计算机体系结构

李龙斌 - 202432466@mail.sdu.edu.cn

1. 计算机概论与设计分析基础

1.1. 引言

1.2. 计算机的分类

1. 物联网/嵌入式计算机
2. 个人移动设备
3. 桌面计算机
4. 服务器
5. 分布式集群（计算机）

1.3. 计算机的组成

1.3.1. 硬件组成部分

1. 输入设备
2. 输出设备
3. 储存器（层次化存储）
4. 运算器
5. 控制器（二者合称处理器；指令级并行、数据级并行、线程级并行）

1.3.2. 软件组成部分

1. 应用软件
2. 系统软件

1.4. 计算机体系结构

计算机体系结构包含计算机的物理实现及其逻辑设计（硬件）、各个部件之间的互连（组成，也称为微体系结构）。

思想	概括
使用抽象简化设计	隐藏低层细节来简化高层设计，类似于函数
加速经常性事件	常见情形出现的更加频繁并且更易优化，效果更好
通过并行提高性能	指令级并行（流水线）、数据级并行（向量体系结构、GPU）、线程级并行（多处理器）
通过流水线提高性能	指令级并行
通过预测提高性能	分支下提前执行可撤回的指令
通过冗余提高可靠性	在发生故障时代替失效组件，维持系统运行

思想	概括
储存层次	缓存机制、虚拟储存
通过冗余提高可靠性	在发生故障时代替失效组件，维持系统运行

1.4.1. 七个伟大思想及量化原理

1.5. 计算机的性能

计算机处于不同场景下对于性能的关注点不一样，如个人用户更关注计算机的响应时间，而服务器更关心它的吞吐量与带宽。对于性能好坏的评价在不同场景下需要使用不同的标准。

1.5.1. 性能的定义与度量

时间是计算机性能的衡量标准

时间的定义	内容
响应时间	完成某项任务需要的总时间，包含一切开销时间
CPU执行时间	程序在CPU上花费的时间（又细分为用户CPU时间与系统CPU时间）
时钟周期数	度量计算机硬件完成基本功能的速度

1.5.2. CPU性能及其度量因素

$$程序的CPU执行时间 = 程序的CPU时钟数 \times 时钟周期时间$$

1.5.3. 指令性能

$$CPU时钟周期数 = 程序的指令数 \times 指令平均时钟周期数（CPI）$$

1.5.4. 经典CPU性能公式

$$CPU时间 = 指令数 \times 指令平均时钟周期数（CPI） \times 时钟周期时间$$

1.5.5. 性能的测量、报告和汇总

基准测试	桌面基准测试	处理器密集型测试、图形密集型测试
	服务器基准测试	事务处理基准测试
性能测试结果	$SPECRatio = \frac{基准计算机上的执行时间}{待评估计算机上的执行时间}$	

1.6. 计算机的发展方向

1. 技术上，由于摩尔定律的失效，缩小晶体管的尺寸与增加晶体管的数量越来越困难，需要从新材料新架构中寻求突破。另外，由于网络速度的加快，远程计算机、云服务将越来越实用与流行。
2. 能耗上，在移动设备与嵌入式设备（物联网）中，在低功耗的情况下实现高性能仍然是突破点
3. 芯片制造中，随着制程工艺的减少，芯片生产难度加大。改良制造工艺提高良率能有效降低成本

1.6.1. 技术趋势

1.6.2. 集成电路中的功耗和能耗趋势

1.6.3. 成本趋势

2. 指令系统

2.1. 汇编语言及其操作数

不同的高级程序语言经过汇编后得到相同的汇编语言。每条汇编指令长度固定，对于大立即数等长度较长的数据，采用多条指令分段载入的方法（lui）。

2.1.1. 存储器操作数

2.1.2. 常数或立即数操作数

2.2. 逻辑操作指令

- 左移
- 右移
- 算术右移
- 按位与
- 或
- 异或
- 取反

2.3. 决策指令

主要使用条件分支指令（beq、bne等）进行回跳（循环）或前跳（if）

2.3.1. 条件分支指令

```
// 如果rs1中的值与rs2中的值相等，那么PC跳转到标签L1处
beq rs1, rs2, L1

// 如果rs1中的值与rs2中的值不相等，那么PC跳转到标签了L2处
bne rs1, rs2, L2
```

2.3.2. 循环

```
// rs1持续加一，直到rs1等于10则退出循环
addi rs1, rs0, 1
addi rs2, rsr0, 10
Loop:
addi rs1, rs1 ,1
beq rs1, rs2, Exit
j Loop
Exit:
//退出循环
```

2.3.3. case/switch语句

两种方法：

1. 将case/switch语句转换为if-then-else语句

2. 使用分支地址表。程序索引到地址表中，然后跳转到对应的地址。

2.4. 计算机硬件对函数的支持

过程（函数）运行的6个步骤： 1. 将参数放在过程可以访问的位置 2. 将控制转交给过程（函数） 3. 获得过程所需的储存资源 4. 执行任务 5. 将结果放在调用程序可以访问的位置 6. 将控制返回初始点

在运行一个过程（函数）时，如果需要使用更多寄存器，那么必须提前将寄存器的值保存。将寄存器的值保存进存储器中，最合适的数据结构是使用栈。

在运行过程（函数）时，在栈中存储局部变量，在堆中存储常量和静态变量

2.4.1. 使用更多的寄存器

2.4.2. 嵌套过程

2.4.3. 在栈中为新数据分配空间

2.4.4. 在堆中为新数据分配空间

2.5. 并行性与指令：同步

1. 通过加锁与解锁来实现对内存单元的独占

2. 使用指令对，保证指令对中间不出现其他操作，确保内存单元未被改变

2.6. 翻译并启动程序

编译器	将高级程序语言编译为汇编语言程序
汇编器	处理汇编语言中的伪指令，将其编译为机器语言。此时只有程序自身的机器语言指令
链接器	将调用到的官方库的机器语言与程序缝合成为可执行的代码
加载器	将机器代码放入存储器中

2.6.1. 动态链接库

> 这个比较麻烦,以后再补吧.

3. 计算机的算术运算

3.1. 算术运算

- 加法和减法
- 乘法
  - 串行版的乘法运算及其硬件实现
  - 带符号乘法
  - 快速乘法
  - RISC-V中的乘法

- 除法
  - 除法运算及其硬件实现
  - 有符号除法、
  - 快速除法
  - RISC-V中的除法
- 浮点运算
  - 浮点表示
  - 例外和中断
  - IEEE754浮点数标准
  - 浮点加法
  - 浮点乘法
  - RISC-V中的浮点指令
  - 精确算术

3.2. 并行性与计算机算术

通过划分进位链，可以同时对多个短向量进行并行操作。即数据级并行

4. 处理器

4.1. 单周期处理器实现

4.1.1. 逻辑设计的基本方法

4.1.2. 数据通路

4.1.3. 实现方案

4.2. 多周期实现

4.3. 流水线概述

使用流水线来使指令能重叠执行，以提高性能。即指令级并行（ILP）。

影响流水线性能的主要为结构冒险、数据冒险、控制冒险。使用动态调度、分支预测等进行优化

4.3.1. 面向流水线的指令系统设计

4.3.2. 流水线数据通路和控制

4.3.3. 利用指令级并行的基本编译器技术

4.4. 冒险与竞争

结构冒险	缺乏硬件支持导致，可以在设计流水线时避免
数据冒险	一个指令必须等待其他指令的结果才能完成导致的停顿为数据冒险，采用前递或旁路、动态调度技术优化

控制冒险	在分支判断结果未出现时，无法得知下一条指令是什么，导致停顿。采用分支预测技术优化

4.5. 例外

4.5.1. RISC-V体系结构中如何处理例外

4.5.2. 流水线实现中的例外

4.6. 指令间的并行性

编译器或处理器来猜测指令的行为并提前开始执行。如果猜测正确则进行指令提交，错误则清除结果并从执行正确的指令。

- 推测的概念
- 基于硬件的推测
- 以多发射和静态调度来利用指令级并行
- 以动态调度、多发射和推测来利用指令级并行
- 用于指令交付和推测的高级技术

5. 存储层次结构

5.1. 存储技术及其优化

- SRAM技术
- DRAM，SDRAM技术
- 闪存、磁盘
- 图形数据RAM
- 堆叠式或嵌入式DRAM
- 相变存储器技术

5.2. 存储层次结构的一般框架

缓存是位于处理器与存储器之间的速度更快的存储器。作用为将存储器中的数据提前放入速度更快的缓存中，处理器读写数据时先在缓存内查找，从而同时获得大容量与高速的存储器。

写穿透	处理器在进行写操作时同时向缓存与主存中写入，为避免写主存引起的长延时，还会增加写缓冲区。
写返回	处理器进行写操作时只对缓存进行写入，并标记脏位。在这个块需要替换时才会写到主存中。此方法减少了对主存的频繁写入。

5.2.1. 块的位置

Table 1. 块的识别方法以及定位方法

机制	定位方法
全相联	查找所有cache表项

机制	定位方法
直接映射	索引
组相联	索引组，查找组中的元素
全相联	查找所有cache表项

5.2.2. 块的识别

5.2.3. 块的替换

5.2.4. 写入策略

5.2.5. 失效的定义

5.2.6. 汉明编码

5.2.7. 3C模型

5.2.8. cache的性能评估

5.2.9. 优化缓存性能

5.3. 提高存储器系统的可靠性

5.4. 使用有限状态自动机控制简单的cache

5.5. 虚拟存储器和虚拟机

要实现多个程序同时运行，共享内存空间。将内存划分并通过页表将程序与真实的物理地址相联系，这样在程序看来是自己独占内存。

虚拟机可以使多个用户共享同一台计算机，且用户本身感知不到其他用户的存在。虚拟机监视器（VMM）决定如何将虚拟资源映射到物理资源上。

5.5.1. 页的存放、查找、失效

5.5.2. 快速地址变换技术（TLB）

5.5.3. 通过虚拟存储器提供保护

5.5.4. 通过虚拟机提供保护

5.5.5. 对虚拟机监视器的要求

5.5.6. 虚拟机的指令集体系结构支持

5.5.7. 虚拟机对虚拟存储器和I/O的影响

5.5.8. 扩展指令集

6. 数据级并行

单指令流多数据流（SIMD）使得一条向量指令代表了多条指令，同时流水化处理多条数据，从而减少了指令获取和解码的带宽。同时由于每条向量指令的行为已知，可以有效避免竞争冒险的出现。

6.1. 向量体系结构

6.1.1. 向量处理器的工作原理

6.1.2. 向量执行时间

6.1.3. 单指令流多数据流（SIMD）

6.1.4. 向量长度寄存器

处理未知向量长度的循环

条带挖掘技术使得每个向量运算都是针对向量大小小于或等于最大向量长度的情况来完成的。

6.1.5. 谓词寄存器（Predicate Registers）

允许处理器在执行指令时跳过某些操作，从而实现分支控制。

6.1.6. 存储体

6.1.7. 步幅

6.1.8. 向量体系中稀疏矩阵的处理

6.1.9. 向量体系结构编程

6.2. 图形处理器

6.2.1. GPU编程

> 写一个整体的简单总结,写不了就留TODO,以后再补.

- NVIDIA GPU拥有强大的并行处理能力和高带宽存储结构，通过大量的核心对大量数据进行并行处理。
- 其本质是一个多线程SIMD处理器，并且拥有更多处理器，每个处理器的通道更多，多线程硬件也更多。
- 适合处理大量相同类型的并行任务。

6.3. 检测与增强循环级并行

6.3.1. 查找相关

6.3.2. 清除相关计算

7. 线程级并行

在多个处理器上同时执行多个线程，提高程序性能及吞吐量。

处理器之间共享数据有两种方法：1.所有处理器共享一块内存（集中式共享存储器/对称共享存储器）。2.每个处理器有自己的内存但其他处理器可以访问（分布式共享存储器）

7.1. 多处理器体系结构

7.2. 集中式/对称共享存储器体系结构

多处理器需要解决缓存一致性问题。

使用监听一致性协议。多核CPU各自保存数据副本，如果一个核心对数据进行了修改，那么其他核心保存的数据将过期。通过写失效来保证数据同步。

7.2.1. 多处理器缓存一致性概念

7.2.2. 一致性的基本实现方案

7.2.3. 监听一致性协议

7.2.4. 基本一致性协议的拓展

7.2.5. 对称共享存储器多处理器与监听协议的局限性

7.2.6. 实现监听缓存一致性

7.3. 集中式/对称共享存储器多处理器的性能

多个处理器共享同一块内存，处理器之间可以很方便的共享资源，并且处理器之间通信比分布式要快。但是处理器访问内存都要占用总线，当处理器数量较多时会因为带宽不足而影响性能。同时也容易出项竞争冒险现象。如果内存损坏，会影响整个系统的工作，稳定性不如分布式共享存储器结构

7.3.1. 商业工作负载对性能的影响

7.3.2. 多道程序和操作系统工作负载对性能的影响

7.4. 分布式共享存储器和目录一致性

集中式/对称共享存储器体系结构由于总线带宽等限制，处理器比较少。分布式共享存储器结构则是每个处理器有独立存储器，以允许增加更多核以及处理器。

同时为了减少带宽占用，使用了目录一致性协议。每个处理器在写数据时，只对目录进行通信。目录记录了数据的所有者以及一致性状态等信息。目录与存储器一起分配，使得不同的一致性请求访问不同的目录，从而防止竞争冒险且减少了带宽占用。

7.4.1. 目录式缓存一致性协议

目录式缓存一致性协议能有效减少维持缓存一致性的流量，可以扩展到大量处理器的系统中去。缺陷是在有较多处理器情况下目录储存开销较大，且访问内存时因为需要查目录，可能增加访问延迟。

当一个处理器请求访问一个内存块时，会首先查询目录以获取状态。

写操作	如果其他处理器内存块内有缓存该内存块，那么目录发出无效化消息通知其他处理器使他们的副本无效。
读操作	目录更新共享列表。

7.4.2. 实例目录协议

7.5. 同步基础

原语不可分割，要么全部执行成功，要么全部执行失败，可以利用它来实现同步机制以及减少竞争冒险现象的发生。

实现自旋锁：

函数不断使用原子操作获取锁，如果已经被占用则一直在循环中自旋等待解锁。

适用于希望短时间获取这个锁以及在锁可用时锁定延迟较低的情形。但是自旋锁会占用CPU资源，不适用于长时间等待以及可能出现死锁的情况。

7.5.1. 基本硬件原语

7.5.2. 使用一致性实现锁

7.6. 存储器一致性模型

存储器一致性模型保证了在多处理器对内存的访问的数据一致性，不同模型决定了处理器如何对待内存访问的顺序性，从而影响程序的正确性和性能。

7.6.1. 简介

	顺序一致性	要求所有处理器的而操作按照程序中规定的顺序执行，且所有处理器看到的操作顺序一致
宽松一致性模型	完全存储排序或处理器一致性	仅放松W→R顺序。保持了写操作之间的顺序
	部分存储排序	放松W→R和W→W顺序
	弱排序	放松所有四种顺序
	释放一致性	放松所有四种顺序。区分了用于获取对共享变量访问的同步操作（标记为S_A）和那些释放对象以允许其他处理器获取访问的同步操作（标记为S_R）

7.6.2. 宽松一致性模型

7.7. 多处理器测试基准和性能模型

- 性能模型
- Roofline模型

7.7.1. 两代Opteron的比较

8. 集群、仓库级计算机（WSC）

高性能计算（HPC）集群与仓库级计算机（WSC）应用领域不同。前者更倾向于线程级并行，主要解决复杂问题。而后者强调请求级并行，同时为多个用户进行服务。

8.1. 仓库级计算机的编程模型与工作负载

8.2. 仓库级计算机的计算机体系结构

8.2.1. 存储

8.2.2. WSC存储器层次结构

8.3. 仓库级计算机的效率与成本

8.3.1. 测量WSC的效率

8.3.2. WSC的成本

8.4. 云计算：效用计算的回报

9. 领域专用体系结构

针对特定领域定制处理器，加速某些应用程序以实现更好的性能与性价比

9.1. DSA指导原则

9.2. 示例领域：深度神经网络

> 写成列表,算法和网络类型分开.写不出来可以不写,留个简单的列表就行

- 算法
  - DNN的神经元
  - 训练与推理
  - 多层感知机
  - 批数据
  - 量化
- 网络类型
  - 卷积神经网络
  - 循环神经网络

9.3. Google的张量处理单元——一种数据中心推理加速器

- TPU的起源
- TPU体系结构
- TPU指令集体系结构
- TPU微体系结构
- TPU实现
- TPU软件
- 改进TPU

9.4. Microsoft Catapult——一种灵活的数据中心加速器

- Catapult实现与体系结构
- Catapult软件
- Catapult上的CNN
- Catapult上的搜索加速
- Catapult Ver 1 的部署
- Catapult Ver 2

9.5. Intel Crest——一种用于训练的数据中心加速器

9.6. Pixel Visual Core——一种个人移动设备图像处理单元

- ISP——IPU的硬连线前身
- Pixel Visual Core 软件
- Pixel Visual Core 体系结构的理念
- Pixel Visual Core 光晕
- Pixel Visual Core 的处理器
- Pixel Visual Core 指令集体系结构
- Pixel Visual Core 示例
- Pixel Visual Core PE
- 二维行缓冲区及其控制器
- Pixel Visual Core 实现

Last updated 2024-12-27 19:16:52 +0800