中山大學
SUN YAT-SEN UNIVERSITY

国家超级计算广州中心
NATIONAL SUPERCOMPUTER CENTER IN GUANGZHOU

多发射技术与并行限制

主讲教师：胡 淼

中山大学计算机学院

2024 年 秋季

中山大學
SUN YAT-SEN UNIVERSITY

- 第 1 章 绪论
- 第 2 章 基准评测集
- 第 3 章 并行计算机的体系结构
- **第 4 章 高性能处理器的并行计算技术**
- 第 5 章 并行计算机的存储层次
- 第 6 章 并行计算机的互连网络
- 第 7 章 异构计算体系结构
- 第 8 章 领域专用体系结构

第 4 章 高性能处理器的并行计算技术

4.1 指令级并行

part3 多发射技术与指令集并行的限制

发掘指令级并行的目的

- 降低程序执行的平均CPI

Pipeline CPI = Ideal pipeline CPI
+ Structural stalls
+ Data hazard stalls
+ Control stalls

参考资料

- Computer Architecture, A Quantitative Approach, 6th Edition.
  - Appendix C. “Pipelining: Basic and Intermediate Concepts”
  - Chapter 3. “Instruction-Level Parallelism and Its Exploitation”
    - 3.7 Exploiting ILP Using Multiple Issue and Static Scheduling

内容纲要

- 超标量
  - 一个时钟发射多条指令
    - 当前处理器均采用的方法
- 超长指令字（VLIW, Very Long instruction Word）
  - 编译器将多条不相关的指令打包成一个“指令包”
- 同时多线程
- 指令集并行的限制因素

内容纲要

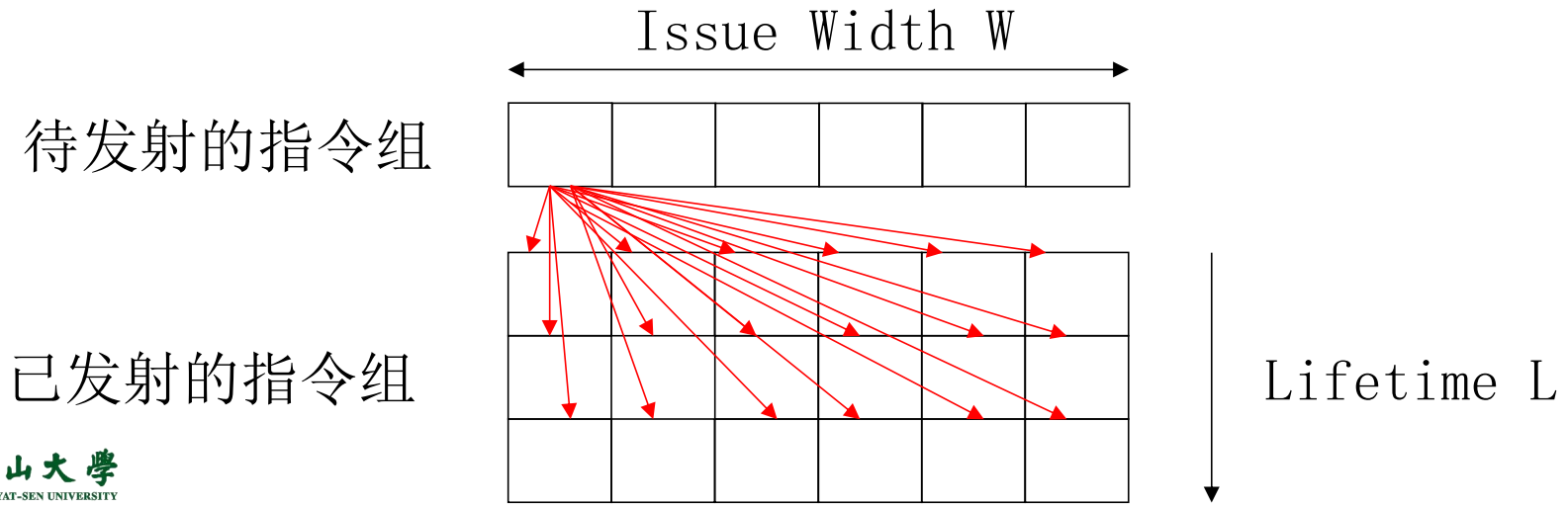
- 超标量
  - 一个时钟发射多条指令
    - 当前处理器均采用的方法
- 超长指令字（VLIW, Very Long instruction Word）
  - 编译器将多条不相关的指令打包成一个“指令包”
- 同时多线程
- 指令集并行的限制因素

超标量的基本原理

- 指令按需发射
  - 每个时钟尝试发射多条指令
- 发射指令时检测所有的冒险
  - 检测正在发射的多条指令之间的冒险
  - 检测正在发射的多条指令与正在流水线中执行指令之间的冒险
- 数据通路中大量的部件冗余
  - 多个浮点单元
  - 多个定点单元
  - 多个访存单元

发射阶段的冒险检测

- 假定每个时钟发射W条指令，流水线中L组指令在执行
  - 需要与W*L条指令做检测
  - 硬件的复杂度为W*（W*L）
- L与正在执行的指令数有关
  - 在按序执行的处理器中，L与流水线的深度有关
  - 在乱序执行的处理器中，L还与指令在ROB中驻留的时间有关
- W增加，需要在更大的指令窗口内寻找可发射的指令
  - 甚至可能导致L的增加，因为ROB可能变长



MIPS结构下典型的二发射

- 定点单元
  - 一个Load、Store
  - 或者一个定点ALU
  - 或者一个分支指令
- 浮点单元
  - 一个浮点的ALU
- 在取指阶段，从指令Cache取出多条指令
  - 判定能发射0条、1条、2条？

	1	2	3	4	5	6	7	8
Integer	IF	ID	EX	MEM	WB			
FP	IF	ID	EX	EX	EX	WB		
Integer		IF	ID	EX	MEM	WB		
FP		IF	ID	EX	EX	EX	WB	
Integer			IF	ID	EX	MEM	WB	
FP			IF	ID	EX	EX	EX	WB
Integer				IF	ID	EX	MEM	WB
FP				IF	ID	EX	EX	EX

浮点数可能执行时间较长

	1	2	3	4	5	6	7	8
Integer	IF	ID	EX	MEM	WB			
FP	IF	ID	EX	EX	EX	WB		
Integer		IF	ID	EX	MEM	WB		
FP		IF	ID	EX	EX	EX	WB	
Integer			IF	ID	EX	MEM	WB	
FP			IF	ID	EX	EX	EX	WB
Integer				IF	ID	EX	MEM	WB
FP				IF	ID	EX	EX	EX

超标量的挑战

- 指令发射的复杂度显著提高
- 将一个时钟内的2条指令发射看作流水线
  - 半个时钟发射一条指令
  - 2条指令在一个时钟发射完
- 增加发射单元的逻辑量
  - 设计能够同时发射2条指令的逻辑单元
- 现代超标量处理器
  - 一个时钟一般能够发射4条指令
  - 以上两种方法同时用到

超标量的挑战

- 需要大量的Forward
  - 保证RAW相关能尽快得到结果
- 浮点运算的瓶颈
  - 包含多个浮点单元，每个浮点单元都是流水线
  - Load-Use停顿可能会影响3条指令
    - 同周期发射的指令，和下一周期发射的2条指令

超标量的Tomasulo算法

- 一些基本假设
  - 每个时钟能发射**一条定点和一条浮点指令**
  - 分支预测完全准确
  - 延迟假设
    - 定点ALU: 1时钟
    - Load指令: 2时钟
    - 浮点ALU: 3时钟

```
Loop:   L.D     F0,0(R1)
        ADD.D   F4,F0,F2
        S.D     F4,0(R1)
        DADDIU  R1,R1,#-8
        BNE     R1,R2,Loop
```

双发射超标量的Tomasulo算法

Iter. #	Instructions	Issues	Exec.	Mem. access	Write CDB	Comment
1	L.D F0,0(R1)	1				First issue
1	ADD.D F4,F0,F2					
1	S.D F4,0(R1)					
1	DADDIU R1,R1,#-8					
1	BNE R1,R2,Loop					

双发射超标量的Tomasulo算法

Iter. #	Instructions	Issues	Exec.	Mem. access	Write CDB	Comment
1	L.D F0,0(R1)	1	2	3	4	First issue
1	ADD.D F4,F0,F2					
1	S.D F4,0(R1)					
1	DADDIU R1,R1,#-8					
1	BNE R1,R2,Loop					

双发射超标量的Tomasulo算法

Iter. #	Instructions	Issues	Exec.	Mem. access	Write CDB	Comment
1	L.D F0,0(R1)	1	2	3	4	First issue
1	ADD.D F4,F0,F2	1				
1	S.D F4,0(R1)					
1	DADDIU R1,R1,#-8					
1	BNE R1,R2,Loop					

双发射超标量的Tomasulo算法

Iter. #	Instructions	Issues	Exec.	Mem. access	Write CDB	Comment
1	L.D F0,0(R1)	1	2	3	4	First issue
1	ADD.D F4,F0,F2	1	5			Wait for L.D
1	S.D F4,0(R1)					
1	DADDIU R1,R1,#-8					
1	BNE R1,R2,Loop					

双发射超标量的Tomasulo算法

Iter. #	Instructions	Issues	Exec.	Mem. access	Write CDB	Comment
1	L.D F0,0(R1)	1	2	3	4	First issue
1	ADD.D F4,F0,F2	1	5		8	3 cycles letncy for ADD.D
1	S.D F4,0(R1)					
1	DADDIU R1,R1,#-8					
1	BNE R1,R2,Loop					

双发射超标量的Tomasulo算法

Iter. #	Instructions	Issues	Exec.	Mem. access	Write CDB	Comment
1	L.D F0,0(R1)	1	2	3	4	First issue
1	ADD.D F4,F0,F2	1	5		8	Wait for L.D
1	S.D F4,0(R1)	2				
1	DADDIU R1,R1,#-8	2				
1	BNE R1,R2,Loop					

双发射超标量的Tomasulo算法

Iter. #	Instructions	Issues	Exec.	Mem. access	Write CDB	Comment
1	L.D F0,0(R1)	1	2	3	4	First issue
1	ADD.D F4,F0,F2	1	5		8	Wait for L.D
1	S.D F4,0(R1)	2	3			
1	DADDIU R1,R1,#-8	2	4			Wait for ALU
1	BNE R1,R2,Loop					

双发射超标量的Tomasulo算法

Iter. #	Instructions	Issues	Exec.	Mem. access	Write CDB	Comment
1	L.D F0,0(R1)	1	2	3	4	First issue
1	ADD.D F4,F0,F2	1	5		8	Wait for L.D
1	S.D F4,0(R1)	2	3	9		Wait for ADD.D
1	DADDIU R1,R1,#-8	2	4		5	1 cycle
1	BNE R1,R2,Loop	3				

双发射超标量的Tomasulo算法

Iter. #	Instructions	Issues	Exec.	Mem. access	Write CDB	Comment
1	L.D F0,0(R1)	1	2	3	4	First issue
1	ADD.D F4,F0,F2	1	5		8	Wait for L.D
1	S.D F4,0(R1)	2	3	9		Wait for ADD.D
1	DADDIU R1,R1,#-8	2	4		5	Wait for ALU
1	BNE R1,R2,Loop	3	6			Wait for DADDIU

双发射超标量的Tomasulo算法

Iter. #	Instructions	Issues	Exec.	Mem. access	Write CDB	Comment
1	L.D F0,0(R1)	1	2	3	4	First issue
1	ADD.D F4,F0,F2	1	5		8	Wait for L.D
1	S.D F4,0(R1)	2	3	9		Wait for ADD.D
1	DADDIU R1,R1,#-8	2	4		5	Wait for ALU
1	BNE R1,R2,Loop	3	6			Wait for DADDIU
2	L.D F0,0(R1)					
2	ADD.D F4,F0,F2					
2	S.D F4,0(R1)					
2	DADDIU R1,R1,#-8					
2	BNE R1,R2,Loop					

双发射超标量的Tomasulo算法

Iter. #	Instructions	Issues	Exec.	Mem. access	Write CDB	Comment
1	L.D F0,0(R1)	1	2	3	4	First issue
1	ADD.D F4,F0,F2	1	5		8	Wait for L.D
1	S.D F4,0(R1)	2	3	9		Wait for ADD.D
1	DADDIU R1,R1,#-8	2	4		5	Wait for ALU
1	BNE R1,R2,Loop	3	6			Wait for DADDIU
2	L.D F0,0(R1)	4				Wait for BNE
2	ADD.D F4,F0,F2	4				
2	S.D F4,0(R1)					
2	DADDIU R1,R1,#-8					
2	BNE R1,R2,Loop					

双发射超标量的Tomasulo算法

Iter. #	Instructions	Issues	Exec.	Mem. access	Write CDB	Comment
1	L.D F0,0(R1)	1	2	3	4	First issue
1	ADD.D F4,F0,F2	1	5		8	Wait for L.D
1	S.D F4,0(R1)	2	3	9		Wait for ADD.D
1	DADDIU R1,R1,#-8	2	4		5	Wait for ALU
1	BNE R1,R2,Loop	3	6			Wait for DADDIU
2	L.D F0,0(R1)	4	7	8	9	Wait for BNE
2	ADD.D F4,F0,F2	4				
2	S.D F4,0(R1)					
2	DADDIU R1,R1,#-8					
2	BNE R1,R2,Loop					

双发射超标量的Tomasulo算法

Iter. #	Instructions	Issues	Exec.	Mem. access	Write CDB	Comment
1	L.D F0,0(R1)	1	2	3	4	First issue
1	ADD.D F4,F0,F2	1	5		8	Wait for L.D
1	S.D F4,0(R1)	2	3	9		Wait for ADD.D
1	DADDIU R1,R1,#-8	2	4		5	Wait for ALU
1	BNE R1,R2,Loop	3	6			Wait for DADDIU
2	L.D F0,0(R1)	4	7	8	9	Wait for BNE
2	ADD.D F4,F0,F2	4	10		13	Wait for L.D
2	S.D F4,0(R1)	5				
2	DADDIU R1,R1,#-8	5				
2	BNE R1,R2,Loop	6				

双发射超标量的Tomasulo算法

Iter. #	Instructions	Issues	Exec.	Mem. access	Write CDB	Comment
1	L.D F0,0(R1)	1	2	3	4	First issue
1	ADD.D F4,F0,F2	1	5		8	Wait for L.D
1	S.D F4,0(R1)	2	3	9		Wait for ADD.D
1	DADDIU R1,R1,#-8	2	4		5	Wait for ALU
1	BNE R1,R2,Loop	3	6			Wait for DADDIU
2	L.D F0,0(R1)	4	7	8	9	Wait for BNE
2	ADD.D F4,F0,F2	4	10		13	Wait for L.D
2	S.D F4,0(R1)	5	8			Wait for ALU for A
2	DADDIU R1,R1,#-8	5				
2	BNE R1,R2,Loop	6				

双发射超标量的Tomasulo算法

Iter. #	Instructions	Issues	Exec.	Mem. access	Write CDB	Comment
1	L.D F0,0(R1)	1	2	3	4	First issue
1	ADD.D F4,F0,F2	1	5		8	Wait for L.D
1	S.D F4,0(R1)	2	3	9		Wait for ADD.D
1	DADDIU R1,R1,#-8	2	4		5	Wait for ALU
1	BNE R1,R2,Loop	3	6			Wait for DADDIU
2	L.D F0,0(R1)	4	7	8	9	Wait for BNE
2	ADD.D F4,F0,F2	4	10		13	Wait for L.D
2	S.D F4,0(R1)	5	8	14		Wait for ADD.D
2	DADDIU R1,R1,#-8	5				
2	BNE R1,R2,Loop	6				

双发射超标量的Tomasulo算法

Iter. #	Instructions	Issues	Exec.	Mem. access	Write CDB	Comment
1	L.D F0,0(R1)	1	2	3	4	First issue
1	ADD.D F4,F0,F2	1	5		8	Wait for L.D
1	S.D F4,0(R1)	2	3	9		Wait for ADD.D
1	DADDIU R1,R1,#-8	2	4		5	Wait for ALU
1	BNE R1,R2,Loop	3	6			Wait for DADDIU
2	L.D F0,0(R1)	4	7	8	9	Wait for BNE
2	ADD.D F4,F0,F2	4	10		13	Wait for L.D
2	S.D F4,0(R1)	5	8	14		Wait for ADD.D
2	DADDIU R1,R1,#-8	5	9		10	Wait for L.D
2	BNE R1,R2,Loop	6	11			Wait for DADDIU

双发射超标量的Tomasulo算法

Iter. #	Instructions	Issues	Exec.	Mem. access	Write CDB	Comment
1	L.D F0,0(R1)	1	2	3	4	First issue
1	ADD.D F4,F0,F2	1	5		8	Wait for L.D
1	S.D F4,0(R1)	2	3	9		Wait for ADD.D
1	DADDIU R1,R1,#-8	2	4		5	Wait for ALU
1	BNE R1,R2,Loop	3	6			Wait for DADDIU
2	L.D F0,0(R1)	4	7	8	9	Wait for BNE
2	ADD.D F4,F0,F2	4	10		13	Wait for L.D
2	S.D F4,0(R1)	5	8	14		Wait for ADD.D
2	DADDIU R1,R1,#-8	5	9		10	Wait for ALU
2	BNE R1,R2,Loop	6	11			Wait for DADDIU
3	L.D F0,0(R1)	7	12	13	14	Wait for BNE
3	ADD.D F4,F0,F2	7	15		18	Wait for L.D
3	S.D F4,0(R1)	8	13	19		Wait for ADD.D
3	DADDIU R1,R1,#-8	8	14		15	Wait for ALU
3	BNE R1,R2,Loop	9	16			Wait for DADDIU

双发射超标量的Tomasulo算法

- 每3个时钟完成一次循环
  - IPC为5/3 = 1.67

Clock	Integer ALU	FP ALU	Data Cache	CDB
1				
2	1/ L.D			
3	1 / S.D		1/ L.D	
4	1 / DADDIU			1/ L.D
5		1 / ADD.D		1 / DADDIU
6				
7	2 / L.D			
8	2 / S.D		2 / L.D	1 / ADD.D
9	2 / DADDIU		1 / S.D	2 / L.D
10		2 / ADD.D		2 / DADDIU
11				
12	3 / L.D			
13	3 / S.D		3 / L.D	2 / ADD.D
14	3 / DADDIU		2 / S.D	3 / L.D
15		3 / ADD.D		3 / DADDIU
16				
17				
18				3 / ADD.D
19			3 / S.D	

功能单元使用情况

内容纲要

- 超标量
  - 一个时钟发射多条指令
    - 当前处理器均采用的方法
- 超长指令字（VLIW, Very Long instruction Word）
  - 编译器将多条不相关的指令打包成一个“指令包”
- 同时多线程
- 指令集并行的限制因素

VLIW的基本原理

- 依赖**编译器**发现指令级并行
  - 硬件不负责检测指令之间的冒险，硬件设计简单
- 将多条**不相关的指令**打包成一个指令包
  - 里面每条指令占用一个独立的功能单元
- 每个指令包用一条“超长”的指令表示：112到168位
  - 一个指令包可能包括：
    - 2个整数运算
    - 2个浮点运算
    - 2个访存操作
    - 1个分支指令
  - 每个功能单元对应的指令包含多个field，长度约16到24位

VLIW的基本原理

- 编译器的任务
  - 在每个超长指令中，尽可能让每个功能单元上有一个操作
  - 如果找不到合适的操作，则填充 NOP
- 硬件的任务
  - 将每个操作的 field 发送到对应的功能单元上
- 对不同的硬件平台，需要重新编译源代码
  - 同一指令集体系结构，但功能单元的设置可能不同
    - X个整数运算
    - X个浮点运算
    - X个访存操作
    - X个分支指令

早期VLIW机器

- FPS AP120B (1976)
  - scientific attached array processor
  - first commercial wide instruction machine
  - hand-coded vector math libraries using software pipelining and loop unrolling
- Multiflow Trace (1987)
  - commercialization of ideas from Fisher's Yale group
  - including "trace scheduling"
  - available in configurations with 7, 14, or 28 operations/instruction
  - 28 operations packed into a 1024-bit instruction word
- Cydrome Cydra-5 (1987)
  - 7 operations encoded in 256-bit instruction word
  - rotating register file

一个VLIW的例子

- 将一个循坏调度到VLIW机器上执行，该机器可同时发射
  - 2个访存操作
  - 2个浮点操作
  - 一个定点操作或分支指令
- 循环的代码
  - Loop x[i] = x[i] + s

```
Loop:   L.D     F0,0(R1)
        ADD.D   F4,F0,F2
        S.D     F4,0(R1)
        DADDUI  R1,R1,#-8
        BNE     R1,R2,Loop
```

一个VLIW的例子

Mem Ref1	Mem Ref2	FP1	FP2	Int/branch
L.D F0,0(R1)	L.D F6,-8(R1)			
L.D F10,-16(R1)	L.D F14,-24(R1)			
L.D F18,-32(R1)	L.D F22,-40(R1)	ADD.D F4,F0,F2	ADD.D F8,F6,F2	
L.D F26,-48(R1)		ADD.D F12,F10,F2	ADD.D F16,F14,F2	
		ADD.D F20,F18,F2	ADD.D F24,F22,F2	
S.D F4,0(R1)	S.D F8,-8(R1)	ADD.D F28,F26,F2		
S.D F12,-16(R1)	S.D F16,-24(R1)			DADDUI R1,R1,#-56
S.D F20,24(R1)	S.D F24,16(R1)			
S.D F28,8(R1)				BNE R1,R2,Loop

- 9个时钟得到7个结果，每1.29个时钟得到一个结果

- 9个时钟完成23条原始指令
  - 资源利用率23/45=51%

一个VLIW的例子

- 基本的MIPS指令执行
  - 一个时刻执行1条指令
- 二发射的超标量处理器
  - 每个时钟发射2条指令，5段流水
- VLIW
  - 同时执行7条基本指令
  - 需要更多的寄存器！

不同VLIW的性能差异

- 最消极的做法
  - 通过循环展开保证每个时钟有指令发射即可
  - 8个时钟得到3个结果，平均2.66个时钟一个结果
  - 功能单元利用率：11/40 = 27.5%

Mem Ref1	Mem Ref2	FP1	FP2	Int/branch
L.D F0,0(R1)	L.D F6,-8(R1)			
L.D F10,-16(R1)				
		ADD.D F4,F0,F2	ADD.D F8,F6,F2	
		ADD.D F12,F10,F2		
				DADDUI R1,R1,#-24
S.D F4,24(R1)	S.D F8,16(R1)			
S.D F12,8(R1)				
				BNE R1,R2,Loop

不同VLIW的性能差异

- 最积极的做法
  - 通过循环展开最大程度的利用浮点单元
  - 10个时钟得到10个结果
  - 功能单元利用率：36/50 =72%

Mem Ref1	Mem Ref2	FP1	FP2	Int/branch
L.D F0,0(R1)	L.D F6,-8(R1)			
L.D F10,-16(R1)	L.D F14,-24(R1)			
L.D F18,-32(R1)	L.D F22,-40(R1)	ADD.D F4,F0,F2	ADD.D F8,F6,F2	
L.D F26,-48(R1)	L.D F30,-56(R1)	ADD.D F12,F10,F2	ADD.D F16,F14,F2	
L.D F34,-64(R1)	L.D F38,-72(R1)	ADD.D F20,F18,F2	ADD.D F24,F22,F2	DADDUI R1,R1,#-80
S.D F4,80(R1)	S.D F8,72(R1)	ADD.D F28,F26,F2	ADD.D F32,F30,F2	
S.D F12,64(R1)	S.D F16,56(R1)	ADD.D F36,F34,F2	ADD.D F40,F38,F2	
S.D F20,48(R1)	S.D F20,40(R1)			
S.D F20,32(R1)	S.D F20,24(R1)			
S.D F20,16(R1)	S.D F20,8(R1)			BNE R1,R2,Loop

VLIW的缺陷

- **代码容量显著增长**
  - 特别依赖循环展开
  - 大量的功能单元需要用NOP填充
- **Lock-step的限制**
  - 一个发射包里面的指令要一起执行完毕
  - 但是包里同时有定点和浮点指令
  - 执行时间以最长的指令为准
  - 任何功能单元（FU）的停顿导致整个处理器停顿
    - 访存操作最容易导致停顿，且无法预测
- **代码的可移植性**
  - 在新的硬件下需要重新编译源代码
  - 代码取决于
    - 指令集、流水线结构、各个功能单元的延迟
  - 不同的流水线结构和功能单元延迟需要不同的代码

一些解决方案

- **针对代码容量显著增长**
  - 在内存中压缩代码，在Cache、或者译码时解压
    - 大量的功能单元用NOP填充，所以压缩比很高
- **针对Lock-step的限制**
  - 放松Lock-step的限制，允许动态调度
    - 编译器只负责找出不相关的指令，生成指令包，发射到功能单元
    - 同一个指令包内的所有操作独立执行，无需同步
  - 动态的VLIW
- **针对代码的可移植性**
  - Object-Code translation

内容纲要

- 超标量
  - 一个时钟发射多条指令
    - 当前处理器均采用的方法
- 超长指令字（VLIW, Very Long instruction Word）
  - 编译器将多条不相关的指令打包成一个“指令包”
- **同时多线程**

多线程技术

- 每个线程具有独立的：
  - 硬件方法
  - 指令、数据、PC、寄存器、页表
  - 它们共享功能单元
  - 多个线程交替在功能单元上执行

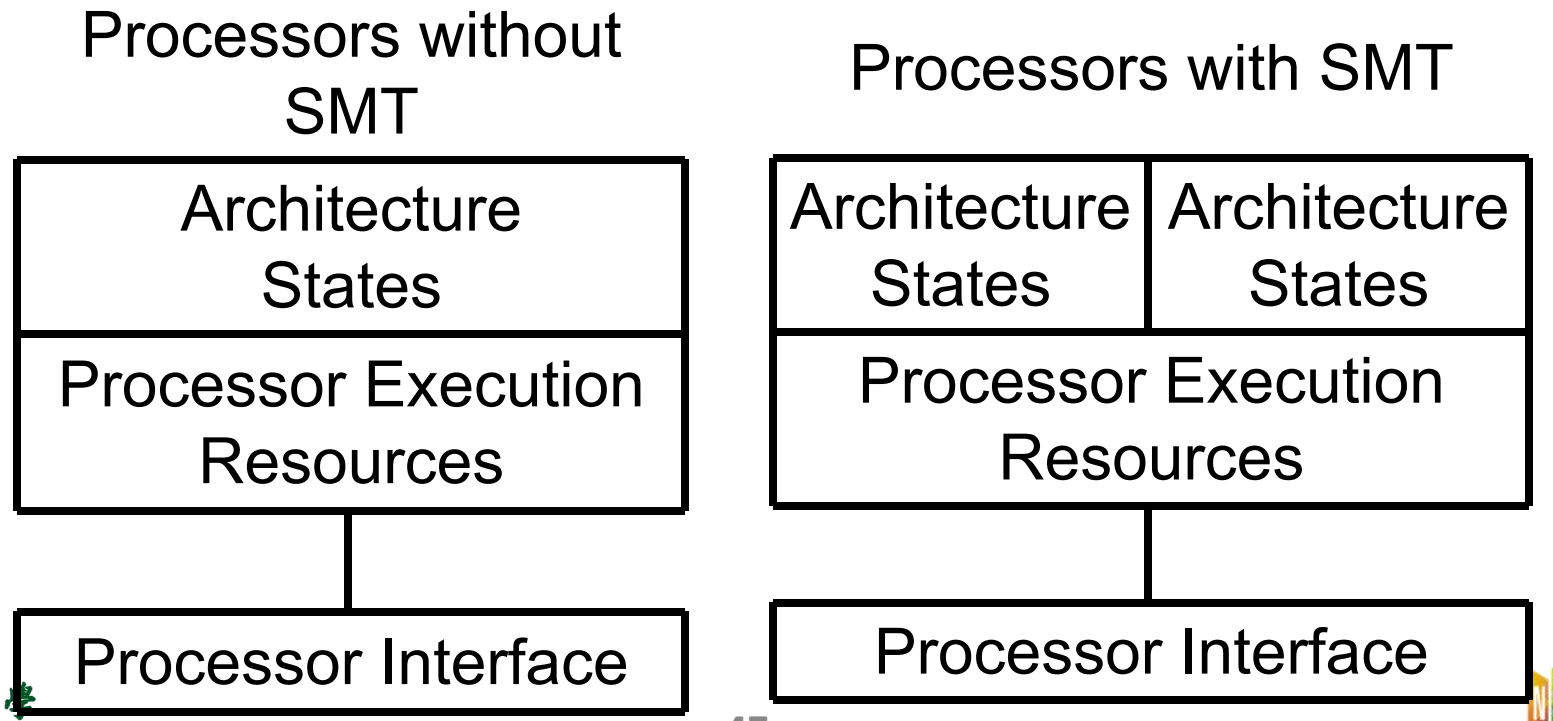
- 线程级并行（TLP）
  - 软件方法
  - 一个程序划分为多个线程
  - 更粗粒度的并行
  - 主要用于提升含有多线程程序的执行速度

多线程技术分类

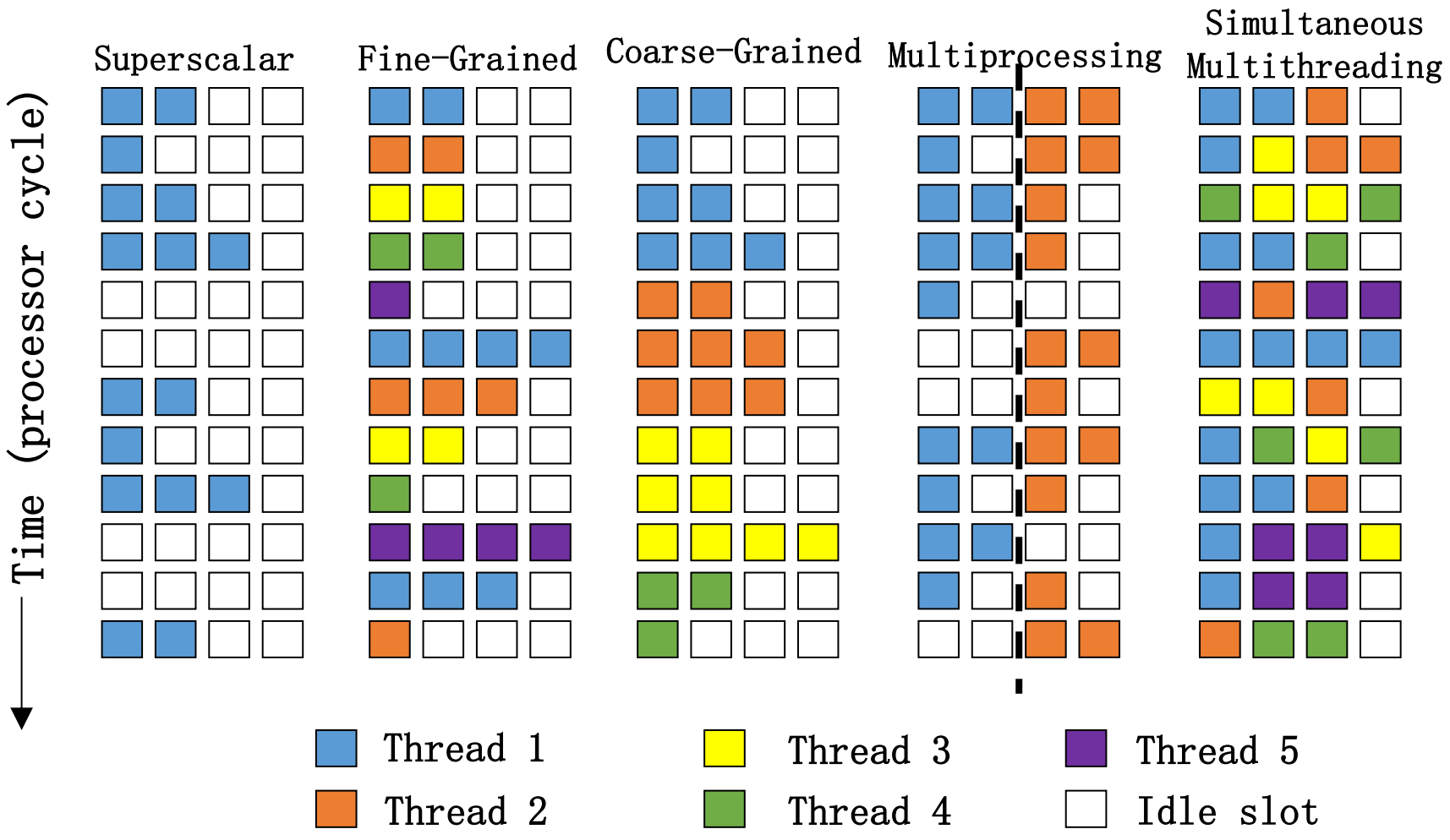
- **细粒度的多线程**
  - 每个时钟发生一次线程切换，多个线程交替执行
    - 发生停顿的线程跳过
  - 优点：功能单元的利用率高
  - 缺点：单个线程的执行时间变长
- **粗粒度的多线程**
  - 当一个线程发生成本较高的流水线停顿时，发生切换
    - 如L2或者L3 Cache缺失
  - 优点：单个线程的执行时间不会显著变长
  - 缺点：功能单元的利用率不高
    - 一个线程发生代价不高的停顿时，不会发生切换，存在气泡
    - 线程切换时，流水线启动时存在气泡
- **同时多线程 Simultaneous Multithreading (SMT)**
  - 细粒度的变种

SMT

- 同时多线程
  - 设置大量的虚拟寄存器，每个线程都有一组寄存器
    - 不同线程之间是独立的
  - 通过寄存器重命名和动态调度实现细粒度的多线程
    - 不同线程的指令之间没有相关性
  - 往往与超标量、动态调度等技术联合使用

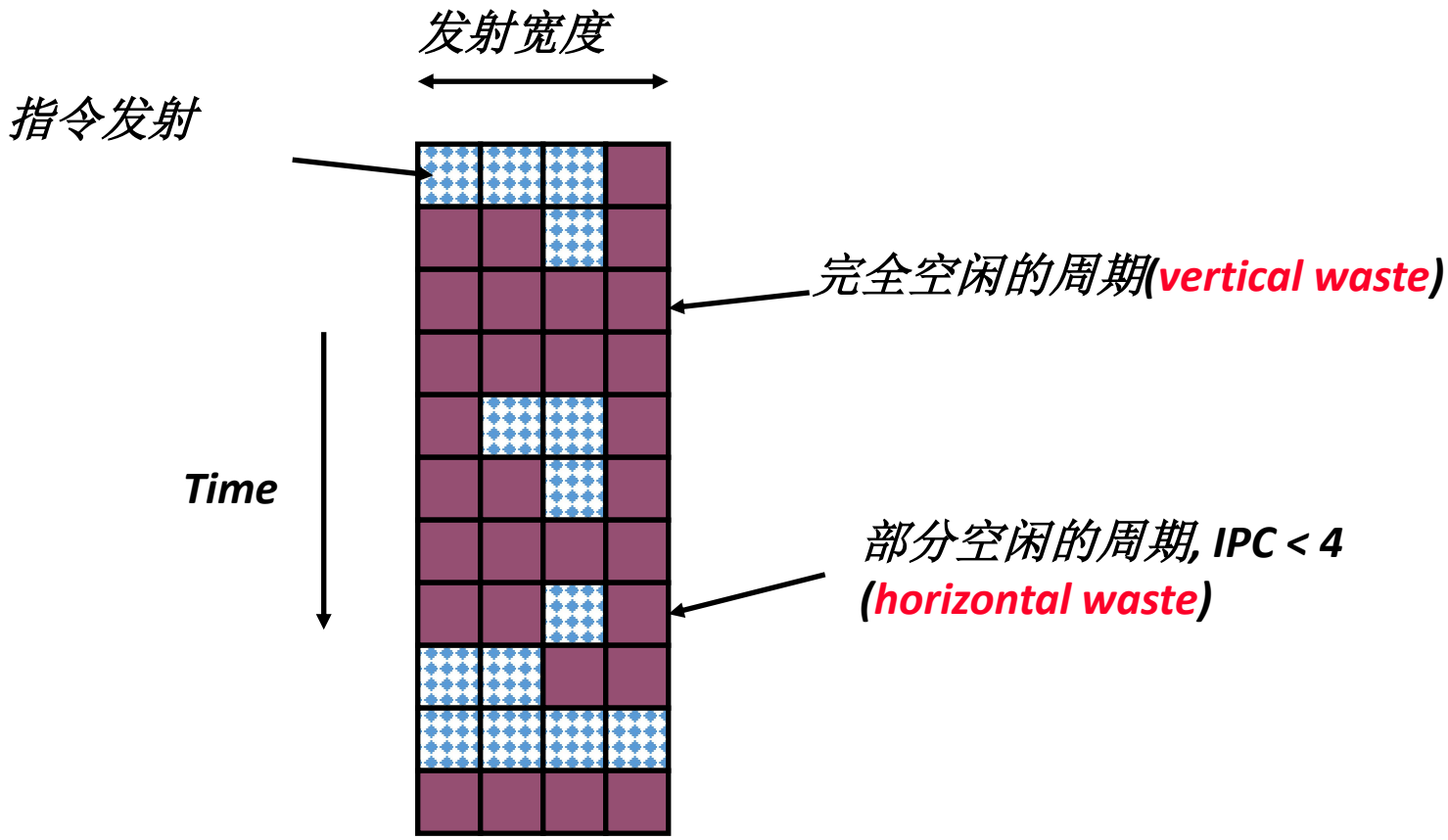


各种执行方式的比较


Superscalar
Fine-Grained
Coarse-Grained
Multiprocessing
Simultaneous Multithreading
Time (processor cycle)
Thread 1
Thread 2
Thread 3
Thread 4
Thread 5
Idle slot

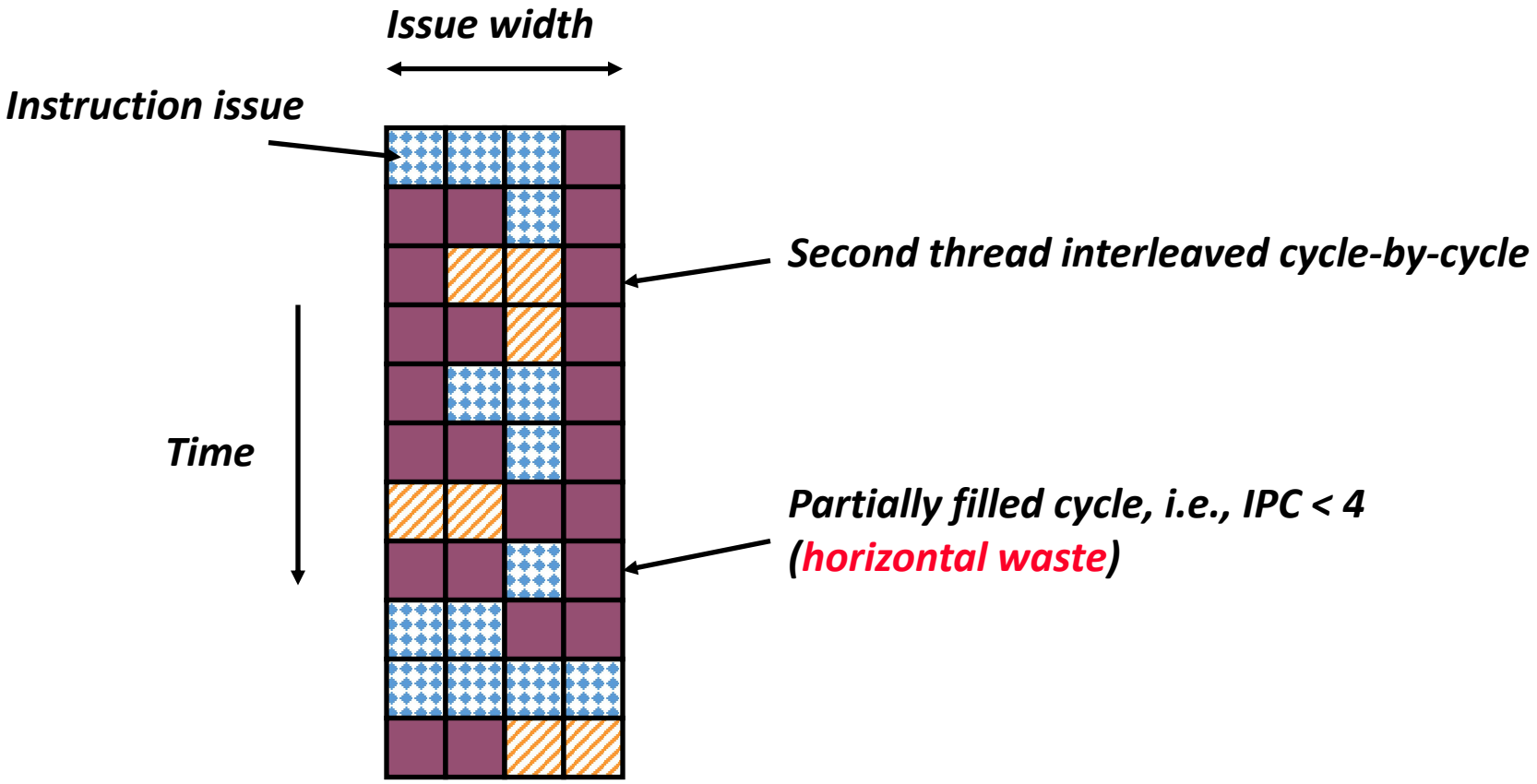
各种执行方式的比较

- 横向的超标量多发射



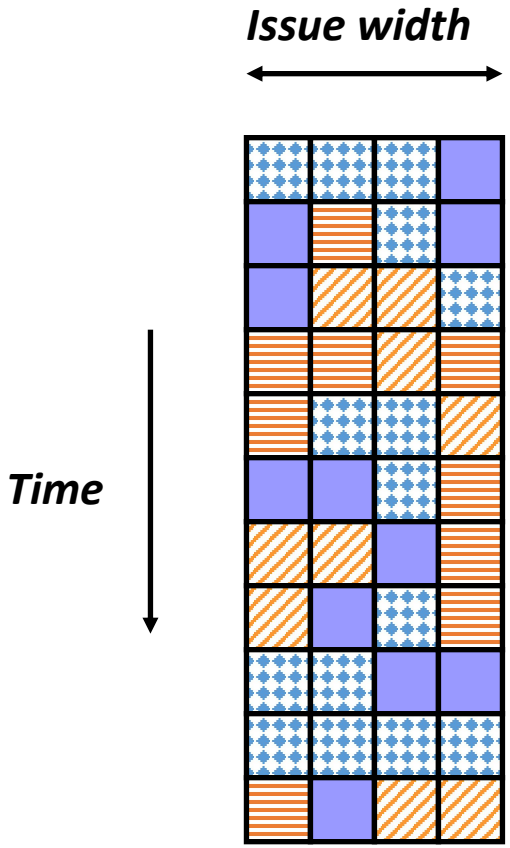
各种执行方式的比较

- 纵向的多线程发射
  - 消除了部分时钟没有指令发射的情况



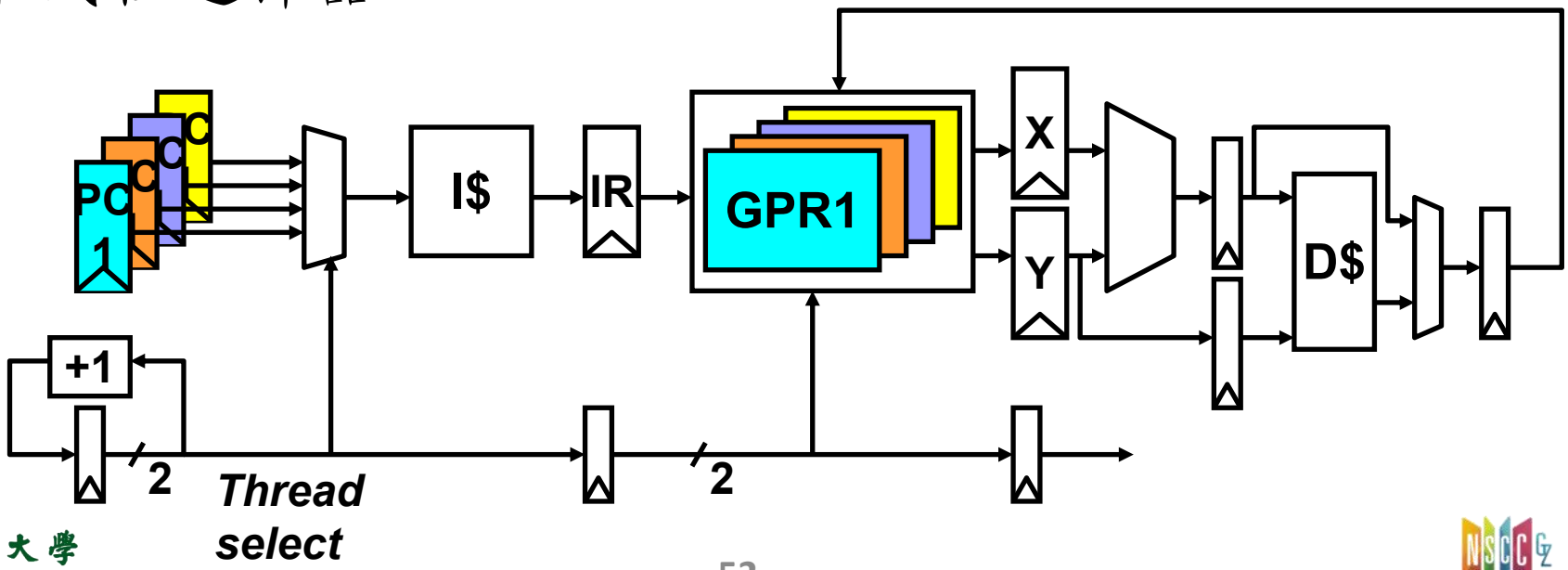
各种执行方式的比较

- 超标量+SMT
  - 从多个线程中寻找可发射的指令



SMT体系结构设计

- 多个线程执行的上下文（Context）
  - 寄存器组、PC、取指单元
  - 允许从多个线程中取指令发射
- 采用超标量技术和乱序（Out-of-Order）执行技术
  - 提高发射宽度
  - 乱序执行
- 一个线程选择器



SMT面临的问题

- 单线程性能和总体吞吐率不能两全
  - SMT提升总体吞吐率（功能单元利用率），但会降低单线程的性能
  - 也可以给某个特定的线程更高的调度优先级
- 需要大量的寄存器组文件来保存多个线程的上下文
- 有可能增加时钟周期的长度，降低主频
  - 发射单元复杂：需考虑多个线程的指令流
  - 指令提交复杂：选择哪条指令提交
- 多线程之间的Cache和TLB的争用有可能降低性能

Intel的超线程就是SMT的典型应用

内容纲要

- 超标量
  - 一个时钟发射多条指令
    - 当前处理器均采用的方法
- 超长指令字（VLIW, Very Long instruction Word）
  - 编译器将多条不相关的指令打包成一个“指令包”
- 同时多线程
- 指令集并行的限制因素

理想的处理器

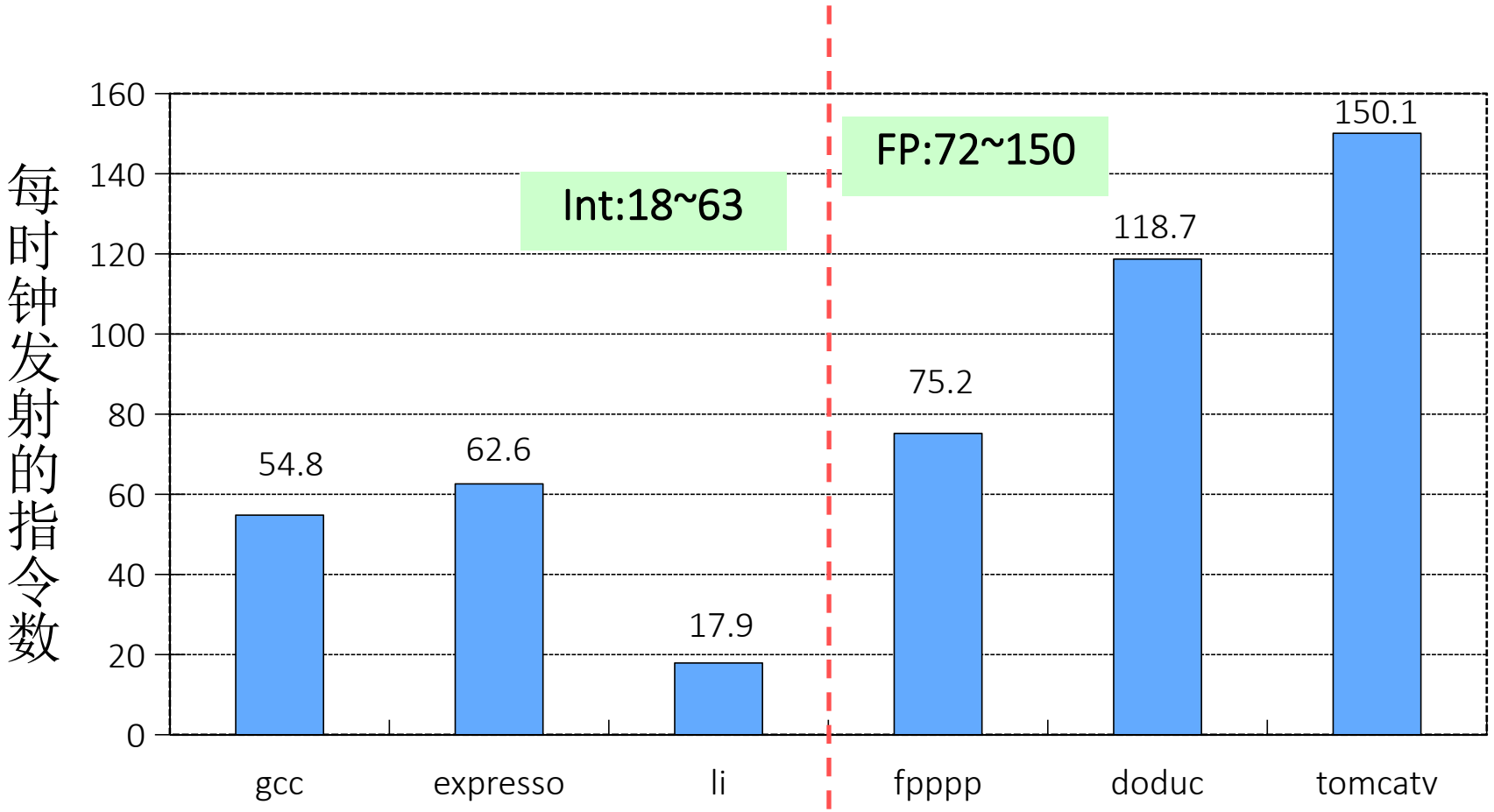
- 1. 无限的虚拟寄存器
  - 无限制的寄存器重命名
  - 消除所有的名字相关(WAW、WAR)
- 2. 完美的分支预测
  - 没有任何控制冒险
- 3. 完美的无条件跳转预测
  - 完美的推测执行
  - 无限的ROB
- 4. 完美的值预测
  - 解决不同名字带来的问题，如：访存
- 5. 无限大的L1缓存
  - 所有的访存一个时钟完成

理想的处理器意味着：

- 消除所有的名字相关
  - Assumptions 1 (registers) and 4 (memory)
- 消除所有的控制相关
  - Assumptions 2 and 3
- 一旦依赖的数据已经就绪，指令可立即执行
  - 真相关很好的解决
- 指令发射单元可以前后看的很远，总是能找到足够的指令填充功能单元

理想的处理器下的性能

- 指令集并行的上限(SPEC92)



当前处理器的实际情况

- 需要考虑的实际问题
  - 指令窗口大小、分支预测准确度
  - 寄存器数量、别名分析结果
  - 指令发射的宽度

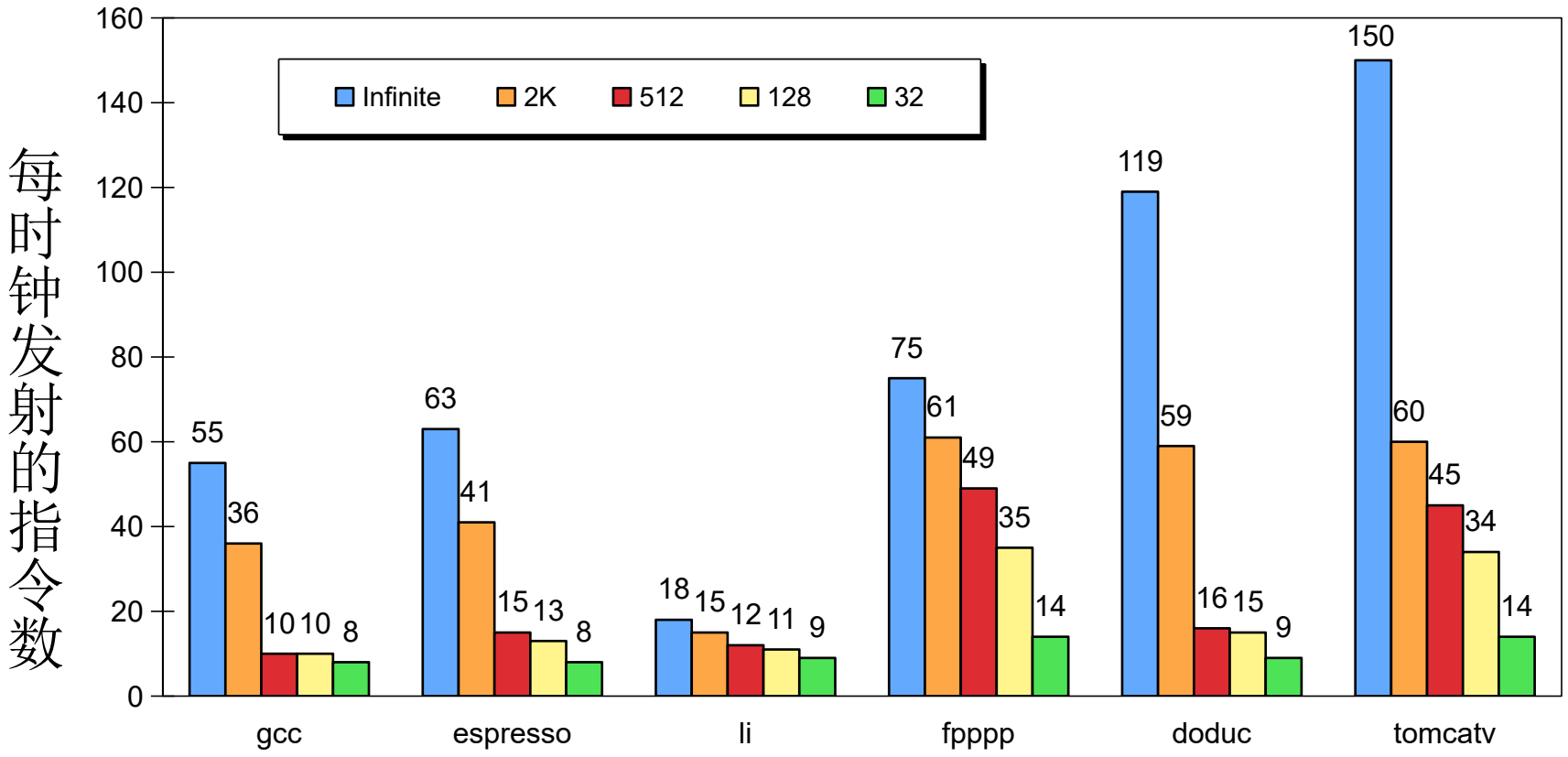
Functions	Ideal Model	Power 8 (2014)
Instructions Issued per clock	Infinite	10
Execution units	Infinite	16
Instruction Window Size	Infinite	Hundreds
Renaming Registers	Infinite	48 integer+40 Fl. Pt. (?)
Branch Prediction	Perfect	2% to 6% misprediction
Cache	Perfect	64KD, 32KI, 512KB L2, 8MB L3 (one core)
Memory Alias Analysis	Perfect	??

指令窗口大小的影响

- 当前指令窗口大小：hundreds量级
  - 窗口大小50约需要2450比较(900 for 30)
  - IBM Power5：超过200条指令正在执行
    - 每时钟向6个功能单元大约发射4条指令
    - 88定点寄存器和88浮点寄存器用于重命名

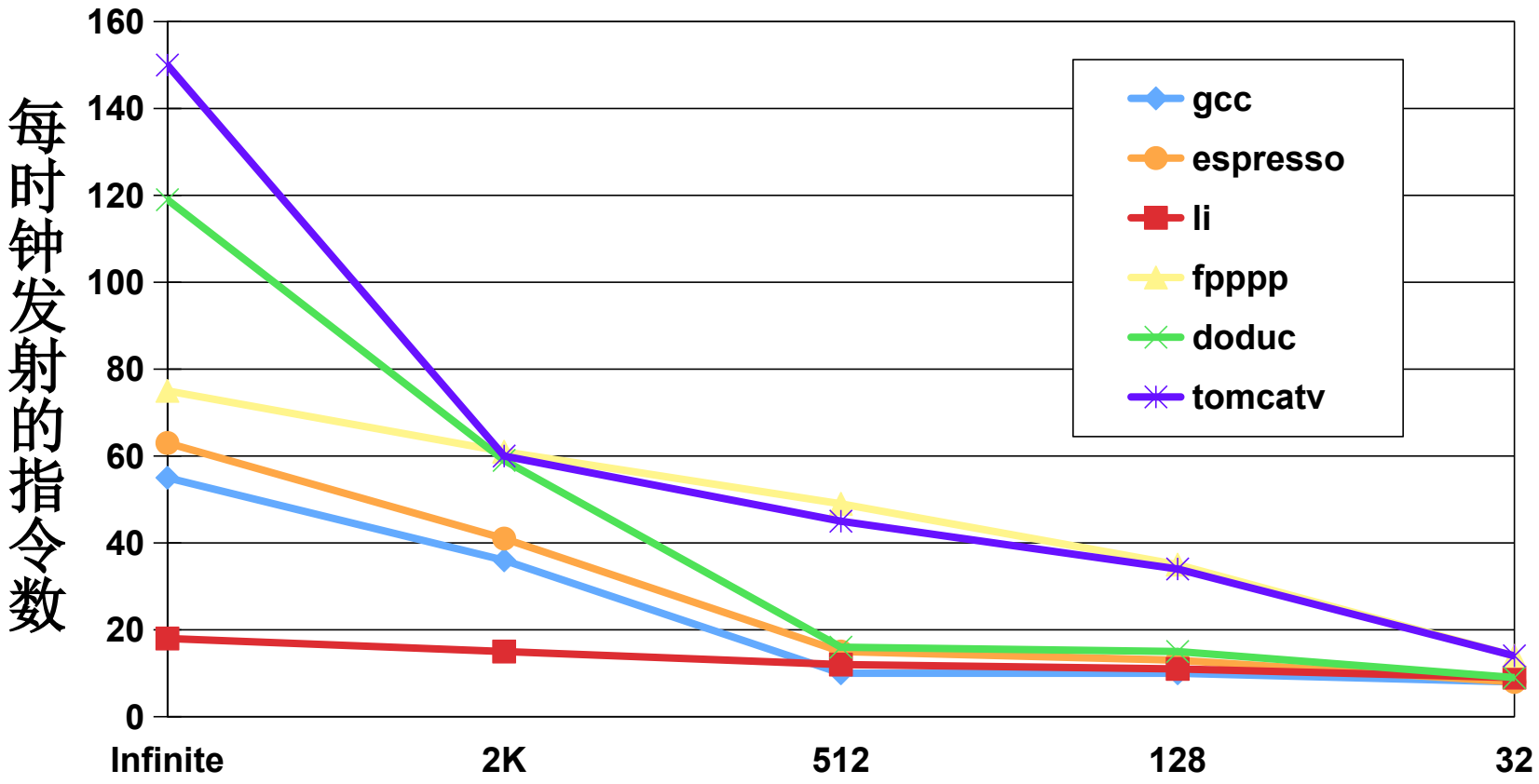
Functions	Real Model	Ideal Model
Instructions issued per clock	Infinite	Infinite
Instruction window size	Infinite, 2K, 512, 128, 32	Infinite
Renaming registers	Infinite	Infinite
Branch prediction	Perfect	Perfect
Cache	Perfect	Perfect
Memory alias	Perfect	Perfect

指令窗口大小的影响


每时钟发射的指令数
160
140
120
100
80
60
40
20
0
Infinite
2K
512
128
32
55
36
10
10
8
63
41
15
13
8
18
15
12
11
9
75
61
49
35
14
119
59
16
15
9
150
60
45
34
14
gcc
espresso
li
fpppp
doduc
tomcatv

指令窗口大小的影响

- 随着指令窗口减小，每时钟能发射的指令数显著降低



分支预测的影响

- 五种解决方案的比较
  - 理想预测器：Perfect
  - 竞赛预测器：Tournament-based branch predictor
    - 2^{13} (8K) entries, 每个entry包含3个2-bit fields
    - correlating 2-bit predictor, with global history
    - noncorrelating 2-bit predictor
    - 2-bit selector: correlating/noncorrelating
  - Standard 2-bit with 512 entries BPB
    - 16 entries to predict returns, in addition
  - Static
    - Using profile history of the program by compiler
  - None
    - Jumps are still predicted

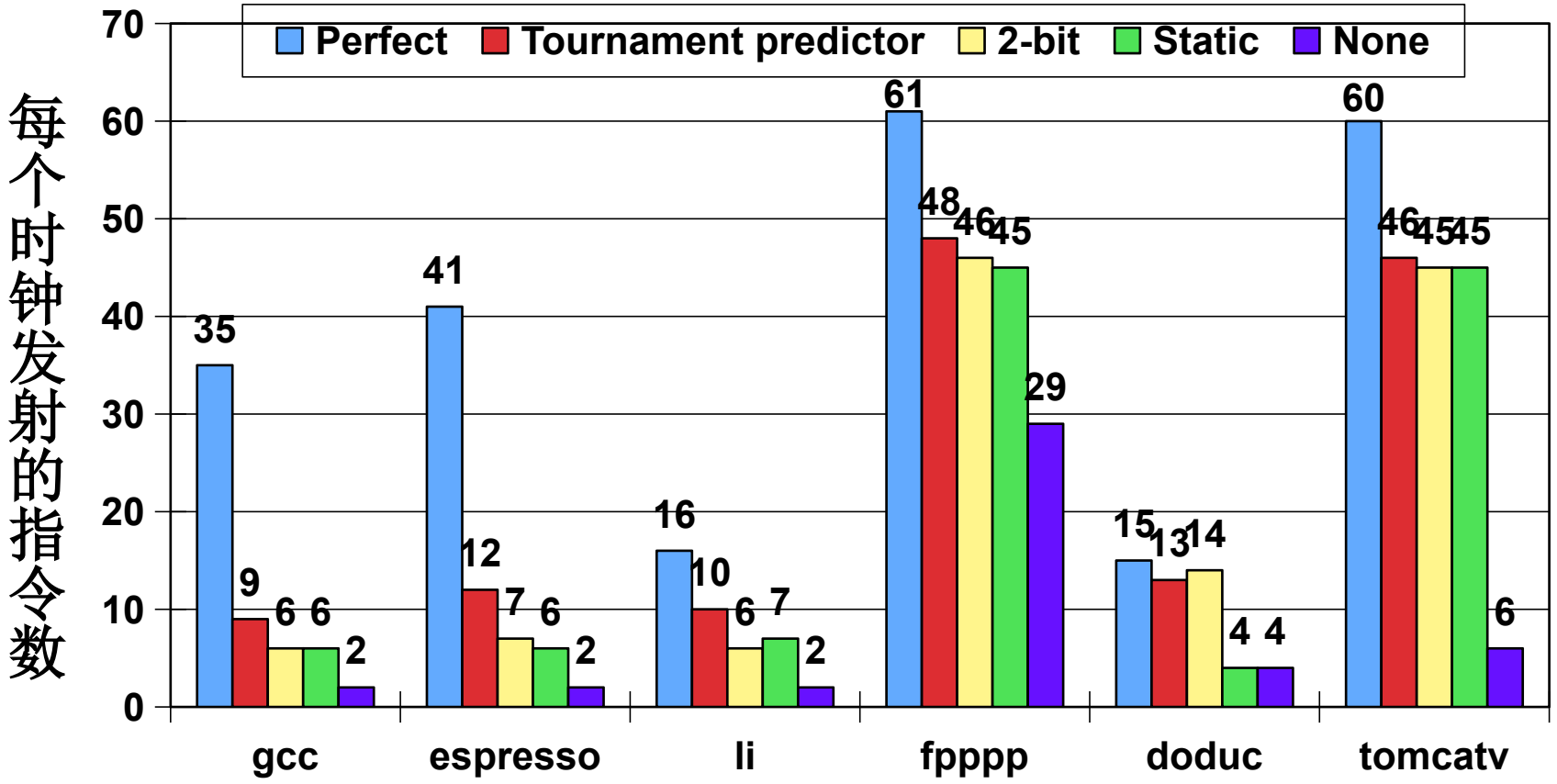
分支预测的影响

- 五种解决方案的比较

Functions	New Model	Model
Instructions issued per clock	64	Infinite
Instruction window size	2048	Infinite
Renaming registers	Infinite	Infinite
Branch prediction	Perfect vs. 8K Tournament vs. 512 2-bit vs. profile vs. None	Perfect
Cache	Perfect	Perfect
Memory alias	Perfect	Perfect

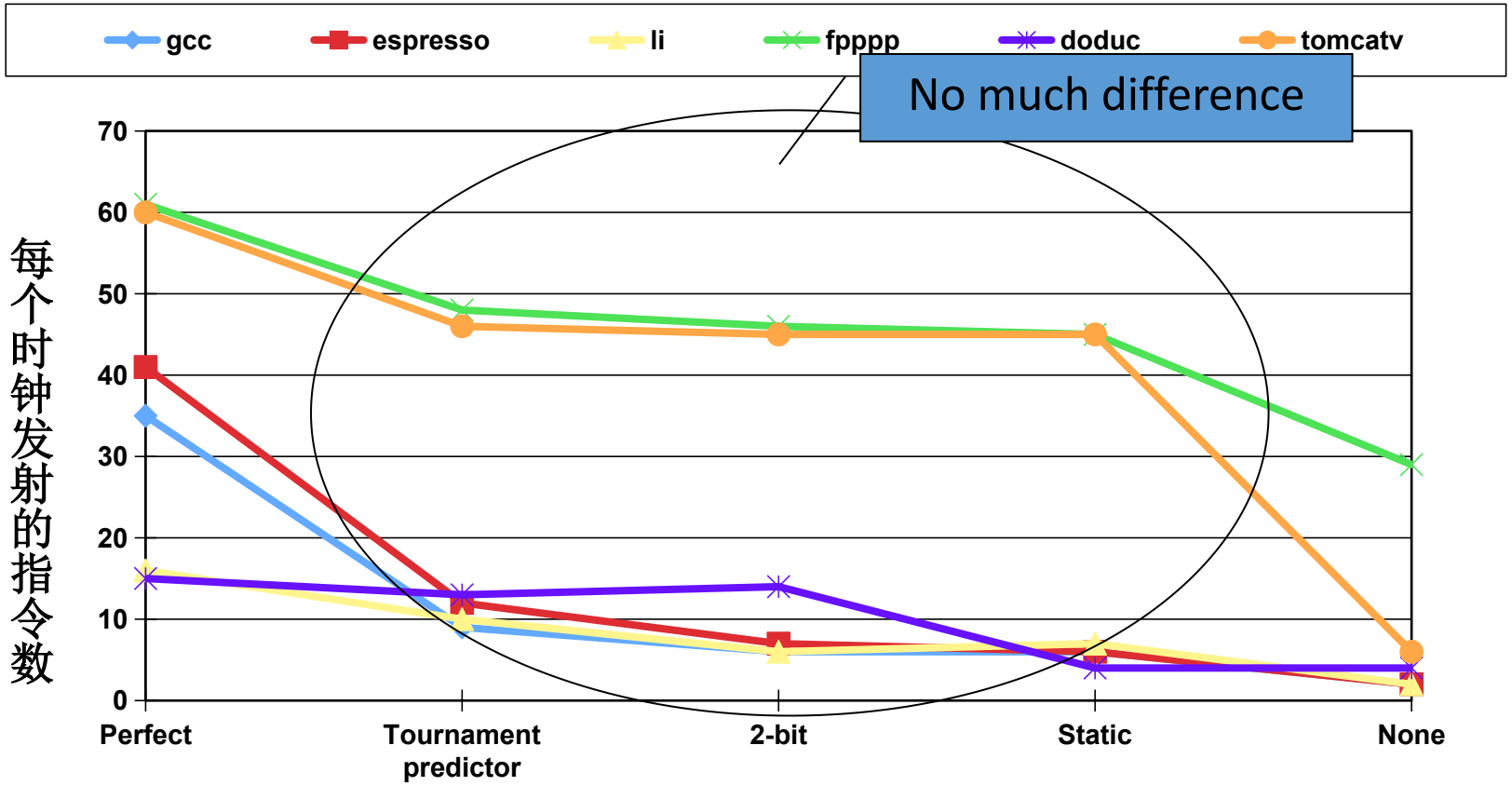
分支预测的影响

- 五种解决方案的比较
  - 预测失败的代价很高



分支预测的影响

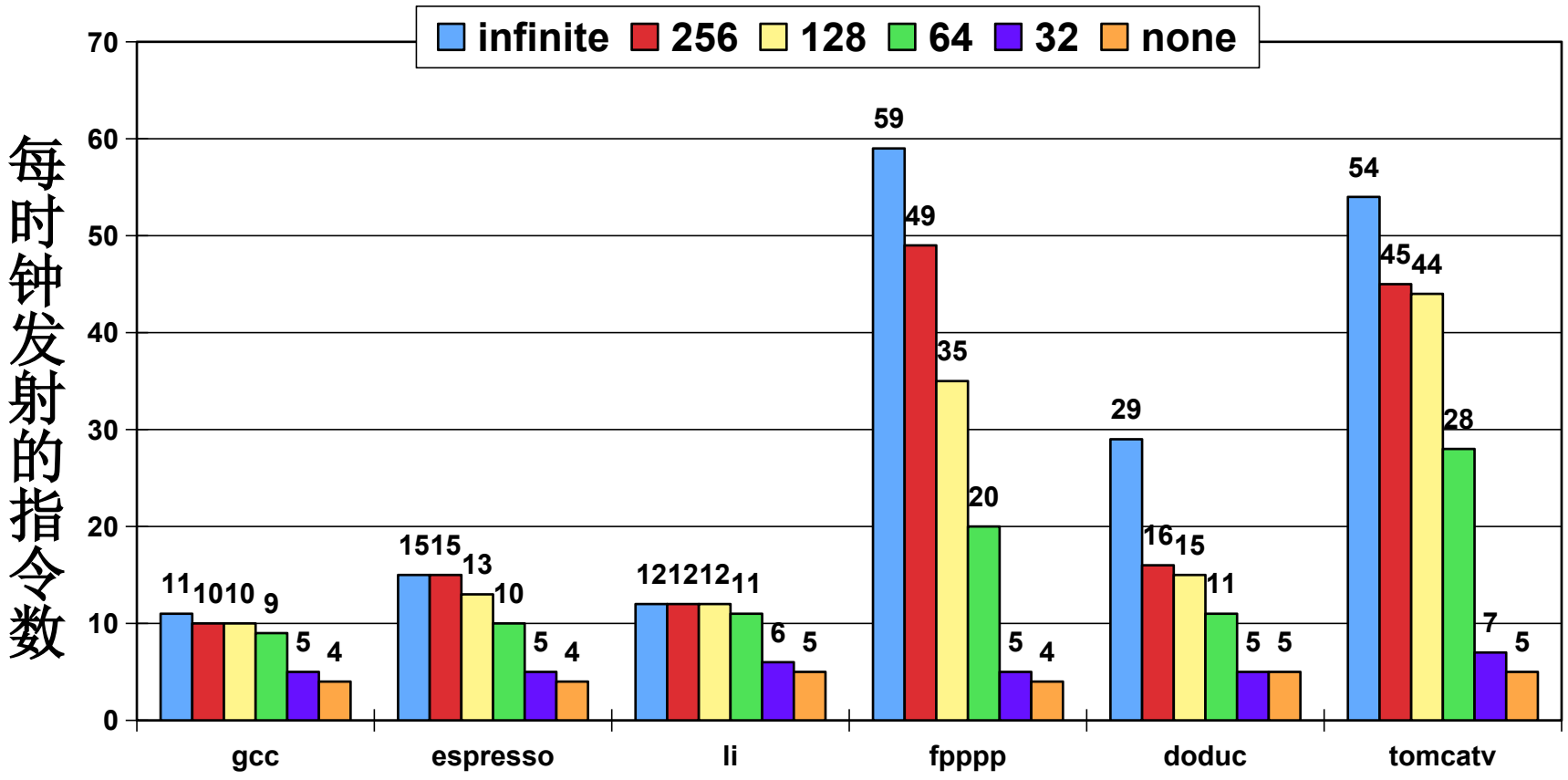
- 影响不是很大



寄存器数量的影响

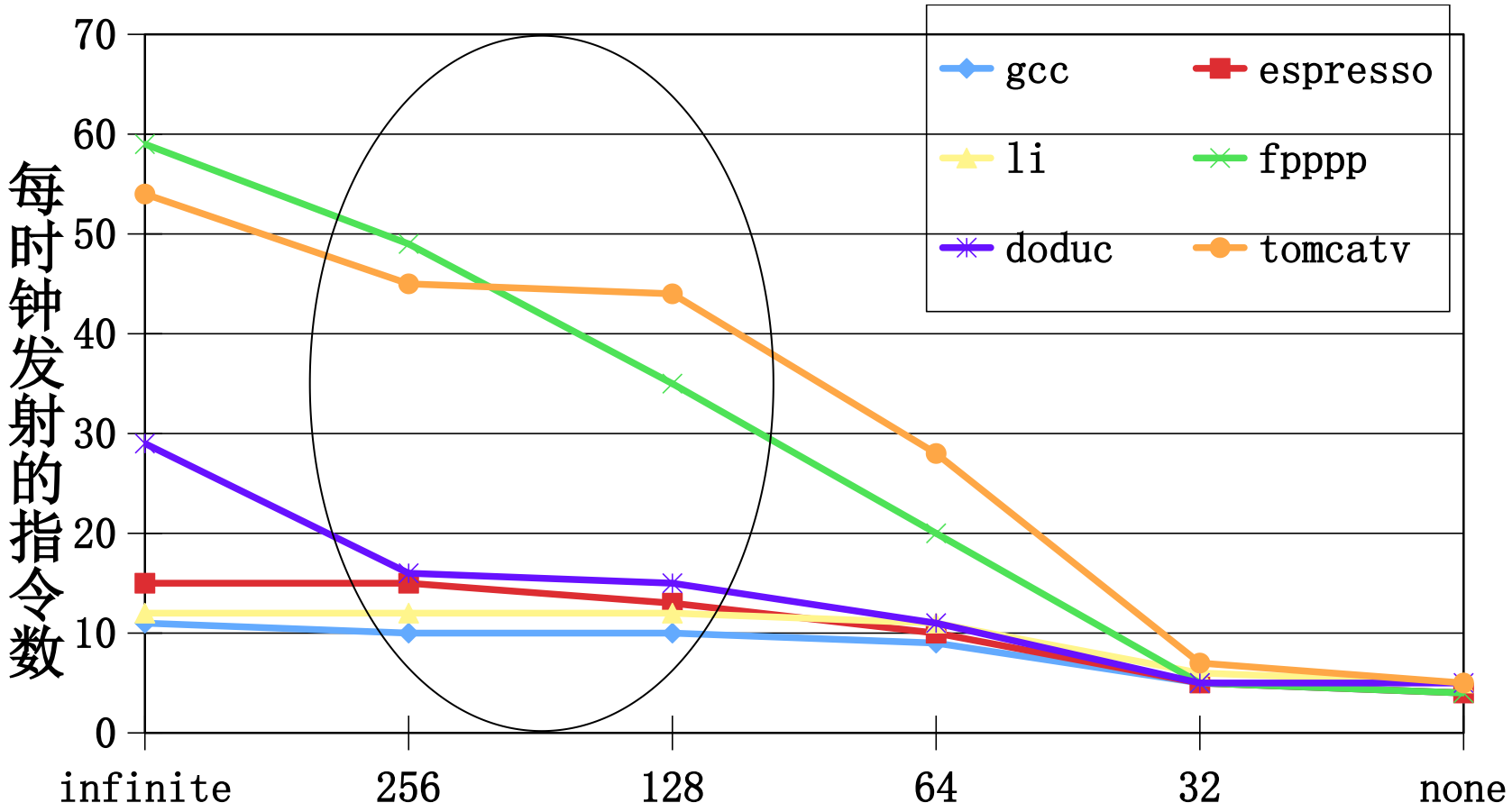
Functions	New Model	Model
Instructions issued per clock	64	Infinite
Instruction window size	2048	Infinite
Renaming registers	Infinite v. 256, 128, 64, 32, none	Infinite
Branch prediction	8K 2-bit	Perfect
Cache	Perfect	Perfect
Memory alias	Perfect	Perfect

寄存器数量的影响


每时钟发射的指令数
infinite
256
128
64
32
none
70
60
50
40
30
20
10
0
11 10 10 9 5 4
15 15 13 10 5 4
12 12 12 11 6 5
59 49 35 20 5 4
29 16 15 11 5 5
54 45 44 28 7 5
gcc
espresso
li
fpppp
doduc
tomcatv

寄存器数量的影响

- 256个寄存器好像就够了



内容小结

- 超标量
  - 一个时钟发射多条指令
    - 当前处理器均采用的方法
- 超长指令字（VLIW, Very Long instruction Word）
  - 编译器将多条不相关的指令打包成一个“指令包”
- 同时多线程
- 指令集并行的限制因素