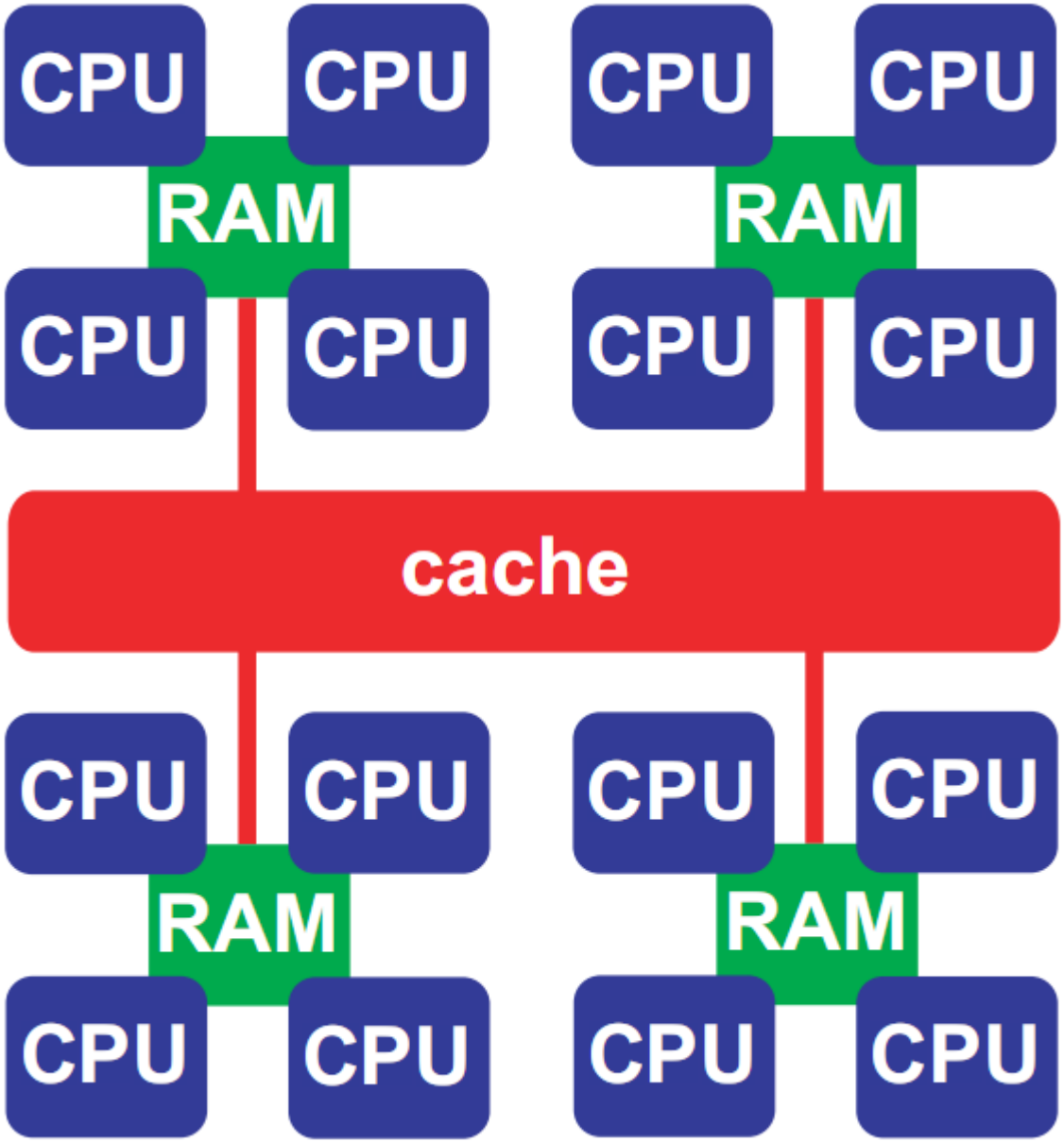
1.2 并行计算平台

- （1）共享存储器（Shared Memory）
  - （a）对称多处理器（SMP）
    - 所有的处理器通过连接共享通用存储器，并以相同速度访问所有存储器单元。
    - 性能限制：处理器/存储器带宽（共享总线带宽）



1.2 并行计算平台

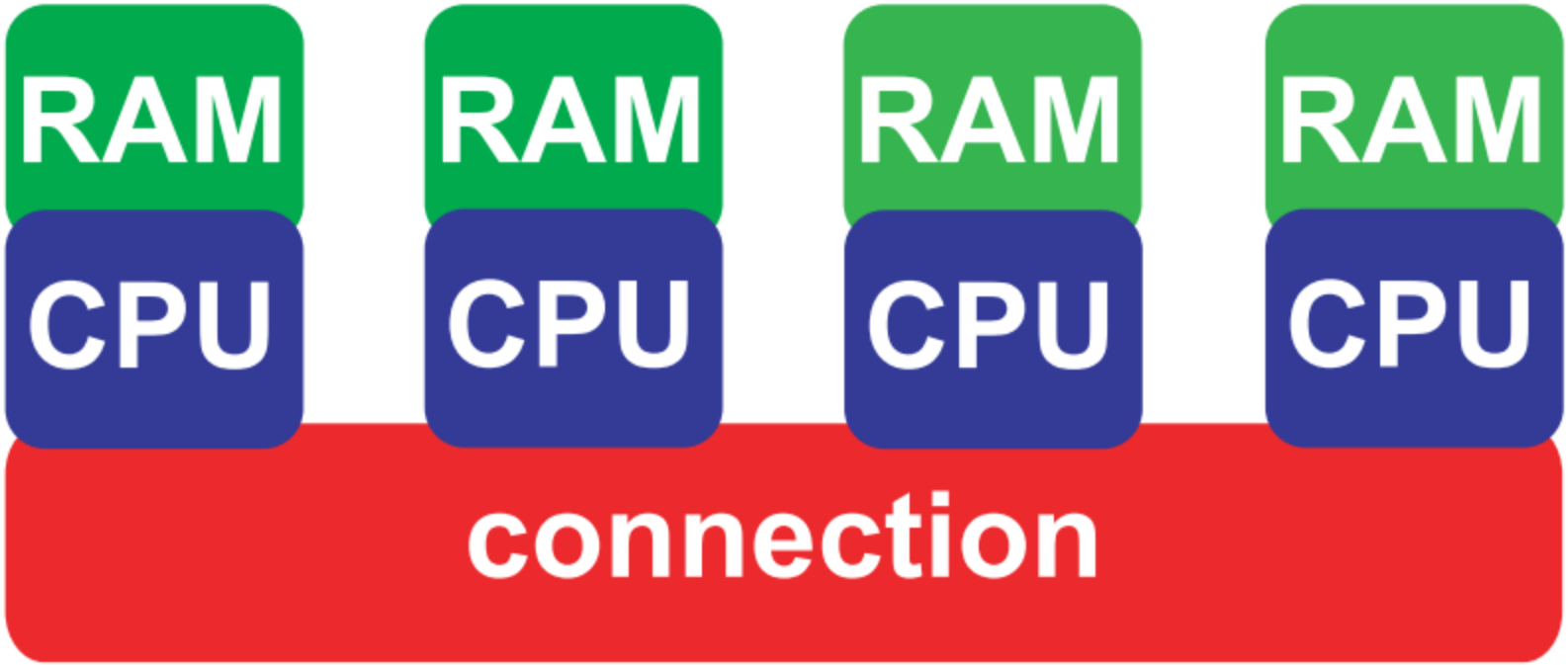
- （1）共享存储器（Shared Memory）
  - （b）非均匀存储器访问（NUMA）
    - 储存器是共享的（即所有的处理器都可以访问它），但是某些存储器对于某些处理器来说，可能物理位置更近一些，这降低了存储器的带宽瓶颈，使得系统能够具有更多的处理器。
  - 这类体系结构的另一个名字：
    - 缓存一致性非均匀存储器访问系统（ccNUMA）
    - cache coherent Non-Uniform Memory Access


CPU
CPU
CPU
CPU
RAM
RAM
CPU
CPU
CPU
CPU
cache
CPU
CPU
CPU
CPU
RAM
RAM
CPU
CPU
CPU
CPU

ccNUMA 系统结构

1.2 并行计算平台

- （2）分布式存储器（Distributed Memory）
  - DM-MIMD（MIMD systems with Distributed Memory）
    - 在一个分布式存储器系统中，每个进程具有它自己的地址空间，进程间通过消息传递（发送和接收消息）进行通讯。

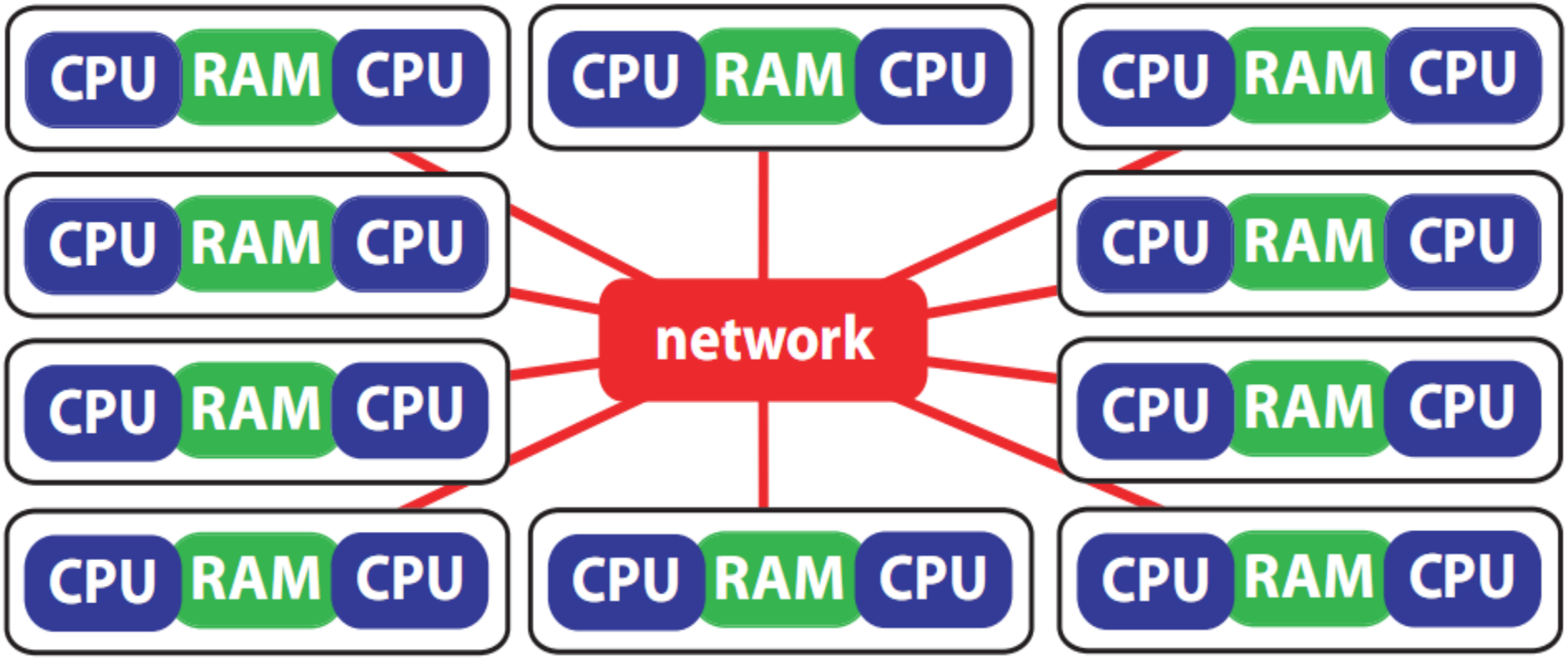


1.2 并行计算平台

- （2）分布式存储器（Distributed Memory）
  - （a）大规模并行处理机
    - Massively Parallel Processors，简称MPP
    - 处理器和网络结构紧耦合，并专门用于并行计算机。
    - 这些系统具有非凡的可伸缩性，在某些示例中，一个系统具有数千个处理器。
  - （b）集群（Cluster）
    - 由商业计算机通过商业网络连接而成的分布式存储器系统
    - 集群为一个组织获得并行计算能力提供了一种廉价的方式。现在可以从很多供货商处获得预配置的集群。一个节俭的团体甚至可以通过使用集群组合过时的PC的能力来构建一个有用的并行系统。
  - （c）网格（Grid）

1.2 并行计算平台

- （3）混合系统（Hybrid System）



SMP节点组成的集群

1.2 并行计算平台

- 小结
  - 根据硬件的结构特征对并行计算平台进行了分类。
  - 并行计算平台会影响用于表示系统并发性的自然编程模型。
  - 注意：
    - 共享存储器机器的编程环境可能为程序员提供分布式存储器和消息传递的抽象---例如：Erlang语言的并发机制。
    - 而分布式存储器系统也可提供分布式存储器机器的共享存储器抽象。
  - 建议阅读资料
    - Overview of recent supercomputers
    - http://www.phys.uu.nl/~euroben/ （每年更新）

内容

- 1. 引言
  - 1.1 并行计算的发展历史与现状
  - 1.2 并行计算平台
  - 1.3 并行计算的性能度量
- 2. 并行程序设计的基本性质
- 3. 并行程序实现环境

1.4 并行计算的性能度量

- 分析：
  - 假定一个计算由3部分组成：
    - 准备（setup）
    - 计算（computation）
    - 结束（finalization）
  - 则：

    在一个处理器（核）上运行的时间为：

① $$T_{total}(1) = T_{setup} + T_{compute}(1) + T_{finalization}$$

1.4 并行计算的性能度量

- 分析：
  - 在一台具有n个处理器（核）的计算机上并行计算
    - 假设准备部分和结束部分无法与其他部分一起并发执行
    - 假定计算部分能够被分解为多个任务，这些任务将独立运行在多个处理器（核）上，而且计算步骤的数目与初始计算中的步骤相同。

② $$T_{total}(n) = T_{setup} + \frac{T_{compute}(1)}{n} + T_{finalization}$$

非常理想化的情形，为什么？

1.4 并行计算的性能度量

- 分析：
  - 加速比（一种更直观的度量方式）
    - 描述了一个问题在并行情况下运行比普通情况下运行要快多少。

③ $$S(n)=\frac{T_{total}(1)}{T_{total}(n)}$$

1.4 并行计算的性能度量

- 分析：
  - 串行比例 γ
    - 执行程序中串行部分的比例。

④ $$\gamma = \frac{T_{setup} + T_{finalization}}{T_{total}(1)}$$

  - 花费在程序的可并行部分的时间比例为1− γ ，于是：

⑤ $$T_{total}(n) = \gamma T_{total}(1) + \frac{(1-\gamma)T_{total}(1)}{n}$$

1.4 并行计算的性能度量

- 分析：
  - 利用公式⑤重写S（n），即Amdahl法则（1967年）：

$$S(n)=\frac{T_{total}(1)}{T_{total}(n)}=\frac{T_{total}(1)}{\gamma T_{total}(1)+\frac{(1-\gamma)T_{total}(1)}{n}}$$

$$S(n)=\frac{1}{\gamma+\frac{1-\gamma}{n}}$$ (Amdahl法则)

1.4 并行计算的性能度量

- 分析：
  - 当n趋于无穷大时，则串行部分执行时间为 γ 的应用程序并行化之后能够达到的加速比上限为：

$$S(n)=\frac{1}{\gamma}$$

1.4 并行计算的性能度量

结论：

1. 无限的处理器核并不能带来性能上的无限增长，即应用程序从可并行部分所获得的性能提升最大值受限于串行部分所占的比例。

2. 对于加速程序性能而言，减少程序中串行部分所占的比例，增加并行部分比例的方法将比增加处理器核的数量的方法更有实际意义。

3. 只有当程序的大部分都是可并行代码的时候，增加处理器核的数量才会比增加并行代码的比例更加有效。

内容

- 1. 引言
  - 1.1 并行计算的发展历史与现状
  - 1.2 并行计算平台
  - 1.3 并行计算的性能度量
- 2. 并行程序设计的基本性质
- 3. 并行程序实现环境

2.1 并行程序设计的基本性质

- 程序设计人员的“思维”转变：

线性程序设计模型

↓

Thinking Parallel！ →

↓

并行程序设计模型

分解（Decomposition）
负载平衡（Load Balancing）
正确性（Correctness）
性能（Performance）
可扩展性（Scalability）
可移植性（Portability）
同步（Synchronization）
通信（Communication）
…
…

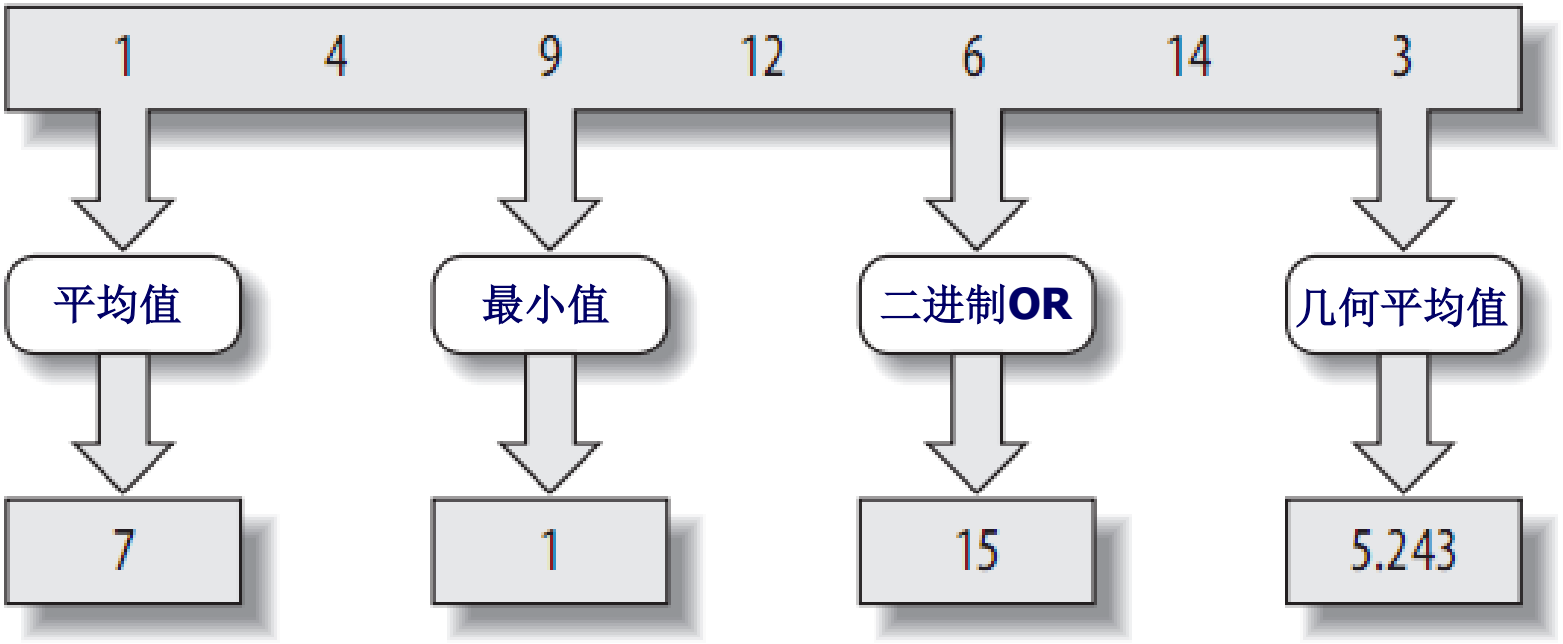
2.1.1 问题分解

- 问题分解（Decomposition）
  - 将应用程序划分成多个独立的任务，并确定这些任务之间相互依赖关系的过程。

分解方式	设计	说明
任务分解	不同的程序行为采用不同的线程实现	常用于GUI应用程序
数据分解	多个线程对不同的数据块执行相同的操作	常用于音频、图像处理和科学计算应用程序
数据流分解	一个线程的输出作为另一个线程的输入	尤其应注意尽量消除启动和排空延迟

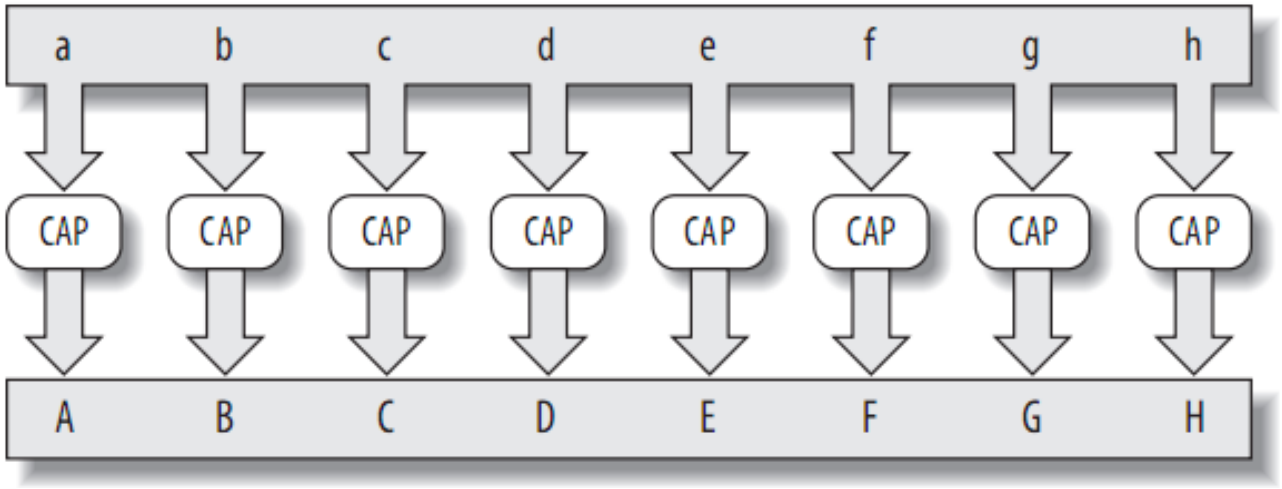
2.1.1 问题分解

- （1）任务分解（Task Decomposition）
  - 对应用程序根据其执行功能进行分解的过程。



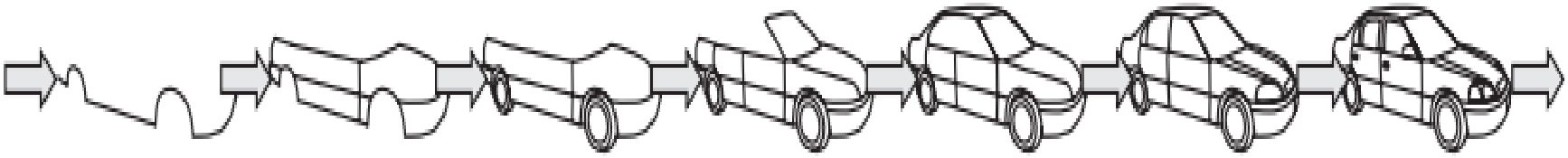
2.1.1 问题分解

- （2）数据分解（Data Decomposition）
  - 数据级并行（data-level parallelism）
  - 密集并行问题（Embarrassingly Parallel Problem）
    - 多个线程对大规模数据集中的不同的数据块执行相同的操作
  - 特点：
    - 数据分解方式所能处理的问题规模随着处理器核数量的增加而增长。



2.1.1 问题分解

- （3）数据流分解（Data Flow Decomposition）
  - Pipelining
  - 根据任务之间的数据流关系对问题进行分解。



  - 生产者/消费者（producer/consumer）问题
    - 两者的相关性带来的延迟
    - 可用线程之间所分配的工作量不平均（负载不平衡）

2.1 并行程序设计的基本性质

- 程序设计人员的“思维”转变：

线性程序设计模型

↓

Thinking Parallel！ →

↓

并行程序设计模型

分解（Decomposition）
负载平衡（Load Balancing）
正确性（Correctness）
性能（Performance）
可扩展性（Scalability）
可移植性（Portability）
同步（Synchronization）
通信（Communication）
…
…

2.1.2 负载平衡

- 负载平衡（Load Balancing）
  - 指的是多个线程之间工作量分布的情况。负载平衡能够使各线程的工作量平均分配。

【例】粉刷房间

假定粉刷5间房间，唯一的大房间是2个单位，其他的每个房间是1个单位。每个房间指定一个人（处理器）粉刷。

- 6个单位中的5个可以并行粉刷
- p=5/6，1-p=1/6
- 故加速比S=1/(1-p+p/n)=1/(1/6+1/6)=3

2.1 并行程序设计的基本性质

- 程序设计人员的“思维”转变：

线性程序设计模型

↓

Thinking Parallel！ →

↓

并行程序设计模型

分解（Decomposition）
负载平衡（Load Balancing）
正确性（Correctness）
性能（Performance）
可扩展性（Scalability）
可移植性（Portability）
同步（Synchronization）
通信（Communication）
…
…

2.1.3 可扩展性

- 可扩展性（Scaling）
  - 所设计的并行程序具有在任何数目的处理器系统上运行良好的能力，是衡量在性能更加强劲的系统上运行软件时能否有效利用更多线程的指标。
  - 任何长期使用的软件（成功的软件）都必须要能扩展以适应“多核Moore定律”性能曲线。

（1）创建可扩展并行计算所需的大多数技术与在多核芯片上生成高效求解的技术是相同的。

（2）目前的多核芯片具有适中数目的处理器，通常是2～8个，在未来几年内每个芯片上的核数肯定会急剧增加，这使得可扩展性并行概念与之直接相关。

2.1 并行程序设计的基本性质

- 程序设计人员的“思维”转变：

线性程序设计模型

↓

Thinking Parallel！ →

↓

并行程序设计模型

分解（Decomposition）
负载平衡（Load Balancing）
正确性（Correctness）
性能（Performance）
可扩展性（Scalability）
可移植性（Portability）
同步（Synchronization）
通信（Communication）
…
…

2.1.4 可移植性

- 性能可移植（Performance Portability）
  - 指的是一个并行程序在多种不同类型的计算机上，只需要通过一些适当的优化（tuning），就能够获得较好的能力。
- 方法：
  - 通过在一个逼真的抽象机模型上进行程序设计，程序员可以将他们的代码移植到不同体系结构的并行计算机上，并且可以期望程序的核心性质与新硬件是相兼容的。虽然调谐是必须的，但是算法就可以不必完全重新考虑。
  - 通过设计并行程序设计语言进行抽象，实现隐式的性能可移植，如ZPL高级并行数组语言。

内容

- 1. 引言
  - 1.1 并行计算的发展历史与现状
  - 1.2 并行计算平台
  - 1.3 并行计算的性能度量
- 2. 并行程序设计的基本性质
- 3. 并行程序实现环境

3. 并行程序实现环境

- (1) 扩展已有的串行编程语言
  - (a) 提供支持并发程序的关键字和语言
    - 高性能Fortran（HPF）语言
    - Current Pascal
    - Current C
    - 基于GPU的CUDA通用并行计算架构，扩展C语言
    - …
  - (b) 利用并行编译器把串行程序变成可执行的并行代码
    - OpenMP
      - 以串行语言为基础，通过使用嵌入的编译器命令和一些库函数来建立并行说明。
      - 然后通过特殊的编译器将源程序编译成可并发执行的代码。
      - 是一种标准，有多种实现，主要用于共享存储系统环境。

3. 并行程序实现环境

- (2) 扩展已有的串行编程语言
  - (c) 使用外部函数库实现并发程序
    - 消息传递库
      - PVM（Parallel Virtual Machine）
      - MPI（Message Passing Interface）
      - 主要用于分布式存储系统环境
    - 线程API
      - 微软Windows线程API（Win32/MFC线程API）
      - 微软.NET框架线程API（最近发布Parallel FX和MPI.NET）
      - POSIX线程（或称为Pthreads）
        - Pthreads 是Linux操作系统中多线程接口的标准
        - Pthread-win32（Windows版本）
    - Intel的Threading Building Blocks库
    - Java，在5.0版中引入了concurrency库