哈尔滨工业大学

**<<计算机体系结构>>**

**实验报告**

**(2021年度秋春季学期)**

|  |  |
| --- | --- |
| **姓名：** | **郑漫莎** |
| **学号：** | **180110711** |
| **学院：** | **计算机科学与技术** |
| **教师：** |  |

# 分析插桩工具的代码框架和执行过程

## 1.1 inscount0

如图1，图2所示：

INS\_AddInstrumentFunction用于添加指令级的插入指令函数。Instruction为插桩代码，即决定在哪里插入。Docount为分析代码，即被插入的代码。在inscount0.cpp中每条指令都进行INS\_InsertCall调用，在每条指令之前IPOINT\_BEFORE插入docount。插入指令为指令数加一。

PIN\_AddFiniFunction在程序退出之前执行Fini函数，即输入指令数icount到输出文件。

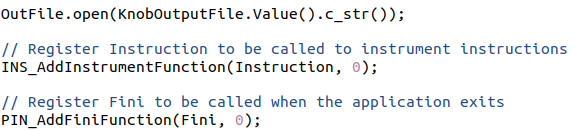


图1-1 inscount0.cpp main

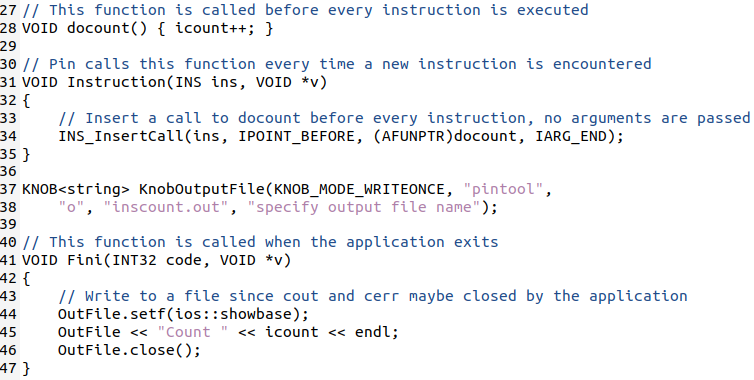


图1-2 inscount0.cpp 其余主要函数

## 1.2 inscount1

如图3，图4所示：

TRACE\_AddInstrumentFunction为轨迹级插桩函数。Trace函数为插桩代码，他在每个bbl块前面进行BBL\_InsertCall调用，参数为BBL\_NumIns(bbl)，即一个BBL块内的指令数。



图1-3 inscount1.cpp main

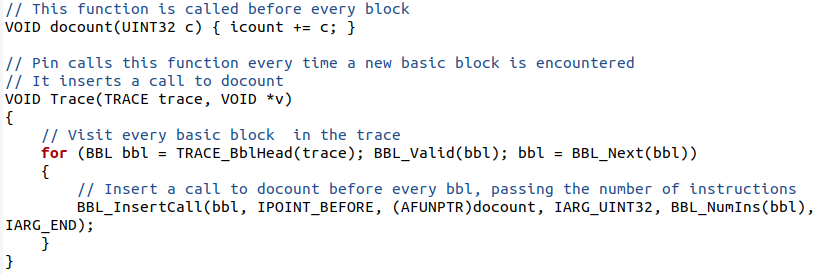


图1-4 inscount1.cpp 插桩代码和分析代码

## 1.3 inscount2

如图5所示：inscount2仍使用TRACE\_AddInstrumentFunction的轨迹级插桩函数进行插桩。插桩代码他在每个bbl块前面进行BBL\_InsertCall调用，参数为BBL\_NumIns(bbl)。IPOINT\_ANYWHERE为对插入指令的任何位置有效。PIN\_FAST\_ANALYSIS\_CALL是为了更快的对分析指令进行链接。

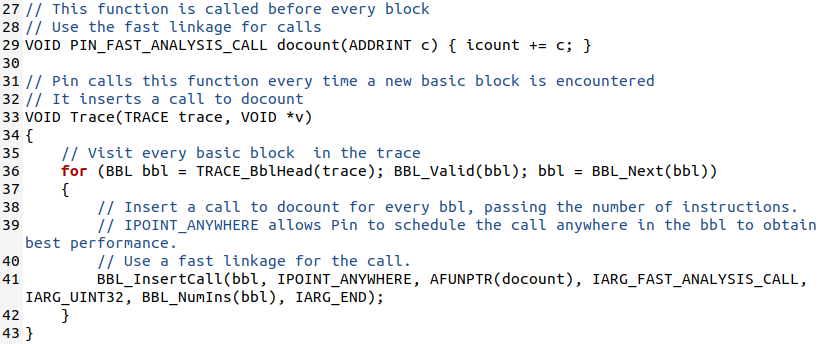


图1-5 inscount2.cpp 插桩代码和分析代码

# 插桩测试和统计

## /bin/ls

如图2-1，2-2所示为/bin/ls函数的指令依赖距离插桩测试结果。可以看出指令依赖距离大小最多的为1，其后呈断崖式下跌。指令依赖距离大小为1，2为超级频繁的一个等级，3-8相对频繁，9-18则相对常用，其后基本占比为0。在一定距离，4, 8, 12,16会有起伏。

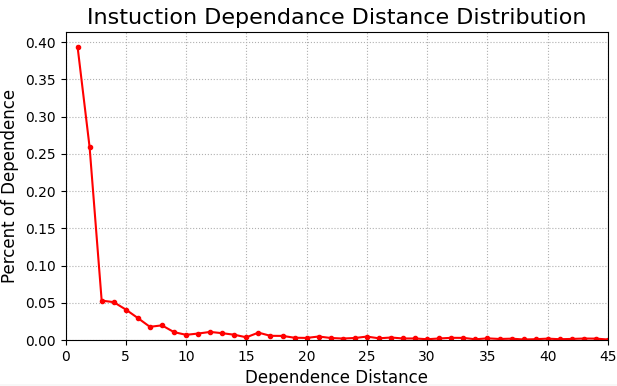


图2-1 /bin/ls 的指令依赖距离插桩测试结果

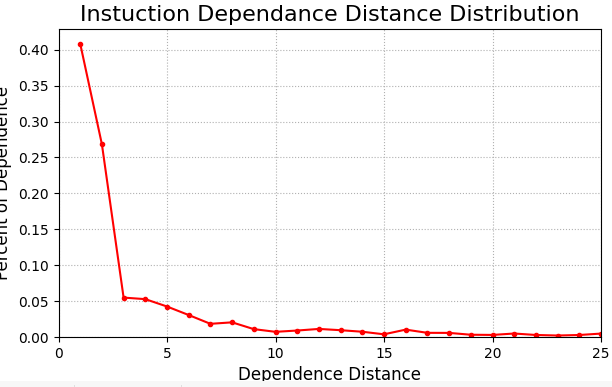


图2-2 /bin/ls 的指令依赖距离插桩测试结果，最大依赖距离25

## Pi

指令如图2-3。

如图2-4，2-5所示为pi函数的指令依赖距离插桩测试结果。可以看出指令依赖距离大小最多的为1，其后呈断崖式下跌。指令依赖距离大小为1，2为超级频繁的一个等级，3-8相对频繁，9-18则相对常用，其后基本占比为0。在一定距离，4, 8, 12,16会有起伏。

其分布和ls分布极为相似。

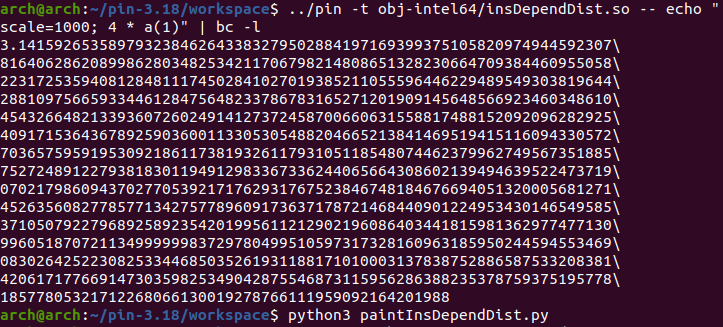


图2-3 pi插桩测试

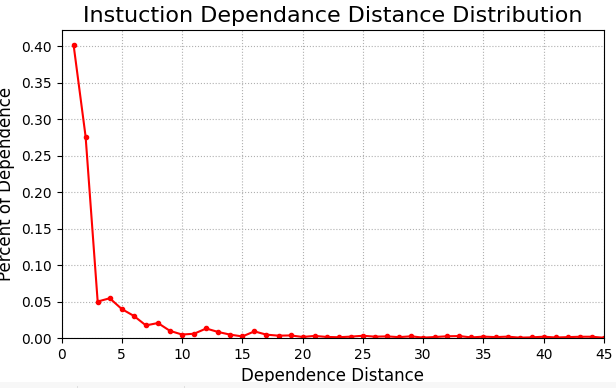


图2-4 pi的指令依赖距离插桩测试结果

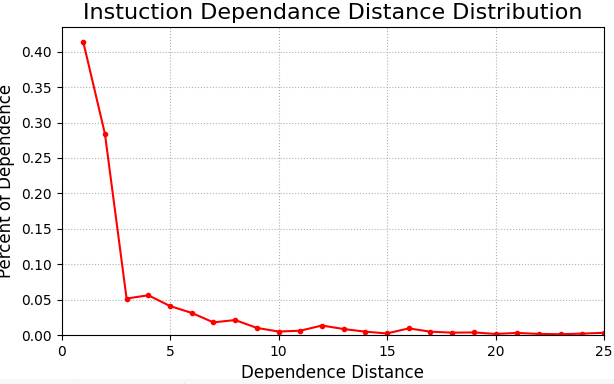


图2-5 pi的指令依赖距离插桩测试结果，最大依赖距离25

## Memtest

指令如图2-6。

如图2-7，2-8所示为memtest函数的指令依赖距离插桩测试结果。可以看出指令依赖距离大小最多的为1，其后呈断崖式下跌。在距离为4、5有明显的上升。在8和11也有小幅度的回调。

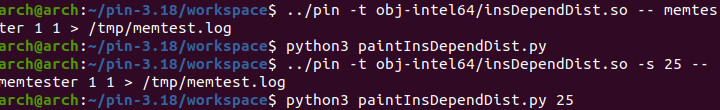


图2-6 Memtest的插桩测试

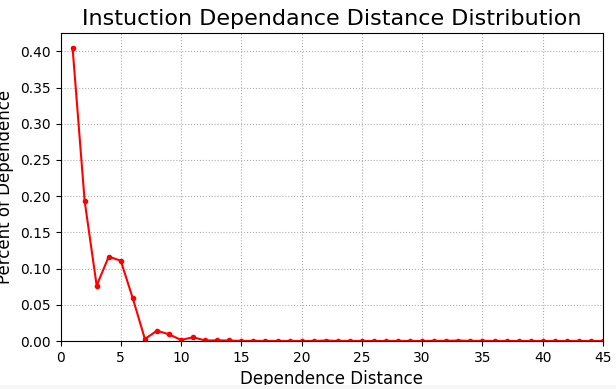


图2-7 memtest的指令依赖距离插桩测试结果

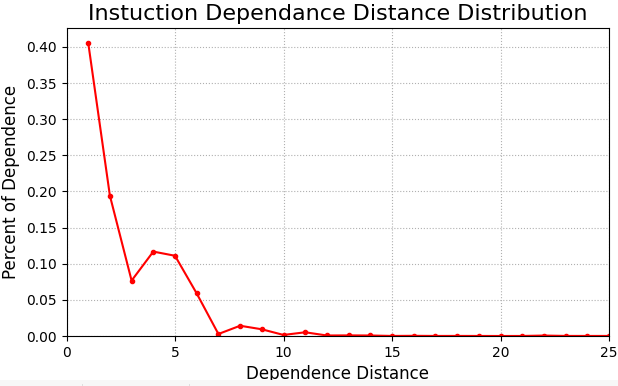


图2-8 memtest的指令依赖距离插桩测试结果，最大依赖距离25

# 思考题

## 指令依赖距离分布图很长的尾巴，可能是由哪些寄存器引起的，为什么？

ESP、EBP：函数调用的时候，CPU机制会通过push、pop指令自动调整ESP的值，同时将EBP的值进行更改。但是一个函数内的指令有很多，且函数内没有进行函数调用，则可能存在很长的函数依赖距离。同时，不同的函数调用，以及相同的函数调用，但是不同的参数可能导致的函数内的指令数不同。

## 架构A、B谁具有更多的寄存器，为什么？

架构B具有更多的寄存器。

因为架构B距离为1的指令相对于架构A要少很多，而距离为2的指令相对于架构A要多，即，相对于架构A，架构B的指令存在的写入后立即读取的状况较少。

可以推断，在架构A的指令对寄存器写后立即读，而后重复，才导致距离为1的分布占绝大多数。

而架构B的指令，在该情况下，可能是写入两个寄存器或三个寄存器后再进行读取。所以可以推断架构B的寄存器多。

## 不同方法解决流水线冲突，指令依赖距离分布图是否相同，为什么？

相同。停顿法是指在遇到数据冲突时，后续流水停顿一段时间，防止寄存器读后写的情况。数据转发法则是定向技术，在计算结果未出来之前，后面的指令不一定立马要用到该指令，将计算结果重定向，直接送到其他指令需要的地方，避免部分停顿。

指令依赖距离分布是以指令为单位进行计算，他依据指令写和读之间的指令间隔数进行统计和分析，而不是根据时钟周期进行判断。而停顿法和数据转发法都是通过对指令的运行方式的改变，来对指令的运行时间进行优化。但是对于指令本身而言，寄存器的读写情况并没有改变，所以，指令依赖距离也没有改变。