北京大学计算机学院本科生课程

计算机组成与系统结构实习

性能分析及模拟器概述

北京大学微处理器研发中心

易江芳 刘先华

2022-10-17

背景

- 人类探索未知世界的三种方法
  - –实验（记录和描述自然现象）
  - –理论（利用模型归纳演绎推理）
  - –计算（对复杂现象进行模拟仿真）

结构的选择是一个复杂的过程

- 结构选择是一个复杂的过程
  - 存在大量的可选方案
  - 可选方案与海量的工作负载交织在一起
  - 如何科学的选择结构是系统设计者需要研究的重大问题

性能分析、建模和测量

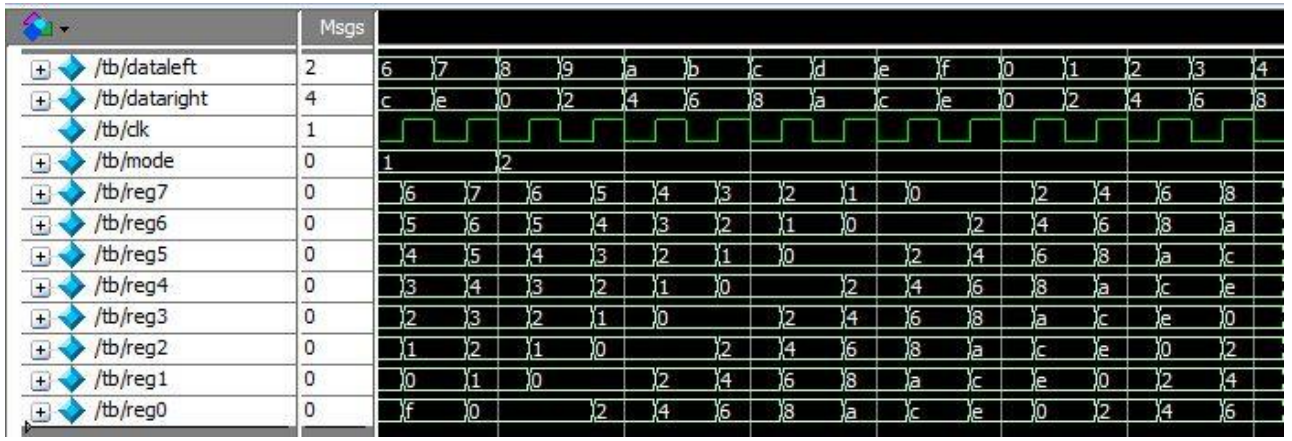
- 性能分析就是**测量***平台及微结构*和*软件及数据结构*之间的交互行为，为设计新结构**提供**必要的**数据**和**理论基础**。
  - COD的姊妹篇，《computer Architecture: A Quantitative Approach》
- 性能建模
  - 基于分析和统计的建模：概率模型，统计模型，petri网模型
  - 基于模拟的建模：本课的主要内容
- 性能测量
  - 硬件监测，软件监测，插桩 等

性能建模和测量的要求

- 建模和测量的**准确性**，直接影响设计权衡和方案选择
- 模型的**速度**，对于设计空间探索的广度和深度有重要影响
- 以上两点最为重要，其他还包括
  - 能够捕获用户态和系统态下的行为
  - 开销不能过大
  - 易于修改和扩展
  - 测量环境应与真实场景接近
  - 等等

模拟(Simulation)和仿真(Emulation)

- 模拟
  - 由另一系统的性能来表示某一个实际系统或抽象系统性能的某些特性
  - 用另一系统来模仿一个系统，原则上由软件完成，使该模仿系统接收与被模仿系统相同的数据，执行相同或相似的计算机程序，得到与被模仿系统相似或相同的结果。

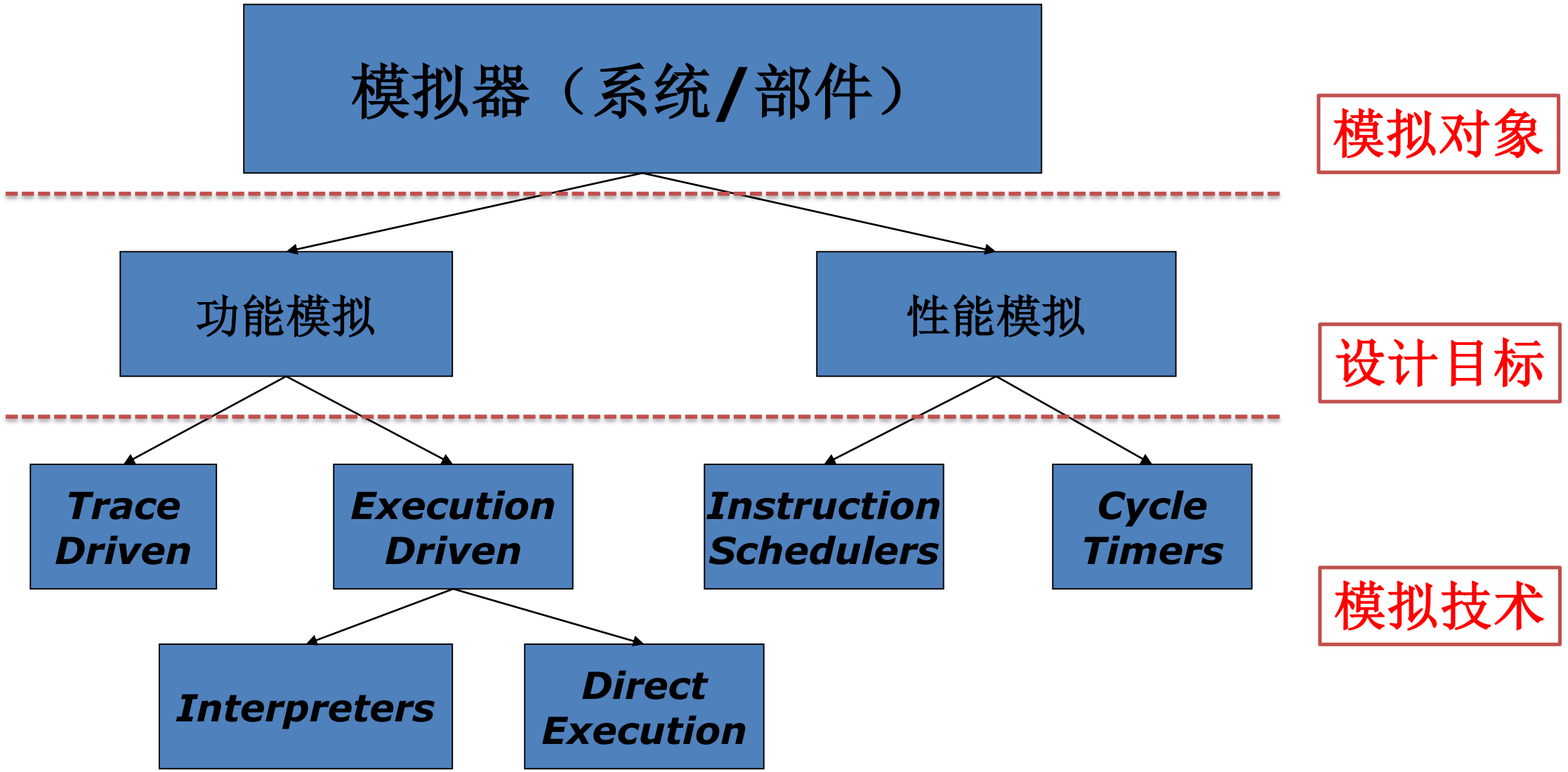


- 仿真
  - 一个系统对另一个系统的部分或全部的模仿，主要通过硬件实现，使模仿的计算机系统与被模仿的计算机系统接收相同的数据，执行相同的程序，获得相同的结果。

为什么采用模拟技术

- 较之具体实现
  - 降低成本
  - 在硬件可用前提供指导
  - 灵活，易于修改以描绘不同的硬件实现
  - 全面，提供不同层次的抽象
- 较之数学模型
  - 准确
  - 详细

模拟器的一种分类


模拟器（系统/部件）
功能模拟
性能模拟
Trace Driven
Execution Driven
Instruction Schedulers
Cycle Timers
Interpreters
Direct Execution
模拟对象
设计目标
模拟技术

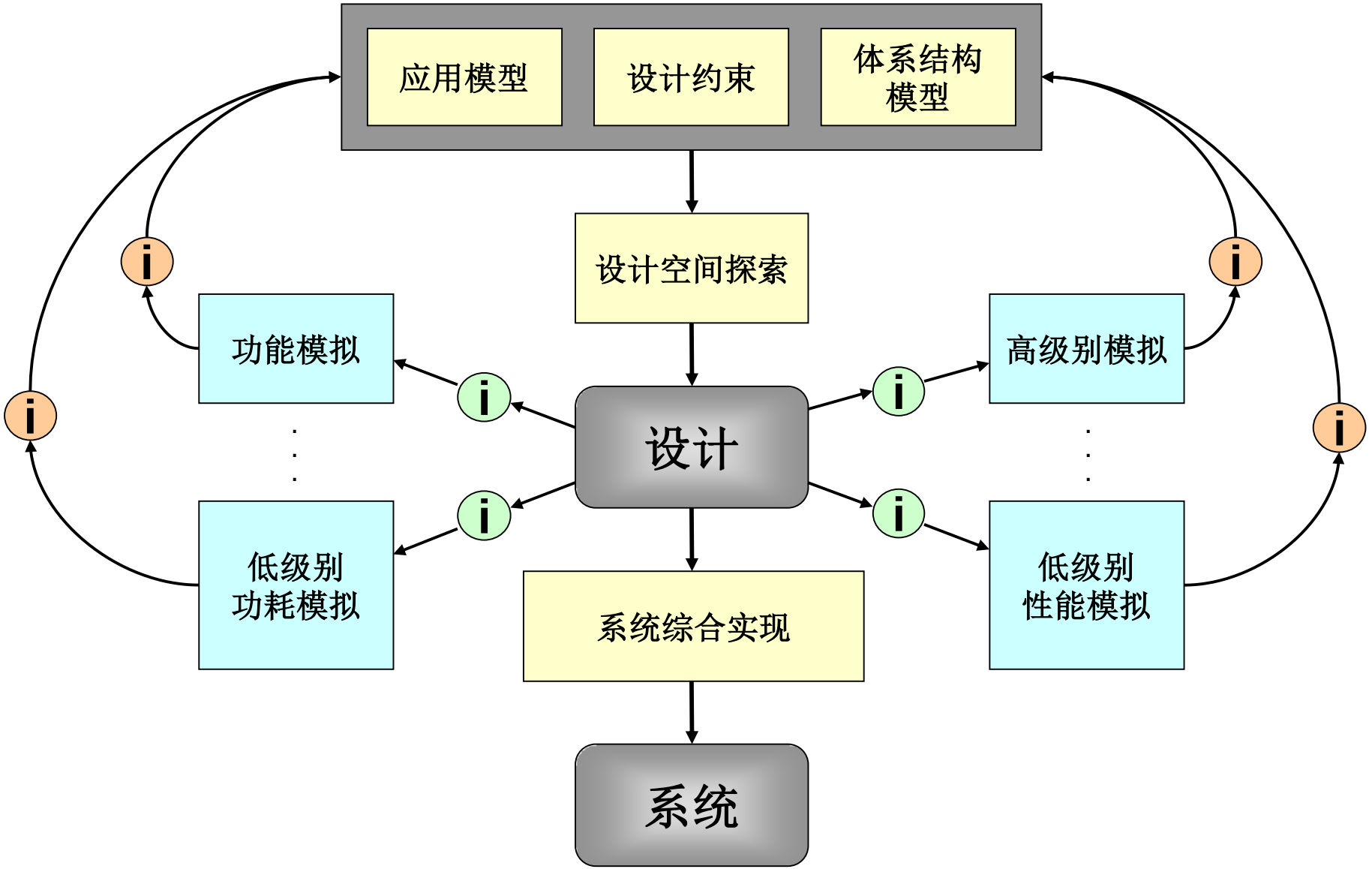
模拟器的分类方法一

模拟器输入 \ 设计目标		功能模拟	功耗模拟	性能模拟	
				数量计数	周期计数
踪迹驱动		N.A.	N.A.	SS/sim-bpred	某些cache模拟器
执行驱动	解释	Armulator	Wattch+SS	**SS/sim-safe**	SS/sim-outorder
	翻译	qemu	Wattch+SimOS	SIMICS SimOS	gem5 SimOS

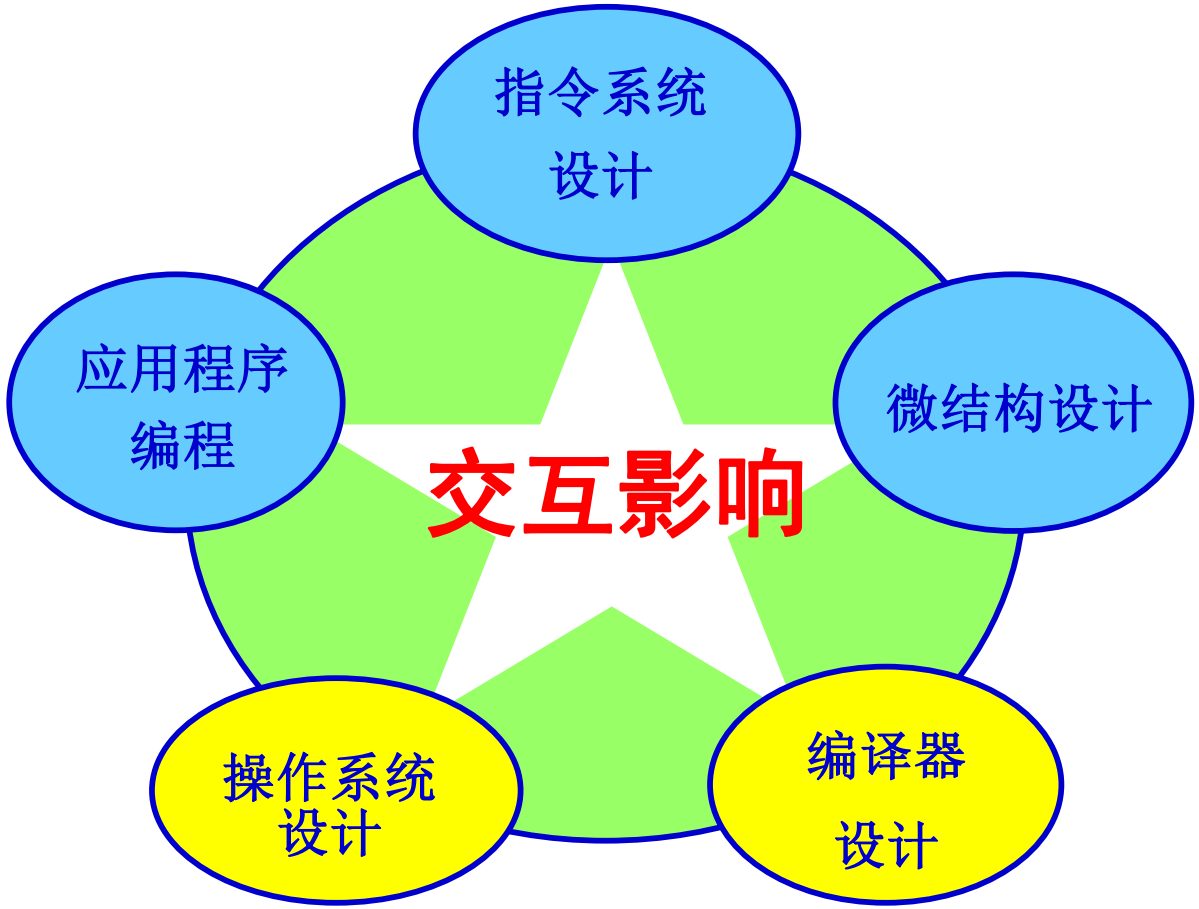
模拟器的分类方法二

支持级别 / 模拟粒度	应用程序级 （仅支持应用程序运行）	全系统级 （支持完整计算机系统运行）
指令级	SS/sim-safe	SIMOS
周期级	SS/sim-outorder	SIMICS/gem5
信号级	SPIM信号级模拟器 一般用于支持验证	过于复杂，不常见

设计流程中的模拟器


应用模型
设计约束
体系结构
模型
设计空间探索
功能模拟
高级别模拟
设计
低级别
功耗模拟
系统综合实现
低级别
性能模拟
系统

模拟技术和其它相关技术的交互



功能模拟和性能模拟

- 功能模拟器只模拟指令功能， 忽略微结构细节
  - 速度快，没有任何结构或微结构信息
  - 主要用在系统软件开发、踪迹信息获取等方面
  - Lab 2.1就是功能模拟
- 性能模拟，或者结构模拟
  - 对处理器微结构中的流水线、多发射、乱序执行、分支预测等操作和对内存系统中的缓存读写操作进行模拟
  - 主要用在性能分析等方面
  - Lab 2.2就是性能模拟

踪迹驱动与执行驱动

- 执行驱动
  - 执行驱动是指模拟器的输入是真实的二进制代码，因此可以在执行驱动的模拟器运行完整的操作系统或者应用程序
  - 可以编译执行，也可以解释执行
  - 比如lab 2中的测试输入，是编译用户程序产生的二进制编码
- 踪迹驱动
  - 踪迹驱动则是指模拟器的输入是一些事先得到的踪迹，这些踪迹可以是真实系统或者驱动执行的模拟器上运行程序的踪迹，也可以是按某种分布函数所产生的伪踪迹
  - 比如lab 3中的测试输入，就是一组截取的访存序列

时钟驱动和事件驱动

- 性能模拟技术的一种分类
  - 时钟驱动
    - 在模拟器中维护一个软件时钟，每个时钟周期调用模拟函数推进一步
  - 事件驱动
    - 事先将所有的事件按时间戳顺序插入到列表中，模拟时按照列表中的事件推进模拟器

- 完成lab 2.2时需要选择

高级的模拟技术

- 时序优先（timing-first）技术
  - 性能模拟器需要模拟时序信息，如果将功能和时序模拟结合在一起，则模拟器实现过于复杂，调试维护难度增大
  - 将功能模拟和时序模拟分开，时序模拟主要提高模拟器的精度，而功能模拟主要是保证模拟的正确性。
  - 时序模块模拟每个部件的微结构，根据微结构当前状态向功能级模拟器发出执行命令，由功能模拟模块完成具体的功能模拟
  - 可以利用现有的功能级模拟器快速开发一个时钟级模拟器，并使模拟器的结构更加清晰，减轻了开发和调试负担
  - 比如，快速改造Lab 2.1，完成Lab 2.2

输入序列的缩减—采样

- 采样：截取程序的一小部分进行模拟，并将其作为整个程序模拟结果的一个近似，以降低模拟时间
- 关键：如何获得这一小段具有代表性的程序?
- 采样的方式
  - 随机采样
  - 周期采样
  - 热点分析

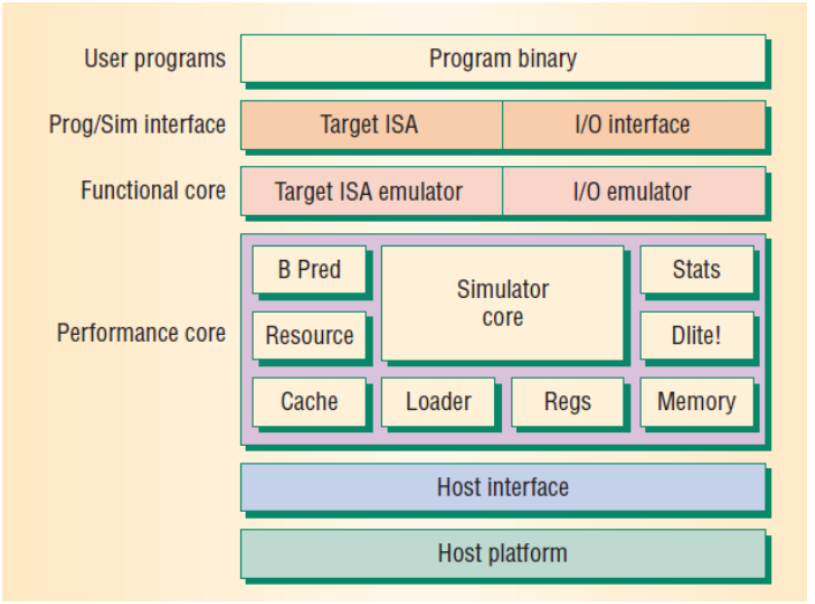
CPU2000程序要达到一定的模拟精度所需要采样的最小指令数目（20-30 million）

执行驱动模拟器：SimpleScalar

- 于1995年由威斯康辛大学推出， 迅速成为研究处理器设计的重要工具
  - 采用了功能模拟和时序模拟分离的方法。功能模拟模块提供指令集和I/O模拟，性能模拟部分提供模拟器运行的性能模型
  - 多种运行模式，如**sim-outorder**模拟了动态调度、推测执行等处理器技术，以及多级**Cache**存储系统
  - 可以研究**CPU**微结构，无法体现操作系统对整个系统的影响
  - **http://www.simplescalar.com**

Table 1. SimpleScalar baseline simulator models.

Simulator	Description	Lines of code	Simulation speed
sim-safe	Simple functional simulator	320	6 MIPS
sim-fast	Speed-optimized functional simulator	780	7 MIPS
sim-profile	Dynamic program analyzer	1,300	4 MIPS
sim-bpred	Branch predictor simulator	1,200	5 MIPS
sim-cache	Multilevel cache memory simulator	1,400	4 MIPS
sim-fuzz	Random instruction generator and tester	2,300	2 MIPS
sim-outorder	Detailed microarchitectural timing model	3,900	0.3 MIPS

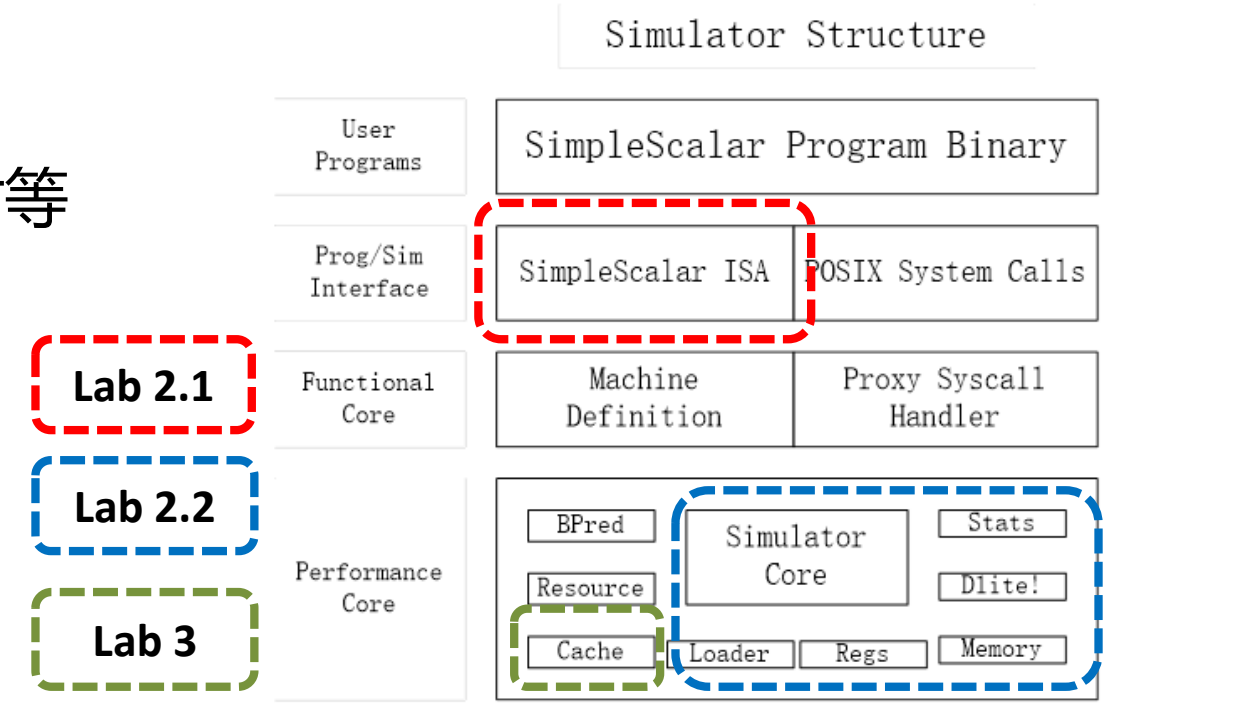


支持的模拟工具

- sim-safe
  - 指令模拟，支持调试
- sim-fast
  - 快速指令模拟，不支持调试，不记录指令相关信息
- sim-outorder
  - 支持乱序执行、分支预测、多发射等超标量特性
- sim-bpred
  - 实现分支预测分析器
- sim-cache
  - 实现cache模拟

Table 1. SimpleScalar baseline simulator models.

Simulator	Description	Lines of code	Simulation speed
sim-safe	Simple functional simulator	320	6 MIPS
sim-fast	Speed-optimized functional simulator	780	7 MIPS
sim-profile	Dynamic program analyzer	1,300	4 MIPS
sim-bpred	Branch predictor simulator	1,200	5 MIPS
sim-cache	Multilevel cache memory simulator	1,400	4 MIPS
sim-fuzz	Random instruction generator and tester	2,300	2 MIPS
sim-outorder	Detailed microarchitectural timing model	3,900	0.3 MIPS



系统级模拟器：QEMU

- QEMU: a generic and open source machine emulator and virtualizer
- 仅模拟系统，不模拟微结构
- 支持两种运行模式：
  - user mode，运行用户程序
  - full system，可运行操作系统
- 最特殊之处
  - 作为KVM和Xen中的虚拟化部分，运行速度接近真机
- https://www.qemu.org/

当今主流：GEM5

- 既可以模拟微结构，也可以模拟全系统
- 可定制化的模拟框架
  - 支持**X86**、 **Alpha**、 **ARM**、 **SPARC**和**MIPS**等指令集
  - RISC-V也有GEM5版本：https://github.com/gem5/gem5.git
  - 可以支持运行应用程序，也可以支持运行操作系统
  - 可以模拟处理器微结构，也可以模拟核间互连
- 当前的主流模拟器
- http://www.gem5.org

FPGA加速模拟技术

- Patterson主持开发了RAMP（Research Accelerator for Multiple Processors）项目
  - 使用**FPGA**进行全系统模拟，目的在于研究**16**个核心以上的多处理器系统，是一个综合的软硬件平台
  - http://ramp.eecs.berkeley.edu/

- 异构计算，各式Acclerator
  - Simulatior + emulator

性能测量

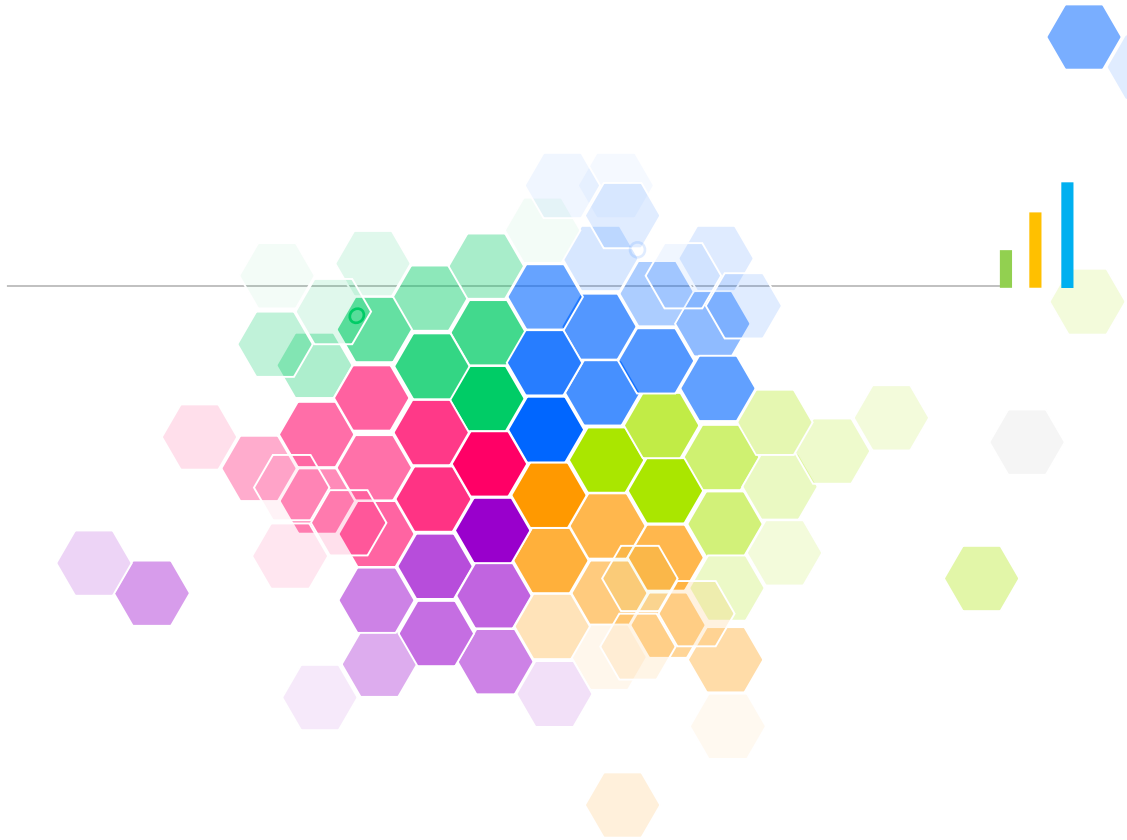
- 目的：对现有系统或程序进行调优
- 主要方法
  - 硬件性能监视器（performance monitor）
  - 软件性能监测
- 性能测量数据的分析
  - Perf
  - gProf
  - OProfile

事件号	umask值	事件	描述
3CH	00H	UnHalted Core cycles	时钟周期或拍数
C0H	00H	Instruction retired	提交的指令数
2EH	4FH	LLC reference	访问最后一级cache的数目
2EH	41H	LLC misses	访问最后一级cache失效的数目
C4H	00H	Branch Instruction Retired	提交的分支指令的数目
C5H	00H	Branch Misses Retired	提交的误预测的分支指令的数目
0BH	01H	MEM_INST_RETIRED.LOADS	提交的load指令的数目
0BH	02H	MEM_INST_RETIRED.STORES	提交的store指令的数目
0EH	01H	UOPS_ISSUED.ANY	从重命名表发射到保留站的微码数目
0FH	02H	MEM_UNCORE_RETIRED.OTHER_CORE_L2_HITM	提交的Load访存操作，命中芯片相邻核L2 cache，并是modified状态
12H	01H	SIMD_INT_128.PACKED_MPY	128位的SIMD定点乘法操作的数目
24H	01H	L2_RQST.LD_MISS	L2 Load请求，L2 cache失效，L2 load包括L1D失效和L1D预取
26H	FFH	L2_DATA_RQSTS.ANY	所有的L2数据请求
40H	0FH	L1D_CACHE_LD.MESI	所有的L1数据cache读请求
C4H	00H	BR_INST_RETIRED.ALL_BRANCHES	提交的分支指令
D2H	0FH	RAT_STALLS.ANY	寄存器分配表引起的堵塞
2AH	01H	UNC_QMC_OCCUPANCY.CH0	内存控制器通道0读请求发生
60H	01H	UNC_DRAM_OPEN.CH0	DRAM通道0由于读或者写发出open命令，因为该page首先需要打开

Intel Nehalem性能计数器事件示例

小结

- 性能分析、模拟和测量
- 根据评测目标，选择不同类型的模拟器
- 模拟器最重要的是精度和速度
- 经典的模拟器
  - SimpleScalar
  - QEMU
  - GEMS/SIMICS/GEM5…
- 测量技术
  - 硬件监视器，软件监测
  - 采样和插桩



欢迎提问

关于SP、PC和GP

- SP: 栈指针，指示栈的起始地址
- PC: 程序指针，指示当前运行的指令地址
- GP：全局数据指针，指示小数据段的起始地址
- 其中，SP和PC属于运行时环境，GP属于二进制规范定义，可以在链接脚本中做出定义。

示例：
risc-tools/riscv-tests/benchmarks/common/test.ld

```
OUTPUT_ARCH( "riscv" )      /*定义目标系统的体系结构*/
SECTIONS
  {
   /* text: test code section */
   . = 0x100;        /*定义代码段的起始地址为绝对地址0x100*/
   .text :
   { crt.o(.text)    /*在代码段一开始增加crt.o的代码段*/
     *(.text)        /*用户程序代码段*/
   }
   /* data segment */
   .data : { *(.data) }
   .sdata :
   {
     _gp = . + 0x800;  /*定义GP寄存器起始地址为0x100+0x800
     …
   }

   /* bss segment */
   .sbss: …
   .bss  : …
   /* End of uninitalized data segement */
   _end = .;
  }
```

GreenCard 勘误

- addiw rd, rs1, imm
  - R[rd] ← SignExt((R[rs1](63:0) +SignExt(imm))[31:0])
- lw rd, offset(rs1)
  - R[rd] ← SignExt(Mem(R[rs1] + offset, word))

感谢杨程旭同学

Instruction	Type	Opcode	Funct3	Funct7/IMM	Operation
add rd, rs1, rs2	R	0x33	0x0	0x00	R[rd] ← R[rs1] + R[rs2]
mul rd, rs1, rs2			0x0	0x01	R[rd] ← (R[rs1] * R[rs2])[31:0]
sub rd, rs1, rs2			0x0	0x20	R[rd] ← R[rs1] - R[rs2]
sll rd, rs1, rs2			0x1	0x00	R[rd] ← R[rs1] << R[rs2
mulh rd, rs1, rs2			0x1	0x01	R[rd] ← (R[rs1] * R[rs2])[63:32]
slt rd, rs1, rs2			0x2	0x00	R[rd] ← (R[rs1] < R[rs2]) ? 1 : 0
xor rd, rs1, rs2			0x4	0x00	R[rd] ← R[rs1] ^ R[rs2]
div rd, rs1, rs2			0x4	0x01	R[rd] ← R[rs1] / R[rs2]
srl rd, rs1, rs2			0x5	0x00	R[rd] ← R[rs1] >> R[rs2]
sra rd, rs1, rs2			0x5	0x20	R[rd] ← R[rs1] >> R[rs2]
or rd, rs1, rs2			0x6	0x00	R[rd] ← R[rs1] | R[rs2]
rem rd, rs1, rs2			0x6	0x01	R[rd] ← (R[rs1] % R[rs2]
and rd, rs1, rs2			0x7	0x00	R[rd] ← R[rs1] & R[rs2]
lb rd, offset(rs1)	I	0x03	0x0		R[rd] ← SignExt(Mem(R[rs1] + offset, byte))
lh rd, offset(rs1)			0x1		R[rd] ← SignExt(Mem(R[rs1] + offset, half))
lw rd, offset(rs1)			0x2		R[rd] ← SignExt(Mem(R[rs1] + offset, word))
ld rd, offset(rs1)			ox3		R[rd] ← Mem(R[rs1] + offset, doubleword)
addi rd, rs1, imm		0x13	0x0		R[rd] ← R[rs1] + imm
slli rd, rs1, imm			0x1	0x00	R[rd] ← R[rs1] << imm
slti rd, rs1, imm			0x2		R[rd] ← (R[rs1] < imm) ? 1 : 0
xori rd, rs1, imm			0x4		R[rd] ← R[rs1] ^ imm
srli rd, rs1, imm			0x5	0x00	R[rd] ← R[rs1] >> imm
srai rd, rs1, imm			0x5	0x20	R[rd] ← R[rs1] >> imm
ori rd, rs1, imm			0x6		R[rd] ← R[rs1] | imm
andi rd, rs1, imm			0x7		R[rd] ← R[rs1] & imm
addiw rd, rs1, imm		0x1B	0x0		R[rd] ← SignExt((R[rs1](63:0) +SignExt(imm))[31:0])
Jalr rd, rs1, imm		0x67	0x0		R[rd] ← PC + 4
					PC ← R[rs1] + {imm, 1b'0}
ecall		0x73	0x0	0x000	(Transfers control to operating system)
					a0 = 1 is print value of a1 as an integer.
					a0 = 10 is exit or end of code indicator.
sb rs2, offset(rs1)	S	0x23	0x0		Mem(R[rs1] + offset) ← R[rs2][7:0]
sh rs2, offset(rs1)			0x1		Mem(R[rs1] + offset) ← R[rs2][15:0]
sw rs2, offset(rs1)			0x2		Mem(R[rs1] + offset) ← R[rs2][31:0]
sd rs2, offset(rs1)			0x3		Mem(R[rs1] + offset) ← R[rs2][63:0]
beq rs1, rs2, offset	SB	0x63	0x0		if(R[rs1] == R[rs2])
					PC ← PC + {offset, 1b'0}
bne rs1, rs2, offset			0x1		if(R[rs1] != R[rs2])
					PC ← PC + {offset, 1b'0}
blt rs1, rs2, offset			0x4		if(R[rs1] < R[rs2])
					PC ← PC + {offset, 1b'0}
bge rs1, rs2, offset			0x5		if(R[rs1] >= R[rs2])
					PC ← PC + {offset, 1b'0}
auipc rd, offset	U	0x17			R[rd] ← PC + {offset, 12'b0}
lui rd, offset		0x37			R[rd] ← {offset, 12'b0}
jal rd, imm	UJ	0x6f			R[rd] ← PC + 4
					PC ← PC + {imm, 1b'0}