YOLO 卷积函数的 SMART 仿真

1.实验目的

通过将 YOLO 源码的卷积函数放在 SMART 上进行仿真和性能监测，进一步介绍嵌入式 C 裸机程序在 C910 上的开发过程和仿真原理。通过观测分支预测开关对程序性能的影响，理解分支预测在现代处理器中的重要性。通过指令分布统计，了解内存墙的概念，并进一步理解缓存在现代处理器中的重要性。

2.实验步骤

（1）文件核对和执行仿真

由于实验 7 中 Core Num 默认设置为 1，因此该部分无需改动；

在反汇编后查看 conv_test.s 文件内容，发现 guard_start()与 guard_end()函数的入口地址分别为 40'h1b50 和 40'h1c10，故在 tb/tb.v 文件中修改开头定义为：`define GUARD_START_PC 40'h1b10 以及 `define GUARD_END_PC 40'h1c10，删除 conv_test.s 后 bkill 当前任务再进行仿真

若反编译后未将 conv_test.s 删除则后续仿真会报错

（2）CPI、Cache 缺失率和分支预测统计

下图为 all prediction on 时的 run.log 截图：

```
        ******LOADING PROGRAM*********

num_cycle is 5131578

num_instret is 9449136

CPI is 0.543074

num_conditional_branch_mis is 5006

num_L1_Dcache_read_access is 3795212

num_L1_Dcache_read_miss is 14227

num_L1_Dcache_write_access is 1898553

num_L1_Dcache_write_miss is 1827

num_L2_Dcache_read_access is 470737

num_L2_Dcache_read_miss is 4164

num_L2_Dcache_write_access is 121008

num_L2_Dcache_write_miss is 399

guard_cycle_count is    5132230

guard_inst_count is    8724951
*********************************************
*    simulation finished successfully        *
*********************************************
```

三次仿真全部运行完毕可得到如下数据表格：

configure	all prediction on	BPE on, BTB off	all prediction off
cycle	5131578	5202663	8936645
insts	9449136	9449136	9449136
CPI	0.543074	0.550597	0.945763
conditional branch miss	5006	4439	253598
L1_Dread access	3795212	3780647	4579889
L1_Dread miss	14227	14250	12291
L1_Dread miss_rate	0.003748671	0.003769196	0.002683689
L1_Dwrite access	1898553	1897811	1920568
L1_Dwrite miss	1827	1858	1805
L1_Dwrite miss_rate	0.000962312	0.000979023	0.000939826
L2_Dread access	470737	469435	481926
L2_Dread miss	4164	4164	4183
L2_Dread miss_rate	0.008845704	0.008870238	0.008679756
L2_Dwrite access	121088	121027	122750
L2_Dwrite miss	399	399	400
L2_Dwrite miss_rate	0.003295124	0.003296785	0.003258656

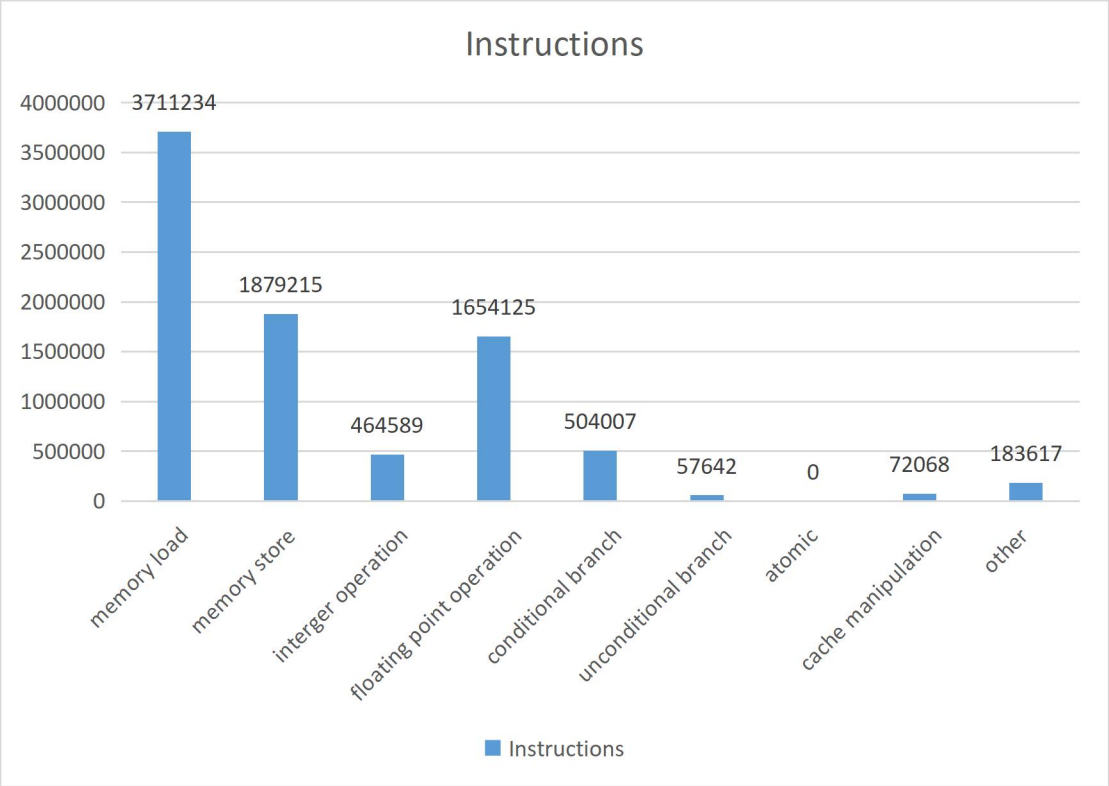
从中可看出 BTB off 对程序运行速度影响较小，BPE on 对程序运行速度影响大；结合下面的数据可发现，分支预测主要是减少了分支跳转错误来加快程序运行的速度，对 L1，L2 Cache 来说，三者的命中率几乎相等

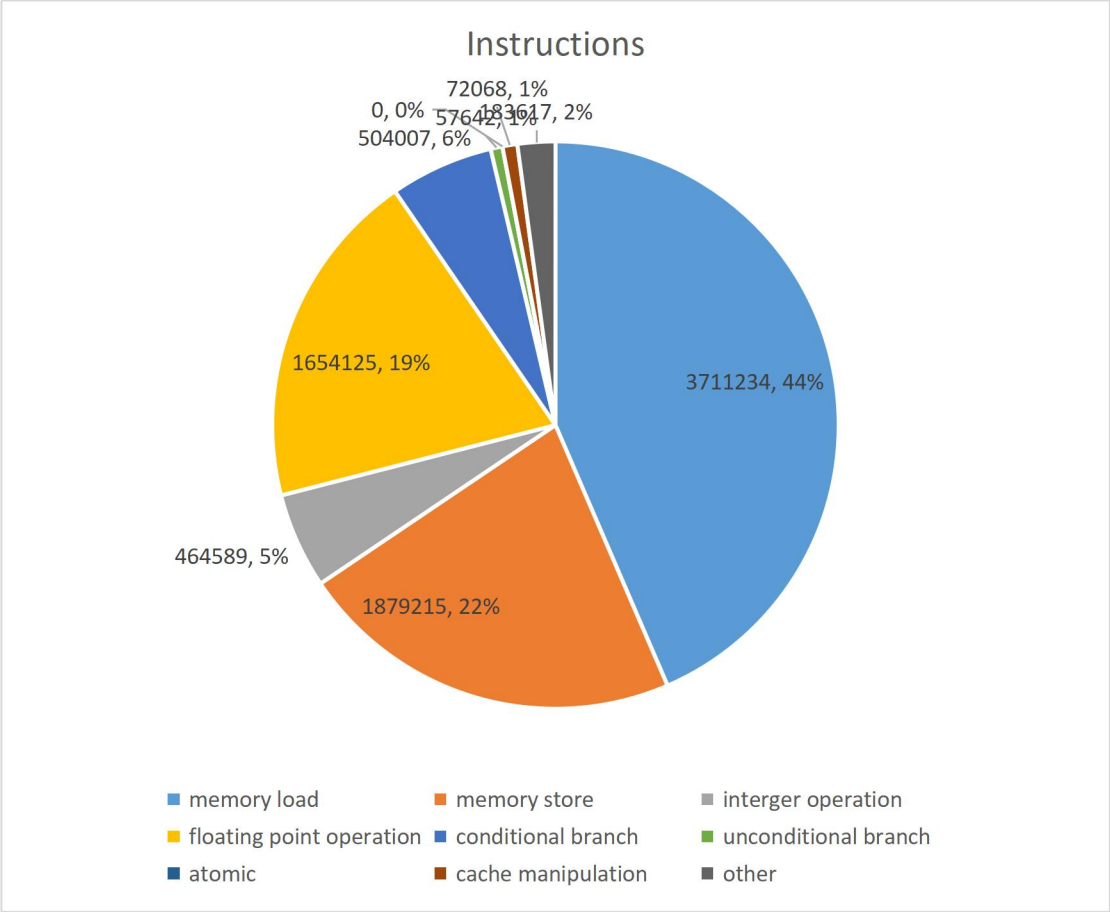
（3）指令分布统计绘图

inst_class 的文件内容：（all prediction on 情况下）

```
memory access instructions:
        memory load:    3711234
        memory store:   1879215

arithmetic  instructions:
        integer operation:      464589
        floating ponit operation:       1654125

branch instructions:
        conditional branch:     504007
        unconditional branch:   57642

atomic instructions:    0
cache manipulation instructions (C910 customized extension):    72068
other instructions:     183617

total:  8526497
```

根据该文件绘制出不同类型的指令数与占比如下：





3.实验分析与总结

本实验前文花大量篇幅详细阐述了 smart 平台上运行 C 语言程序的步骤，从源码编译，内存划分与链接开始引入，深入分析了以下几部分内容：

（1）编译阶段 Makefile 如何准备好程序代码段数据与输入的测试数据

（2）SMART 平台如何将载入在 CPU 里的程序（机器码）根据内存映射规则载入到 RTL 仿真 Memory 中

（3）使用 Verilog 的系统字符代替 printf 进行输出打印；使用 SMART 总线中闲置的内存模块，开辟出可存放大量输入数据的空间，之后在 C 源码中通过 volatile 直接为变量指定载

入地址，即可完成输出

之后针对性能分析，在熟悉的 run.log 给出的指标以外，通过在源码加入两个哨兵函数，可以精确解析运行程序所执行的指令总数，并根据 opcode 分析出不同指令类别

执行仿真后就可分别在 run.log 与 inst_count 中分析 CPI，Cache 缺失率，分支预测统计和指令类别数

4.实验收获

实验花大篇幅进行背景知识介绍，对于拥有一定 FPGA 开发经验的我来说，这部分内容从另一个方面阐述了 FPGA 嵌入式系统运行程序与 RTL 仿真平台运行程序的不同点。尽管这部分内容繁杂，限于篇幅实验介绍中也只能粗略介绍，但信息量足够多了。