

计算机体系结构（实验）

2022年·秋
江仲鸣



HITSZ 实验与创新实践教育中心
Education Center of Experiments and Innovations, HITSZ

• 实验项目

	实验项目名称	学时	分数
实验1	Pin入门及指令依赖距离分析	4	8
实验2	分支预测器设计	8	12
实验3	Cache设计	8	12
实验4	层次存储系统分析	4	8

• 实验成绩（共40分）

- 除实验1：验收30%，代码+报告70%
- **附加题**：在实验总分的基础上加分，满40分为止

• 实验指导书：<https://hitsz-cslab.gitee.io/arch/>

- 实验包及PPT下载：见实验指导书首页



计算机体系结构

实验1 PIN入门及指令依赖距离分析



HITSZ 实验与创新实践教育中心
Education Center of Experiments and Innovations, HITSZ

- 了解评估CPU特性和程序特性的基本方法
- 了解程序插桩分析的基本原理，掌握Pin插桩工具的编写和使用方法
- 了解指令依赖距离的概念，掌握使用指令依赖距离分布图分析程序特性的方法

- 1、Why Pin?

- 为了设计高性能的CPU/程序，必须充分了解其特性

- 评估方法
 - 软件模拟 建立CPU模型，在模型上运行测试程序
优点：信息更丰富
缺点：慢
 - 程序插桩 在测试程序中注入代码，在物理CPU运行
优点：快，容易实现
缺点：与模拟相比，获得的信息没那么丰富

• 1、Why Pin?

- 程序插桩
 - 源码插桩 直接修改测试程序的源码
 - 优点：直观
 - 缺点：需重新编译、效率低
 - 二进制插桩 直接在二进制可执行文件中注入代码
 - 优点：不需重新编译，效率高

- Pin: **动态二进制**插桩工具包

动 态——运行时注入，不生成额外的文件

二进制——直接对可执行文件进行插桩，不需重新编译



• 2、插桩原理

- 基本原理：将额外的代码插入到目标程序，以收集目标程序的信息。
- 插桩工具 = 插桩代码 + 分析代码

决定在哪里插入哪些代码

被插入的代码

- inscount0:

```
// This function is called before every instruction is executed
VOID docount() { icount++; }

// Pin calls this function every time a new instruction is encountered
VOID Instruction(INS ins, VOID *v)
{
    // Insert a call to docount before every instruction, no arguments are passed
    INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR)docount, IARG_END);
}
```



• 2、插桩原理

- 插桩工具也需考虑效率
- Pin插桩粒度：
 - 指令级插桩 ——以指令为插桩的基本单位，最细粒度
 - 轨迹级插桩 ——以基本块BBL为插桩的基本单位
 - 函数级插桩 ——镜像加载时，对整个函数进行遍历和插桩
 - 镜像级插桩 ——镜像加载时，对整个镜像进行遍历和插桩

• 2、插桩原理

- 插桩工具开销 { 插桩代码开销
分析代码开销 (**执行次数** × **执行开销**)

- 插桩代码：只在可执行文件的指令被**首次**执行时执行

- 分析代码：**每次**执行指令时都会执行

- 原则：尽可能减少分析代码**执行次数**、减小分析代码**执行开销**

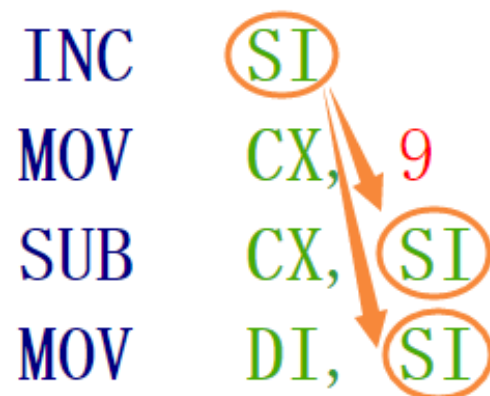
使用大粒度的插桩

简化分析代码

- 优化方法：分析代码任务转移、控制流消除、编译器优化, etc.

• 3、指令依赖距离

- **定义：** 一个寄存器被某条指令写入，且被后续指令读取时，这两条指令之间的指令数

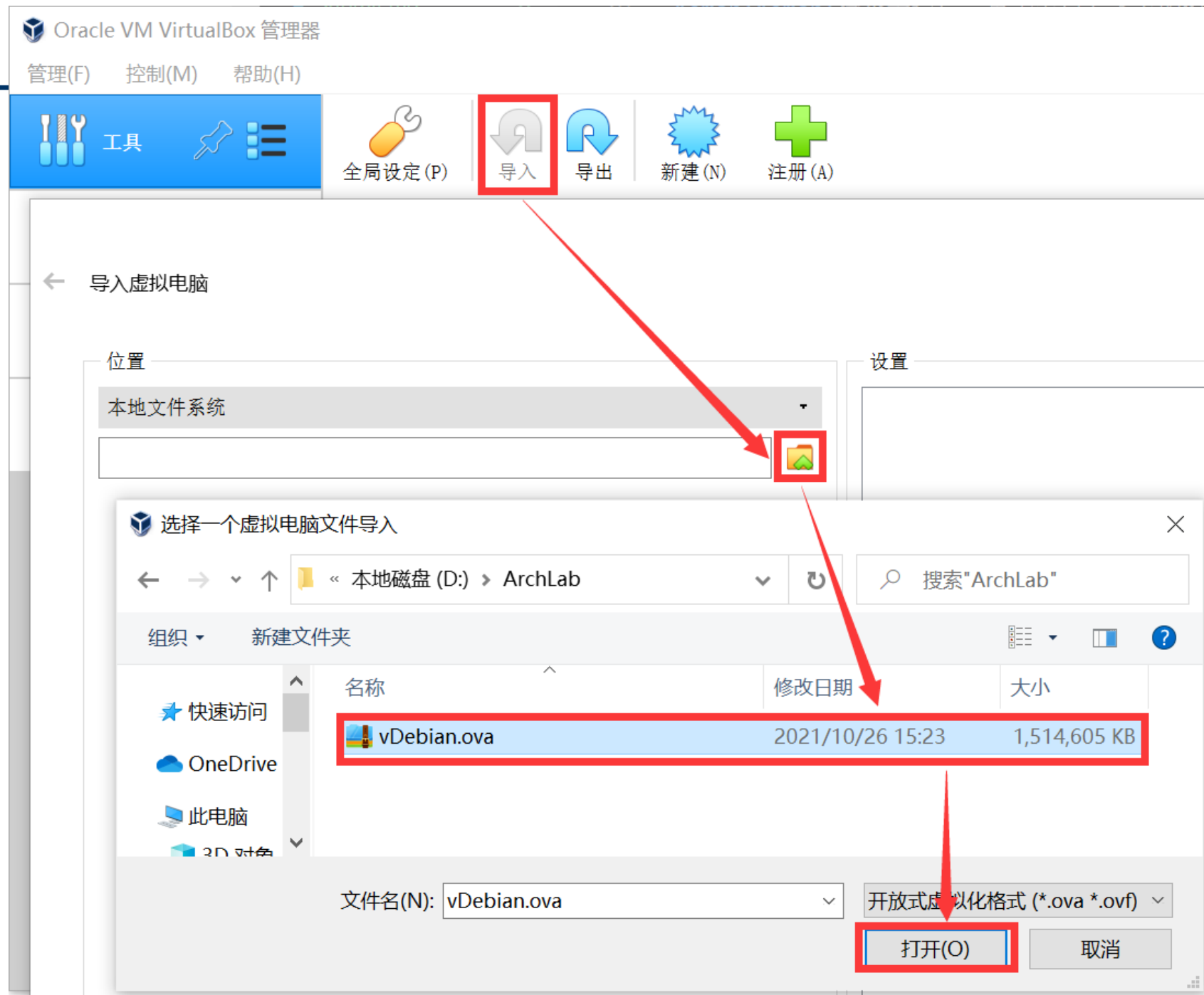


• 1. 导入虚拟机

虚拟机中包含有
实验1实验材料

✓ Pin-3.23

✓ workspace



- 2. 运行Pin

- (1) 进入范例程序的目录：source/tools/ManualExamples

```
debian@debian:~$ cd pin-3.24/  
debian@debian:~/pin-3.24$ cd source/tools/ManualExamples/ && ls  
buffer_linux.cpp      isampling.cpp  
buffer_windows.cpp    itrace.cpp  
countreps.cpp         little_malloc.c  
detach.cpp            makefile
```

- (2) 执行make命令编译插桩工具
- (3) 使用pin命令运行插桩工具

```
pin -t <toolname> [-o <output_file>] -- <target_executable>
```

```
$ ../../../../pin -t obj-intel64/inscount0.so -- pwd
```

- 3. 编写insDependDist插桩工具
 - (1) 参考范例程序，补全insDependDist.cpp中的代码
 - (2) 在makefile.rules中添加新的插桩工具
 - (3) 编译并运行，记录并分析实验结果
 - (4) 按指导书要求，完成实验报告



- 题目：插桩效率优化（+2分）
 - insDependDist工具基于指令级插桩粒度开发，效率较低
 - 内容：
 - 参考inscount1和inscount2源码，修改insDependDist代码，使其能够基于轨迹级粒度进行插桩，以提高效率
 - 要求：
 - 比较优化前后的插桩效率
 - 将设计思路、关键代码实现、测试方法及分析写入实验报告

- 无课堂检查
- 将**源码**、**实验报告**打包提交
 - 命名规则：**学号_姓名_ARCH实验1.zip**
 - 提交方法：<https://hitsz-cslab.gitee.io/arch/ojguide>
 - Deadline：下周同一上课时间前
- 附加题：将设计思路、关键代码等写入报告，与源码一起打包提交（**+2**分）

开始实验



HITSZ 实验与创新实践教育中心
Education Center of Experiments and Innovations, HITSZ