高等计算机体系结构
第一讲: 概论

栾钟治
北京航空航天大学 计算机学院 中德联合软件研究所
2020-03-06

1

你们为什么而来

“C”：一种计算模式　　计算机系统工作的程序员视角

数字逻辑：一种计算模式　　计算机系统工作的硬件设计者视角

2

2

你们为什么而来

“C”：一种计算模式　　计算机系统工作的程序员视角

■这中间发生了什么?

数字逻辑：一种计算模式　　计算机系统工作的硬件设计者视角

3

3

你们为什么而来

“C”：一种计算模式　　计算机系统工作的程序员视角

■汇编程序最终是如何按照数字逻辑的方式执行的?

■这中间发生了什么?

■一台用逻辑门和连线设计实现的计算机是如何满足那些特定目标的?

数字逻辑：一种计算模式　　计算机系统工作的硬件设计者视角

4

4

你们为什么而来

“C”：一种计算模式

计算机系统工作的程序员视角

■汇编程序最终是如何按照数字逻辑的方式执行的？

■这中间发生了什么？

■一台用逻辑门和连线设计实现的计算机是如何满足那些特定目标的？

体系架构师/微架构师的视角：
如何设计一台能够满足
系统设计目标的计算机。

数字逻辑：一种计算模式

计算机系统工作的硬件设计者视角

5

5

你们为什么而来

“C”：一种计算模式

计算机系统工作的程序员视角

■汇编程序最终是如何按照数字逻辑的方式执行的？

■这中间发生了什么？

■一台用逻辑门和连线设计实现的计算机是如何满足那些特定目标的？

体系架构师/微架构师的视角：
如何设计一台能够满足
系统设计目标的计算机。

这里如何选择将严重影响软件设计者和硬件设计者

数字逻辑：一种计算模式

计算机系统工作的硬件设计者视角

6

6

计算机系统的层次结构

高级语言程序(e.g. C)

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

10

7

计算机系统的层次结构

高级语言程序(e.g. C)

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

编译器

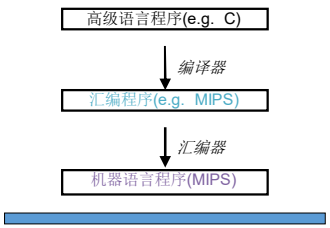
汇编程序(e.g. MIPS)

```
Lw   $t0, 0($2)
Lw   $t1, 4($2)
Sw   $t1, 0($2)
Sw   $t0, 4($2)
```

11

8

计算机系统的层次结构




```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

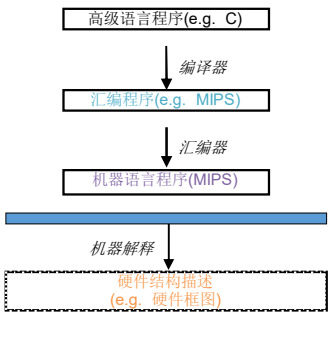
```
Lw   $t0, 0($2)
Lw   $t1, 4($2)
Sw   $t1, 0($2)
Sw   $t0, 4($2)
```

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

12

9

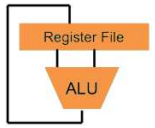
计算机系统的层次结构




```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
Lw   $t0, 0($2)
Lw   $t1, 4($2)
Sw   $t1, 0($2)
Sw   $t0, 4($2)
```

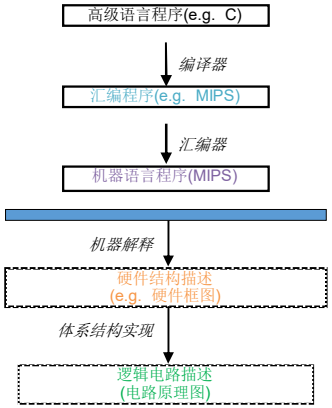
```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```



13

10

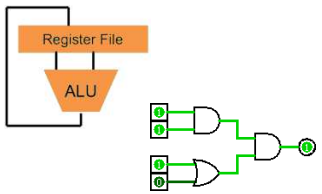
计算机系统的层次结构




```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
Lw   $t0, 0($2)
Lw   $t1, 4($2)
Sw   $t1, 0($2)
Sw   $t0, 4($2)
```

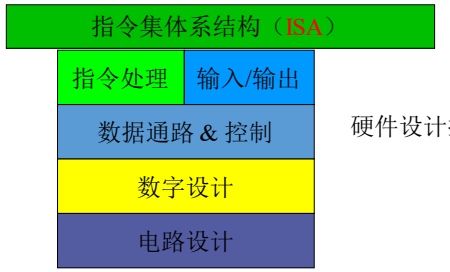
```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```



14

11

计算机系统的层次结构

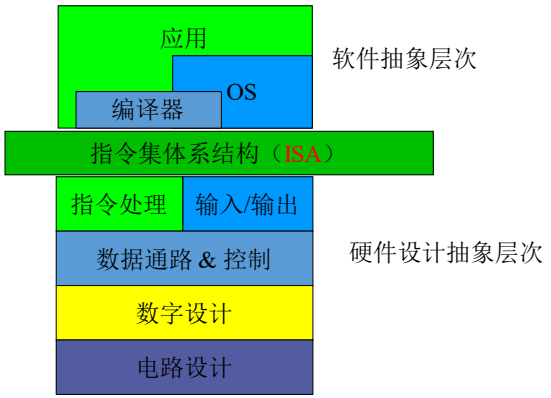


硬件设计抽象层次

16

12

计算机系统的层次结构



软件抽象层次

硬件设计抽象层次

17

13

系统层次结构

"The purpose of computing is insight" (*Richard Hamming*)
我们在解决问题过程中获得洞察
如何确保可以用电子解决问题?

问题

电子

14

14

系统层次结构

"The purpose of computing is insight" (*Richard Hamming*)
我们在解决问题过程中获得洞察
如何确保可以用电子解决问题?

问题
算法

电子

15

15

系统层次结构

"The purpose of computing is insight" (*Richard Hamming*)
我们在解决问题过程中获得洞察
如何确保可以用电子解决问题?

问题
算法
程序/语言

电子

16

16

系统层次结构

"The purpose of computing is insight" (*Richard Hamming*)
我们在解决问题过程中获得洞察
如何确保可以用电子解决问题?

问题
算法
程序/语言
运行时系统
(VM, OS, MM)

电子

17

17

系统层次结构

"The purpose of computing is insight" (*Richard Hamming*)
我们在解决问题过程中获得洞察
如何确保可以用电子解决问题?

问题
算法
程序/语言
运行时系统
(VM, OS, MM)
ISA (体系结构)

电子

18

18

系统层次结构

"The purpose of computing is insight" (*Richard Hamming*)
我们在解决问题过程中获得洞察
如何确保可以用电子解决问题?

问题
算法
程序/语言
运行时系统
(VM, OS, MM)
ISA (体系结构)
微体系结构

电子

19

19

系统层次结构

"The purpose of computing is insight" (*Richard Hamming*)
我们在解决问题过程中获得洞察
如何确保可以用电子解决问题?

问题
算法
程序/语言
运行时系统
(VM, OS, MM)
ISA (体系结构)
微体系结构
逻辑

电子

20

20

系统层次结构

"The purpose of computing is insight" (*Richard Hamming*)
我们在解决问题过程中获得洞察
如何确保可以用电子解决问题?

问题
算法
程序/语言
运行时系统
(VM, OS, MM)
ISA (体系结构)
微体系结构
逻辑
电路
电子

系统层次结构

"The purpose of computing is insight" (*Richard Hamming*)
我们在解决问题过程中获得洞察
如何确保可以用电子解决问题?

问题
算法
程序/语言
运行时系统
(VM, OS, MM)
ISA (体系结构)
微体系结构
逻辑
电路
电子

抽象的力量

- 分层描述系统产生了抽象
  - 抽象: 上一层只需要知道跟下一层的接口，而不需要知道下一层是如何实现的
  - 例如, 高级语言的程序员并不需要“真的”了解什么是ISA以及计算机是如何执行指令的
- 抽象提高了生产效率
  - 不需要考虑底层所做出的决策
  - 例如, Java编程 vs. C编程 vs. 汇编语言编程 vs. 二进制代码 vs. 每一个时钟周期作用在每个晶体管上的电信号

抽象的力量

- 分层描述系统产生了抽象
  - 抽象: 上一层只需要知道跟下一层的接口，而不需要知道下一层是如何实现的
  - 例如, 高级语言的程序员并不需要“真的”了解什么是ISA以及计算机是如何执行指令的
- 抽象提高了生产效率
  - 不需要考虑底层所做出的决策
  - 例如, Java编程 vs. C编程 vs. 汇编语言编程 vs. 二进制代码 vs. 每一个时钟周期作用在每个晶体管上的电信号
- 那么，为什么你想要知道到底下一层或者上一层发生了什么呢?

跨越抽象层次

- 如果一切顺利，不知道别的层次发生了什么也无关紧要
- 但是，如果
  - 你写的程序跑得很慢呢?
  - 你写的程序执行得不正确呢?
  - 你写的程序能耗太高呢?

25

25

跨越抽象层次

- 如果一切顺利，不知道别的层次发生了什么也无关紧要
- 但是，如果
  - 你写的程序跑得很慢呢?
  - 你写的程序执行得不正确呢?
  - 你写的程序能耗太高呢?
- 但是，如果
  - 在你设计的硬件上编程太困难呢?
  - 你设计的硬件效率太低因为你为软件提供的原语不够合理呢？

26

26

跨越抽象层次

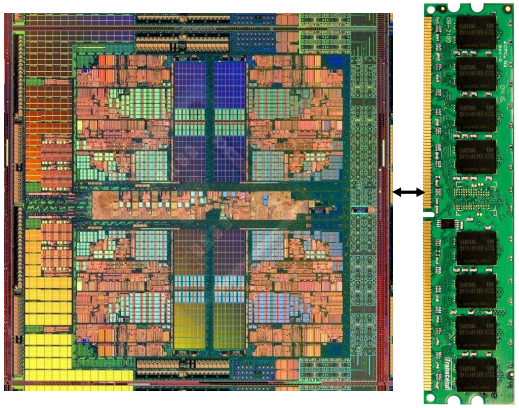
- 如果一切顺利，不知道别的层次发生了什么也无关紧要
- 但是，如果
  - 你写的程序跑得很慢呢?
  - 你写的程序执行得不正确呢?
  - 你写的程序能耗太高呢?
- 但是，如果
  - 在你设计的硬件上编程太困难呢?
  - 你设计的硬件效率太低因为你为软件提供的原语不够合理呢？
- 这门课的一个目标就是理解一个处理器是如何工作在软件层面之下以及硬件的决策是如何影响软件和程序员的

27

27

一个例子

多核芯片



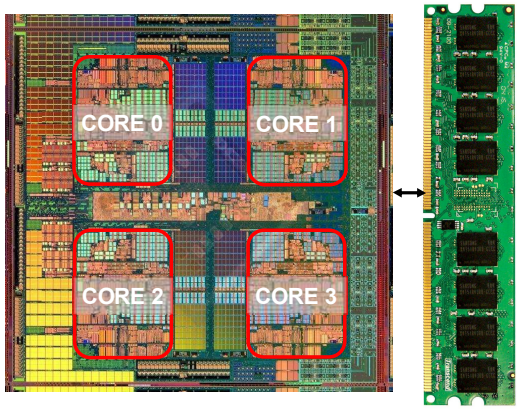
* AMD Barcelona

28

28

一个例子

多核芯片



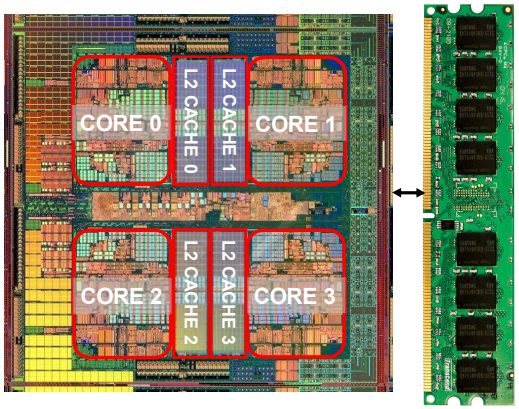
* AMD Barcelona

29

29

一个例子

多核芯片



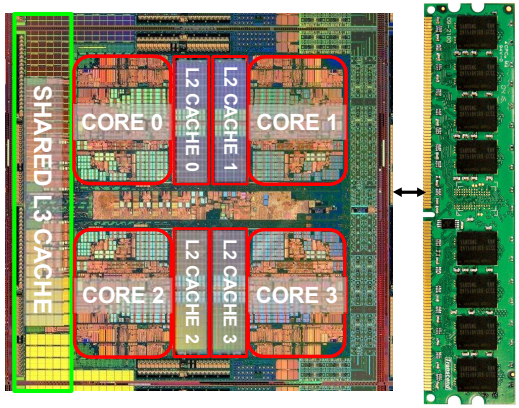
* AMD Barcelona

30

30

一个例子

多核芯片



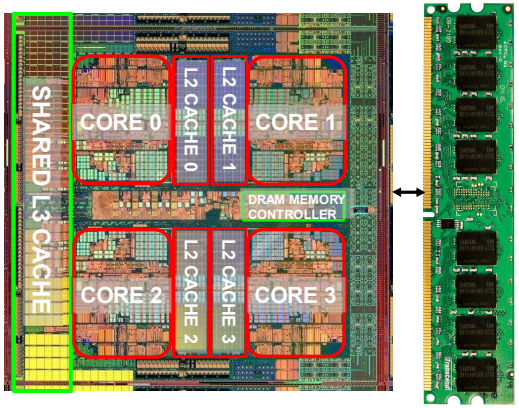
* AMD Barcelona

31

31

一个例子

多核芯片



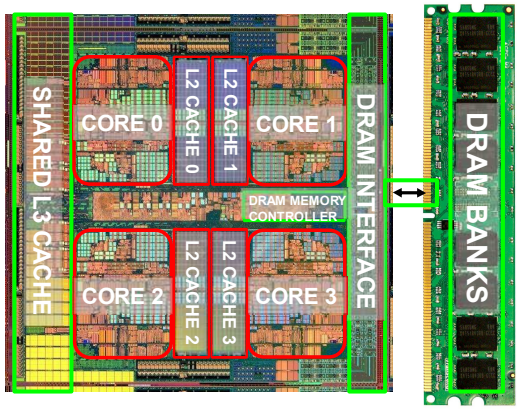
* AMD Barcelona

32

32

一个例子

多核芯片

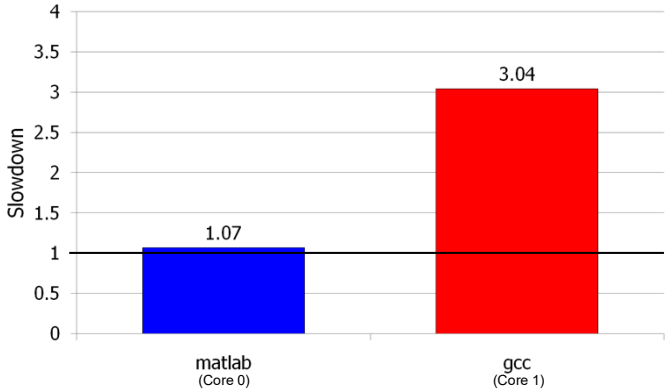


* AMD Barcelona

33

33

多核环境下的性能下降现象

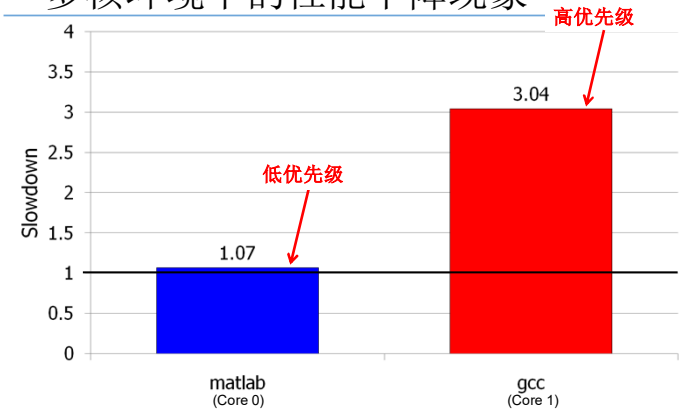


Moscibroda and Mutlu, "Memory performance attacks: Denial of memory service in multi-core systems," USENIX Security 2007.

34

34

多核环境下的性能下降现象

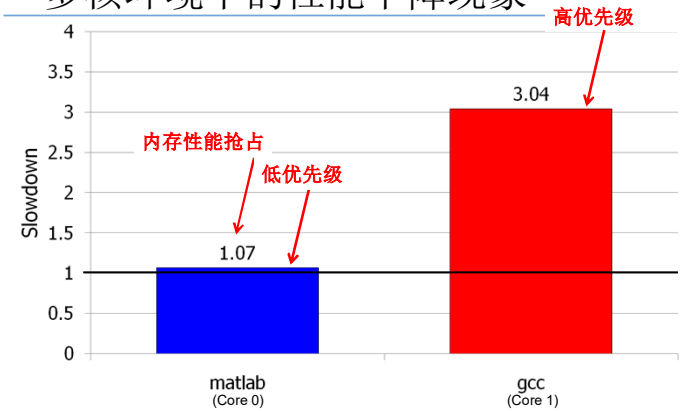


Moscibroda and Mutlu, "Memory performance attacks: Denial of memory service in multi-core systems," USENIX Security 2007.

35

35

多核环境下的性能下降现象



Moscibroda and Mutlu, "Memory performance attacks: Denial of memory service in multi-core systems," USENIX Security 2007.

36

36

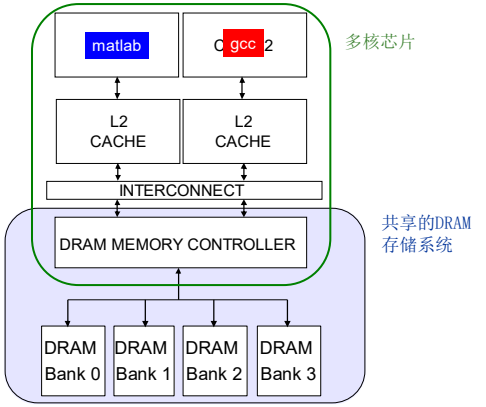
问题

- 如果你不知道处理器是如何执行程序的，你是否有可能找出性能下降差异的原因？
- 你能在不知道底层发生了什么的情况下解决问题吗？

37

37

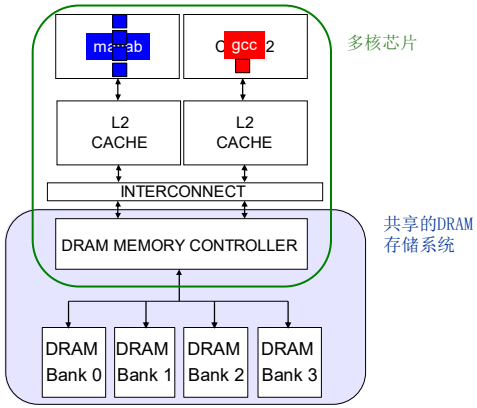
为什么性能下降会有这么大的差异?



38

38

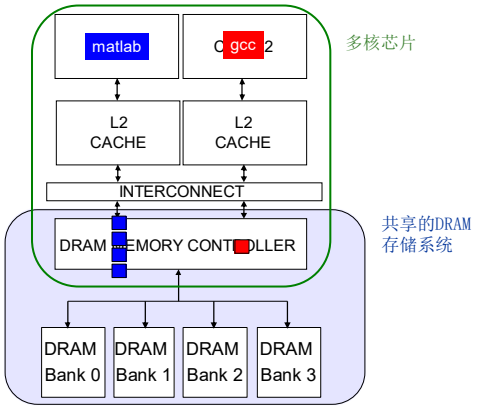
为什么性能下降会有这么大的差异?



39

39

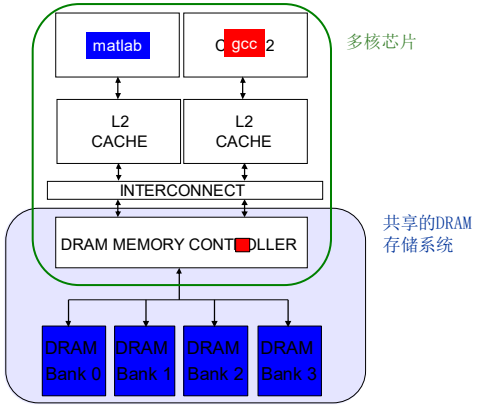
为什么性能下降会有这么大的差异?



40

40

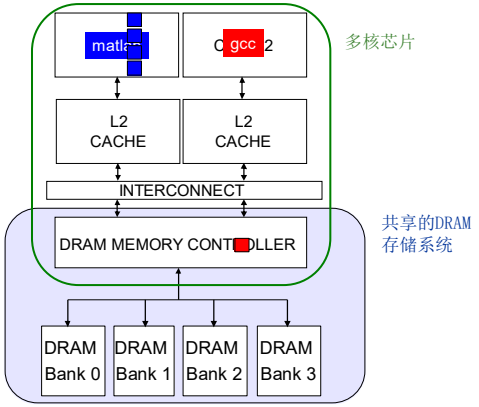
为什么性能下降会有这么大的差异?



41

41

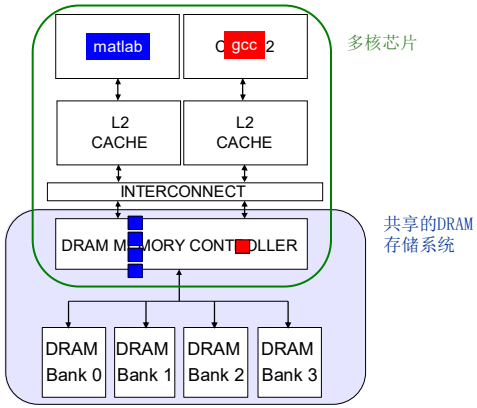
为什么性能下降会有这么大的差异?



42

42

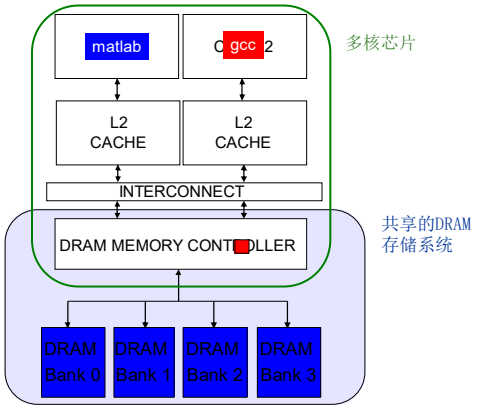
为什么性能下降会有这么大的差异?



43

43

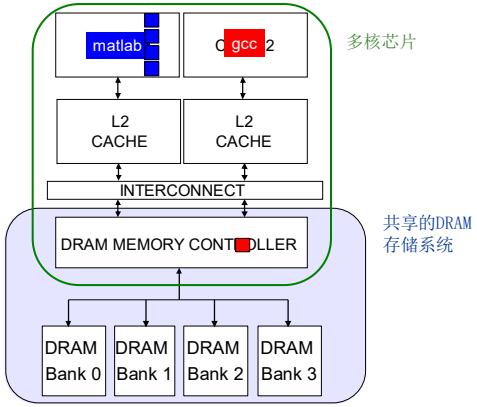
为什么性能下降会有这么大的差异?



44

44

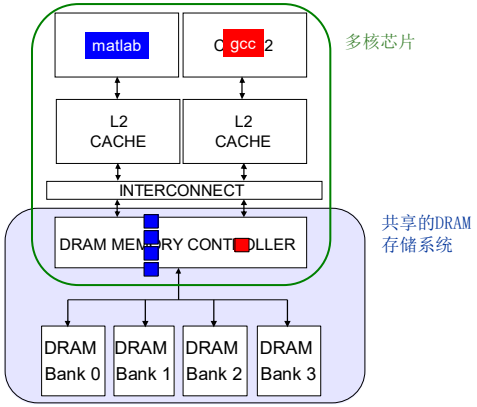
为什么性能下降会有这么大的差异?



45

45

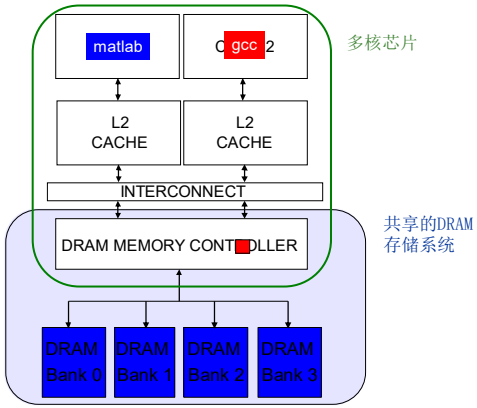
为什么性能下降会有这么大的差异?



46

46

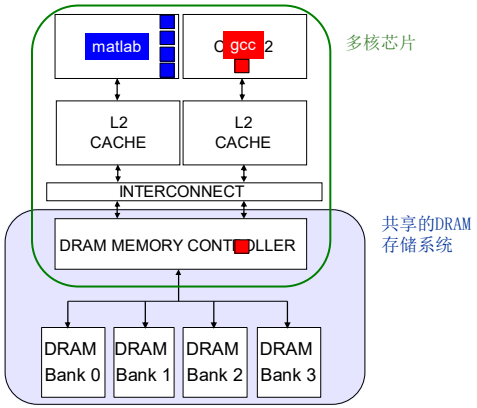
为什么性能下降会有这么大的差异?



47

47

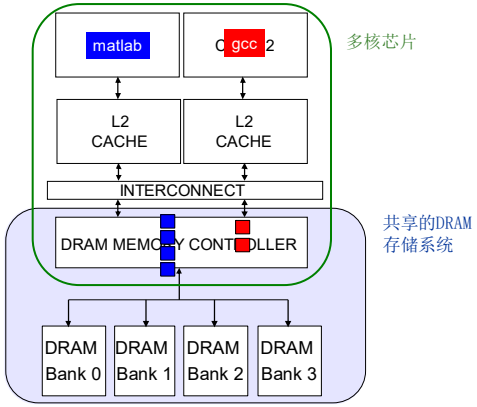
为什么性能下降会有这么大的差异?



48

48

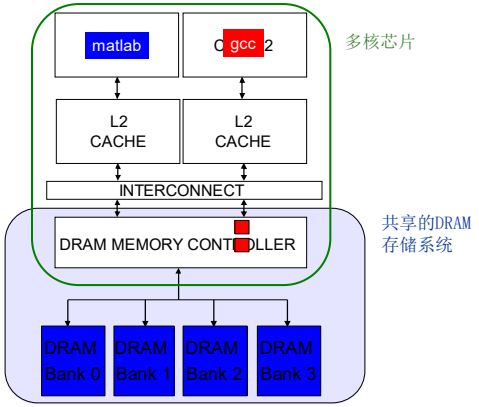
为什么性能下降会有这么大的差异?



49

49

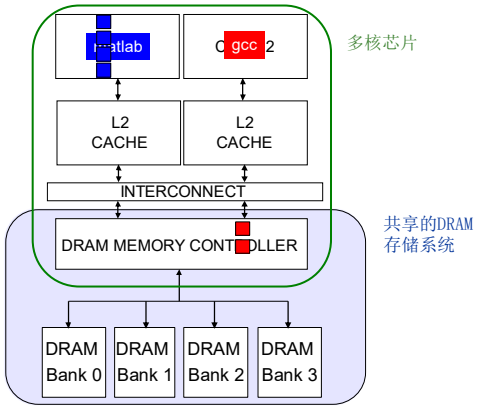
为什么性能下降会有这么大的差异?



50

50

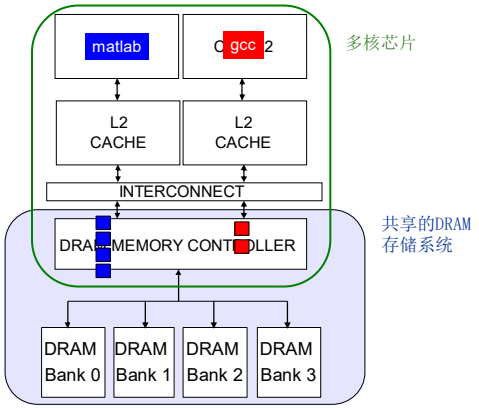
为什么性能下降会有这么大的差异?



51

51

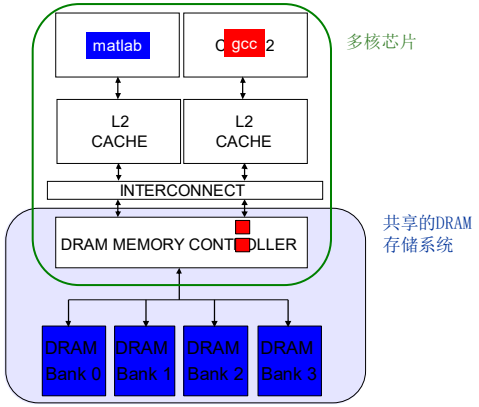
为什么性能下降会有这么大的差异?



52

52

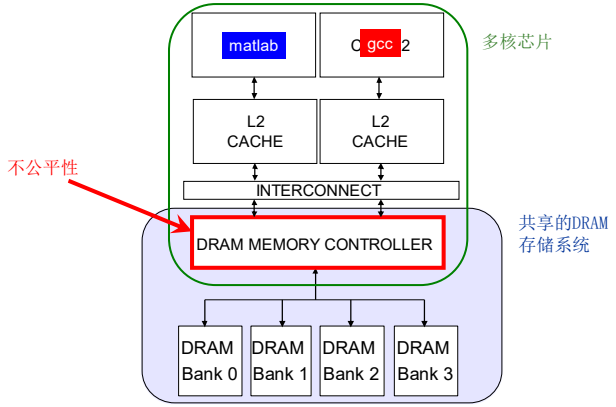
为什么性能下降会有这么大的差异?



53

53

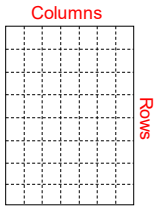
为什么性能下降会有这么大的差异?



54

54

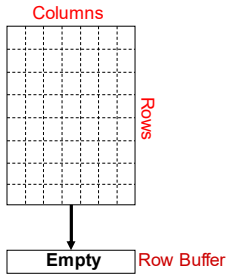
DRAM Bank的操作



55

55

DRAM Bank的操作

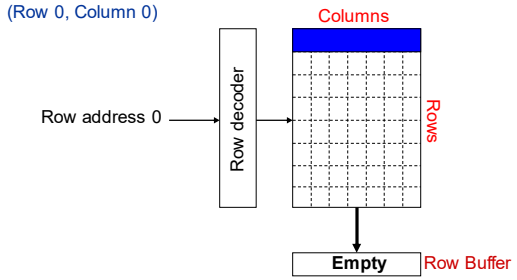


56

56

DRAM Bank的操作

访问地址:
(Row 0, Column 0)

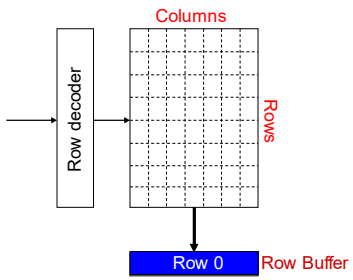


57

57

DRAM Bank的操作

访问地址:
(Row 0, Column 0)

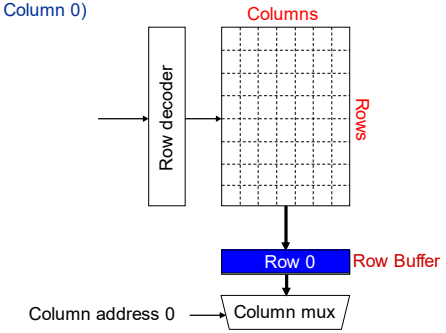


58

58

DRAM Bank的操作

访问地址:
(Row 0, Column 0)

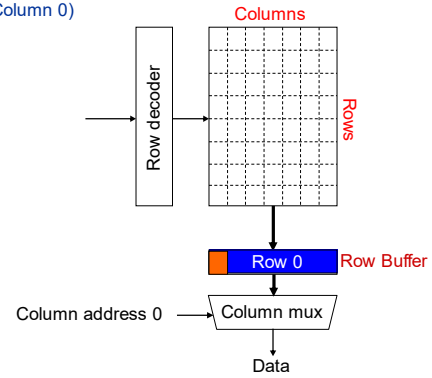


59

59

DRAM Bank的操作

访问地址:
(Row 0, Column 0)

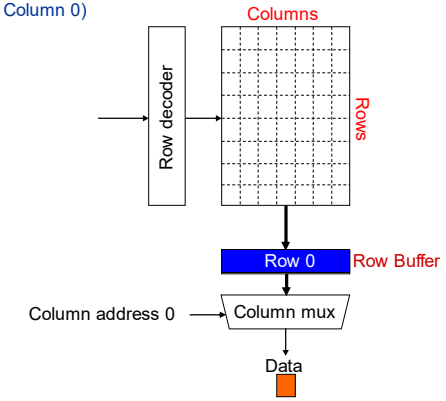


60

60

DRAM Bank的操作

访问地址:
(Row 0, Column 0)

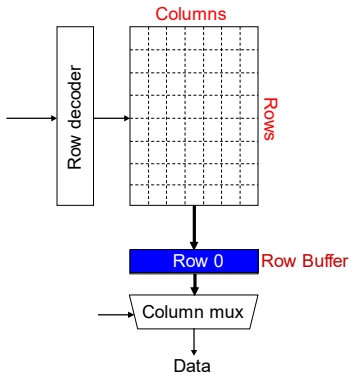


61

61

DRAM Bank的操作

访问地址:
(Row 0, Column 0)
(Row 0, Column 1)

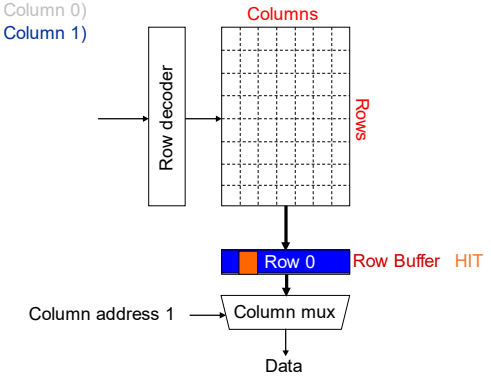


62

62

DRAM Bank的操作

访问地址:
(Row 0, Column 0)
(Row 0, Column 1)

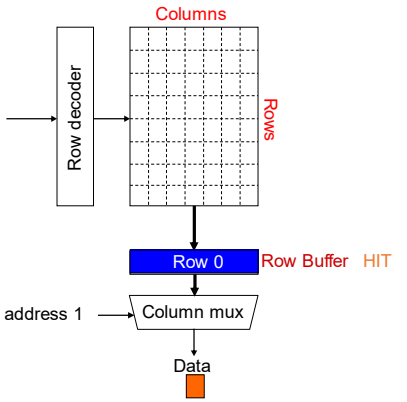


63

63

DRAM Bank的操作

访问地址:
(Row 0, Column 0)
(Row 0, Column 1)

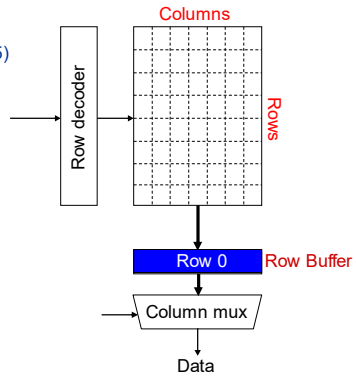


64

64

DRAM Bank的操作

访问地址:
(Row 0, Column 0)
(Row 0, Column 1)
(Row 0, Column 85)

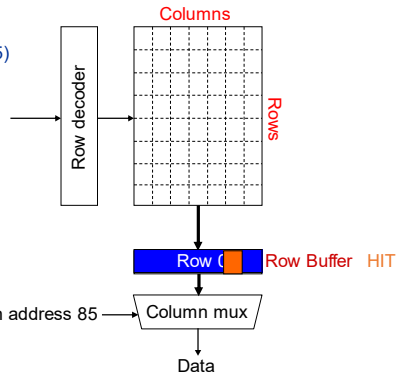


65

65

DRAM Bank的操作

访问地址:
(Row 0, Column 0)
(Row 0, Column 1)
(Row 0, Column 85)

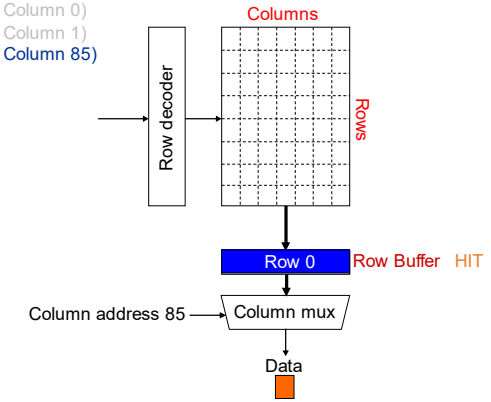


66

66

DRAM Bank的操作

访问地址:
(Row 0, Column 0)
(Row 0, Column 1)
(Row 0, Column 85)

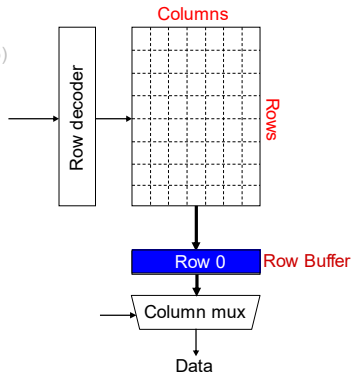


67

67

DRAM Bank的操作

访问地址:
(Row 0, Column 0)
(Row 0, Column 1)
(Row 0, Column 85)
(Row 1, Column 0)

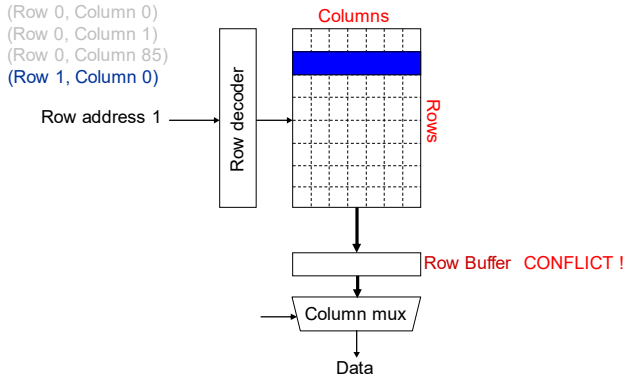


68

68

DRAM Bank的操作

访问地址:
(Row 0, Column 0)
(Row 0, Column 1)
(Row 0, Column 85)
(Row 1, Column 0)

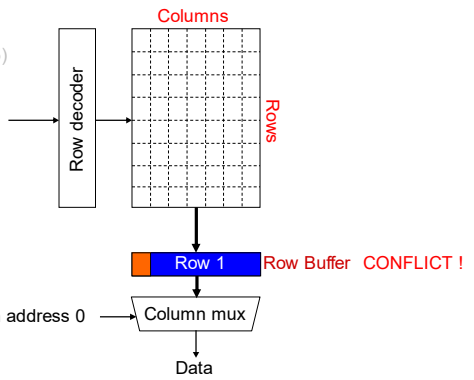


69

69

DRAM Bank的操作

访问地址:
(Row 0, Column 0)
(Row 0, Column 1)
(Row 0, Column 85)
(Row 1, Column 0)

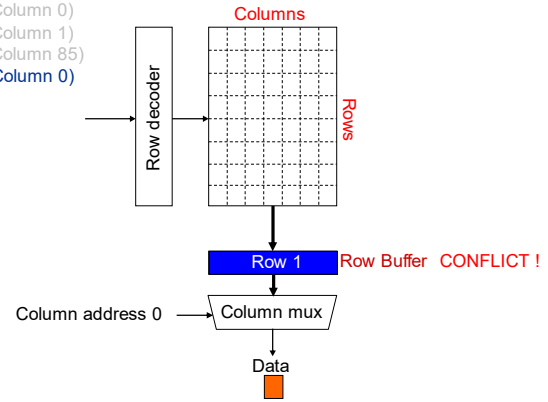


70

70

DRAM Bank的操作

访问地址:
(Row 0, Column 0)
(Row 0, Column 1)
(Row 0, Column 85)
(Row 1, Column 0)



71

71

DRAM 控制器

- 行不命中时的内存访问时间远远大于行命中时的访问时间
- 现在的控制器得益于行缓冲（row buffer）
- 通常采用的调度策略 (FR-FCFS) [Rixner 2000]*
  (1) Row-hit first: 优先行缓冲命中
  (2) Oldest-first: 然后先来先服务
- 这一调度策略的目的在于最大化DRAM的吞吐率

*Rixner et al., "Memory Access Scheduling," ISCA 2000.

72

72

问题

- 多线程共享DRAM控制器
- DRAM 控制器设计用来最大化DRAM吞吐率

- DRAM调度策略是“线程非公平”的
  - Row-hit first: 导致对于高“行缓冲局部性”线程的“不公平优先”
  - Oldest-first: 导致对于内存密集型线程的“不公平优先”

问题

- 多线程共享DRAM控制器
- DRAM 控制器设计用来最大化DRAM吞吐率

- DRAM调度策略是“线程非公平”的
  - Row-hit first: 导致对于高“行缓冲局部性”线程的“不公平优先”
  - Oldest-first: 导致对于内存密集型线程的“不公平优先”

- DRAM 控制器容易受到拒绝服务攻击
  - 可以写程序来利用这种不公平性

例子

```
// initialize large arrays A, B

for (j=0; j<N; j++) {
   index = j*linesize;   streaming
   A[index] = B[index];
   …
}
```

流

- 顺序的访存
- 极高的行缓冲局部性（96% 命中率）
- 内存密集型

例子

```
// initialize large arrays A, B

for (j=0; j<N; j++) {
   index = j*linesize;   streaming
   A[index] = B[index];
   …
}
```

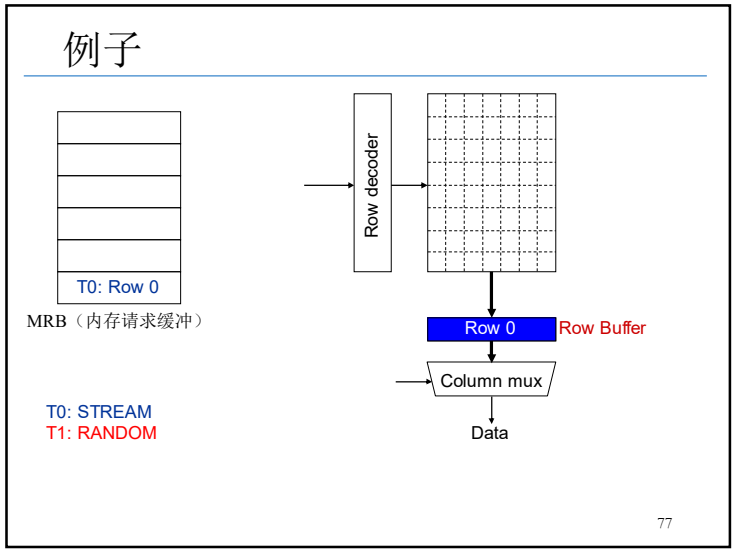
流

- 顺序的访存
- 极高的行缓冲局部性（96% 命中率）
- 内存密集型

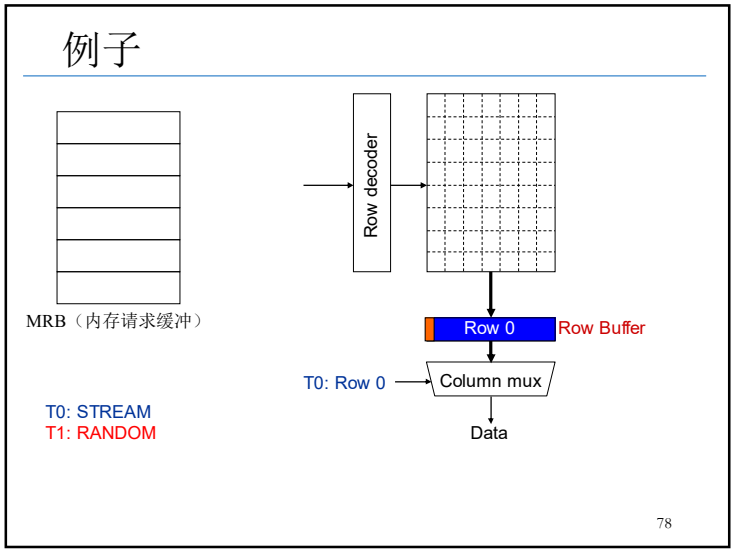
```
// initialize large arrays A, B

for (j=0; j<N; j++) {
   index = rand();   random
   A[index] = B[index];
   …
}
```

随机

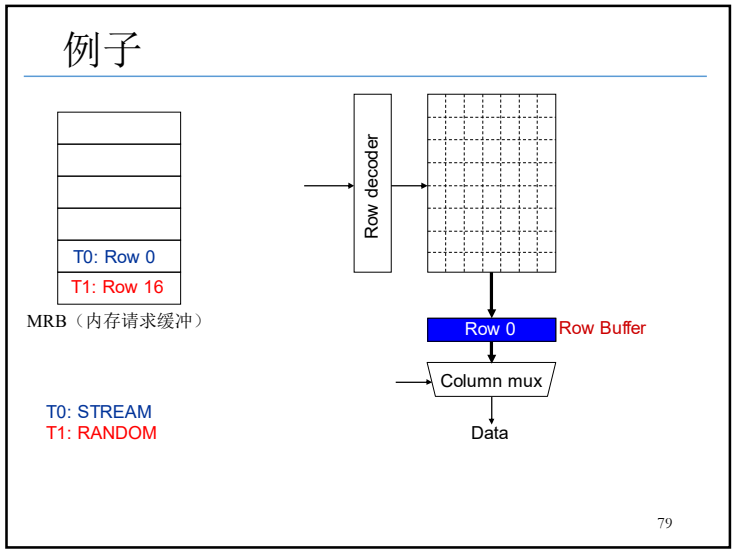
- 随机访存
- 极低的行缓冲局部性(3% 命中率)
- 同样是内存密集型


例子
T0: Row 0
MRB（内存请求缓冲）
Row decoder
Row 0
Row Buffer
Column mux
Data
T0: STREAM
T1: RANDOM
77

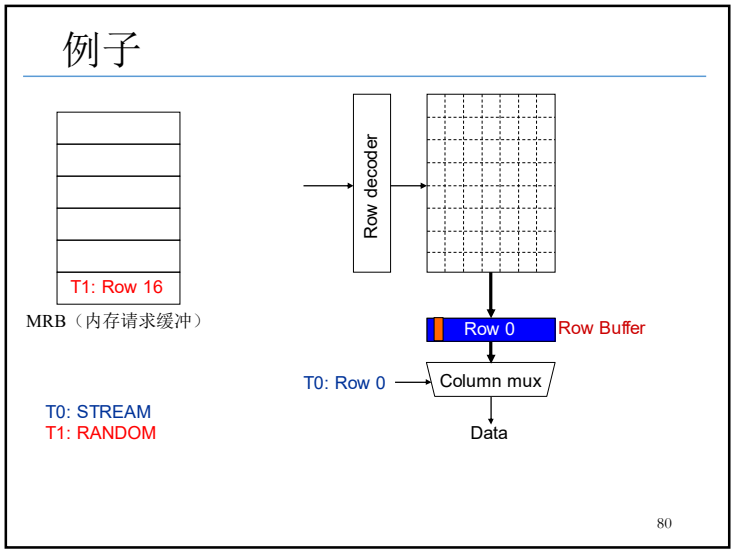
77


例子
MRB（内存请求缓冲）
Row decoder
Row 0
Row Buffer
T0: Row 0
Column mux
Data
T0: STREAM
T1: RANDOM
78

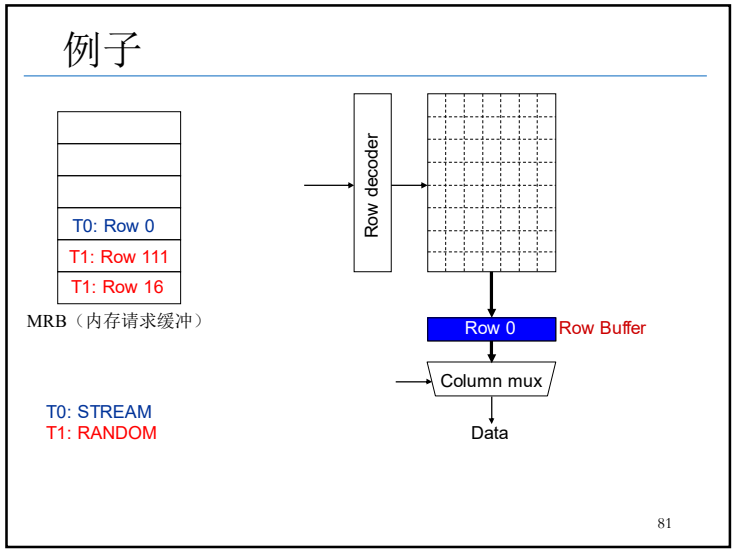
78


例子
T0: Row 0
T1: Row 16
MRB（内存请求缓冲）
Row decoder
Row 0
Row Buffer
Column mux
Data
T0: STREAM
T1: RANDOM
79

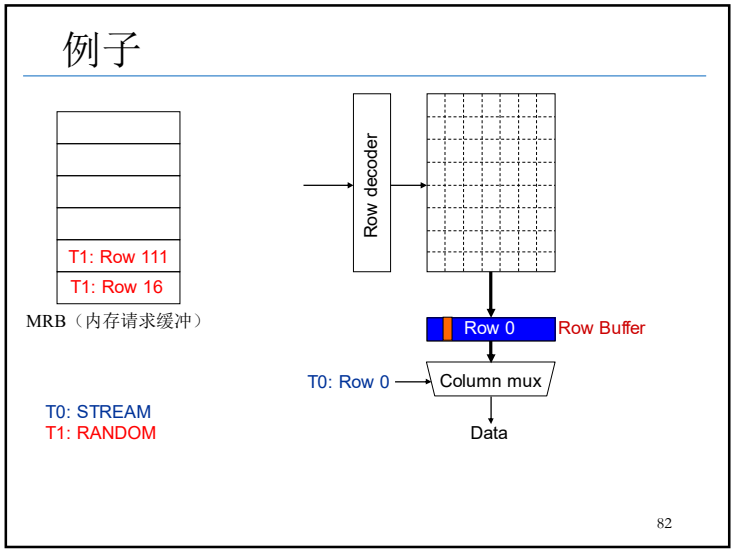
79


例子
T1: Row 16
MRB（内存请求缓冲）
Row decoder
Row 0
Row Buffer
T0: Row 0
Column mux
Data
T0: STREAM
T1: RANDOM
80

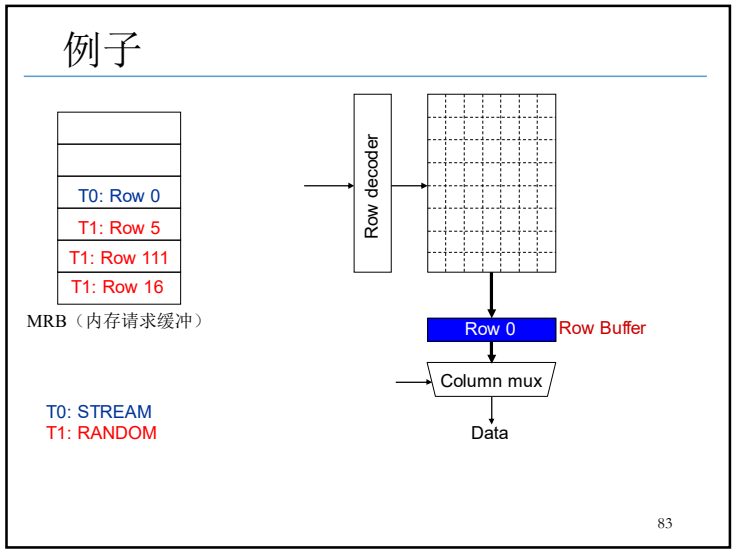
80


例子
T0: Row 0
T1: Row 111
T1: Row 16
MRB（内存请求缓冲）
Row decoder
Row 0
Row Buffer
Column mux
Data
T0: STREAM
T1: RANDOM
81

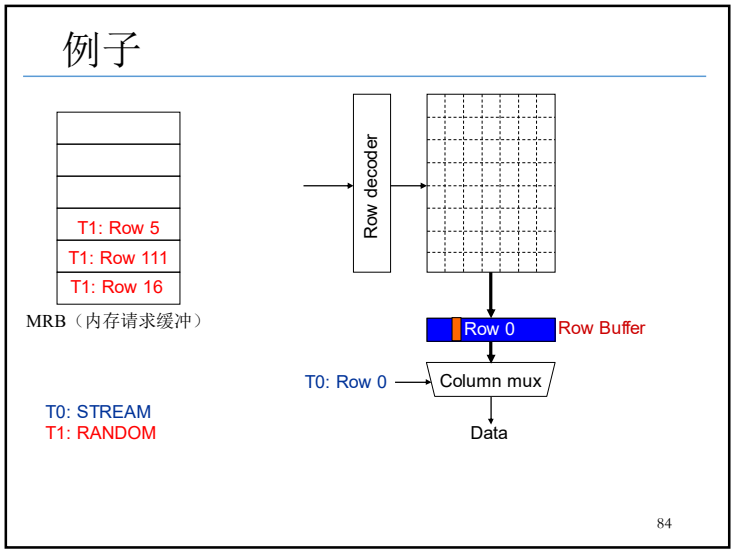
81


例子
T1: Row 111
T1: Row 16
MRB（内存请求缓冲）
Row decoder
Row 0
Row Buffer
T0: Row 0
Column mux
Data
T0: STREAM
T1: RANDOM
82

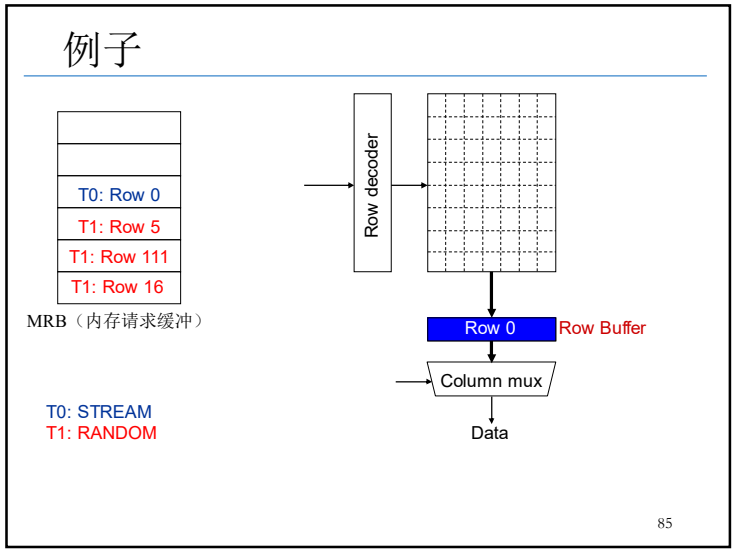
82


例子
T0: Row 0
T1: Row 5
T1: Row 111
T1: Row 16
MRB（内存请求缓冲）
Row decoder
Row 0
Row Buffer
Column mux
Data
T0: STREAM
T1: RANDOM
83

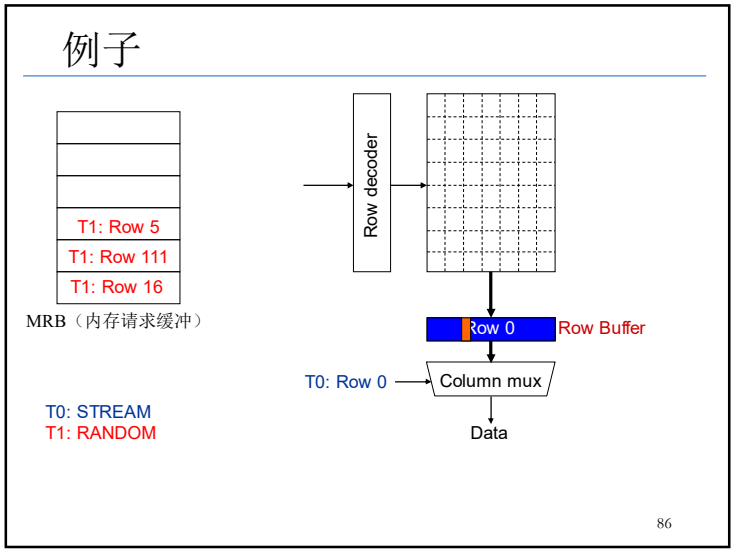
83


例子
T1: Row 5
T1: Row 111
T1: Row 16
MRB（内存请求缓冲）
Row decoder
Row 0
Row Buffer
T0: Row 0
Column mux
Data
T0: STREAM
T1: RANDOM
84

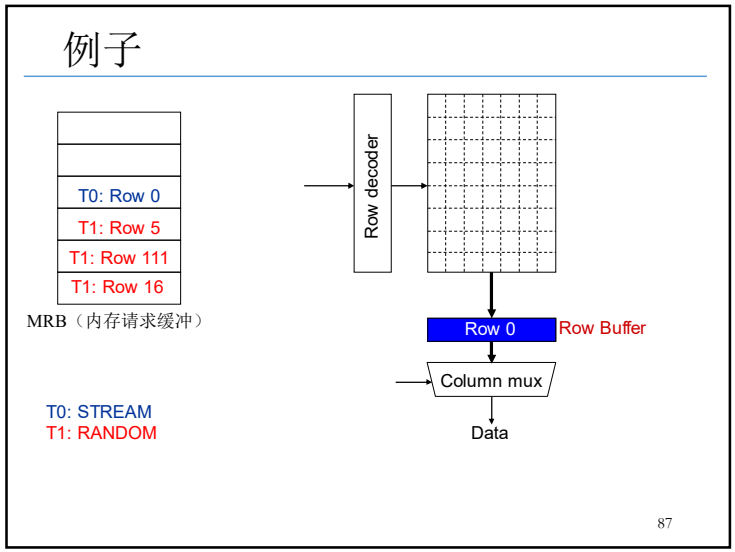
84


例子
T0: Row 0
T1: Row 5
T1: Row 111
T1: Row 16
MRB（内存请求缓冲）
Row decoder
Row 0
Row Buffer
Column mux
Data
T0: STREAM
T1: RANDOM
85

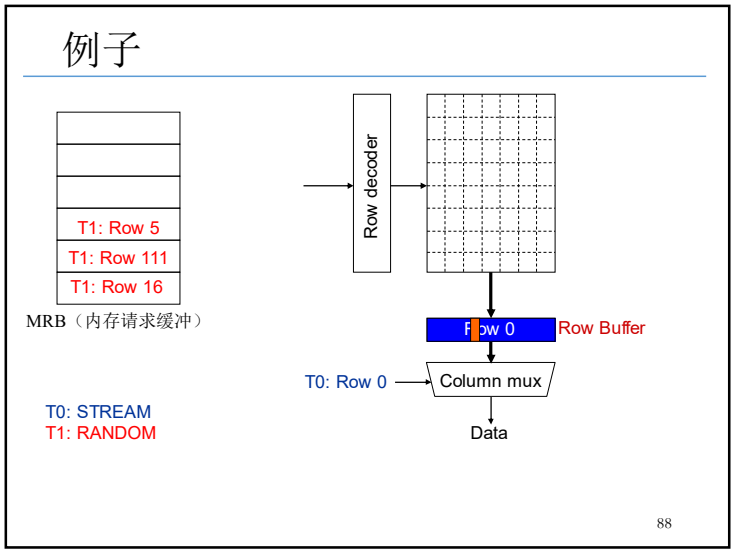
85


例子
T1: Row 5
T1: Row 111
T1: Row 16
MRB（内存请求缓冲）
Row decoder
Row 0
Row Buffer
T0: Row 0
Column mux
Data
T0: STREAM
T1: RANDOM
86

86

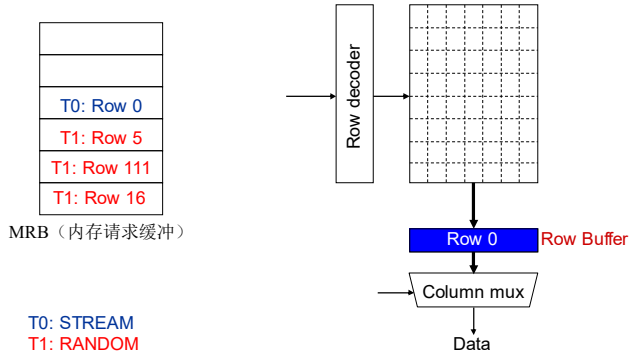

例子
T0: Row 0
T1: Row 5
T1: Row 111
T1: Row 16
MRB（内存请求缓冲）
Row decoder
Row 0
Row Buffer
Column mux
Data
T0: STREAM
T1: RANDOM
87

87


例子
T1: Row 5
T1: Row 111
T1: Row 16
MRB（内存请求缓冲）
Row decoder
Row 0
Row Buffer
T0: Row 0
Column mux
Data
T0: STREAM
T1: RANDOM
88

88

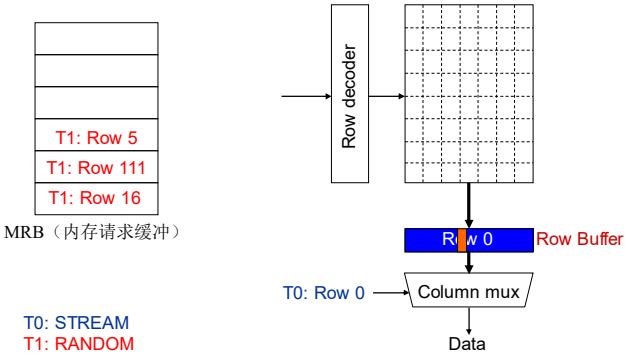
例子



89

89

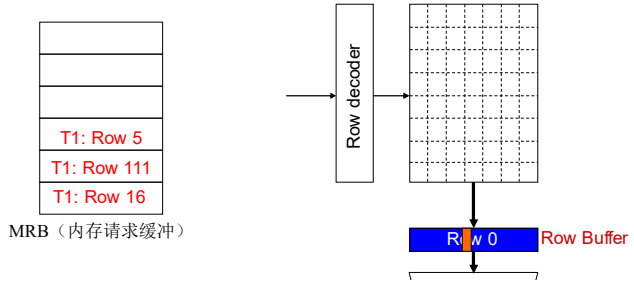
例子



90

90

例子



假如 Row: 8KB, cache line: 64B
T0可能有128次请求优先于T1的请求被服务

91

91

现在我们知道底层会发生什么

- 怎么解决这样的问题呢?

92

92

现在我们知道底层会发生什么

■怎么解决这样的问题呢?

■在什么位置解决呢?
- ❑程序员?
- ❑系统软件?
- ❑编译器?
- ❑硬件 (内存控制器)?
- ❑硬件 (DRAM)?
- ❑电路?

93

93

现在我们知道底层会发生什么

■怎么解决这样的问题呢?

■在什么位置解决呢?
- ❑程序员?
- ❑系统软件?
- ❑编译器?
- ❑硬件 (内存控制器)?
- ❑硬件 (DRAM)?
- ❑电路?

■本课程的另外两个目标
- ❑批判性的思考
- ❑全局性的思考

94

94

简要回顾: 课程目标

- 理解处理器如何工作
- 实现一个简单的处理器
- 理解硬件层面的决策如何影响软件/程序员以及硬件设计本身
- 解决问题时的批判性思维
- 在不同层次之间的全局性思维
- 理解在设计时如何分析并且做出折衷（tradeoff）

95

95

提纲

- 课程简介
  - 课程基本信息
  - 课程的主要内容
  - 关于实验、作业、阅读等
- 接下来两周的工作
  - 布置并提交作业0
  - 布置作业1
  - 布置第一次实验
  - 阅读
- 计算机体系结构的基本概念

96

96

课程信息: 我们

- 主讲教师: 栾钟治
  - luan.zhongzhi@buaa.edu.cn，rick710055@263.net
  - qq:9281971
  - 微信:rickluan
  - 办公室:新主楼G座413
  - 计算机学院中德联合软件研究所
  - 研究方向和讲授课程:
    - 分布式处理和分布式系统
    - 计算机组成和计算机体系结构
    - 并行计算和高性能计算
    - 计算机网络技术

97

97

课程信息: 我们

- 助讲教师
  - 杨海龙
    - hailong.yang@buaa.edu.cn
    - 办公室：新主楼G座823
- 助教
  - 孙庆骁
    - qingxiaosun@buaa.edu.cn
    - 新主楼G座413

98

98

你们

- 你是谁?
- 作业 0
  - 告诉我们你是谁
  - 截止提交日期：**3月13日, 上课前**
  - 提交到课程中心
  - 贴上你的照片

99

99

你们

- 你是谁?
- 作业 0
  - 告诉我们你是谁
  - 截止提交日期：**3月13日, 上课前**
  - 提交到课程中心
  - 贴上你的照片
- 后续的得分基于作业 0是否提交
  - P = (homework 0 turned in == true)
  - Grade = P ? ActualGrade : 0

100

100

在哪里获得不断更新的课程信息?

- 课程网站: http://course.buaa.edu.cn/
  - 讲义
  - 实验信息
  - 作业信息
  - 课程的计划、要求阅读的材料等等

  请务必经常浏览课程网站!
- 微信群
- qq群

101

101

课程的时间地点

- 课程:
  - 每周五 9:50-12:15
  - qq群/B101
  - 不点名。。。但是来听课对你掌握知识有好处!
  - 可能会根据情况安排实验内容和习题的讲解

102

102

初步的课程安排

- 初步的课程安排在课程网站的教学大纲和日程表中
  - 计算机体系结构基本概念
  - ISA设计和折衷的基本概念、原则和实现基础
  - 单周期、多周期和流水线微体系结构
  - 流水线的数据相关、控制相关、精确异常和动态调度/系统性能分析
  - 分层存储体系结构、Cache、主存储器、内存控制和隐藏访存延迟
  - 系统性能和能耗分析
  - 多处理器与多处理、互连和片上多核
- 但是……
- 运行最好的系统通常是动态调度的
  - 静态 vs. 动态调度
  - 编译时vs. 运行时

103

103

实验（计划中的Project）

- Project1：用Logisim设计1个7指令单周期MIPS CPU
  - 实验1：掌握处理器的基本原理、结构、构造方法
- Project2：学习Bluespec语言
  - 实验2：构建一个右移位器
  - 实验3：实现一个pipelined右移位器
- Project3：设计pipelined CPU
  - 实验4：学习SMIPS处理器代码，改造unpipelined的SMIPS处理器；改造为pipelined的SMIPS
- Project4：提升CPU性能
  - 实验5：进一步改造pipelined的SMIPS，并尽可能最大化性能
- 实验基本要求
  - 独立实验
  - 我们只提供实验环境、实验素材、实验内容
  - 你必须主动去学习大量内容

104

104

你们能学到什么

- **计算机体系结构: 通过硬件组件的设计、选择、互连以及软硬件接口的设计来创造计算系统的科学与艺术，它使得创造出的计算系统能够满足功能、性能、能耗、成本以及其他特定的目标。**
- **传统的定义:** “体系结构这个术语用来描述程序员所观察到的系统属性，也就是那些不同于数据流和控制流的组织、逻辑设计以及物理实现的概念性的结构和功能性的行为。” *Gene Amdahl*, IBM Journal of R&D, 1964年4月

105

105

你们能学到什么

- **计算机体系结构: 通过硬件组件的设计、选择、互连以及软硬件接口的设计来创造计算系统的科学与艺术，它使得创造出的计算系统能够满足功能、性能、能耗、成本以及其他特定的目标。**
- **传统的定义:** “体系结构这个术语用来描述程序员所观察到的系统属性，也就是那些不同于数据流和控制流的组织、逻辑设计以及物理实现的概念性的结构和功能性的行为。” *Gene Amdahl*, IBM Journal of R&D, 1964年4月

Dr. Amdahl holding a 100gate LSI air-cooled chip. On his desk is a circuit board with the chips on it. This circuit board was for an Amdahl 470 V/6 (photograph dated March 1975).

106

106

计算机体系结构在系统层次中的位置

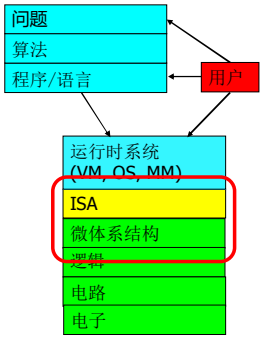
- 阅读：Patt, “Requirements, Bottlenecks, and Good Fortune: Agents for Microprocessor Evolution,” Proceedings of the IEEE 2001.

107

107

重新审视系统的层次结构

- 以用户为中心的视角: 为用户设计的计算机



- 整个层次结构都需要为用户需求来优化

108

108

你们能学到什么

- 设计软硬件接口和现代微处理器主要组件时的基本原则和方法
  - 关注一些新的研究和进展
  - 如何做出折衷

109

109

你们能学到什么

- 设计软硬件接口和现代微处理器主要组件时的基本原则和方法
  - 关注一些新的研究和进展
  - 如何做出折衷
- 如何设计、实现和评价一个功能性的现代处理器
  - 通过实验
  - RTL级别的实现和更高层模拟相结合
  - 关注功能性(个别关注 “如何可以更好”)

110

110

你们能学到什么

- 设计软硬件接口和现代微处理器主要组件时的基本原则和方法
  - 关注一些新的研究和进展
  - 如何做出折衷
- 如何设计、实现和评价一个功能性的现代处理器
  - 通过实验
  - RTL级别的实现和更高层模拟相结合
  - 关注功能性(个别关注 “如何可以更好”)
- 如何查找信息, 如何思考
- 如何更加努力的工作!

111

111

课程目标

- 目标 1: 熟悉计算机系统（处理器、内存、平台架构）设计的基本操作原则和折衷方法。
  - 重视基础概念和折衷的设计
- 目标 2: 通过亲自动手实验（RTL级和C语言级的实现）获得现代处理器设计、实现和评价的必要经验。
  - 重视功能性和实验

112

112

关于软件和硬件

- 本课程的分类*大约*是在“计算机硬件”之下
- 但是，如果你能够同时掌握软件和硬件（包括两者的接口）
  - 理解底层硬件能帮助你开发更好的软件
  - 理解软件的执行将使你能更好的设计硬件
  - 两者都理解使你能够设计更好的计算系统
- 本课程涵盖软硬件接口以及微体系结构
  - 我们将关注设计时的折衷以及它是如何影响软件的

113

113

对各位的希望

- 需要的背景知识: (数字逻辑, RTL 实现, Verilog, 计算机组成原理, 汇编语言)
- 好好学习
  - 来上课（非强制）, 阅读布置的材料, 做作业
- 努力工作
  - 问问题、记笔记、参与！
- **来上课请不要迟到！**
- 提早开始，按时完成，不拖延
- 欢迎来办公室骚扰

“Chance favors the prepared mind.” (Pasteur)

114

114

对各位的希望

- 需要的背景知识: (数字逻辑, RTL 实现, Verilog, 计算机组成原理, 汇编语言)
- 好好学习
  - 来上课（非强制）, 阅读布置的材料, 做作业
- 努力工作
  - 问问题、记笔记、参与！
- **来上课请不要迟到！**
- 提早开始，按时完成，不拖延
- 欢迎来办公室骚扰

“Chance favors the prepared mind.” (Pasteur)

115

115

对各位的希望

- 时间的管理非常重要
- 几乎每周都会布置一些任务
  - 计划5次实验和6次左右作业
- 是一门重课！

116

116

最终成绩的构成

- 作业: 20%
- 实验: 50%
- 期末考试: 25%
- 印象分: 5%
  - 参与程度
  - 阅读的情况
  - ……

117

117

关于作业和实验的补充

- 作业
  - 做作业是为了更好地理解而不是为了分数
  - 内容是关于课堂知识、阅读材料、实验和讨论的
  - 所有交来的作业***必须是你自己完成***的
    - 可以和其他人讨论
  - 迟交的作业一律算作未交

118

118

关于作业和实验的补充

- 作业
  - 做作业是为了更好地理解而不是为了分数
  - 内容是关于课堂知识、阅读材料、实验和讨论的
  - 所有交来的作业***必须是你自己完成***的
    - 可以和其他人讨论
  - 迟交的作业一律算作未交
- 实验
  - 实验会占用不少时间，尤其是一开始
  - 需要早一点动手并且努力
  - ***必须自己完成***除非有特别的安排
  - 最后可能会根据情况酌情推迟实验结果的提交但是不会超过一周时间

119

119

关于作弊和学术欺骗

- 对作弊零容忍
  - 长得一样的作业、考卷和相关实验数据，不区分抄与被抄，均按0分对待
  - 没有例外
  - 可能还有更严重的后果
- 大家已经是成年人，我们会按成年人对待大家

120

120

作业和实验预告

- 作业 0
  - 已发布在课程网站，截止时间下周五(3月13日) 上课前
- 作业 1
  - 下周布置，截止提交时间3月27日, 上课前提交,内容会发到课程网站
  - MIPS 热身, ISA 的概念, 基本的性能评价
- 实验1
  - 下周布置，预计完成截止时间为4月10日
  - 需要提前启动，因为你们要学习很多东西

121

121

阅读材料布置

- Patt, "Requirements, Bottlenecks, and Good Fortune: Agents for Microprocessor Evolution," Proceedings of the IEEE 2001.
- Patt & Patel's *Introduction to Computing Systems: From Bits and Gates to C and Beyond*（计算机系统概论）第一章 (基础)
- Patterson & Hennessy's *Computer Organization and Design: The Hardware/Software Interface*（计算机组成与设计：软硬件接口）第一章和第二章 (概述, 抽象, ISA, MIPS)
- 其他相关材料
  - MIPS 手册

122

122

关于教材

- 没有要求
- 但是希望大家能够找一些书来阅读
  - David Patterson & John Hennessy, 《计算机组成与设计：软硬件接口》第4版
  - Yale Patt & Sanjay Patel, 《计算机系统概论》第2版
  - David Harris & Sarah Harris, 《数字设计和计算机体系结构》第2版
  - ……

123

123

基本概念和ISA

124

为什么要学习计算机体系结构?

125

125

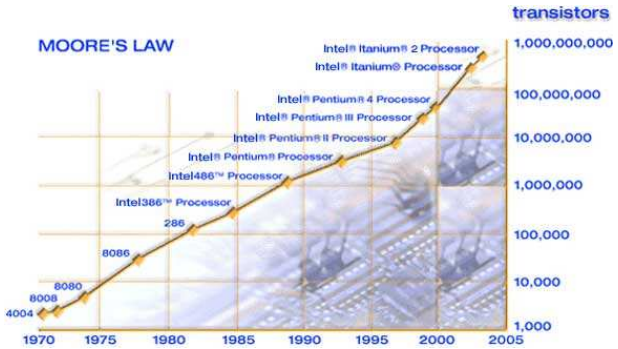
什么是计算机体系结构?

- **通过硬件组件的设计、选择、互连以及软硬件接口的设计来创造计算系统的科学与艺术，它使得创造出的计算系统能够满足功能、性能、能耗、成本以及其他特定目标的要求。**
- 你们很快将会看到体系结构和微体系结构之间的区别

126

126

使能: Moore's Law



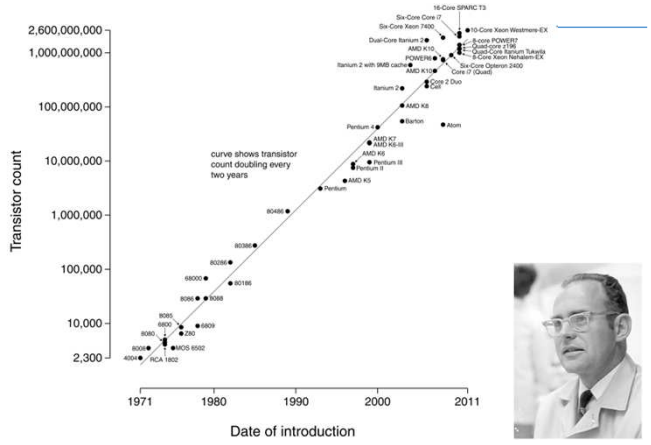
Moore, "Cramming more components onto integrated circuits," Electronics Magazine, 1965. 组件数量每隔一年翻一番

Image source: Intel

127

127

Microprocessor Transistor Counts 1971-2011 & Moore's Law



集成电路上晶体管的数量大约每两年翻一番

Image source: Wikipedia

128

128

我们用这些晶体管来做什么

- 请大家在课下的阅读中思考…
  - Patt, "Requirements, Bottlenecks, and Good Fortune: Agents for Microprocessor Evolution," Proceedings of the IEEE 2001.

129

129

为什么要学习计算机体系结构?

- 更好的系统: 使计算机更快、更便宜、更小、更可靠…
  - 通过利用底层技术的进步
- 更新的应用
  - 三维可视化
  - 虚拟现实
  - 个人基因组
- 更好的解决问题
  - 软件的创新与计算机体系结构的发展和改进紧密融合
    - 每年超过一半的硬件性能提升支撑了软件的创新
- 理解计算机为什么会这样工作

130

130

今天的计算机体系结构(I)

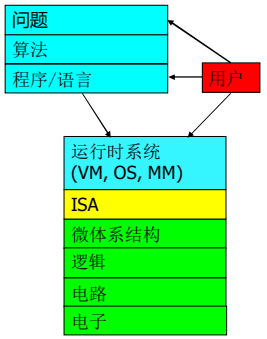
- 计算机工业进入一个大的转变时期:**多核-更多的核**
  - 很多潜在的不同的系统设计可能性
- 很多困难的问题***驱动了***这种转变同时也***由于***这些转变变得更困难
  - 功耗/能耗约束
  - 设计的复杂性→ 多核?
  - 技术扩展的困难→ 新的技术?
  - 存储墙(wall/gap)
  - 可靠性墙/问题(wall/issues)
  - 可编程性墙/问题(wall/problem)
- 这些问题都没有清晰、确定的答案

131

131

今天的计算机体系结构(II)

- 这些问题影响着计算系统的各个层次– 如果我们不改变我们设计系统的方式



132

132

今天的计算机体系结构(III)

- 如果你理解软件和硬件并且对它们作相应的改变，你可能革新构建计算机的方法
- 你可以发明新的计算、通信和存储模式

133

133

今天的计算机体系结构(III)

- 如果你理解软件和硬件并且对它们作相应的改变，你可能革新构建计算机的方法
- 你可以发明新的计算、通信和存储模式
- 推荐一本书: Kuhn, "The Structure of Scientific Revolutions" (1962)
  - 前范式阶段
  - 常规阶段
  - 反常阶段
  - 危机阶段
  - 革命阶段

134

134

… 但是, 首先 …

- 让我们来理解基本的概念…
- 只有当你理解得足够好之后才有可能改变…
  - 尤其是那些过去和现在起主导地位的范式
  - 同时，它们的优点和缺点 -- tradeoffs

135

135